

SECURE
GROUP
COMMUNICATIONS
Over Data Networks

XUKAI ZOU

BYRAV RAMAMURTHY

SPYROS S. MAGLIVERAS

Secure Group Communications over Data Networks

SECURE GROUP COMMUNICATIONS OVER DATA NETWORKS

XUKAI ZOU

Indiana University - Purdue University, Indianapolis
Indianapolis, IN 46202, USA

BYRAV RAMAMURTHY

University of Nebraska-Lincoln
Lincoln, NE 68588, USA

SPYROS S. MAGLIVERAS

Florida Atlantic University
Boca Raton, FL 33431, USA

Prof. Xukai Zou
Dept. of Computer & Information Science
Purdue University,
School of Science at Indianapolis
723 W. Michigan St. SL280E
Indianapolis, IN 46202 USA
xkzou@cs.iupui.edu

Prof. Byrav Ramamurthy
Dept. of Computer Science & Eng.
University of Nebraska-Lincoln
256 Avery Hall
Lincoln, NE 68588-0115 USA
byrav@cse.unl.edu

Prof. Spyros S. Magliveras
Dept. of Mathematical Sciences
Florida Atlantic University
Boca Raton, FL 33431 USA
spyros@fau.edu

Secure Group Communications Over Data Networks

Library of Congress Cataloging-in-Publication Data

A C.I.P. Catalogue record for this book is available
from the Library of Congress.

ISBN 0-387-22970-1

e-ISBN 0-387-22971-X

Printed on acid-free paper.

© 2005 Springer Science+Business Media, Inc.

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, Inc., 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use in this publication of trade names, trademarks, service marks and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed in the United States of America.

9 8 7 6 5 4 3 2 1

SPIN 11053323

springeronline.com

*This book is dedicated to our
wives Suqin, Bhuvana, and
Leanne.*

Contents

Dedication	v
List of Figures	xi
List of Tables	xv
Preface	xvii
Acknowledgments	xix
1. INTRODUCTION	1
1.1 Overview of Secure Group Communications	1
1.2 Preliminaries	3
1.3 Enabling Technologies	8
1.3.1 Multicast	8
1.3.1.1 Directed, Subgroup, Scoped and Multiple Multicasts	9
1.3.2 Cryptosystems	9
1.3.3 Two-party Diffie-Hellman Key Exchange	11
1.4 Group Dynamics and Security	13
1.5 Outline	15
2. TYPICAL GROUP KEY MANAGEMENT SCHEMES	17
2.1 Classification of Typical Group Key Management Schemes	18
2.2 Public-Key based Secure Group Communication Schemes	20
2.2.1 Reversible Parametric Sequence (RPS)	20
2.2.2 Secure Transmission Backbone (STB)	22
2.3 Secret-Key based Secure Group Communication Schemes	23
2.3.1 Core Based Tree (CBT)	23
2.3.2 Iolus	24
2.3.3 Dual Encryption Protocol (DEP)	25
2.4 Group Key Management based on Hierarchical Clusters	28

2.4.1	Layers, Clusters and Keys	28
2.4.2	Key Management	30
2.4.3	Clustering Protocol	33
2.5	N-party Diffie-Hellman Key Exchange Suite	37
2.5.1	ING Protocol	37
2.5.2	BD Protocol	39
2.5.3	GDH Protocols	39
2.5.3.1	GDH.1	39
2.5.3.2	GDH.2	41
2.5.3.3	GDH.3	42
2.5.4	STR Protocol	43
2.5.5	A Protocol without Member Serialization	45
2.5.6	Summarization of n -party Diffie-Hellman Protocols	46
3.	TREE BASED KEY MANAGEMENT SCHEMES	49
3.1	Centralized Key Distribution based on Tree Structures	50
3.1.1	Key Tree – Logical Key Hierarchy (LKH)	50
3.1.2	Bursty Behavior and its Efficient Implementation	52
3.1.2.1	Bursty Behavior and Properties	52
3.1.2.2	Bursty Algorithm	54
3.1.2.3	Theoretical Analysis	57
3.1.3	d – ary Key Tree	59
3.1.3.1	Member-Oriented Rekeying	60
3.1.3.2	Key-Oriented Rekeying	62
3.1.3.3	Group-Oriented Rekeying	63
3.1.3.4	Performance of Bursty Operation in d – ary Key Trees	64
3.1.4	One-way Function Tree (OFT)	65
3.1.5	One-way Function Chain (OFC)	66
3.1.6	Collusion Attacks on OFT and Improvement	69
3.1.7	Group Key Management based on Boolean Function Minimization Technique	72
3.2	Distributed Key Agreement based on Tree Structures	76
3.2.1	Tree based Group Diffie-Hellman Scheme (TGDH)	76
3.2.2	Block-Free TGDH Key Agreement (BF-TGDH)	78
3.2.2.1	BF-TGDH Principle	78
3.2.2.2	BF-TGDH Rekeying Operations	81
3.2.2.3	Performance and Security of BF-TGDH	82
3.2.3	DIstributed Scalable sEcure Communication (DISEC)	83
3.2.3.1	Discovery of the Neighbor	85

3.2.3.2	Key Association Groups	85
3.2.3.3	Join Operation	86
3.2.3.4	Leave Operation	88
4.	DYNAMIC CONFERENCING SCHEMES	91
4.1	Dynamic Conferencing and a Naive Solution	92
4.2	Public-Key based Dynamic Conferencing Scheme (PKDC)	92
4.3	Chinese Remainder Theorem based (Secure Lock) Dynamic Conferencing Scheme	93
4.4	Symmetric Polynomial based Dynamic Conferencing Scheme	94
4.4.1	Limited Symmetric Polynomial based DC Scheme	94
4.4.2	Extended Symmetric Polynomial based DC Scheme	95
4.5	Tree based Dynamic Conferencing Scheme	96
4.5.1	Key Tree based Interval Multicast and DC (IDC)	96
4.5.2	An Efficient and Scalable Key Tree based DC Scheme (KTDC)	98
4.6	BF-TGDH based Dynamic Conferencing (BF-TGDH DC)	101
4.7	Discussion and Comparisons	101
5.	SECURE GROUP COMMUNICATIONS WITH HIERARCHICAL ACCESS CONTROL	105
5.1	Classification	105
5.2	Unconditionally Secure Keying Schemes for HAC	106
5.3	One-way Function Schemes for HAC	108
5.3.1	The Akl-Taylor Scheme	109
5.3.2	Flexible Access Control with Master Keys	110
5.3.3	Lin's Scheme	112
5.3.4	Sandhu's Scheme	113
5.4	Index based Scheme for SGC with HAC	114
5.4.1	Principle	114
5.4.2	Key Management	116
5.4.2.1	Key Distribution	116
5.4.2.2	Subgroup Dynamics	116
5.4.2.3	Member Dynamics	117
5.4.3	Performance Analysis	118
5.4.4	Security Issues	119
5.4.5	Properties	121
5.5	CRT Based Scheme for SGC with HAC	122
5.5.1	CRTHACS Components and Initialization	122

5.5.2	Data Communication	124
5.5.3	Dynamic Key Management	125
5.5.4	Security and Performance Analysis	125
6.	SGC CHALLENGES AND SGC FOR WIRELESS NETWORKS	127
6.1	Factors Enabling SGC Functionality	127
6.1.1	Admission Control and Membership Management	127
6.1.2	Message/Packet Source Authentication	128
6.1.3	Coordination: Timing and Versioning in SGC	129
6.1.4	Broadcast Authentication	130
6.2	SGC in Wireless Environments	130
6.2.1	Topology Matching Key Management	132
6.2.1.1	Key Management for TMKM	133
6.2.2	Administration-scoped Key Management	135
6.2.3	SGC over Ad Hoc Networks	140
6.2.3.1	Securing Ad Hoc Networks with a Large Number of Nodes	140
6.2.3.2	Desired Properties for SGC over Ad Hoc Networks	145
7.	CONCLUDING REMARKS	149
7.1	Summary of Book Contents	149
7.2	Exemplary Applications	150
7.2.1	Secure Teleconferencing	150
7.2.2	Virtual Private Networks (VPN)	150
7.2.3	Secure Grid Computing	151
7.2.4	Secure Collaborative Work	151
7.3	Conclusion	151
	References	153
	About the Authors	161
	Index	165

List of Figures

1.1	Multicast: multicast capable routers make copies of a multicast packet.	9
1.2	Diffie-Hellman key exchange.	12
1.3	Man-in-The-Middle attack.	13
2.1	RPS scheme.	21
2.2	Secure Transmission Backbone.	22
2.3	Core in Core Based Tree.	24
2.4	Subgroups and GSIs in Iolus scheme.	24
2.5	Subgroups and key subgroups in DEP.	26
2.6	Layered clusters and keys for ten members.	29
2.7	Member D , a non-cluster leader, leaves.	31
2.8	Member E , a cluster leader, leaves.	32
2.9	Member F 's leave causes clusters to merge.	33
2.10	Member discovery: periodic multicast.	35
2.11	Member discovery: member join.	35
2.12	Member discovery: new parent discovery.	36
2.13	ING protocol: round $r \in [0, n - 2]$.	37
2.14	ING example for m_2 .	38
2.15	BD protocol: two rounds.	39
2.16	GDH.1 protocol: Stage 1 ($n - 1$ rounds), Stage 2 ($n - 1$ rounds).	40
2.17	GDH.1: an example for 5 members.	40
2.18	GDH.2 protocol: Stage 1 ($n - 1$ rounds), Stage 2 (one round).	41
2.19	GDH.2: an example for 5 members.	41

2.20	GDH.3 protocol: Four stages with stage 1 having $n - 2$ rounds.	42
2.21	GDH.3: an example for 5 members.	43
2.22	Steer's protocol: Stage 1 (broadcast), Stage 2 ($n-2$ rounds).	44
2.23	Steer's protocol: an example for 5 members.	44
2.24	STR protocol: Stage 1 (broadcast), Stage 2 (broadcast).	45
3.1	Key tree: each member assigned the keys from its leaf to the root.	50
3.2	m_2 leaves the group.	51
3.3	Member m_7 joins the group.	51
3.4	Bursty behavior: multiple joins and leaves simultaneously.	53
3.5	Binary key tree.	54
3.6	Changed keys are determined by the comparisons of neighboring members.	55
3.7	Rekeying-in-Key-Tree Algorithm.	57
3.8	Worst case scenario.	58
3.9	3 - ary key tree and join/leave operation.	61
3.10	Logical Key Hierarchy for 8 members.	67
3.11	LKH operation when m_0 leaves.	67
3.12	OFT operation when m_0 leaves.	68
3.13	OFC operation when m_0 leaves.	68
3.14	OFT collusion scenario 1.	70
3.15	OFT collusion scenario 2.	70
3.16	OFT collusion scenario 3.	71
3.17	OFT improvement.	71
3.18	Auxiliary Key Tree.	73
3.19	m_2 leaves.	74
3.20	m_0 and m_4 leave.	75
3.21	Tree based Group Diffie-Hellman key agreement.	77
3.22	Block-Free Tree based Group Diffie-Hellman key agreement for seamless SGC.	80
3.23	One-way function tree in DISEC.	84
3.24	Find_Neighbor Algorithm.	85
3.25	Find_Key_Association Algorithm.	86
3.26	Join operation in DISEC.	87
3.27	Leave operation in DISEC.	88

4.1	Key tree: each member assigned the keys from its leaf to the root.	97
4.2	Conference members are covered under nodes.	98
4.3	Algorithm to determine the keys of the conference in a key tree.	100
5.1	User multiple keying.	107
5.2	Resource multiple keying.	108
5.3	Mixed-Keying.	109
5.4	Assignment of primes, t_i s, and k_i s.	109
5.5	Assignment of primes, t_i s, and k_i s in the modified scheme.	110
5.6	Multiple root vertices and master keys.	111
5.7	Adding a new vertex to the access hierarchy.	111
5.8	Independent keys and the corresponding r_{ji} s.	112
5.9	Keys computed from their ancestors' keys in Sandhu's scheme.	113
5.10	Hierarchical key management using <i>indices</i> .	115
5.11	Members from subgroup 2, 11, and 13 collude.	120
6.1	A cellular network model.	132
6.2	Topology matching key tree.	133
6.3	ALX tree.	135
6.4	Two level rekeying.	136
6.5	Inter-area rekeying: Baseline Rekeying.	138
6.6	Inter-area rekeying: Immediate Rekeying.	138
6.7	Inter-area rekeying: Delayed Rekeying.	139
6.8	A pebblenet.	140
6.9	Pebble clusters.	142
6.10	Neighboring clusterhead discovery.	143
6.11	Backbone in clusters.	144

List of Tables

2.1	States for ING example	38
2.2	Comparison of n -party Diffie-Hellman protocols	47
3.1	Comparison of three rekeying strategies	64
4.1	Comparison of dynamic conferencing schemes	102
5.1	Performance characteristics in CRTHACS	126

Preface

The ubiquitous nature of the Internet is enabling a new generation of applications to support collaborative work among geographically distant users. Security in such an environment is of utmost importance to safeguard the privacy of the communication and to ensure the integrity of the applications.

'Secure group communications' (SGC) refers to a scenario in which a group of participants can receive and send messages to group members, in a way that outsiders are unable to glean any information even when they are able to intercept the messages. SGC is becoming extremely important for researchers and practitioners because many applications that require SGC are now widely used, such as teleconferencing, tele-medicine, real-time information services, distributed interactive simulations, collaborative work, grid computing, and the deployment of VPN (Virtual Private Networks). Even though considerable research accomplishments have been achieved in SGC, few books exist on this very important topic.

The purpose of this book is to provide a comprehensive survey of principles and state-of-the-art techniques for secure group communications over data networks. The book is targeted towards practitioners, researchers and students in the fields of networking, security, and software applications development.

The book consists of 7 chapters, which are listed and described as follows.

Chapter 1. This chapter presents an introductory overview of SGC. The chapter begins by introducing central problems of SGC such as group key management and membership management. It proceeds to introduce secure two party communication techniques and enabling technologies for SGC. Finally, an outline of the subsequent chapters of the book is presented.

Chapter 2. This chapter begins by classifying typical key management protocols based on different criteria such as *public key based* vs. *secret key based*, *centralized* vs. *distributed*, *unconditionally secure* vs. *computationally secure*, and *monolithic* vs. *subgroup-oriented*. Then it explains, in detail, several typical key management protocols including *Naive protocol*, *Secure lock*, *Reversible Parametric Sequence (RPS)*, *Secure Transmission Backbone (STB)*, *Core Based Tree (CBT)*, *Iolus*, *Dual Encryption Protocol (DEP)*, and a suite of *n-party Diffie-Hellman* schemes.

Chapter 3. This chapter focuses on a specific family of tree-based key management schemes. Tree-based key management is efficient and scalable, and can be used in one-to-many multicast applications as well as many-to-many group communications. Moreover, it can be centralized as well as distributed. This chapter discusses *Centralized key-tree*, *Centralized One-way Function Tree (OFT)*, *Distributed Scalable Secure Communication (DISEC)*, *Distributed Tree-based Group Diffie-Hellman key agreement (TGDH)* and *Block-Free Tree based Group Diffie-Hellman key agreement (BF-TGDH)*.

Chapter 4. In this chapter, we discuss a specific scenario of SGC, called *dynamic conferencing*, and related key management solutions. By ‘dynamic conferencing’, we mean that the members of an arbitrary subset of a group are able to form a privileged subgroup and communicate securely within the subgroup. This chapter introduces several schemes such as *naive solution*, *public key based dynamic conferencing*, *secure lock based dynamic conferencing*, *symmetric polynomials based dynamic conferencing*, and *key tree based dynamic conferencing*.

Chapter 5. This chapter presents another specific SGC scenario, namely *SGC with hierarchical access control (HAC)*, and related solutions. By ‘SGC with HAC’, we mean that members in a group are divided into a number of subgroups located at different privileged levels and the members in a higher-level subgroup can receive and decrypt messages from members in any of their descendant lower-level subgroups, but the reverse is not allowed. The following schemes for SGC with HAC are discussed in the chapter: *Akl-Taylor’s scheme*, *Lin’s scheme*, *Sandhu’s scheme* and *Chinese Remainder Theorem based scheme*.

Chapter 6. Apart from group key management, there are other issues which are necessary or helpful for SGC, include *membership management*, *access control*, *message/source authentication*, and *non-repudiation*. Chapter 6 addresses some of these topics. Since wireless/mobile networks (and, in particular, ad hoc networks) are becoming increasingly prevalent, SGC will be widely used in wireless environments. However, the specific properties and limitations of wireless/mobile networks pose many new problems in deploying SGC in these environments. The chapter addresses these challenging problems and proposes certain guidelines for solving them.

Chapter 7. This chapter summarizes the topics in the book and concludes by discussing several typical SGC applications including *secure tele-conferencing*, *secure collaborative work*, *virtual private networks*, and *secure grid computing*.

As a note to the reader, we wish to mention that we have chosen not to pursue a high degree of integration in the subject matter of this book. Thus, the reader will find different nomenclature, terminology and treatment for similar, or even identical entities, such as, for example, the *Trusted Authority (TA)*, versus the *Group Controller (GC)*, etc. We have stayed rather close to the original presentation of research papers, seeking integration of material in terms of scope, but without identifying and fusing commonalities. In time, when the SGC subject matter matures, further integration and identification of equivalent notions might be desirable, and even achievable.

We hope you will enjoy reading this book!

Acknowledgments

During the writing of this book we have been helped by many friends and colleagues to whom we are indebted for their comments, criticisms and suggestions. Particular thanks go to Professor Jean-Camille Birget and Professor Guevara Noubir for research collaboration with us in the areas of secure group communications and hierarchical access control. Many thanks are due to all authors whose research results have been included in the book, and to Mr. Alex Greene and Ms. Melissa Sullivan of Kluwer Academic Publishers for their patience and help with the publication of the book. We are indebted to Mr. Ravi Kumar Balachandran who read and edited the manuscript, provided many suggestions, and drew figures. We are grateful to Professor Wandu Wei, Messrs. Daniel Socek and Michal Sramka as well as Ms. Deborah Derrick for reading segments of the text and making useful and significant suggestions. Thanks also to Messrs. Sarang Deshpande, Amandeep Thukral, and Yogesh Karandikar for drawing figures. Finally, our special thanks go to the U.S. National Science Foundation for partial support of this work under grant CCR-0311577.

Chapter 1

INTRODUCTION

“*Secure group communications*” (SGC) refers to a scenario in which a group of participants can send and receive messages to group members, in a way that outsiders are unable to glean any information even when they are able to intercept the messages. Secure group communications rely on the protocols and fast developing theoretical tools of modern cryptology.

1.1 Overview of Secure Group Communications

With the exponential growth in modern communications, SGC is becoming an extremely important research area because of the need to provide privacy and authentication in communications. Many applications that require SGC are now widely used, such as teleconferencing, tele-medicine, real-time information services, distributed interactive simulations, collaborative work, interactive games and the deployment of VPN (Virtual Private Networks).

Because of the importance of SGC, the initial research group SMuG (Secure Multicast Research Group) in IRTF (Internet Research Task Force) for secure multicast has included SGC as its critical research area and a working group, MSEC (Multicast SECURITY) in IETF (Internet Engineering Task Force) was established in 2001 to standardize and provide drafts for SGC. Moreover, another research group GSEC (Group SECURITY) in IRTF was established in 2001 to investigate problems specific to SGC.

Group communications are implemented using broadcast or multicast mechanisms, such as IP (Internet Protocol) multicast, to provide efficient transmission of group messages. Similar to ‘secure two-party communication’ (STPC), security of group communication is enforced by cryptographic techniques such

as *encryption*, *signatures*, *authentication* and *integrity*. However, because of the intrinsic differences between two-party communications and group communications, techniques, such as key agreement for STPC cannot be directly used for SGC and there are many more problems which need to be solved before SGC can be used in real systems and applications. As a result, even though mature techniques exist for STPC, including many standards and systems, there are currently no standards and available systems for SGC. A few current implementations for SGC such as SecureRing [Kihlstrom et al., 1998], Horus/Ensemble [Rodeh et al., 2001, Rodeh et al., 1998], Rampart [Reiter, 1994], and API's such as CLIQUES [Steiner et al., 1997] and SPREAD (also Secure SPREAD) [Amir et al., 2003] are primarily for research/experimental purpose, and hopefully some of them can be deployed broadly and adapted as standards in the future. There is a considerable number of articles and reports about SGC techniques in the literature. However, there is a dearth of comprehensive books which provide an integrated view of the state-of-the-art in SGC. The objective of this book is to alleviate this problem.

We note that the terms *set* and *group* are synonymous in SGC literature, and so are the terms *subset* and *subgroup*. Thus, *group* and *subgroup* as used here do not have the standard algebraic meaning.

Like in STPC, the secrecy/privacy of group communications is achieved by encrypting group messages with a shared secret, called *group key* in group communication. The group key is only accessible to group members and thus, only group members are able to decrypt the messages. Therefore, the first (and most important) problem facing SGC is *key management*, that is, the methodology that enables group members to establish and distribute shared group keys. A primary feature in SGC is *group dynamics* (also called *member dynamics*). For two-party communications, after the establishment of a communication session, if either party leaves or stops the conversation, the session terminates. However, during a group communication session, members can join or leave the session or group at any time, while the group communication continues. Whenever members join and/or leave the group, the group key needs to be changed in order to guarantee secrecy for the residual, continuing communication group. How to change the key, both efficiently and scalably, is a considerable challenge.

There are some other scenarios related to SGC, which require group key management, such as *dynamic conferencing* and *SGC with hierarchical access control*. 'Dynamic conferencing' refers to a situation where there is a universal group of participants¹ U and any subset S of participants in U is able to form

¹The terms *participants*, *members*, *users*, and *communicants* are used interchangeably in the book.

a privileged subgroup. We call each possible subgroup S a *conference*. The participants in a conference S are able to communicate in such a way that any participant not in the conference (and consequently anyone outside U) is unable to glean any messages directed to the conference. It is required, moreover, that communications among different conferences will not interfere with one another.

‘Secure group communication with hierarchical access control (SGC with HAC)’ refers to a scenario where a universal group U of members is divided into a number of subgroups located at different privilege levels so that a high-level subgroup S can receive and decrypt messages directed to S or any of its descendant lower-level subgroups, while lower-level subgroups are not able to decrypt messages directed to parent subgroups. HAC is generally enforced using cryptography based techniques [Birget et al., 2001, Zou et al., 2003] i.e., cryptographic keys play a primary role in access control. Here, members in a higher level subgroup S possess or are able to derive the key of a descendant subgroup T , and consequently are able to access all messages communicated within T . In this volume we discuss typical key management protocols for *dynamic conferencing* and *SGC with hierarchical access control*.

Apart from group key management for SGC, there are some other issues which are necessary or helpful for SGC such as membership management, admission control, message/source authentication (MSA), and non-repudiation. Moreover, SGC in wireless environments is a new application paradigm. We will point out the challenging problems facing all these aspects of SGC and suggest possible solutions in the book.

1.2 Preliminaries

In this section, we present basic concepts and preliminaries used in the rest of the book. These include one-way functions, (one-way) hash function, the Chinese Remainder Theorem (CRT), the Discrete Logarithm Problem (DLP), and symmetric polynomials.

DEFINITION 1.1 A function $y = f(x)$ is said to be a one-way function if it is easy to compute y from x however it is computationally difficult to compute a preimage x given y .

DEFINITION 1.2 A (one-way) function $y = f(x)$ is said to be a (cryptographic) hash function if it is a mapping from a bit-string of arbitrary length to a bit-string of fixed length (i.e., $\{0, 1\}^* \rightarrow \{0, 1\}^n$) and satisfies the following three properties:

- (1) One-way, that is: given $y \in \{0, 1\}^n$, it is difficult find an $x \in \{0, 1\}^*$ such that $y = f(x)$.
- (2) Matching resistant, that is: given $x \in \{0, 1\}^*$, it is difficult to find a $x' \in \{0, 1\}^*$ such that $f(x') = f(x)$

(3) *Collision resistant, that is, it is difficult to find $x, x' \in \{0, 1\}^*$ such that $f(x') = f(x)$.*

The most important feature of a cryptosystem or secure group communication scheme (SGCS) is its degree of security, i.e. its strength in preventing opponents (attackers) from breaking the system. There are two basic approaches which can be used in discussing the security of a cryptosystem (or an SGCS): *computational security* and *unconditional security*.

DEFINITION 1.3 *A cryptosystem is defined to be computationally secure if the best algorithm for breaking it requires at least N operations, where N is some specified, very large number. On the other hand, a cryptosystem is defined to be unconditionally secure if it cannot be broken by attackers, even if the attackers collude and have infinite computational resources [Stinson, 1995].*

Unconditional security implies that when an opponent does not know the secret key k , then k appears to him as a random element in the key space; that is, the opponent has no way of distinguishing the key k from any other element in the key space. As for computational security, in general, a computationally secure cryptosystem utilizes one-way functions to implement secure communications and other cryptographic protocols. A prominent example is a public-key based system, which is always computationally secure, but never unconditionally secure. On the contrary, secret-key based systems generally belong to the category of unconditional security.

Another approach used to discuss the security of secure group communications (with hierarchical access control), is that of *k-resilient security*.

DEFINITION 1.4 *A scheme (system) is said to have k-resilient security if it is secure against the collusion of any subset of up-to k opponents but cannot defend against the collusion of some $k + 1$ opponents.*

For *k-resilient security*, the value k is a system security parameter and can be set/selected during the system setup. In general, the larger the k is, the more secure a system will be. In contrast, the system will become more inefficient in terms of both space and time complexity.

There may exist certain scenarios where a group of users are required to decrypt an encrypted message, rather than a single user. A threshold cryptosystem [Desmedt and Frankel, 1989] can be used in such situations. A (t, n) threshold scheme does not reveal a secret S unless any t out of n participants, or *shadowholders* work together. Each participant i will have a unique shadow k_i which he/she must keep secret. When any $t-1$ shadowholders work together, they cannot receive any information about the secret S . In this way, a secret can be shared by many people. If a share is burned in a fire or someone forgets

his/her shadow, there should be enough shareholders to recover the secret. An example of a threshold cryptosystem is a scheme based on Lagrange interpolation developed by Shamir [Shamir, 1979].

The Discrete Logarithm Problem (DLP) is the basis of many cryptographic primitives including the ElGamal public-key cryptosystem [ElGamal, 1985, Stinson, 1995], the Diffie-Hellman key exchange protocol, and the ElGamal signature scheme [ElGamal, 1985, Stinson, 1995]. The following is its definition [Stinson, 1995].

DEFINITION 1.5 *Let G be a cyclic group of very large finite order n , which without loss of generality we denote multiplicatively. Let $\alpha \in G$ be a generator of G , thus, for any element $\beta \in G$, there is a unique positive integer a , $0 \leq a \leq n - 1$ such that $\beta = \alpha^a$. The related Discrete Logarithm Problem (DLP) can be stated as follows: Given the group G , the generator α and an element $\beta \in G$, determine the unique $a \in [0, n - 1]$ such that $\beta = \alpha^a$. We denote the integer a by $\log_\alpha \beta$.*

We wish to point out that the intractability (or otherwise) of DLP is not intrinsically a group-theoretic property, but rather depends on the representation of the particular cyclic group G . Note for example that every cyclic group of order n is isomorphic to the additive group \mathbb{Z}_n under addition modulo n . However, if α is a generator of \mathbb{Z}_n then $\gcd(\alpha, n) = 1$, and if β is an arbitrary element of \mathbb{Z}_n , the discrete logarithm problem now becomes $a\alpha = \beta$. However, since $\gcd(\alpha, n) = 1$, α has a multiplicative inverse, say γ , in the ring $(\mathbb{Z}_n, +, \cdot)$, which can be efficiently computed by means of the Euclidean algorithm. We then have that $a = \beta\gamma \bmod n$, and the DLP has a trivial solution, independently of n .

In general, the DLP is intractable in the cyclic multiplicative group \mathbb{F}^* of a finite field $\mathbb{F} = \mathbb{F}_q = GF(q)$ of order $q = p^m$. DLP is also generally intractable in very large cyclic components of a chosen elliptic curve \mathcal{E} over a finite field \mathbb{F}_q . In both the finite field and elliptic curve case the order of the cyclic group must be such that certain well known attacks cannot be mounted.

In the case where we work with a field \mathbb{Z}_p of prime order p , The DLP is described in a simpler way as follows:

DEFINITION 1.6 *Let p be a large prime and $\alpha \in \mathbb{Z}_p$ a primitive element (i.e., generator) of \mathbb{Z}_p^* . The question is: given any $\beta \in \mathbb{Z}_p^*$, find the unique integer a , $0 \leq a \leq p - 2$ such that $\alpha^a \equiv \beta \pmod{p}$. As before we denote the integer a by $\log_\alpha \beta$.*

In the latter case also, care should be exercised in the choice of p so that certain known attacks will not compromise the DLP. It is rather clear that for secure choices of the parameters, the DLP problem induces, in fact, a one-

way function. Thus, any system or protocol based on DLP is *computationally secure*.

Here is a simple example of DLP. Let $p = 17, \mathbb{Z}_p^* = \{1, 2, \dots, 16\}$ and $\alpha = 3$. Given $a = 10, \beta = 3^{10} \bmod 17 = 8$ is easy to calculate. But given $\beta = 8$ it is difficult to calculate a . When p is a large prime number then the problem becomes very time consuming to solve.

The Chinese Remainder Theorem (CRT) is used as a theoretical foundation and practical tool in many cryptosystems as well as schemes for secure group communications, such as the Secure Lock [Chiou and W.T.Chen, 1989]. We review the CRT below as follows: (See also [Stinson, 1995].)

THEOREM 1.7 *Suppose N_1, \dots, N_r are pairwise relatively prime positive integers, i.e., $\gcd(N_i, N_j) = 1$ if $i \neq j$, and let $N = N_1 \cdot N_2 \cdots N_r$. For any given integers a_1, \dots, a_r , consider the following system of congruences:*

$$\begin{aligned} x &\equiv a_1 \pmod{N_1} \\ x &\equiv a_2 \pmod{N_2} \\ &\vdots \\ x &\equiv a_r \pmod{N_r}. \end{aligned}$$

Then the system has a unique solution modulo N , namely :

$$x \equiv \sum_{i=1}^r a_i n_i y_i \pmod{N}$$

where $n_i = N/N_i$ and $y_i = n_i^{-1} \pmod{N_i}$ (i.e. y_i is the multiplicative inverse of n_i modulo N_i). On the other hand, given any $x \in \mathbb{Z}_N$, a_1, \dots, a_r can be uniquely computed as $a_i \equiv x \pmod{N_i}$.

Let us look at an example of CRT. Suppose $r = 3, N_1 = 5, N_2 = 12$ and $N_3 = 19$. Then $N = 1140$. We compute $n_1 = N/N_1 = 228, n_2 = N/N_2 = 95, n_3 = N/N_3 = 60$, and then $y_1 = 2, y_2 = 11$ and $y_3 = 13$. Suppose $a_1 = 3, a_2 = 7$ and $a_3 = 10$, i.e.,

$$\begin{aligned} x &\equiv 3 \pmod{5} \\ x &\equiv 7 \pmod{12} \\ x &\equiv 10 \pmod{19} \end{aligned}$$

Then $x = ((228 \cdot 2 \cdot 3) + (95 \cdot 11 \cdot 7) + (60 \cdot 13 \cdot 10)) \bmod 1140 = (1368 + 7315 + 7800) \bmod 1140 = 16483 \bmod 1140 = 523$. Conversely, from $x = 523$, it is easy to compute $a_1 = 523 \bmod 5 = 3, a_2 = 523 \bmod 12 = 7$, and $a_3 = 523 \bmod 19 = 10$.

We will be using the CRT to fuse partial key information in constructing group keys.

RSA is one of the most widely-used public-key cryptosystems. RSA is based on the fact that no efficient algorithms are known (except on a quantum computer) to find the factorization of a large integer as the product of prime factors. In contrast, the problem of determining whether a given integer is prime (or composite) can be solved efficiently by both probabilistic algorithms [Koblitz, 1994, Menezes et al., 1996], and recently by a deterministic algorithm [Agrawal et al., 2002]. In particular, if $n = pq$ is the product of two primes, the multiplicative group of units \mathbb{Z}_n^* , of the ring \mathbb{Z}_n , has order $\phi(n) = (p-1)(q-1)$, where $\phi(\cdot)$ denotes Euler's ϕ function. It is now easy to see that knowledge of $\phi(n)$ is equivalent to knowing the factorization of n . If $x \in \mathbb{Z}_n^*$ then $x^{\phi(n)} \equiv 1 \pmod{n}$, (Fermat's little theorem), and RSA is based on this fact.

Because the RSA cryptosystem involves arithmetic with very large integers, the encryption and decryption operations are rather slow and the system is used mostly in a large number of key management and other cryptographic protocols rather than for message encryption. It is important to note that from a theoretical point of view RSA can be seen as a system based on the fact that the order of the underlying group (\mathbb{Z}_n^*) is not publicly known.

The formal definition of RSA is as follows.

DEFINITION 1.8 *Let $n = pq$ where p and q are large primes. Let the ciphertext space and the plaintext space be both \mathbb{Z}_n , and define*

$$\mathcal{K} = \{(n, p, q, a, b) : ab \equiv 1 \pmod{\phi(n)}\}.$$

For $K = (n, p, q, a, b)$, define

$$\text{the encryption rule as } e_K(x) = x^b \pmod{n}$$

and

$$\text{the decryption rule as } d_K(y) = y^a \pmod{n}$$

($x, y \in \mathbb{Z}_n$). The values n, b comprise the public key, and the values p, q, a form the private key.

It is rather interesting to note that $x^{ab} \equiv x \pmod{n}$ for all $x \in \mathbb{Z}_n$ rather than just the units of \mathbb{Z}_n .

Secret sharing schemes are secure schemes which recover a *secret* (the key) from an appropriate number of *shares* belonging to particular privileged subsets S_i of participants. Combining the shares of all the members of such a privileged subset S_i yields complete knowledge of the secret, but combining shares of members of any proper subset $T \subset S_i$ yields absolutely no information about the secret. Some secret sharing schemes [Menezes et al., 1996, Stinson, 1995] use symmetric polynomials as their theoretical basis. The definition of symmetric polynomials is as follows.

DEFINITION 1.9 A polynomial

$$f(x_1, \dots, x_n) = \sum_{i_1=0}^{i_1=w} \cdots \sum_{i_n=0}^{i_n=w} a_{i_1, \dots, i_n} x_1^{i_1} \cdots x_n^{i_n}$$

is called a symmetric polynomial if $a_{i_1, \dots, i_n} = a_{\pi(i_1), \dots, \pi(i_n)}$ for all permutations π of $\{1, \dots, n\}$.

For example, suppose $n = 3$ and $w = 2$. Let a_{i_1, i_2, i_3} be as follows:

$$\begin{aligned} a_{i_1, i_2, i_3} &= 0 \quad \text{except for :} \\ a_{0,0,0} &= 13 \\ a_{0,0,1} &= a_{0,1,0} = a_{1,0,0} = 3 \\ a_{0,0,2} &= a_{0,2,0} = a_{2,0,0} = 7 \\ a_{0,1,1} &= a_{1,0,1} = a_{1,1,0} = 4 \\ a_{0,1,2} &= a_{0,2,1} = a_{1,0,2} = a_{1,2,0} = a_{2,0,1} = a_{2,1,0} = 8 \\ a_{0,2,2} &= a_{2,0,2} = a_{2,2,0} = 9 \\ a_{1,2,2} &= a_{2,1,2} = a_{2,2,1} = 11 \\ a_{2,2,2} &= 5 \end{aligned}$$

Then the following polynomial is a symmetric one.

$$\begin{aligned} f(x, y, z) = & 13 + 3(x + y + z) + 7(x^2 + y^2 + z^2) + 4(xy + xz + yz) + \\ & 8(xy^2 + xz^2 + yz^2 + x^2y + x^2z + y^2z) + \\ & 9(x^2y^2 + x^2z^2 + y^2z^2) + \\ & 11(xy^2z^2 + x^2yz^2 + x^2y^2z) + 5x^2y^2z^2 \end{aligned}$$

1.3 Enabling Technologies

Certain underlying technologies enable group communication both securely and efficiently, such as *multicast* and *cryptosystems*. These technologies are discussed in this section.

1.3.1 Multicast

Multicast is a packet transmission mechanism that propagates a data packet to multiple receivers. Instead of sending a separate copy of the packet to each individual receiver, a sender just transmits one copy of the packet. The multicast-capable routers make copies of the packet on its way from the sender to multiple receivers located at different segments of the network topology (See Figure 1.1). Multicast provides an efficient message transmission support for SGC since a message directed to a group S needs to be transmitted to all members of S .

The IP multicast protocol has been in use for many years. Thus, SGC spanning geographically large areas over the Internet can be operated by utilizing IP multicast. When all participants in an SGC system are located within a single LAN, the broadcast capability inherently provided by LANs supports the efficient transmission of group messages to group members.

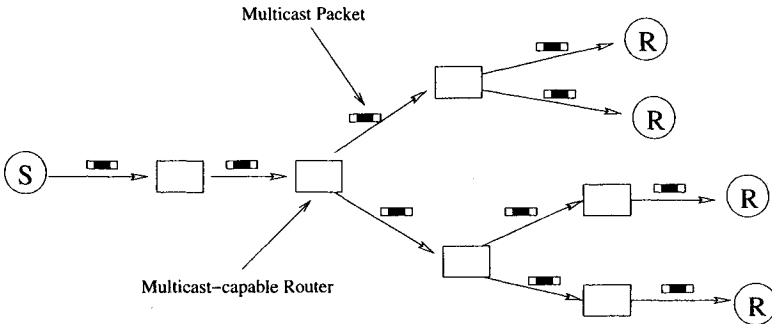


Figure 1.1. Multicast: multicast capable routers make copies of a multicast packet.

1.3.1.1 Directed Multicast, Subgroup Multicast, Scoped Multicast and Multiple Multicasts

In the above generic multicast, when a sender multicasts a packet to multiple receivers, a multicast delivery tree is formed from the sender to all the receivers, by connecting via multicast-capable routers. In the rekeying period of secure group communication, there exist scenarios in which the rekeying materials are supposed to reach certain members. Thus some other multicast mechanisms are more efficient [Banerjee and Bhattacharjee, 2002] than (generic) multicast. *Directed multicast* allows a sender to multicast a packet to individual subtrees rooted at a specific router on the multicast delivery tree. This is extremely useful for many network services such as NAK (negative acknowledgment) suppression. Even though directed multicast is not currently a part of the IP multicast standard, it has been proposed in research, e.g., AIM [Levine and Garcia-Luna-Aceves, 1997] and is currently being considered by the IETF as part of the PGM [Speakman et al., 2000] and GRA [Cain et al., 2001] efforts. *Subgroup multicast* provides a flexible capability that a message can be multicast to any specific subset of members. It would be particularly important and efficient for reliable multicast and rekeying of subset members in SGC if the network could support subgroup multicast. *(TTL) Scoped multicast* is a mechanism that uses TTL-scoping (Time-To-Live) to limit packets to a certain part of the multicast tree. In general, the network load for using scoped multicast is significantly higher than pure directed multicast. *Multiple multicast* implies using a different multicast address for each subset of nodes that have to be addressed. TTL-scoped multicast can be applied to each of multiple multicasts.

1.3.2 Cryptosystems

There are two kinds of cryptosystems: *secret-key* (or *symmetric*) *cryptosystems* and *public-key* (or *asymmetric*) *cryptosystems*. In a secret-key cryptosys-

tem, the sender and receiver share a common secret key k , belonging to a very large set of possible keys, the *key space* \mathcal{K} . Each key $k \in \mathcal{K}$ uniquely determines functions e_k (the *encryption transformation*) and d_k (the *decryption transformation*). The bijections e_k and d_k are inverses of each other under composition of functions, and each can easily be derived from the other. Therefore secret-key cryptosystems are also called *symmetric cryptosystems*. The domain \mathcal{P} and co-domain \mathcal{C} of e_k are very large sets, and in many applications, $\mathcal{P} = \mathcal{C}$, in which case e_k and d_k are permutations on the *message space* $\mathcal{M} = \mathcal{P} = \mathcal{C}$.

Two problems become apparent with secret-key cryptosystems. One is that for secure communications each pair of users need to have a secret key different from that of any other pair of users. Thus, if there are n users in the system, every user needs to store and remember $n - 1$ keys and a total $\binom{n}{2}$ keys are needed for the system. This is very difficult to manage when n is large. A second problem is that before a secure communication session can take place, the secret key k needs to be established between sender and receiver, using a secure channel. Again, this is very difficult to achieve in practice. Public-key cryptosystems were invented to alleviate these, and many other related problems.

In *public-key* cryptosystems, two transformations e_A (*encryption*) and d_A (*decryption*) are determined by each communicant A . These transformations are bijections and inverses of each other as in the case of secret-key cryptosystems above. However, in this case it is computationally infeasible to determine d_A given e_A , and equally infeasible to determine e_A from d_A . The transformation e_A is made public, while d_A is kept secret, known only by A . When participant B wishes to send a secure message x to A , he/she looks up e_A from a public directory, computes $y = e_A(x)$ and sends y to A . To recover the message, A computes $d_A(y) = d_A(e_A(x)) = x$. By means of a clever protocol, messages can also be authenticated (signed) by senders, so that the identity of the sender can be positively verified, while at the same time the receiver can verify the integrity of the message, i.e. be sure that the message has not been tampered with. To accomplish this, sender B first *signs* the message by computing $d_B(x)$, encrypts the result by computing $e_A(d_B(x))$ and sends the message over to A . Assuming that A knows this protocol is being used and that the arriving message is from B , A computes $d_A(e_A(d_B(x))) = d_B(x)$, and then computes $e_B(d_B(x)) = x$. The latter is possible because the encryption transformation e_B is publicly known to every participant of the system.

If x makes sense, then A knows that (i) Indeed it was B that sent the message, (ii) The message has not been tampered with, and (iii) The message is x .

In real applications it is not practical to authenticate each packet x of information using the above method. Instead, other much more efficient protocols

are used. In contrast to *secret-key cryptosystems*, *public-key cryptosystems* are also called *asymmetric cryptosystems*. The main problem with most presently known public-key cryptosystems, of proven security, is that the encryption and decryption operations are extremely slow.

Public-key cryptosystems also suffer from the Man-in-the-Middle attack (see the next Section for detail), where a malicious third party can intercept and modify the messages communicated between two parties. For example, when A is sending its public key P_A to B (or B is retrieving A's public key P_A from a public directory), a malicious third party C intercepts the transmission and replaces P_A with its own public key P_C . B will encrypt messages with P_C and send them to A. C intercepts the messages, decrypts them using its private key, re-encrypts them with P_A , and sends the re-encrypted messages to A. To remedy this problem of authentication, the services of a Certificate Authority (CA) are used. The CA is trusted by both users who wish to communicate. A CA issues a certificate (also called public key certificate) to each user, after proper verification. A certificate binds a user's identity with his/her public key and is later used by the user to prove his/her identity. The role/availability/trust-level of a CA are important issues that need to be resolved before two parties can securely communicate. Certificate issuance is also a time consuming process which must be done offline.

1.3.3 Two-party Diffie-Hellman Key Exchange

As indicated above, one of the most difficult problems in secret-key cryptosystems is to establish a secret key between two communicants. In 1976, Diffie and Hellman proposed an elegant protocol for two parties to achieve a shared secret key over an insecure channel. This is the well known *Diffie-Hellman key exchange* protocol [Diffie and Hellman, 1976b].

Although the *Diffie-Hellman key exchange* protocol can be described in general for a cyclic group G in which DLP is intractable, we present, without much loss of generality the instance when the group is the multiplicative group of a field \mathbb{Z}_p . Suppose p is a large prime and g is a generator of \mathbb{Z}_p^* such that the DLP problem in \mathbb{Z}_p^* is intractable. Parameters p and g are publicly known to members m_1 and m_2 . Member m_1 selects a random number $a_1 \in [1, p - 2]$, computes² $b_1 = g^{a_1} \bmod p$, and sends b_1 to m_2 over the insecure channel. Similarly, m_2 selects a random number $a_2 \in [1, p - 2]$, computes $b_2 = g^{a_2} \bmod p$, and sends b_2 to m_1 . As a result, each of m_1, m_2 is able to compute the shared secret key $SK = b_2^{a_1} = g^{a_1 a_2} = b_1^{a_2} \bmod p$ (See Figure 1.2). However any individual other than m_1 or m_2 cannot compute SK even if he/she knows the public parameters p, g and has intercepted the public components b_1 and

²In a Diffie-Hellman based system, any operation will be performed by mod p . We often omit the mod p for simplicity.

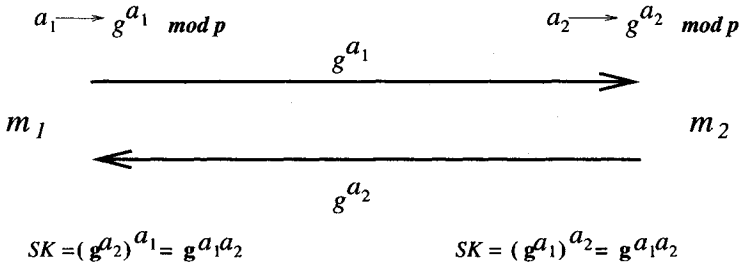


Figure 1.2. Diffie-Hellman key exchange.

b_2 . It is clear that a third party who can break DLP in \mathbb{Z}_p^* is able to recover SK . However, it is worth noticing, that the problem of recovering SK from g, p, b_1 , and b_2 is called the *Diffie-Hellman problem* (DHP), and is ‘weaker’ than the DLP, in the sense that solving DLP solves DHP, but possibly not conversely. To date, no general techniques are known for breaking the DH protocol without solving DLP.

For $i \in \{1, 2\}$, a_i is called the *Diffie-Hellman private share* or simply *private share* of m_i , and b_i the *Diffie-Hellman disguised public share* or simply *public share* of m_i . We call the derived secret key $SK = g^{a_1 a_2}$ a DH key.

The most serious problem with the Diffie-Hellman key exchange is the Man-in-the-Middle attack, as shown in Figure 1.3. In this scenario, the exchanged values $b_1 = g^{a_1}$ and $b_2 = g^{a_2}$ are replaced by $b_o = g^{a_o}$. The key computed by m_1 (or m_2) becomes $g^{a_1 a_o}$ (or $g^{a_2 a_o}$) and is mistaken to be the secret key shared between m_1 and m_2 . When m_1 transmits messages, encrypted with $g^{a_1 a_o}$, and directed to m_2 , m_o intercepts the messages, decrypts them, re-encrypts them using $g^{a_2 a_o}$, and sends them to m_2 . As a result, m_o obtains all the messages in cleartext form, while m_1 and m_2 are totally unaware that their communications have been compromised.

There are two typical ways to defend against the above attack [Kaufman et al., 2002]: (i) Using *authenticated Diffie-Hellman* and (ii) Using *published Diffie-Hellman shares*. In authenticated Diffie-Hellman, m_1 and m_2 know some sort of secret with which they can authenticate each other, such as a shared secret key or knowledge of each other’s public keys. They can use this secret to prove that it was they who generated the Diffie-Hellman public shares. The proof can be performed simultaneously with transmission of the Diffie-Hellman public shares or after the Diffie-Hellman key exchange. The following are examples of authentication:

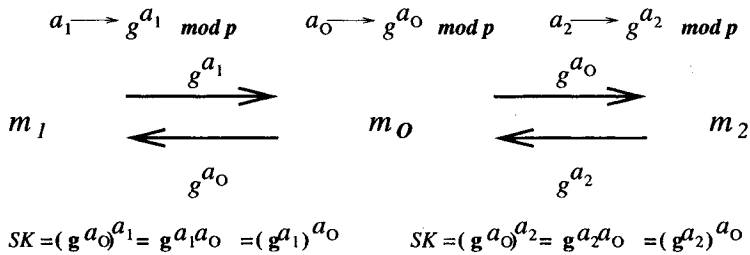


Figure 1.3. Man-in-The-Middle attack.

- Using the shared secret to encrypt the Diffie-Hellman public shares.
- Using the shared secret to transmit a keyed hash of the agreed-upon shared Diffie-Hellman key and participants' identity following exchange.
- Using public key encryption, i.e., encrypt the Diffie-Hellman public shares with each other's public key.
- Using a public key signature, i.e., sign the Diffie-Hellman (public) share and send the share, participants' signature and their public key certificate.

In published Diffie-Hellman shares, every individual has a publicly known, permanent, public share. Public shares can reliably be made public using public share certificates as public key certificates. As a result, the public shares cannot be forged or replaced.

The Diffie-Hellman key exchange protocol forms a basis for a suite of group key agreement protocols in secure group communications.

1.4 Group Dynamics and Security

An important feature in secure group communications is *group dynamics*. That is, group members may join or leave (or be evicted from) the group at any time. The presence of group dynamics makes group key management very difficult. Moreover, multiple group members may join, or leave or be evicted simultaneously. This kind of dynamics involves what is called *bursty* or *bulk* operations. Therefore, in general, a group key management protocol should consider the following operations:

Group initialization (setup). The group is *formed* by legitimate members and the initial group key is distributed to the members.

Member join. During group communication, a user *joins* the group to become a member of the group and participate in present and future group communications.

Member leave. During group communication, a group member *leaves* or is evicted from the group and will no longer be able to comprehend (decrypt) the group communication messages.

Bulk operation. Multiple members *join*, multiple members *leave*, multiple members join and others leave the group simultaneously. In some situations, when a member requests joining or leaving the group, the joining or leaving operation is not conducted immediately, but is postponed for some time. The purpose of postponing operations is to accumulate more joining and leaving requests so that these multiple requests can be processed in bulk. This can also be considered as a *bursty* or *aggregate* operation. The objective of bursty operations is to increase performance.

Periodic rekeying. The group key is changed periodically, independently of members joining or leaving.

Group splitting (partition). In certain applications, a group may be divided into two or more independent subgroups. Another scenario for group splitting is necessitated by a physical network partition. The group is then partitioned into multiple temporary subgroups.

Group merge. Two groups may fuse into a larger group in some SGC applications. Moreover, previously split groups due to an earlier network partition may need to merge again without member interaction.

From the security point of view, secure group key management is trying to guarantee that only current group members can possess the current group key. This means that outsiders, old group members, and future members should not be able to obtain the current group key. Therefore, there are at least two security requirements in SGC:

Backward secrecy. When a member joins a group, the group key needs to be changed so that the new member cannot obtain the old group keys to decrypt previous messages (in case he/she has intercepted and stored such earlier messages).

Forward secrecy. Whenever a member leaves a group, the group key needs to be changed so that the ex-member does not possess the current group key, otherwise he/she would be able to decrypt future group messages.

In general, backward secrecy is much easier to accomplish than forward secrecy because a leaving member possesses the current group key. One simple solution for implementing backward secrecy is as follows: whenever a member joins a group, a new group key is selected and is encrypted with the current group key. The new group key is then broadcast to the group. All group mem-

bers will decrypt the new group key. Moreover, the new group key will be transmitted to the joining member through a secure channel.

Another solution for implementing backward secrecy is using a one-way function. When a member joins a group, all group members compute the new group key by passing the old group key through a one-way function. The new group key is also sent to the joining member by a secure channel.

However, when a member leaves a group, since the member already possesses the current group key, the new group key cannot be computed from the current group key or encrypted by means of the current group key. The naive solution is to send the new group key to every individual member by means of a secure channel. However, this method is not scalable. Therefore, the key management problem becomes a search for solutions which can deal efficiently and scalably with members leaving.

1.5 Outline

The book covers three classes of key management protocols: (i) Key management protocols for SGC, (ii) Key management protocols for dynamic conferencing, and (iii) Key management protocols for SGC with hierarchical access control. These are discussed in Chapters 2, 4, and 5 respectively. Because tree-based SGC key management protocols constitute a considerable part of SGC, we discuss them separately in Chapter 3. In Chapter 6, we discuss other issues related to SGC, and present challenging problems when SGC is used in wireless environments. Finally, in Chapter 7, we present our concluding remarks.

Chapter 2

TYPICAL GROUP KEY MANAGEMENT SCHEMES

In order to achieve secure group communications, a group key must be shared only by group members. Group communication messages are encrypted by the group key, thus preventing outsiders from decrypting the messages. Almost all the schemes to be discussed below use the notion of a *central trusted authority* which is required in the organization of the schemes. This entity is also called the *group controller* (GC), or simply the *trusted authority* (TA). Its primary function is to distribute current and valid group keys and to change the group key with time as members join or leave the group.

A naive solution, called GKMP [Hamey and Muckenhim, 1997a, Hamey and Muckenhim, 1997b], works as follows: The GC selects a group key k_O and distributes this key to all group members. It is assumed that there exists a secure channel between the GC and every group member. Such a secure channel may be implemented using a public-key cryptosystem. Whenever a member joins, the GC selects a new group key, k_N , encrypts the new group key with the old group key ($k' = \{k_N\}_{k_O}$) and broadcasts k' to the group. Moreover the GC sends k_N to the joining member, via the secure channel between the GC and the new member. It is clear now that all members of the enlarged group can recover k_N . Whenever a member leaves, the GC selects a new group key k_N and sends it to each member of the reduced group, via the secure channels, one at a time.

Obviously, the naive solution is not scalable, and it frequently requires the use of secure channels. Secure channels are not always easy to establish. For these reasons a number of group key management protocols have been developed, each with different properties and performance. In this chapter we discuss some of these protocols.

Tree-based key management protocols are important in that they are not only used for one-to-many multicast/broadcast applications but also for many-

to-many group communications. In addition, since there are many variations of tree-based protocols, we discuss them separately in the next chapter.

2.1 Classification of Typical Group Key Management Schemes

In this section, we summarize and classify secure group communication schemes based on different criteria including *the number of senders, group key formation method, kind of cryptosystem used, structural organization, and kind of security*.

Based on *the number of senders*, secure group communication applications can typically be divided into two categories. The first category involves *broadcast* or *multicast* communication, i.e., one sender and multiple receivers, or *one-to-many* communication. The second is *group* (or *many-to-many*) communication. Here, every sender is also a receiver¹. Correspondingly, some secure group communication schemes such as those described in [Bakkardie, 1996, Chiou and W.T.Chen, 1989, Dondeti et al., 1999, Dondeti, 1999, Du et al., 1999, Molva and Pannetrat, 1999] are suitable for broadcast applications. Some other schemes such as those described in [Burmaster and Desmedt, 1995, Burmaster and Desmedt, 1996, Dondeti, 1999, Ingemarsson et al., 1982, Sherman and McGrew, 2003, Steer et al., 1990, Steiner et al., 1996] are suitable for many-to-many applications. There are also a few schemes such as those described in [Caronni et al., 1998, Mitra, 1997, Noubir, 1998, Wong et al., 1998] which are suitable for both kinds of applications.

Based on *how the group key is formed*, some schemes require a CA, TA or GC which generates the group key and distributes the key to all the group members. This kind of key management is called *centralized key distribution*. In other schemes [Burmaster and Desmedt, 1995, Burmaster and Desmedt, 1996, Dondeti, 1999, Dondeti et al., 2000, Ingemarsson et al., 1982, Kim et al., 2000, Kim et al., 2001, Steer et al., 1990, Steiner et al., 1996], the group key is generated by uniform contributions from all group members. This kind of key management is called *distributed* or *contributory key agreement*. The contributory key agreement protocols are primarily different variations of n -party Diffie-Hellman key exchange [Burmaster and Desmedt, 1995, Ingemarsson et al., 1982, Kim et al., 2000, Kim et al., 2001, Steer et al., 1990, Steiner et al., 1996].

Based on the *kind of cryptosystem used*, secure group communication schemes can be divided into two classes. One class is based on *public-key* cryptosystems, called *public-key based schemes* [Chiou and W.T.Chen, 1989, Du et al., 1999, Molva and Pannetrat, 1999]. The other class is based on *secret-*

¹The authors of [Dondeti, 1999] define one more category, namely *few-to-many* communications.

key (i.e., symmetric) cryptosystems, called *secret-key based schemes*, where the underlying secure channel may be implemented by public-key cryptosystems. In the latter case, the group key management is the core problem. Many schemes [Bakkardie, 1996, Beimel and Chor, 1994, Beimel and Chor, 1996, Blom, 1985, Blundo and Cresti, 1995, Blundo et al., 1998, Blundo et al., 1993, Burmester and Desmedt, 1995, Burmester and Desmedt, 1996, Caronni et al., 1998, Dondeti, 1999, Dondeti et al., 1999, Fiat and Naor, 1994, Ingemarsson et al., 1982, Sherman and McGrew, 2003, Mittra, 1997, Noubir, 1998, Steer et al., 1990, Steiner et al., 1996, Stinson, 1997, Wong et al., 1998] have been proposed to deal with key management.

Based on the *structural organization* of group members, most schemes do not split members whereas some schemes [Dondeti et al., 1999, Mittra, 1997, Molva and Pannetrat, 1999] divide group members into distinct subgroups. In the latter case, every subgroup has a *subgroup manager*² and a *subgroup key* managed by the subgroup manager. The subgroup manager also has access to the subgroup key of a higher subgroup (generally his subgroup's parent) and is responsible for "relaying" messages between its subgroup and the one above. A subgroup manager may or may not be a member of the group, depending on the schemes. Furthermore, some schemes [Du et al., 1999, Mittra, 1997] trust subgroup managers while other schemes [Dondeti et al., 1999, Molva and Pannetrat, 1999] do not. In a typical scenario of the first case, a subgroup manager may be given access to all keys related to its subgroup plus the subgroup key of its parent node. In a different scenario, a subgroup manager is only given access to partial (half or incomplete) keys.

Based on the *kind of security*, the schemes for secure group communications may be classified as *unconditionally secure* or *computationally secure* [Stinson, 1995, Stinson, 1997]. Furthermore some schemes are secure against any number of colluding adversaries, whereas others [Beimel and Chor, 1994, Beimel and Chor, 1996, Blom, 1985, Blundo and Cresti, 1995, Blundo et al., 1998, Blundo et al., 1993, Fiat and Naor, 1994, Stinson, 1997] are only secure against the collusion of up to k adversaries, i.e., they are k -resilient schemes, where k is a security parameter in these schemes.

Among all key management protocols, the key tree scheme [Caronni et al., 1998, Noubir, 1998, Wong et al., 1998, Zhang et al., 2001] is a very powerful approach. The scheme can be used for both multicast/broadcast (i.e., one-to-

²A subgroup manager is usually a computer based *agent* (or process) responsible for the above task.

many) communication and group (i.e., many-to-many) communication. It can also be used for centralized key distribution as well as distributed (contributory) key agreement.

This chapter is organized as follows. We discuss key management protocols in the following two sections based on the classification of public-key based and secret-key based schemes. Then we discuss a number of n -party Diffie-Hellman key agreement protocols. These protocols may be considered as public-key based schemes, as well as secret-key based schemes, because the final result of the protocols is to generate a shared secret key.

2.2 Public-Key based Secure Group Communication Schemes

In this section, we discuss two public-key cryptosystem based schemes: ‘*Reversible Parametric Sequence*’ (RPS) and ‘*Secure Transmission Backbone*’ (STB).

2.2.1 Reversible Parametric Sequence (RPS)

The notion of a *Reversible Parametric Sequence* (RPS) [Molva and Panertrat, 1999] is based on a one-way function (see Definition 1.1). It is supposed to be embedded in a multicast tree and be used for secure multicast (one-to-many) communications.

Let N be a finite set, and suppose $f : N \times N \rightarrow N$ is a one-way function, so that if $y = f(x, a)$, it is computationally infeasible to compute a given x and y . Suppose further that (a_1, \dots, a_n) is a finite sequence of n elements, $a_i \in N$. Define now a new sequence (S_0, \dots, S_n) of $n + 1$ elements of N as follows: S_0 is a randomly selected initial element, and for $i > 0$, $S_i = f(S_{i-1}, a_i)$.

DEFINITION 2.1 A sequence (S_0, \dots, S_n) as defined above is called a *Reversible Parametric Sequence associated with f* , denoted as RPS_f , if for each pair (i, j) , $0 < i < j \leq n$, there exists a computable function $h_{(i,j)}$ such that $S_i = h_{(i,j)}(S_j)$.

Thus, if (S_0, \dots, S_n) is an RPS_f , and $h_{(0,n)}$ is known, S_0 can be computed from S_n as: $S_0 = h_{(0,n)}(S_n)$.

Two typical examples of RPS_f are constructed from DLP and RSA. The DLP derived RPS_f is as follows: Let p be a large prime, (a_1, a_2, \dots, a_n) a sequence of elements in the finite field $\mathbb{F}_p = \mathbb{Z}_p$ and let g be a generator of \mathbb{F}_p^* . Define f by $f(x, a) = x^a \bmod p$. Then the following sequence is an RPS_f : $S_0 = g$, $S_i = f(S_{i-1}, a_i)$, where

$$h_{(i,j)}(S_j) = S_j^{D_i} \bmod n$$

and $D_i \cdot (a_{i+1} \cdots a_j) \equiv 1 \pmod{(p-1)}$. Thus, D_i is the multiplicative inverse of $a_{i+1} \cdots a_j$ modulo $(p-1)$ and can be efficiently computed using the *Extended Euclidean Algorithm*.

The RSA-derived RPS_f is as follows: Let p and q be large primes, $n = pq$ and (a_1, a_2, \dots, a_n) a sequence of units in the ring \mathbb{Z}_n so that $\gcd(a_i, \phi(n)) = 1$. Define f by $f(x, a) = x^a \pmod n$. Then the following sequence is an RPS_f : $S_0 = x$, $S_i = f(S_{i-1}, a_i)$, where

$$h_{(i,j)}(S_j) = S_j^{D_i} \pmod n$$

Here $D_i \cdot (a_{i+1} \cdots a_j) \equiv 1 \pmod{\phi(n)}$ and D_i is the multiplicative inverse of $a_{i+1} \cdots a_j$ modulo $\phi(n)$ and can be efficiently computed.

Let us consider multicast communication and the notion of a *multicast tree*. When a sender sends messages to multiple receivers, the paths from the sender to all the receivers form a tree. The root node is the sender and the intermediate nodes are routers (or gateways) whereas the leaf nodes are sets of receivers (hosts). Then an RPS_f is embedded in the tree as follows. Every intermediate node is assigned an RPS_f element a_i , the sender holds a_1 , and every leaf node (the set of receivers) is assigned a function $h_{(0,n_m)}$. The function f is publicly known to all nodes. See Figure 2.1.

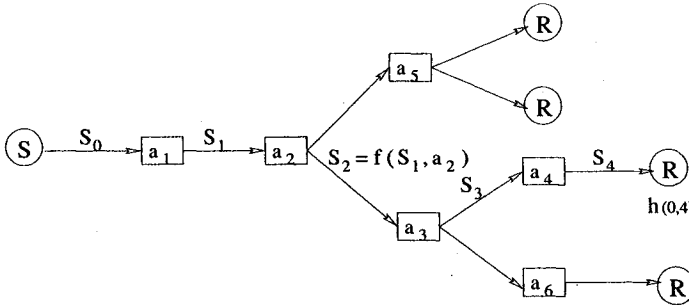


Figure 2.1. RPS scheme (Source: [Molva and Pannetrat, 1999], © ACM).

The sequence (S_0, \dots, S_n) is obtained as follows. The message³ M to be sent is chosen as S_0 . The sender computes $S_1 = f(S_0, a_1)$ and multicasts S_1 to all its neighboring nodes in the multicast tree. When an intermediate node N_j receives S_{j-1} , N_j computes $S_j = f(S_{j-1}, a_j)$ and multicasts S_j to all its children's nodes in the multicast tree. When a receiver receives S_n , it can recover the initial message $M = S_0 = h_{(0,n)}(S_n)$, using $h_{(0,n)}$.

³When M is a long message, a random key k is generated, M is encrypted with k , and k is chosen as S_0 .

The RPS scheme assumes that the intermediate nodes are trustworthy and they participate in multicast routing, as well as in security enforcement. The biggest problem with the RPS scheme is that the sender needs to know the exact paths to all receivers so that it can assign values a_i to intermediate routers and $h_{(i,j)}$ to receivers. The RPS scheme assumes that there is a secure channel between the sender and each of the intermediate nodes and receivers so that the a_i and $h_{(i,j)}$ can be sent to them securely. Moreover, this scheme is only suitable for one-to-many applications. If used in many-to-many communications, the routers need to store the a_i for every potential sender, i.e., for every member in the group, in fact, for all members of all groups. Clearly, this is not practical.

2.2.2 Secure Transmission Backbone (STB)

Secure Transmission Backbone, STB [Du et al., 1999] is similar to RPS in the sense that the intermediate routers are trusted and participate in the secure delivery of messages. STB is based on a public-key cryptosystem such as RSA. Moreover, STB tries to solve the scalability problem by deleting the shared group key.

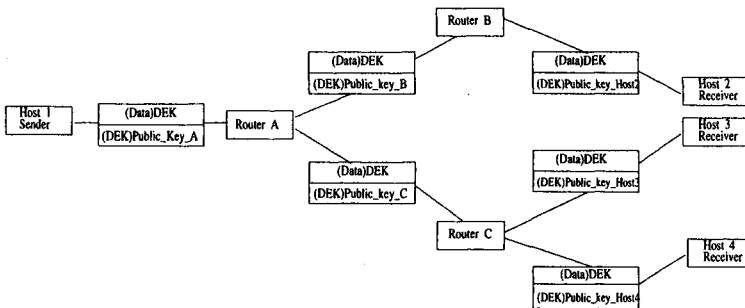


Figure 2.2. Secure Transmission Backbone (Source: [Du et al., 1999], © IEEE).

STB consists of a number of trusted routers. Each router and each receiver have their pairs of public and private keys. The public keys are made public, or at least a router should know the public keys of its neighbors. When a sender sends a message, it selects a random session key called DEK (Data Encryption Key), encrypts the message using DEK, then encrypts DEK using the first router's public key and sends the encrypted message along with the encrypted DEK to the first router. Upon receiving a packet, a router decrypts the DEK and re-encrypts the DEK using the public keys of the neighboring routers/hosts. The data segment (i.e., the encrypted message) of the packet is not modified. Next, the new packet is forwarded to the next routers/hosts. This

process is repeated until the packets are transmitted to receivers. A receiver can decrypt the DEK using its private key and then the message using DEK. See Figure 2.2.

The objective of STB [Du et al., 1999] is to achieve a general solution for both secure multicast and secure unicast. Moreover, there is no need for an individual multicast group to maintain its own keys, and therefore the key management problem is solved naturally.

The most serious problem with STB is that an end router needs to take care of the management of all groups. For example, a router may be connected to a LAN which has several members belonging to different groups. Assume, for example, that H_1 belongs to groups G_1 and G_2 , H_2 belongs to G_2 and G_3 , and H_3 belongs to G_1 and G_3 . The router must remember these membership relations so that when a packet for group (e.g., G_1) arrives, the router will know which members belong to this group so that it can encrypt the DEK with the public keys of these members (e.g., H_1 and H_3). This is a considerable burden to routers. Moreover, the existing protocols for routing and group management need to be modified.

2.3 Secret-Key based Secure Group Communication Schemes

Public-key based key management protocols are computationally inefficient. Therefore there are some key management protocols which are based on secret-key cryptosystems. The next section discusses such schemes.

2.3.1 Core Based Tree (CBT)

The Core Based Tree (CBT) [Bakkardie, 1996] approach works as follows: A *Group Initiator* (GI) designates a number of routers as *cores*, and the routing tree connecting these cores is called a *core tree* (see Figure 2.3). Each core receives the membership control information (the same for all core nodes) from the GI, which is in the form of either a *member inclusion list* or *member exclusion list*. When a member wishes to join the group, it sends a request to one of the cores. After the request is approved by that core, the path connecting the member to the core tree forms a new part of the group multicast tree. This link could connect the new member through a sequence of nodes, some of which may not be members. There is a *Key Distribution Center* (KDC), which may be the GI, and which generates two keys: the *Data Encryption Key* (DEK) and the *Key Encryption Key* (KEK). When a member joins, the KDC sends these two keys to the new member via a secure channel. To rekey the DEK, the KDC multicasts to the group the new DEK encrypted with KEK. The members in the group can communicate securely using the DEK.

CBT assumes that a member will stop receiving any multicast messages once he/she leaves or is removed from the group. In other words, the scheme assumes that leaving members will not harm the group after they leave the group. Of course, this assumption is invalid in most applications.

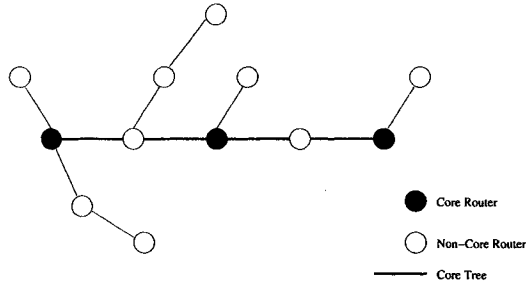


Figure 2.3. Core in Core Based Tree (Source: [Bakkardie, 1996]).

2.3.2 Iolus

In *Iolus* [Mittra, 1997], a large group is divided into a number of subgroups. Thus when a member m joins or leaves, only the key of the subgroup to which m belongs needs to be changed, while the keys of all other subgroups are not affected. *Iolus* relies on *relay* nodes for rekeying, performing admission control, and relaying messages.

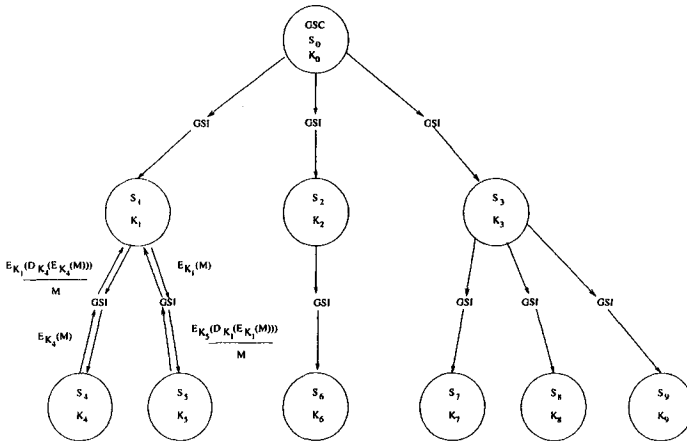


Figure 2.4. Subgroups and GSIs in Iolus scheme.

In Iolus a group is decomposed into a number of subgroups which are located at different levels of a tree. Every subgroup has a controller called a *Group Security Agent (GSA)*. The GSA of the root subgroup is also called the *Group Security Controller (GSC)*. The other GSAs are also called *Group Security Intermediates (GSI)* (see Figure 2.4). Every subgroup has its own independent key. A GSI is a bridge between its parent subgroup and its own subgroup, and it possesses both subgroup keys. When a member sends a message, it encrypts the message using its subgroup key. When a GSI receives a message from one subgroup, it decrypts the message, re-encrypts the message with the key of the *adjacent* subgroup and retransmits the message to the adjacent subgroup. We illustrate this operation by means of an example related to Figure 2.4. Suppose a user u belonging to subgroup 7 wishes to send a message M to the entire group. u broadcasts $y_7 = \{M\}_{K_7}$. All members of subgroup 7, including GSI_7 , are able to decrypt y_7 and obtain M . GSI_7 now broadcasts $y_3 = \{M\}_{K_3}$. All members of subgroup 3, including GSI_8 , GSI_9 and GSI_3 , know K_3 and are able to recover M from y_3 . Now, for each $i = 8, 9$, GSI_i broadcasts $\{M\}_{K_i}$. Thus the message is propagated to all members of subgroups 8 and 9. GSI_3 broadcasts $y_0 = \{M\}_{K_0}$ to subgroup 0, thus all members of subgroup 0, including the GSC, GSI_1 , and GSI_2 , are able to decrypt y_0 and obtain M . GSI_1 and GSI_2 can recursively transmit M securely to the remaining subgroups 1,2,4,5 and 6, using exactly the same procedure.

When a member u belonging to a subgroup S receives a message, it decrypts the message using the key for S . When a member u joins or leaves a subgroup S , the GSA of S selects a new subgroup key and sends the key to all members in the new S . As a result, only the key for S and its members are affected, while all other subgroup keys and all members in all other subgroups are not affected.

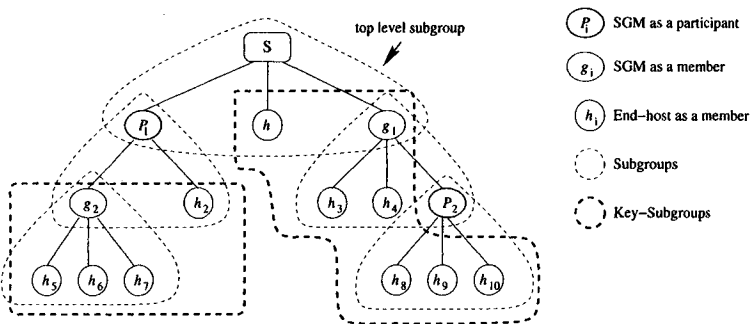
2.3.3 Dual Encryption Protocol (DEP)

In Iolus, all information (keys, messages) is exposed to the GSAs. The GSAs are generally third party entities such as Internet Service Providers (ISPs). This means that the system is not secure if some GSA is un-trustworthy or unreliable. In order to ameliorate this problem, the *Dual Encryption Protocol (DEP)* was proposed in [Dondeti, 1999]. We present the protocol in this section⁴. In what follows, we use the term SGM (SubGroup Manager) in place of GSA.

⁴Portions reprinted, with permission of the authors and the publisher, from L. R. Dondeti et al., *A dual encryption protocol for scalable secure multicasting* in Proceedings of the 4th IEEE Symposium on Computers and Communications, July, 1999, pages: 2–8. © 2002 IEEE.

DEP is a one-to-many multicast protocol, i.e., there is one only sender. The members (i.e., the sender and all receivers) are divided into hierarchical subgroups as in Iolus and the sender belongs to the root subgroup. Every subgroup has an independent key and an SGM. An SGM is responsible for decrypting messages coming from its parent subgroup, for example encrypted data packets and the encrypted DEK coming along with these packets (see below). An SGM is also responsible for encrypting and sending messages to its own subgroup. Thus, as in Iolus, an SGM will have both the key of its own subgroup (i.e., the Local Subgroup Key) and the key of its parent subgroup.

Group members are also divided into another kind of subgroup called *key-subgroup*. The subgroups and key-subgroups are independent. However, in general, a subgroup other than the root subgroup belongs to one key-subgroup, i.e., all members in a subgroup except for a *participant SGM* (see below) belong to the same key-subgroup. This means that, excluding participant SGMs, a subgroup is a subset of some key-subgroup. Thus, excluding participant SGMs, a key-subgroup is always the union of certain subgroups. The subgroup keys are generated and distributed by their corresponding SGMs while all key-subgroup keys are generated and distributed by the sender. Every member belongs to only one subgroup and only one key-subgroup. An SGM, if trusted, will also belong to one subgroup and one key-subgroup. An SGM of this kind is called a *member SGM*. If a particular SGM is not trusted, it will belong to a subgroup but will not belong to any key-subgroup and is then called a *participant SGM*.



All members receive their subgroup keys and their key-subgroup keys. An SGM possesses the subgroup-key for its own subgroup as well as the subgroup-key for its parent subgroup. A member-SGM such as g_1 , receives its key-subgroup key, however, a participant-SGM such as P_1 receives no key-subgroup key.

Figure 2.5. Subgroups and key subgroups in DEP (Source: [Dondeti, 1999], © IEEE).

Let $\{x\}_k$ denote the result of encrypting x under key k , and under some particular fixed cryptosystem. There are three kind of keys: (i) A DEK (Data Encryption Key), one for the entire group and used for encrypting data messages. Thus a message M is encrypted as $\{M\}_{DEK}$. (ii) There are KEK's (Key Encryption Keys), one for each key-subgroup and these are used for encrypting the DEK respectively, resulting in $\{DEK\}_{KEK_T}$ for key-subgroup T . (iii) Finally there are LSKs (Local Subgroup Keys), one for each subgroup S , and these are used for encrypting the above encrypted DEK respectively, resulting in $\{\{DEK\}_{KEK_T}\}_{LSK_S}$, where $S \subseteq T$. Hence, the DEK is encrypted twice: once by a KEK and then by an LSK. Consequently the scheme is called a *Dual Encryption Protocol*. Every group member will be given the KEK of its key-subgroup by the sender when the member joins the group and also the LSK of its subgroup by the SGM of its subgroup (see Figure 2.5). Member SGMs are also given the KEK of their corresponding key-subgroups, however participant SGMs will not be given any KEK.

The data communication is as follows. The sender first encrypts the message M with a randomly selected DEK , i.e., he/she computes $\{M\}_{DEK}$. The sender then encrypts the DEK with each KEK_T for each existing key-subgroup T in the structure. Thus, the sender computes $\{DEK\}_{KEK_T}$, over all possible key-subgroups $\{T\}$. Then, the sender encrypts all these $\{DEK\}_{KEK_T}$ with the LSK of the root subgroup (LSK_0), resulting in $\{\{DEK\}_{KEK_T}\}_{LSK_0}$. Finally the sender broadcasts the encrypted message $\{M\}_{DEK}$ as well as all dually encrypted versions of DEK , namely the various $\{\{DEK\}_{KEK_T}\}_{LSK_0}$.

A member of the sender's subgroup can decrypt the DEK using its own LSK and KEK. An SGM decrypts the dually encrypted DEK using its parent LSK to get $\{DEK\}_{KEK}$, encrypts $\{DEK\}_{KEK}$ using its own LSK and multicasts $\{\{DEK\}_{KEK}\}_{LSK}$ to its subgroup. A member-SGM, can also decrypt the DEK using its KEK. However, a participant-SGM can not decrypt the DEK because it has no KEK. After a member has obtained the DEK, it can decrypt $\{M\}_{DEK}$ to recover the message M .

When a member joins the group, the following operations take place. The sender sends to the joining member the KEK of the key-subgroup to which the joining member will belong. The SGM of the subgroup to which the joining member belongs changes its LSK and multicasts the LSK to its members, also sending the LSK to the joining member. When a member leaves, its SGM changes the subgroup LSK and sends the new LSK to the remaining members of the subgroup. All KEKs and all other LSKs need not be changed. Since

the DEK will be dually encrypted with KEKs and LSKs, the leaving member cannot decrypt the DEK in the future because it does not have the new LSK.

As indicated above, this scheme can only be used for one-to-many communications. A trivial collusion attack still exists: a leaving user may transfer his/her KEK to a participant-SGM, so that the participant-SGM will acquire the KEK.

2.4 Group Key Management based on Hierarchical Clusters

In [Banerjee and Bhattacharjee, 2002], a cluster based group key management with hierarchical structure was proposed. We present the scheme in this section⁵.

2.4.1 Layers, Clusters and Keys

The group members are initially aggregated into clusters, at a lowest level L_0 , called layer 0, by a member clustering protocol (see Section 2.4.3 for a description of the protocol). Each cluster selects one of its members as the cluster leader. All cluster leaders at L_0 become the members of the next higher layer L_1 . The members of L_1 are again grouped into clusters at L_1 by the same clustering protocol. The cluster leaders at L_1 become the members of layer L_2 , etc. This process continues until finally there is a single cluster at some level L_{m-2} and its cluster leader becomes the only member at the next highest level L_{m-1} . Thus group members form different clusters at different levels, configured in a hierarchy satisfying the following properties: (i) Each cluster has between k and $2k - 1$ members, for some fixed k ; (ii) No two clusters at the same level share a member; (iii) Cluster members are close to each other on the multicast delivery path.

Figure 2.6 shows a snapshot of the structure of hierarchical clusters for ten members $A - J$. From the figure, it is clear that the ten L_0 members have been partitioned by the clustering protocol into three clusters: $[ABC]$, $[DEFJ]$, and $[GHI]$. The cluster leaders of the three clusters, C , E , and H , join layer L_1 to form the single cluster $[CEH]$ after the execution of another instance of the clustering protocol. The leader H of cluster L_1 joins layer L_2 , the top layer in this example. When a new member requests to join, it is placed in one of the clusters of layer L_0 by the clustering protocol. The addition of a new member to or departure of an existing member from a cluster may cause a change or reselection of the cluster leader. Changing a cluster leader at one level will

⁵Portions reprinted, with permission of the authors and the publisher, from S. Banerjee and B. Bhattacharjee, *Scalable Secure Group Communication Over IP Multicast* in IEEE Journal on Selected Areas in Communications, 20(8), 2002, pages: 1151–1527. © 2002 IEEE.

cause a membership change at the next level, and this kind of change may propagate up to the highest level. Occasionally the arrival of a new member will split a cluster and the departure of an existing member will result in the merging of clusters. All these decisions are made by the clustering protocol.

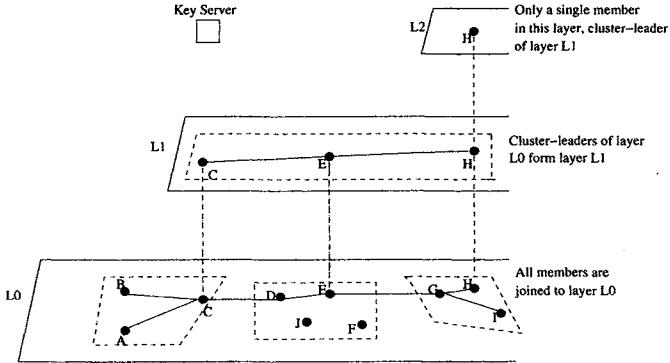


Figure 2.6. Layered clusters and keys for ten members (Source: [Banerjee and Bhattacharjee, 2002], © IEEE).

There are three types of keys: *layer keys*, *cluster keys*, and *pairwise keys*. For each layer L_i in the hierarchy, there is a secret *layer key*, LK_i , associated with the layer. All members of a layer, and only these members have access to the associated layer key. Layer keys are generated by a *Group Controller*, also called the *key server*. There is a secret *cluster key* associated with each cluster $C_{i,j}$ (i.e., the j^{th} cluster in the i^{th} layer), denoted by $CK_{i,j}$. All members in a cluster and only these members will have access to the cluster key. The cluster keys are generated by cluster leaders. If $l_{i,j}$ is the cluster leader and m any member of $C_{i,j}$, there is a pairwise secret key shared between $l_{i,j}$ and m . Since a cluster leader belongs to at least two adjacent layers it is clear that, in general, a member may belong to multiple layers and multiple clusters (one per layer). Thus, a member m may have multiple layer keys, one for each layer it belongs to, multiple cluster keys, one for each cluster it belongs to, and multiple pairwise secret keys. The latter are shared between m and each of its cluster leaders, or between m and each of its cluster members if m happens to be a cluster leader. Finally, all members belong to the L_0 layer, so the key (LK_0) for L_0 is shared by all members and used as the group key for secure communication.

2.4.2 Key Management

Key management is based on layers, assisted by clusters, with goal that the layer key of each layer is available to a member if and only if the member joins that layer. Thus, whenever a member joins or leaves a layer, a new layer key is required for that layer. The new layer key is generated by the key server and distributed securely to all members of that layer via the layer's cluster leader. This guarantees that key LK_0 for group communication is available only to the current group members, i.e., the members of L_0 . The rekeying operations for different scenarios are discussed below.

Whenever a member requests to join the group⁶, the member is assigned to an L_0 cluster by the clustering protocol at layer L_0 . The cluster leader, CL , performs the following:

- 1 The CL and the joining member establish a pairwise secret key by means of the Diffie-Hellman key exchange protocol.
- 2 The CL sends a new layer key request to the key server. At the same time, CL generates a new cluster key and distributes the key to all its cluster members⁷, including the joining member.
- 3 The key server generates the new L_0 -layer key LK_0 , encrypts the new LK_0 by LK_1 of L_1 , which is the next higher layer, and multicasts the encrypted new LK_0 to all the members of L_1 .
- 4 After receiving the new layer key LK_0 , all members in L_1 , being leaders of the clusters in L_0 , extract the new LK_0 , encrypt the new LK_0 with their respective cluster keys, and multicast it to members of their respective clusters. Note that each CL encrypts the new LK_0 with its new cluster key and the joining member can thus obtain LK_0 after receiving the CL 's multicast message.

Whenever a member m leaves the group, the rekeying operations will differ depending on whether the member is a cluster leader or not. If the leaving member is not a cluster leader, the rekeying operation will be limited to layer L_0 . The leader of the cluster to which the leaving member belongs will initiate the rekeying operation, following steps 2 - 4 as in the above rekeying operation for a join. The only difference here would be that the cluster leader must

⁶The member needs to be authenticated by the key server using a predefined authentication method.

⁷A method for distributing the new cluster key to cluster members was not discussed in [Banerjee and Bhattacharjee, 2002]. One way of doing this is for the CL to unicast the new cluster key, encrypted by pairwise keys, to each of the cluster members including the joining member. Another way would be for the CL to encrypt the new cluster key with the current cluster key and multicast the encrypted new cluster key to its cluster members, while also unicasting the new cluster key to the joining member.

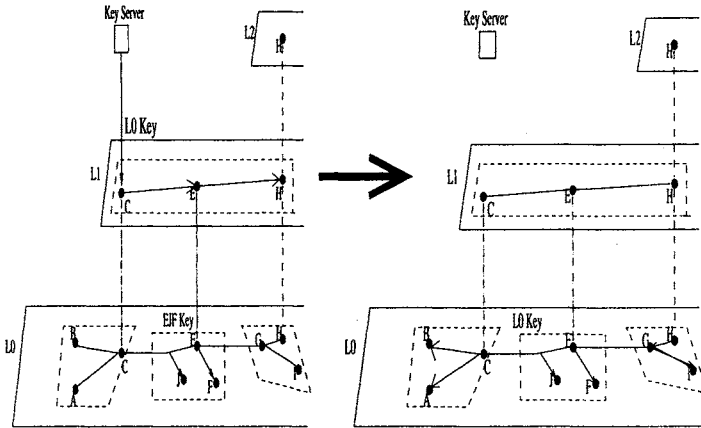


Figure 2.7. Member D , a non-cluster leader, leaves (Source: [Banerjee and Bhattacharjee, 2002], © IEEE).

unicast the new cluster key to each of the remaining members by the pairwise keys between itself and each remaining member.

For example, Figure 2.7 illustrates the operations when member D , a non cluster leader, leaves the group. The cluster leader E generates a new cluster key and unicasts the key to J and F . The key server, after receiving the request from E , generates a new layer key LK_0 and multicasts LK_0 to L_1 members (the left hand segment in the figure). Then cluster leaders in L_0 multicast LK_0 , encrypted with their respective cluster keys, to members of their respective clusters (the right hand segment of the figure).

If a leaving member m is a cluster leader in L_0 , then m is also a member of L_1 . Moreover, if m is a cluster leader in L_1 , it is also a member at layer L_2 , etc. Suppose that m is a member of all the layers up to some layer L_s (i.e., m is a cluster header in $\{L_0, \dots, L_{s-1}\}$ and a non-leader member of cluster C_s , at layer L_s ⁸). In this case the rekeying operations are as follows.

- 1 **Cluster leader re-selection:** For every cluster in layers $\{L_0, \dots, L_{s-1}\}$, which is affected by m 's departure, a new cluster leader is chosen by the clustering protocol. In case that m was originally the only member at the highest layer L_s , the new cluster leader NCL of the cluster at L_{s-1} will

⁸In the extreme case, L_s is at the highest layer in the hierarchy, so that m is the only member of L_s

become the only member at L_s . Thus a new layer key LK_s is generated by the key server and sent to the new leader NCL .

- 2 *Cluster rekey*: The new cluster leaders generate new cluster keys and distribute them securely to the remaining members in their respective clusters.
- 3 *Layer rekey*: The new layer keys LK_0, LK_1, \dots, LK_s are requested by the leader of the highest affected cluster C_s , and the key server generates these new layer keys. Then for each layer i ($= s, s - 1, \dots, 0$), the server encrypts the layer key LK_i with its immediate higher layer key LK_{i+1} , and multicasts the encrypted LK_i to all members of L_{i+1} , namely to the cluster leaders of the clusters of L_i . Then, these members decrypt LK_i (this is possible because they all have LK_{i+1}), encrypt LK_i with their respective cluster keys, and multicast the encrypted LK_i to their clusters. As a result, all members⁹ at layer L_i receive the new layer key LK_i . In the final round $i = 0$, all members receive the new layer key LK_0 .

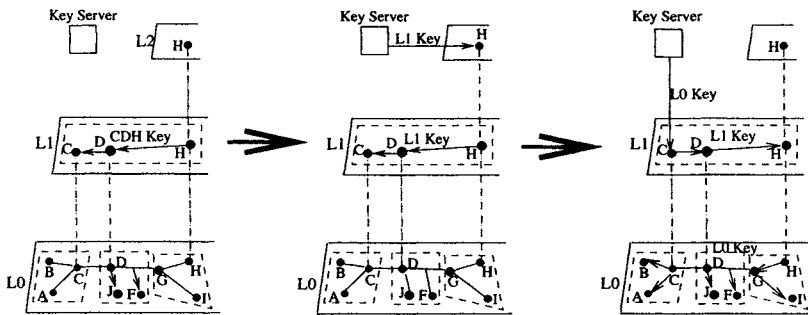


Figure 2.8. Member E , a cluster leader, leaves (Source: [Banerjee and Bhattacharjee, 2002], © IEEE).

For example, Figure 2.8 shows the departure of cluster leader E from the initial configuration of Figure 2.6. E was part of layers L_0 and L_1 , so not only the cluster leader needs to be re-selected, but also more than one cluster keys and layer keys need to be changed. On the left hand side of Figure 2.8, i) D replaces E as cluster leader of cluster $[DJF]$ in L_0 , ii) D generates and unicasts a new cluster key for cluster $[DJF]$, iii) D joins layer L_1 and the new cluster $[CDH]$, and iv) H generates and unicasts a new cluster key for cluster $[CDH]$. In the middle segment of Figure 2.8, the key server generates a new LK_1 and multicasts it to the L_2 members, while subsequently the L_2 members (cluster leaders of L_1) multicast the new LK_2 to their respective clusters. On the right hand segment of Figure 2.8, the new LK_0 is generated, encrypted by

⁹note that m has already been excluded from this layer at this time

the new LK_1 , and multicast by the key server to the members of L_1 . Finally, the members of L_1 , being the cluster leaders of L_0 , encrypt the new LK_0 using their cluster keys, and multicast it to their respective layer- L_0 clusters.

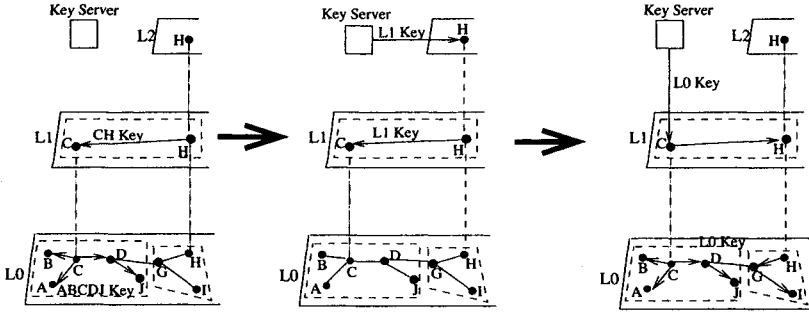


Figure 2.9. Member F 's leave causes clusters to merge (Source: [Banerjee and Bhattacharjee, 2002], © IEEE).

It should be clear that any change in membership in a specific layer L_j for $j > 0$ results in corresponding changes in all lower layers L_i , $0 \leq i < j$. Moreover, the departure of a member may cause some clusters to merge and the number of layers to reduce. For example, Figure 2.9 shows the merging of clusters when F departs following E 's leave. As shown on the left hand side of Figure 2.8, two clusters merge into L_0 , D drops out of L_1 , and new cluster keys are generated for cluster $[ABCDJ]$ of L_0 and cluster $[CH]$ of L_1 , by cluster leaders C and H . In the middle of the figure, a new LK_1 is generated since the membership of L_1 has changed (D is no longer a member of L_1 , as it is not a cluster leader in L_0). On the right hand segment, the new layer key LK_0 is generated and distributed as before.

2.4.3 Clustering Protocol

As indicated in the above description, the clustering protocol plays an important role in the rekeying process. The protocol is executed at each layer to form clusters whenever there is a membership change at that layer. The protocol consists of two sub-protocols: (1) *member discovery protocol* and (2) *clustering protocol*. The input to the member discovery protocol is a multicast topology and the output a *member overlay tree*, which specifies parent-child relationships among the members of the multicast tree. The input to the clustering protocol is the member overlay tree generated above. The output of the clustering protocol is a collection of clusters. Below, we describe the two sub-protocols in detail.

The parent-child relationship in the member overlay tree is determined by the distance, in router hops, from a multicast source S (i.e. the root). Let

$d(u, v)$ denote the distance between member u and member v along the multicast delivery path rooted at source S . Moreover, for any member $v \in L_0$ define the set of members $P_v = \{x \in L_0 : d(x, S) \leq d(v, S)\}$. Now, for any member v , u is defined to be the parent of v if and only if the following three conditions hold:

C1: $u \in P_v$, i.e. u is closer to the source S than v ($d(S, u) \leq d(S, v)$).

C2: u is the closest member satisfying condition C1, i.e., if $w \in P_v$ then $d(u, v) \leq d(w, v)$.

C3: u is the lexicographically smallest member among all closest members, i.e., $\forall x$ that satisfy C2, $u \leq x$ (lexicographically).

There are two periodic signals used by the *member discovery protocol* to discover and maintain the parent-child relationships. One is the *source heartbeat message*, which is multicast periodically by the root S to all the members of the group. Every member of the group can infer its distance to S from the source heartbeat message. The second signal is the *TTL-scoped heartbeat message*. Every member u periodically transmits a TTL-scoped heartbeat message to its parent and all of its children over the overlay tree. The message contains the tuples $\langle d(S, u), P(u) \rangle$, where $P(u)$ denotes the parent of u in the overlay tree. The initial guess of the parent-child relation based on the source heartbeat message and previous knowledge may be incorrect. After receiving the TTL-scoped heartbeat message from its parent or child, a member can further infer the parent-child relations. When the protocol converges to a stable state, the correct member overlay tree is determined.

An example from [Banerjee and Bhattacharjee, 2002] is shown in Figure [2.10, 2.11, 2.12] and illustrates the tree construction. In Figure 2.10, E transmits a message with TTL five¹⁰ and the message reaches its parent B . Suppose at some moment a new member C arrives and joins the multicast group (as shown in Figure 2.11). Then the TTL-scoped message from B arrives at C and C can infer its distance from B . Hence, C concludes that its parent is B , according to the two specified rules. However, if the original TTL-scoped message from B died before reaching C , then after a timeout, C launches an extended search to locate its parent. When B receives a query from C , it updates its TTL-scope value accordingly. As a result, the next TTL-scoped message from B will reach C . Finally, C begins to heartbeat its periodic TTL-scoped message, which is scoped to reach its parent B (see Figure 2.12). The message can also be heard by E . From condition C1, E then concludes that C is its new

¹⁰A TTL (Time To Live) value of five means that the message can only travel 5 hops from the originating node.

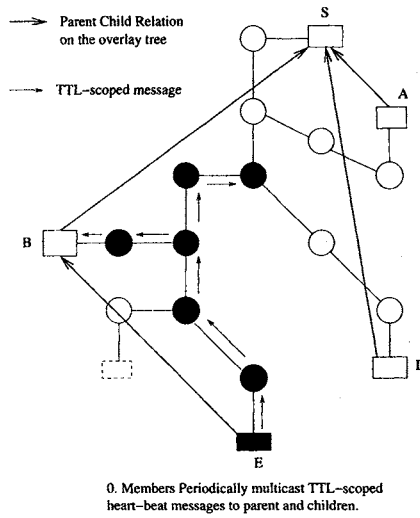


Figure 2.10. Member discovery: periodic multicast (Source: [Banerjee and Bhattacharjee, 2002], © IEEE).

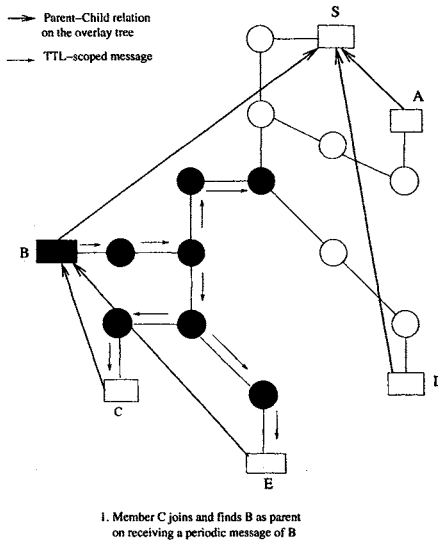


Figure 2.11. Member discovery: member join (Source: [Banerjee and Bhattacharjee, 2002], © IEEE).

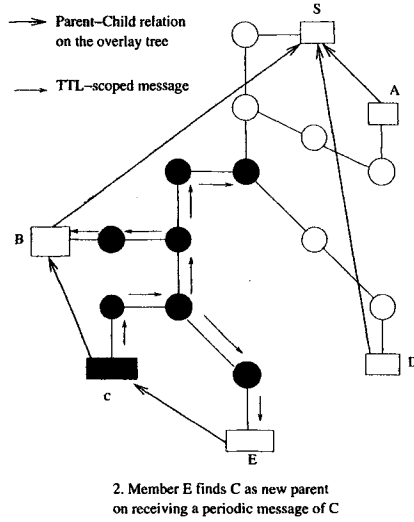


Figure 2.12. Member discovery: new parent discovery (Source: [Banerjee and Bhattacharjee, 2002], © IEEE).

parent on the member overlay tree and adjusts its TTL-scope value (limited to reach C). B is unable to hear periodic messages from E and concludes it is no longer E 's parent. Note that even though E cannot hear the TTL-scoped message from C , it is guaranteed that the periodic message from E , scoped to reach its current parent B , reaches C . Thus C would conclude that it is preferable for himself to be E 's parent and would update its TTL-scope value of the messages intended to reach E .

The clustering protocol will traverse the member overlay tree to create clusters. Apart from the overlay tree, a parameter for cluster size k is provided as input to the clustering protocol. The traversal is performed from the leaves, upward, to the root. During the traversal, whenever members in the interval $[k, 2k - 1]$ are found, they are grouped into a cluster. The clustering protocol then prunes these members from the overlay tree and continues to traverse the remaining overlay tree. At the end of the traversal, all members are grouped into appropriately-sized clusters except the last cluster which may have fewer than k members. The small size of the root cluster does not affect the rekeying process.

2.5 N-party Diffie-Hellman Key Exchange Suite

As described earlier in section 1.3.3 of Chapter 1, the Diffie-Hellman key exchange is a widely used protocol for two parties to agree on a shared secret key over an insecure channel. The protocol is extended to a group of participants to establish a shared secret group key for secure group communications, called n -party Diffie-Hellman key exchange. Similar to the two party Diffie-Hellman protocol, suppose p is a large prime and g is a generator of \mathbb{Z}_p^* . Suppose there are n group members which are labeled as m_0, m_1, \dots, m_{n-1} . Every member m_i selects its own secret component s_i . We discuss here four different schemes for establishing a group key K_G for the group $G = \{m_0, \dots, m_{n-1}\}$ as presented by the scheme proposers.

- (i) $K_G = g^{s_0 s_1 \dots s_{n-1}} \pmod p$
- (ii) $K_G = g^{s_0 s_1 + s_1 s_2 + \dots + s_{n-1} s_0} \pmod p$
- (iii) $K_G = g^{s_{n-1} g^{s_{n-2} g^{\dots g^{s_1} g^{s_0}}}} \pmod p$
- (iv) $K_G = g^{g^{g^{s_0 s_1} g^{s_2 s_3} g^{s_4 s_5} g^{s_6 s_7}}} \pmod p$ (suppose $n=8$)

The schemes (i) through (iv) were proposed by [Ingemarsson et al., 1982, Steiner et al., 1996], [Burmester and Desmedt, 1995, Burmester and Desmedt, 1996], [Steer et al., 1990, Kim et al., 2001], and [Becker and Wille, 1998, Kim et al., 2000], respectively. All these protocols establish a contributory group key with each member having equal contribution to the group key. The protocols (i) to (iii) are discussed in this section and protocol (iv) will be discussed in the next chapter. For the purposes of these protocols, the n members are positioned cyclically at the nodes of an n -gon, so that $m_k = m_j$ when $k \equiv j \pmod n$. It is worth mentioning that in this section, all computations will be performed by modulo p , which may be omitted for simplicity. All these protocols require member serialization. In subsection 2.5.5, we discuss a protocol which removes this limitation.

2.5.1 ING Protocol

The earliest attempt for extending the two-party Diffie-Hellman key exchange to a group key agreement protocol is due to Ingemarsson et al., and

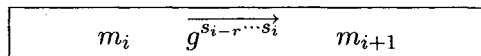


Figure 2.13. ING protocol: round $r \in [0, n - 2]$.

is called ING [Ingemarsson et al., 1982]. The ING protocol consists of $n - 1$ rounds and group members perform every round in synchronization. As already mentioned the members are arranged in a cycle. During the 0^{th} round

each member m_i computes g^{s_i} and passes it to member $m_{(i+1) \bmod n}$. At the end of round $r - 1$, member m_i receives $h_{r-1,i-1} = g^{s_{i-r} \cdots s_{i-1}}$ from m_{i-1} and executes round r , i.e., computes $h_{r,i} = (h_{r-1,i-1})^{s_i} = g^{s_{i-r} s_{i-r+1} \cdots s_i}$ and passes $h_{r,i}$ to m_{i+1} . After the final round, every member is in possession of the group key $g^{s_0 s_1 \cdots s_{n-1}}$.

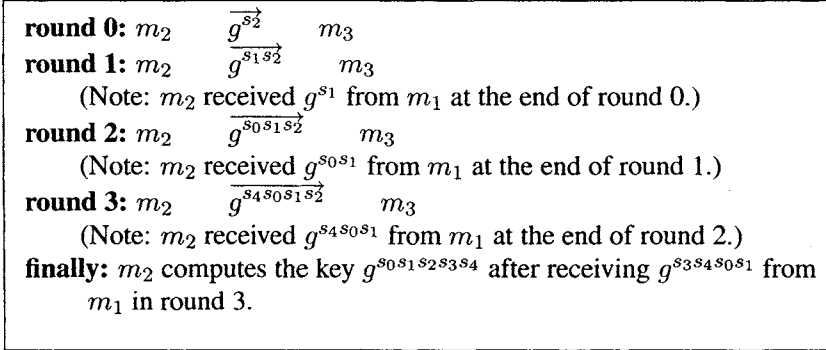


Figure 2.14. ING example for m_2 .

Table 2.1. States for ING example

r		m_0	m_1	m_2	m_3	m_4
0	receive comp/send	-	-	g^{s_2}	g^{s_3}	g^{s_4}
1	receive comp/send	g^{s_4}	g^{s_0}	g^{s_1}	g^{s_2}	g^{s_3}
		$g^{s_4 s_0}$	$g^{s_0 s_1}$	$g^{s_1 s_2}$	$g^{s_2 s_3}$	$g^{s_3 s_4}$
2	receive comp/send	$g^{s_3 s_4}$	$g^{s_4 s_0}$	$g^{s_0 s_1}$	$g^{s_1 s_2}$	$g^{s_2 s_3}$
		$g^{s_3 s_4 s_0}$	$g^{s_4 s_0 s_1}$	$g^{s_0 s_1 s_2}$	$g^{s_1 s_2 s_3}$	$g^{s_2 s_3 s_4}$
3	receive comp/send	$g^{s_2 s_3 s_4}$	$g^{s_3 s_4 s_0}$	$g^{s_4 s_0 s_1}$	$g^{s_0 s_1 s_2}$	$g^{s_1 s_2 s_3}$
		$g^{s_2 s_3 s_4 s_0}$	$g^{s_3 s_4 s_0 s_1}$	$g^{s_4 s_0 s_1 s_2}$	$g^{s_0 s_1 s_2 s_3}$	$g^{s_1 s_2 s_3 s_4}$
4	receive compute	$g^{s_1 s_2 s_3 s_4}$	$g^{s_2 s_3 s_4 s_0}$	$g^{s_3 s_4 s_0 s_1}$	$g^{s_4 s_0 s_1 s_2}$	$g^{s_0 s_1 s_2 s_3}$
		$g^{s_1 s_2 s_3 s_4 s_0}$	$g^{s_2 s_3 s_4 s_0 s_1}$	$g^{s_3 s_4 s_0 s_1 s_2}$	$g^{s_4 s_0 s_1 s_2 s_3}$	$g^{s_0 s_1 s_2 s_3 s_4}$

We consider an example. Suppose there are five members in the group. The steps at member m_2 for each round are shown in Figure 2.14. For clarity, all actions performed are displayed in Table 2.1. Note that steps 0 and 4 are only half rounds. Step 0 does not involve receiving data from a neighboring member, while step 4 does not involve sending data to a neighboring member. Thus, the protocol requires the equivalent of 4 ($= n - 1$) full rounds.

2.5.2 BD Protocol

In 1995, Burmester and Desmedt proposed an efficient group key agreement scheme, called the BD protocol, which requires only two rounds (three steps) [Burmester and Desmedt, 1995, Burmester and Desmedt, 1996]. The BD protocol generates key $K_G = g^{s_0 s_1 + s_1 s_2 + \dots + s_{n-1} s_0} \pmod p$ and is described below. The group members are arranged cyclically as in ING.

1. Every m_i computes and broadcasts $b_i = g^{s_i}$. (b stands for blinded secret.)
2. Every m_i computes and broadcasts $X_i = (b_{i+1}/b_{i-1})^{s_i}$.
3. Every m_i now computes the key $K_i = b_{i-1}^{n s_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \dots X_{i-2}$.

Figure 2.15. BD protocol: two rounds.

As an example, let us examine the protocol from the viewpoint of member m_2 in a group of five members. In the first round, m_2 receives $b_1 = g^{s_1}$ and $b_3 = g^{s_3}$ (and all other b_i). m_2 computes $X_2 = (b_3/b_1)^{s_2} = g^{s_3 s_2 - s_1 s_2}$ and broadcasts X_2 . In the second round, m_2 receives all the remaining X_i . Finally m_2 computes the key $K_G = (b_1)^{5 s_2} \cdot X_2^4 \cdot X_3^3 \cdot X_4^2 \cdot X_0 = g^{5 s_1 s_2} \cdot g^{4 s_2 s_3 - 4 s_1 s_2} \cdot g^{3 s_3 s_4 - 3 s_2 s_3} \cdot g^{2 s_4 s_0 - 2 s_3 s_4} \cdot g^{s_0 s_1 - s_4 s_0} = g^{s_0 s_1 + s_1 s_2 + s_2 s_3 + s_3 s_4 + s_4 s_0}$.

2.5.3 GDH Protocols

In 1996, Steiner et al. proposed a suite of group Diffie-Hellman schemes called GDH.1, GDH.2, GDH.3 [Steiner et al., 1996, Steiner et al., 2000]. The key generated by all three schemes is $K_G = g^{s_0 s_1 \dots s_{n-1}}$. We discuss them one at a time.

2.5.3.1 GDH.1

In GDH.1, there are two stages: *upflow* and *downflow*. In the upflow stage, there are $n - 1$ rounds. The purpose of the upflow stage is to collect contributions from all group members. In round i , m_i receives an ordered collection of intermediate values from m_{i-1} and raises the last intermediate value, i.e., $g^{s_0 \dots s_{i-1}}$, to the power of its secret component s_i , appends it to the incoming flow and forwards the new ordered list to m_{i+1} . For example, m_3 receives from m_2 $\{g^{s_0}, g^{s_0 s_1}, g^{s_0 s_1 s_2}\}$, raises the last value to obtain $g^{s_0 s_1 s_2 s_3}$ and forwards $\{g^{s_0}, g^{s_0 s_1}, g^{s_0 s_1 s_2}, g^{s_0 s_1 s_2 s_3}\}$ to m_4 . In the downflow stage (also $n-1$ rounds), member m_i receives from m_{i+1} an ordered list of intermediate values and raises all of them to the power of its secret component s_i . The last entry of the resulting vector is the group key and m_i forwards all other values to m_{i-1} .

For example, m_3 receives from m_4 $\{g^{s_4}, g^{s_0s_4}, g^{s_0s_1s_4}, g^{s_0s_1s_2s_4}\}$, obtains the key $g^{s_0s_1s_2s_3s_4}$ and forwards $\{g^{s_3s_4}, g^{s_0s_3s_4}, g^{s_0s_1s_3s_4}\}$ to m_2 .

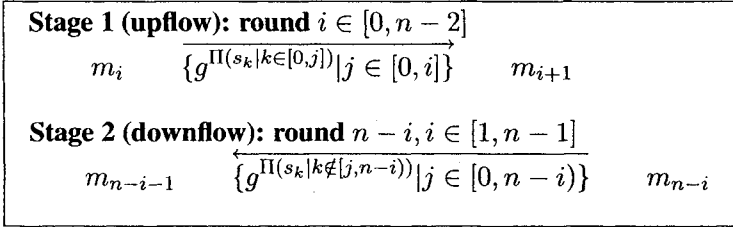


Figure 2.16. GDH.1 protocol: Stage 1 ($n - 1$ rounds), Stage 2 ($n - 1$ rounds).

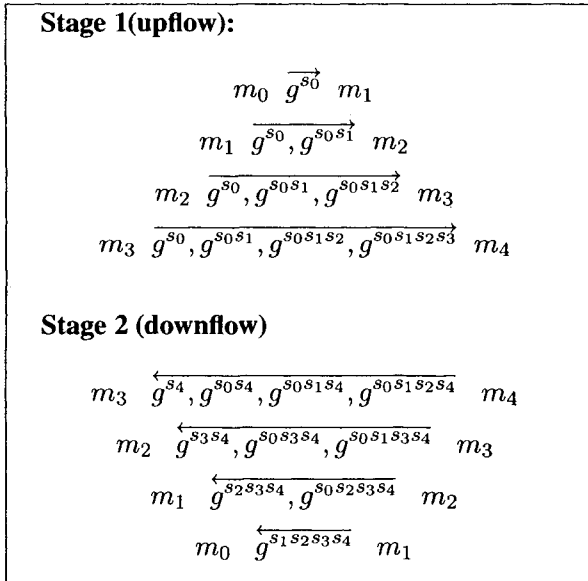


Figure 2.17. GDH.1: an example for 5 members.

The GDH.1 protocol is shown in Figure 2.16 and an example of 5 members is shown in Figure 2.17.

2.5.3.2 GDH.2

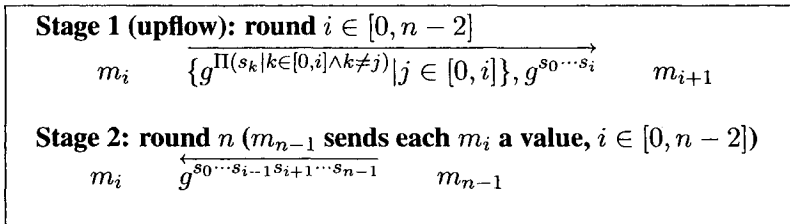


Figure 2.18. GDH.2 protocol: Stage 1 ($n - 1$ rounds), Stage 2 (one round).

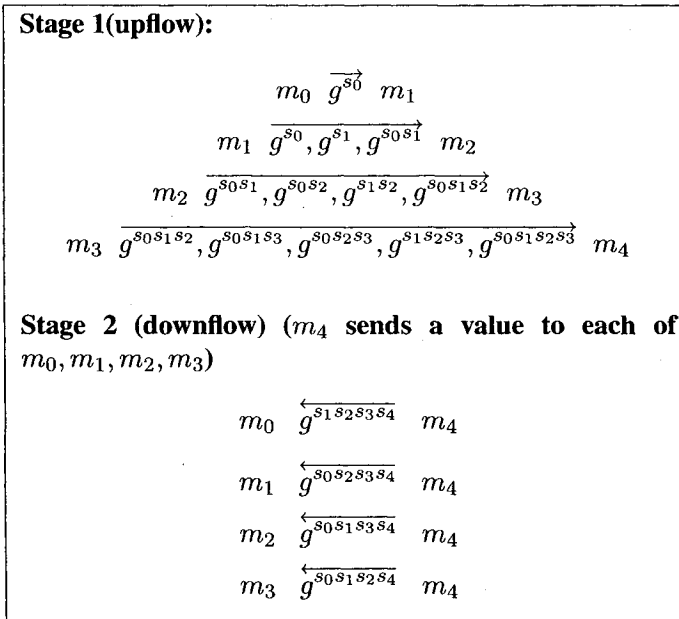


Figure 2.19. GDH.2: an example for 5 members.

GDH.2 reduces the number of rounds in GDH.1 by collecting contributions from all members in stage 1 (upflow) but broadcasting messages by m_{n-1} to all other members in Stage 2 (downflow). In the upflow, every member m_i receives some values from its previous member m_{i-1} , raises all of the values to the power of its secret component s_i and sends them to m_{i+1} . For example,

m_3 will receive from m_2 $\{g^{s_0s_1}, g^{s_0s_2}, g^{s_1s_2}, g^{s_0s_1s_2}\}$ and sends $\{g^{s_0s_1s_2}, g^{s_0s_1s_3}, g^{s_0s_2s_3}, g^{s_1s_2s_3}, g^{s_0s_1s_2s_3}\}$ to m_4 . The GDH.2 protocol is presented given in Figure 2.18 and an example for 5 members is given in Figure 2.19.

2.5.3.3 GDH.3

In GDH.1 and GDH.2, every member m_i will perform $(i + 1)$ exponentiations which is computationally inefficient. The resulting computational overhead is considerable when the group size is large. GDH.3 solves the problem by only requiring every member (except for m_{n-1}) to perform a constant, small number of exponentiations.

GDH.3 involves four stages. The first stage consists of $n - 2$ rounds and is used to collect contributions from members m_0, m_1, \dots, m_{n-2} . In the second stage, m_{n-2} broadcasts $g^{s_0 \dots s_{n-2}}$ to all members, thus m_{n-1} can obtain the key by raising the received value to the power of s_{n-1} . In stage 3, every member m_i extracts its own component s_i from $g^{s_0 \dots s_{n-2}}$ and sends the result to m_{n-1} . Finally in stage 4, m_{n-1} raises all the values to its secret component s_{n-1} and broadcasts these values to members. After receiving values from m_{n-1} , every member can compute the group key. Protocol GDH.3 is shown in Figure 2.20 and an example of 5 members is shown in Figure 2.21.

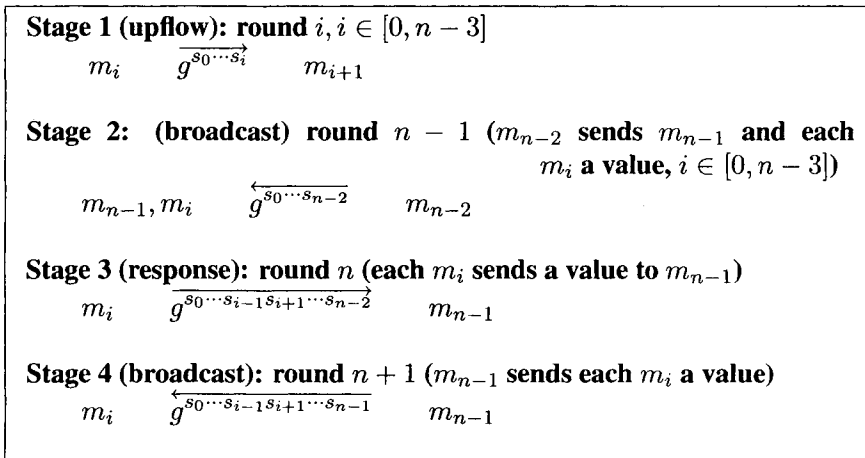


Figure 2.20. GDH.3 protocol: Four stages with stage 1 having $n - 2$ rounds.

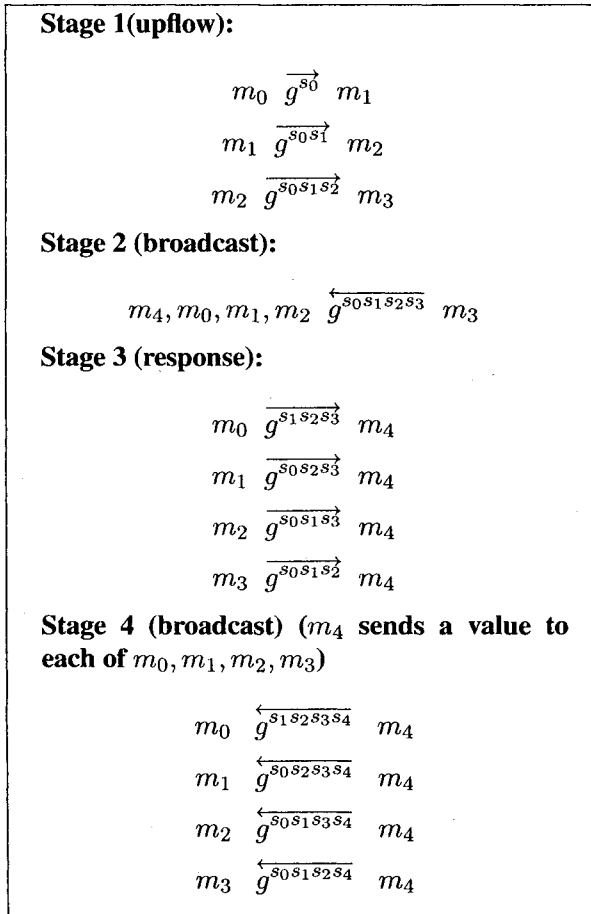


Figure 2.21. GDH.3: an example for 5 members.

2.5.4 STR Protocol

In 1988, Steer et al. [Steer et al., 1990] proposed a group key agreement protocol which generates the key best described by the following recursive definition. Consider the group $\{m_0, m_1, \dots, m_{n-1}\}$, and let p, g and s_i be as in the earlier protocols of this section. Let q_m be defined recursively by:

- (i) $q_0 = s_0$, and
- (ii) $q_m = g^{s_m q_{m-1}}$ for $0 < m < n$

Thus, $q_3 = g^{s_3 q_2} = g^{s_3 g^{s_2 q_1}} = g^{s_3 g^{s_2 g^{s_1 q_0}}} = g^{s_3 g^{s_2 g^{s_1 s_0}}$. The group key generated by Steer's protocol is just $K_G = q_{n-1}$. The protocol processes in two stages with $n - 1$ rounds (See Figure 2.22). An example for 5 members is shown in Figure 2.23.

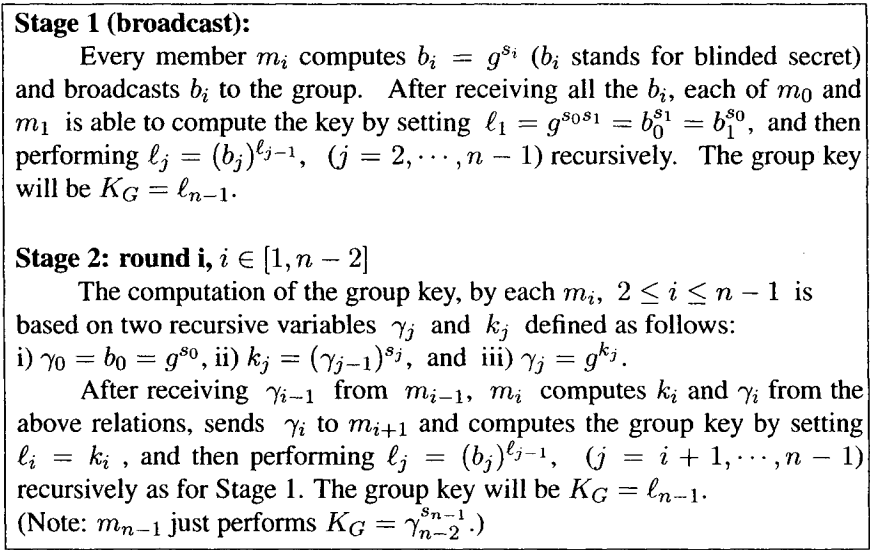


Figure 2.22. Steer's protocol: Stage 1 (broadcast), Stage 2 (n-2 rounds).

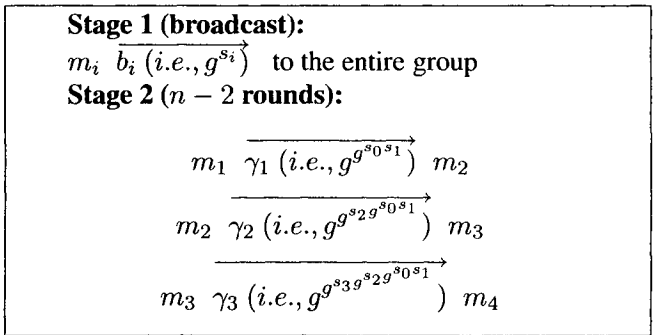


Figure 2.23. Steer's protocol: an example for 5 members.

Steer’s protocol was improved by Kim et al. in 2001 [Kim et al., 2001]. The improved protocol reduces the $n - 1$ rounds in Steer’s protocol to just 2 rounds and is called the STR protocol and is shown in Figure 2.24.

The scheme in [Becker and Wille, 1998, Kim et al., 2000] bases the formation of the group key on a binary tree structure, which will be discussed in detail in Section 3.2.1 of Chapter 3.

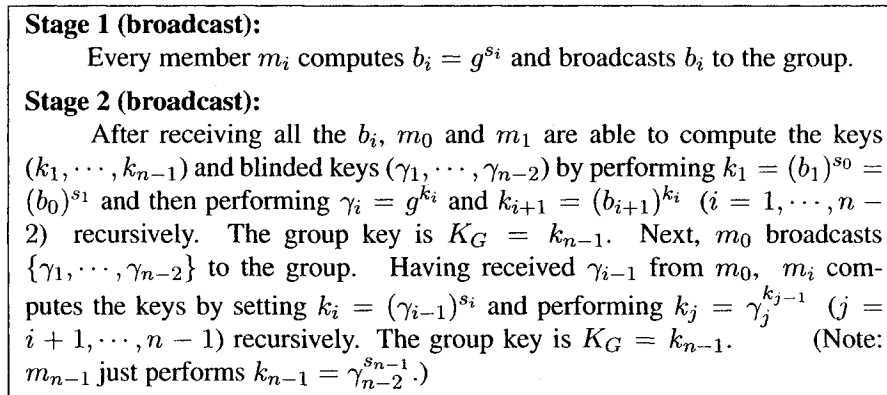


Figure 2.24. STR protocol: Stage 1 (broadcast), Stage 2 (broadcast).

2.5.5 A Protocol without Member Serialization

The above protocols have a requirement of serialization, that is, group members must be numbered serially and key computations must be executed serially as well. This fails to be efficient in certain environments such as large ad hoc networks ¹¹. In [Yasinsac et al., 2002], the authors propose a protocol which removes this limitation. We discuss this protocol in what follows.

As in the previous protocols, suppose the group members¹² are m_0, m_1, \dots, m_{n-1} and g and p are publically known. The members first select a coordinator (e.g., m_c) if they want to perform group communication. Then the protocol is executed in two rounds: (1) Each non-coordinator member m_i ($i \neq c$) selects its private share s_i and broadcasts its public share g^{s_i} , (2) The coordinator m_c generates its private share s_c and a random number z , computes its

¹¹Ad hoc networks are wireless networks formed by communicating nodes “on the fly” in an “ad hoc” manner without requiring any installed infrastructures (such as access points/base stations)

¹²The member names can be considered as their identifications. There is no need for these names to be serially ordered.

public share g^{s_c} , computes $g^{s_c s_i}$ for each $i \neq c$, encrypts z with each $g^{s_i s_c}$ respectively, and broadcasts g^{s_c} along with multiple encryptions¹³ of z , i.e., $(g^{s_c}, (\{z\}_{g^{s_c s_0}}, m_0), \dots, (\{z\}_{g^{s_c s_{c-1}}}, m_{c-1}), (\{z\}_{g^{s_c s_{c+1}}}, m_{c+1}), \dots, (\{z\}_{g^{s_c s_{n-1}}}, m_{n-1}))$. After receiving the broadcast from m_c , each m_i ($i \neq c$) computes $g^{s_i s_c}$, decrypts z , and computes the group key as $K_G = g^{f(g^{s_0}, g^{s_1}, \dots, g^{s_n}) \circ z}$, where f is some predefined combining function such as a secure hash function and \circ is a binary operator such as XOR [Yasinsac et al., 2002]. Clearly, m_c can also compute K_G .

There are two problems with this protocol. First, the protocol requires that a coordinator be selected prior to the key generation process. This will postpone and delay group communications. Secondly, the coordinator can become a single point of failure. The protocol can be modified without an initial selection of coordinator as follows. In the first round, each member m_i selects its private share s_i , computes its public share g^{s_i} , and broadcasts (g^{s_i}, m_i) . In the second round, each member m_i selects a secret number z_i , computes $g^{s_i s_j}$ for all $j \neq i$, encrypts z_i with $g^{s_i s_j}$ for all $j \neq i$ respectively, and broadcasts $((\{z_i\}_{g^{s_i s_0}}, m_0), \dots, (\{z_i\}_{g^{s_i s_{i-1}}}, m_{i-1}), (\{z_i\}_{g^{s_i s_{i+1}}}, m_{i+1}), \dots, (\{z_i\}_{g^{s_i s_{n-1}}}, m_{n-1}))$. The group key is $K_G = f(z_0, \dots, z_{n-1})$ where f is some predefined function such as $f(z_0, \dots, z_{n-1}) = z_0 \oplus \dots \oplus z_{n-1}$. All the members can compute the group key K_G after receiving the broadcast messages in the second round. We call this modified protocol the *non-serialized n-party Diffie-Hellman key exchange* protocol, denoted by “NON-SERIAL”.

2.5.6 Summarization of n -party Diffie-Hellman Key Agreement Protocols

We provide a comparison summary of current n -party Diffie-Hellman protocols in Table 2.2.

¹³ $\{x\}_k$ stands for encrypting x under key k using some secret-key cryptosystem.

Chapter 3

TREE BASED KEY MANAGEMENT SCHEMES

Among all group key management protocols, the *key tree* scheme¹ provides a very powerful approach which can be used for one-to-many communications, many-to-many communications, in centralized key distribution systems, as well as systems utilizing distributed, contributory key-agreement protocols.

Tree based schemes were independently proposed by several research groups. The first such scheme was proposed in July, 1997 by Wallner et al. [Wallner et al., 1998] and at the suggestion of A. Sherman was named *Logical Key Hierarchy* (LKH). The name LKH was adopted by Harney and Harder in their 1999 implementation of the algorithm [Harney and Harder, 1999]. In July 1997, Wong et al. [Wong et al., 1998] proposed a generalized key tree scheme called *key graph*. In 1998, Caronni et al. [Caronni et al., 1998] published an idea similar to LKH, based on set theory. In 1998 Noubir [Noubir, 1998] proposed another idea similar to LKH.

A more efficient scheme than LKH, based on the idea of a *One-way-Function Tree* (OFT) was proposed by A. Sherman et al. [Sherman and McGrew, 2003]. Moreover a scheme proposed by Canetti et al. [Canetti et al., 1999a, Canetti et al., 1999b] was named *One-way Function Chain* (OFC) by Sherman, and can be considered as a variant of OFT.

As indicated in Chapter 1, bursty behavior is an important feature in SGC and bulk operations are effective in reducing the number of rekeying mes-

¹[Canetti et al., 1999a, Caronni et al., 1998, Noubir, 1998, Sherman and McGrew, 2003, Wallner et al., 1998, Wong et al., 1998, Zhang et al., 2001]

sages. In this chapter, we discuss LKH, OFT, OFC, efficient bursty operations and implementation. LKH, OFT, and OFC are centralized key distribution schemes. They have been extended or modified to distributed, contributory key agreement schemes. In this chapter, we also discuss tree based distributed schemes such as *Distributed Scalable sE cure Communication* (DISEC), *Tree based Group Diffie-Hellman* (TGDH), and *Block-Free TGDH* (BF-TGDH).

3.1 Centralized Key Distribution based on Tree Structures

3.1.1 Key Tree – Logical Key Hierarchy (LKH)

We briefly introduce key tree based key management [Caronni et al., 1998, Noubir, 1998, Wallner et al., 1998, Wong et al., 1998]. In such schemes there is an associated binary tree, called the *key tree* or the *Logical Key Hierarchy* (LKH). There is also a Group Controller (GC), which maintains the multicast group and manages a virtual tree (see Figure 3.1). The members of the group are placed at leaf nodes of the tree. The nodes in the tree are assigned keys. The key at the root is the so called *Traffic Encryption Key* (TEK) and all other keys are *Key Encryption Keys* (KEKs). Every member is assigned the keys along the path from its leaf to the root. Keys at leaf nodes are possessed by individual members, while keys at an internal node are shared by its descendant members. Finally, the key at the root node (TEK) is shared by all members. When a message is sent, it is encrypted with TEK, and all the members can decrypt it using TEK.

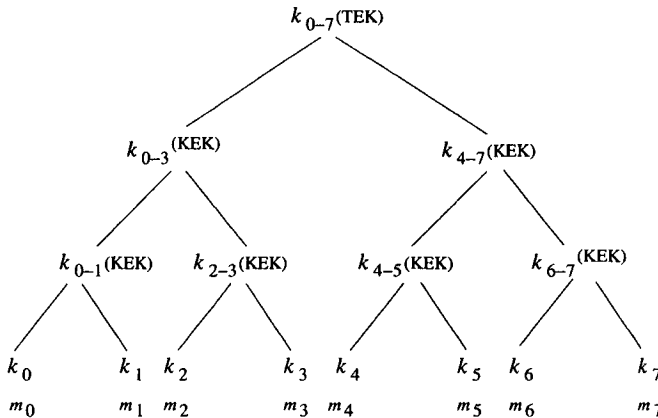


Figure 3.1. Key tree: each member assigned the keys from its leaf to the root.

When a member m leaves the group, all the keys that m knows and shares with other members (i.e., the keys from the root to the parent of m) need to be changed. The GC changes these keys from the bottom up. Every changed key

is encrypted using its children's keys (some of which are new changed keys) and the result is broadcast to all members. For example, when member m_2 leaves the group (see Figure 3.2), keys $k_{2-3}, k_{0-3}, k_{0-7}$ are initially changed

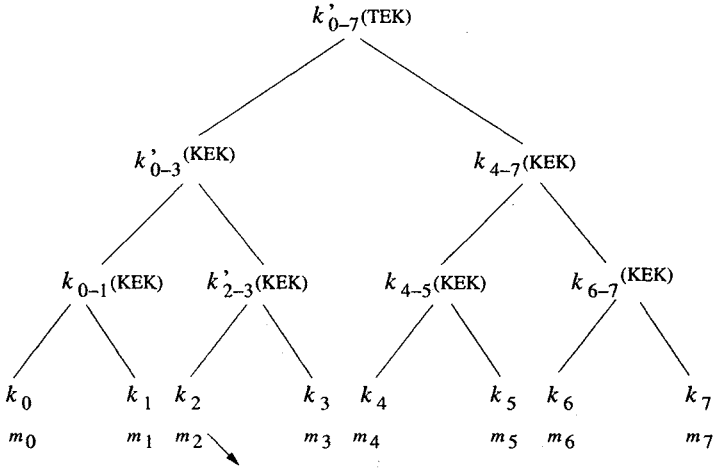


Figure 3.2. m_2 leaves the group.

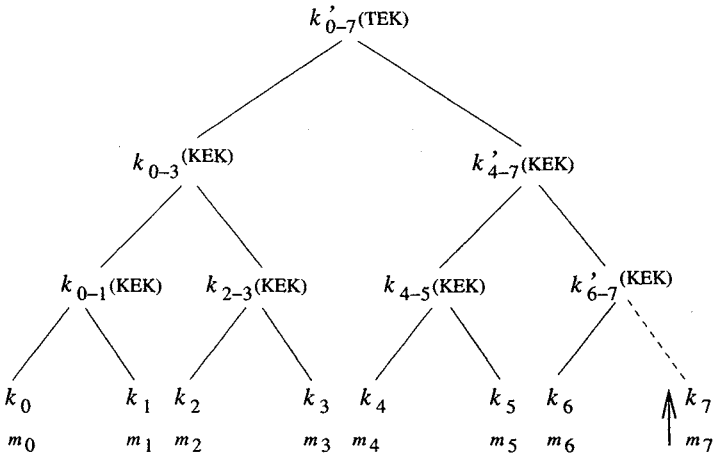


Figure 3.3. Member m_7 joins the group.

to $k'_{2-3}, k'_{0-3}, k'_{0-7}$. Then, k'_{2-3} is encrypted using key k_3 ; k'_{0-3} using keys k_{0-1} and k'_{2-3} ; and k'_{0-7} using k'_{0-3} and k_{4-7} .

When a member m joins, having been authenticated by the GC, the GC decides where it should be placed in the tree and changes all the keys from the parent of the joining member to the root. Then the GC encrypts each changed key, at node x , using the keys of the children of x , and broadcasts the encrypted keys to the group. For example, suppose the current group consists of members m_0 to m_6 and that member m_7 is to join the group (see Figure 3.3). The new, changed keys k'_{6-7} , k'_{4-7} and k'_{0-7} are encrypted using keys k_6 and k_7 , k_{4-5} and k'_{6-7} , and k_{0-3} and k'_{4-7} respectively. Consider also the case where there is no space at the lowest level for a joining member. Suppose, for instance, that m_8 is to join the group while the level of m_0, \dots, m_7 is completely occupied. Then, node m_0 becomes an internal node $m_{0,8}$, with two descendants m_0 and m_8 . Thus a new level is created and more members can be accommodated.

One specific feature associated with *key tree* is that joining and leaving can be processed in the same way. Moreover the number of keys which need to be changed for a join or leave is $O(\log(n))$ and the number of encryptions is $O(2\log(n))$ where n is the number of members in the multicast group. The encrypted keys can be sent as separate rekeying messages or can be packed in a single rekeying message.

3.1.2 Bursty Behavior and its Efficient Implementation

3.1.2.1 Bursty Behavior and Properties

First, we present an example to illustrate how an *aggregate operation* reduces the number of keys which need to be changed (see Figure 3.4). Suppose the current members in the group are $\{m_0, m_1, m_2, m_3, m_4, m_6, m_7\}$, and that four members $\{m_0, m_1, m_4, m_6\}$ will be leaving, while, at the same time, five other members will be joining the group. If we use a *split operation*, nine separate operations are needed with $\log_2(8) = 3$ keys that need to be changed for each operation. Therefore, the total number of changed keys is $9 \times 3 = 27$. If we process the four leaves and five joins at the same time, i.e., in one *aggregate operation*, also called *bulk operation*, then the total number of keys that need to be changed is 6 (all internal keys except for the parent of m_2 and m_3) (see Figure 3.4). Thus, an 88% (i.e., $1 - \frac{6}{27}$) reduction in the total number of key changes is achieved. The reason for this improvement is that in an *aggregate operation* any shared key is changed only once.

Bursty behavior has some special properties, which can be clarified by means of the two lemmas that follow (their proofs are omitted because they are rather obvious). The first lemma states that without considering the keys on leaf nodes², the operations of joining and leaving are equivalent. The second lemma

²Because the keys on leaf nodes are only possessed by individual members and are only used to encrypt messages between the group controller and individual members, whether or not they need to be changed or

states that when j joins and ℓ leaves are combined in one *aggregate operation*, the cost for rekeying $\min\{j, \ell\}$ keys will be completely saved (covered by $\max\{j, \ell\}$). Therefore, we uniformly call both joins and leaves *updates*.

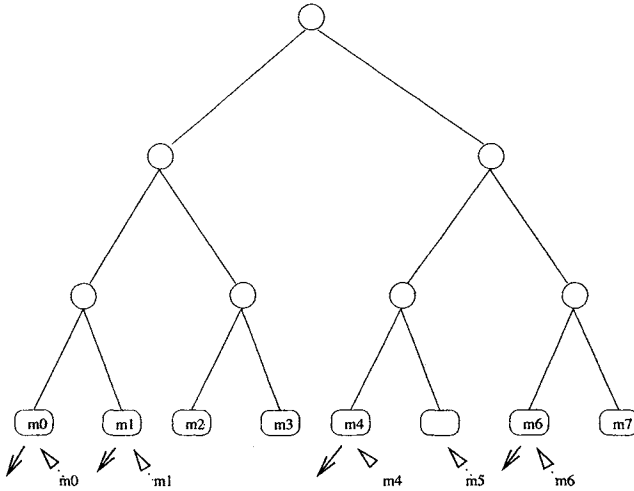


Figure 3.4. Bursty behavior: multiple joins and leaves simultaneously.

LEMMA 3.1.1 Suppose the number of joins is j and the number of leaves is ℓ . If $j = \ell$ and the positions of joining members in the tree are the same as the positions of the leaving members, then the keys that need to be changed for the joins in one aggregate operation are exactly the same as the keys that need to be changed for the leaves in one aggregate operation.

LEMMA 3.1.2 Suppose the number of joins is j and the number of leaves is ℓ . If we perform both joins and leaves in one aggregate operation, the number of keys that need to be changed for these $j + \ell$ members is the same as the number of keys that need to be changed for $m = \max\{j, \ell\}$ members.

how they will be changed does not affect an *aggregate operation*. In fact, we may assume that keys on leaf nodes are the public keys of individual members.

3.1.2.2 Bursty Algorithm

In order to perform multiple joins and/or multiple leaves in one *aggregate operation*, it is necessary to find the common keys shared by joining and/or leaving members in an efficient way. In order to do that, we represent keys by bit-strings and use only binary *right shift* operations. The detailed mechanism is described below.

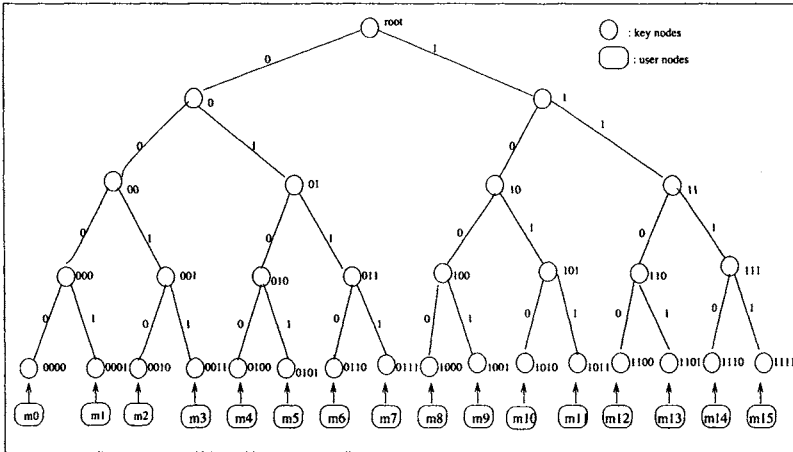


Figure 3.5. Binary key tree.

All nodes are represented by bit-strings. The root is represented by the empty string ε . Suppose a node is labeled by bit-string v , then its left child will be represented by $v0$ and its right child by $v1$. The key associated with node v is represented by k_v . Thus, the root key is $k_\varepsilon = k$. The keys at level 1 are k_0, k_1 ; the keys at level 2 are $k_{00}, k_{01}, k_{10}, k_{11}$ and so on (see Figure 3.5).

In general, a node at level ℓ is represented by some bit-string $i_{\ell-1} \cdots i_0$ of length ℓ , and its key by $k_{i_{\ell-1} \cdots i_0}$. Each member is assigned a random non-negative integer as its identification (ID). These integer ID's can be represented as bit-strings³. Suppose that the tree height is h , so that the group size could be as large as 2^h , and let a member m_i have ID equal to i . We represent i in binary form as $i_{h-1} \cdots i_0$. The member m_i is placed at node $i_{h-1} \cdots i_0$ and is assigned the keys along the path from the node to the root, i.e., $k_{i_{h-1} \cdots i_1 i_0}, k_{i_{h-1} \cdots i_1}, \dots, k_{i_{h-1}}, k$.

³When an integer i is stored in the computer, it is generally in the binary form; hence, there is no need to perform representation transformations.

Next we consider the process of finding the common keys shared by members. From Lemma 3.1.2, it is reasonable to assume that all members in question join a group or they all leave a group. We consider two cases separately: two members and many members.

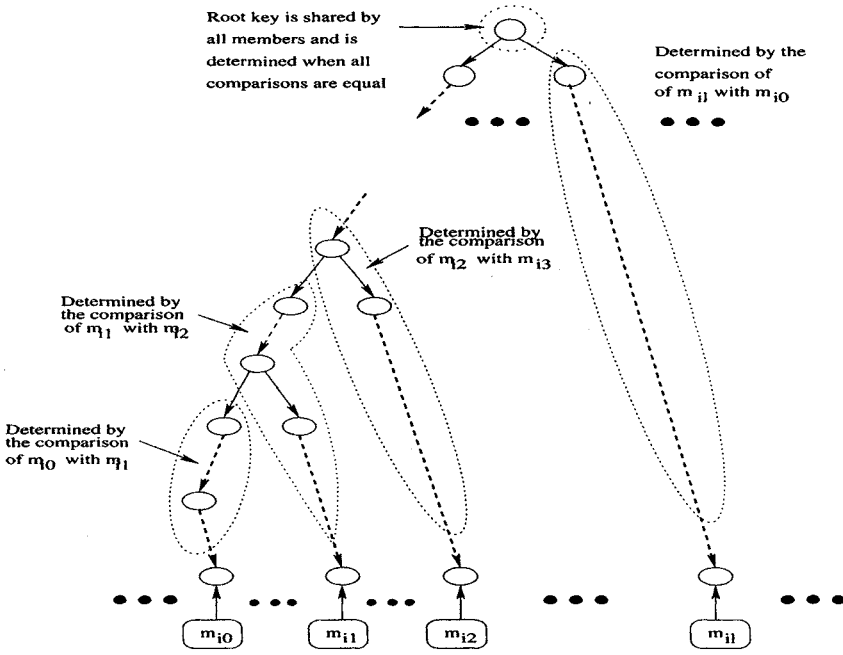


Figure 3.6. Changed keys are determined by the comparisons of neighboring members.

Suppose the two members in question are m_i and m_j . We can decide their shared keys by comparing their corresponding bit-strings as discussed below. Since $i \neq j, i_{h-1} \dots i_1 i_0 \neq j_{h-1} \dots j_1 j_0$ and $k_{i_{h-1} \dots i_1 i_0} \neq k_{j_{h-1} \dots j_1 j_0}$. Let s be the smallest index such that $i_{h-1} \dots i_s = j_{h-1} \dots j_s$, then $s \geq 1, k_{i_{h-1} \dots i_s} = k_{j_{h-1} \dots j_s}$, and all keys along the path from node $i_{h-1} \dots i_s$ to the root are shared keys. The only operation needed here is to find the prefix of an integer, which can be implemented by the *right shift* operation \gg .

Let us now consider $m > 2$ members. We have the following theorem.

THEOREM 3.1 *Suppose members $m_{i_0}, \dots, m_{i_{m-1}}$ are to be processed in one aggregate operation and assume they are already sorted by their ID. The keys that need to be changed can be determined by the comparisons of IDs of neigh-*

boring members in a cyclic manner, i.e., i_0 with i_1 , i_1 with i_2 , ..., i_{m-2} with i_{m-1} , and i_{m-1} with i_0 . The keys shared by all the members are determined when all neighboring comparisons yield equalities.

Proof: First let us consider any key at level ℓ which is only possessed by one member m_{i_j} . Then this key will be $k_{i_{j_{h-1}} \dots i_{j_{h-\ell}}}$. Consider the next member's ID i_{j+1} (if m_{i_j} is the last member $m_{i_{m-1}}$ then consider the first member's ID i_0). It is true that $i_{j_{h-1}} \dots i_{j_{h-\ell}} \neq i_{(j+1)_{h-1}} \dots i_{(j+1)_{h-\ell}}$, otherwise, $k_{i_{j_{h-1}} \dots i_{j_{h-\ell}}}$ will be shared by these two members. Therefore, this key can be found by comparing $i_{j_{h-1}} \dots i_{j_{h-\ell}}$ with $i_{j+1_{h-1}} \dots i_{j+1_{h-\ell}}$. Next let us consider any key at level ℓ which is shared by a subset of members $m_{i_j}, m_{i_{j+1}}, m_{i_{j+t}}$. Then again, this key will be $k_{i_{j_{h-1}} \dots i_{j_{h-\ell}}}$. It is true that $i_{j_{h-1}} \dots i_{j_{h-\ell}} = i_{(j+1)_{h-1}} \dots i_{(j+1)_{h-\ell}} = \dots = i_{(j+t)_{h-1}} \dots i_{(j+t)_{h-\ell}}$. Consider the next member's ID i_{j+t+1} . If $m_{i_{j+t}}$ is the last member, then consider i_0 . Since $i_{(j+t)_{h-1}} \dots i_{(j+t)_{h-\ell}} \neq i_{(j+t+1)_{h-1}} \dots i_{(j+t+1)_{h-\ell}}$, the shared key $k_{i_{j_{h-1}} \dots i_{j_{h-\ell}}}$ can be found by comparing $i_{(j+t)_{h-1}} \dots i_{(j+t)_{h-\ell}}$ with $i_{(j+t+1)_{h-1}} \dots i_{(j+t+1)_{h-\ell}}$. Finally, consider a key at level ℓ which is shared by all members $m_{i_0}, \dots, m_{i_{m-1}}$. It is true that $i_{0_{h-1}} \dots i_{0_{h-\ell}} = i_{1_{h-1}} \dots i_{1_{h-\ell}} = \dots = i_{(m-1)_{h-1}} \dots i_{(m-1)_{h-\ell}}$. Thus, all neighboring comparisons yield equalities. Or, to say it in another way, when all comparisons of neighboring members are equal, a shared key is found. Once a key shared by all members is found, there is no need to do further comparisons. The theorem is proved. \square

Figure 3.6 illustrates the comparisons of neighboring members. In order to make the comparison of i_{m-1} with i_0 be the same as for the other comparisons, we add an $(m + 1)$ st member and let its ID be the ID of the first one, i.e., $i_m = i_0$. The algorithm **Rekeying-in-Key-Tree** ($h, m, i[]$) (see Figure 3.7) finds all keys which need to be changed. A changed key will be encrypted, using the keys of its two children in the tree, and will be broadcast separately (thus, requiring $2 \times \text{Numberofkeyschanged}$ rekeying messages), or in one packet (requiring $\text{Numberofkeyschanged}$ rekeying messages). It is also possible that all encrypted changed keys are put in one packet, requiring one rekeying message.

```

Rekeying-in-Key-Tree ( $h, m, i[\ ]$ ):

  /* $h$ : tree height;  $m$ : NO. of updates;  $i_0, i_1, \dots, i_{m-1}, i_0$ : update IDs*/
  /* $i_0$  is appended to simplify comparison between  $i_{m-1}$  and  $i_0$ .*/
  NumberOfKeysChanged = 0;
  for ( $s = 1; s \leq h; s++$ ) {
    SameSubtree=true;
    for ( $j = 0; j < m; j++$ )
      if ( $((i_j \gg s) \neq (i_{j+1} \gg s))$ ) {
        SameSubtree=false;
        the key  $k_{i_{j_{h-1}} \dots i_{j_s}}$  is not a shared key and change it;
        NumberOfKeysChanged++; }
    if (SameSubTree) {
      /*the keys shared by all members are found*/
      change  $k_{i_{0_{h-1}} \dots i_{0_s}}, k_{i_{0_{h-1}} \dots i_{0_{s+1}}}, \dots, k$ ;
      NumberOfKeysChanged +=  $h - s$ ;
      break; }
  }
  /*exit from for loop*/

```

Figure 3.7. Rekeying-in-Key-Tree Algorithm.

3.1.2.3 Theoretical Analysis

In this section we analyze the performance of the algorithm, from a theoretical point of view, in terms of the number of keys that need to be changed. We consider the best, worst and average cases. Suppose the group size is $n = 2^h$ where h is the height of the key tree. We denote the leaf nodes by $0, 1, \dots, n - 1$. Suppose there are m ($m < n$) updates and $k = \lfloor \log(m) \rfloor$. If $m = n$, then the number of changed keys in the best, worst, and average cases are the same, i.e., all $n - 1$ keys need to be changed.

Best case performance It is clear that when all updates occur at consecutive positions, the updates share more keys. Furthermore, it can be proved easily that when m updates occur at leaf nodes $0, \dots, m - 1$, the number of keys needing to be changed is minimized. It is not hard to derive that the number of keys that need to be changed in this case is:

$$N_{best} = \sum_{\ell=0}^{h-1} \left(1 + \left\lfloor \frac{m-1}{2^{h-\ell}} \right\rfloor\right) = h + \sum_{\ell=0}^{h-1} \left\lfloor \frac{m-1}{2^{h-\ell}} \right\rfloor$$

where $(1 + \lfloor \frac{m-1}{2^{h-\ell}} \rfloor)$ is the number of keys that need to be changed at level ℓ .

Worse case performance Intuitively, if the m updates are uniformly distributed among leaf nodes $0, \dots, n - 1$, the keys shared among the m updates is

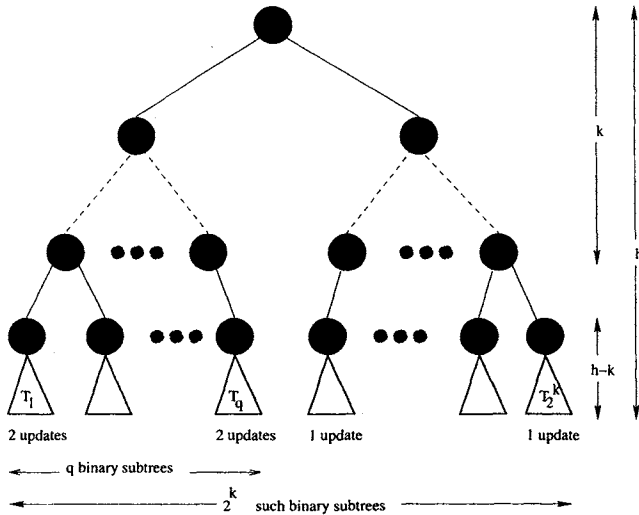


Figure 3.8. Worst case scenario.

minimized, and consequently the number of keys needing to be changed would be maximized. Suppose $2^k \leq m < 2^{k+1}$ where $m = 2^k + q$, $0 \leq q < 2^k$. Figure 3.8 shows⁴ one of the worst case scenarios. Basically, a worst case scenario occurs when the m updates are evenly distributed among the binary subtrees T_1, \dots, T_{2^k} . It is not hard to derive the number of keys changed in the worst case scenario:

$$N_{worst} = m \cdot (h - k - 1) + (2^{k+1} - 1)$$

where the second term indicates all shared keys among the m updates while the first item indicates the keys contributed individually by each of the m updates.

Average case performance For a given m , what is the average number of keys changed? We deal with this question in two steps. We first consider the average number of keys changed at a certain level. Then we sum the average number of changed keys over all levels to get the result. In the discussion that follows we use the convention that the root is at level 0, and the leaf nodes are at level h , where $h = \log(n)$.

Note that all nodes at the same level are equivalent. If we can determine the probability that a key at a node needs to be changed, then we can find the total

⁴The figure is drawn following a figure in [Li et al., 2000]

number of keys changed at that level, by multiplying the probability with the number of nodes at that level.

Consider a node x at level ℓ , $0 \leq \ell \leq h - 1$. As indicated in the previous section, we do not need to consider keys at level h because they are individual keys. The binary subtree rooted at node x has $n' = 2^{h-\ell}$ leaves. If one of these leaves is updated, then the key at node x needs to be changed, otherwise, the key at node x will remain unchanged. The probability that none of the m updates is located in the binary subtree rooted at x is:

$$prob(\ell) = \frac{\binom{n-n'}{m}}{\binom{n}{m}}$$

where $\binom{a}{b}$ is the usual binomial coefficient, counting the number of subsets of size b in a set of size a . We use the fact that $\binom{a}{b} = 0$ if $b > a$.

Thus, the number of keys changed at level ℓ is $2^\ell(1 - prob(\ell))$. Summing up over all levels, we obtain the average number of changed keys:

$$N_{average} = \sum_{\ell=0}^{h-1} 2^\ell \left(1 - \frac{\binom{n-n/2^\ell}{m}}{\binom{n}{m}}\right) = n - 1 - \sum_{\ell=0}^{h-1} 2^\ell \cdot \frac{\binom{n-n/2^\ell}{m}}{i} \binom{n}{m}$$

3.1.3 $d - ary$ Key Tree

In previous sections, we discussed group key management based on a binary key tree. Such a binary key tree can be generalized to a $d - ary$ key tree ($d \geq 2$) [Wong et al., 1998, Wong et al., 2000]. Moreover, based on a fine-tuned multicast capability, i.e., subgroup or directed multicast, the rekeying strategies can be classified in detail as *member-oriented*⁵, *key-oriented*, and *group-oriented*⁶. We discuss the generic key tree in this section⁷.

In a $d - ary$ key tree, each internal node can have up to d children. The members are placed at the leaf nodes. For the same number of members, the larger d is, the shorter the key tree. Each member has the keys of all nodes in the path from its leaf to the root. The root key is shared by all the members and is the group key. Whenever a member joins or leaves the group, the keys along the path from the member to the root need to be changed and distributed to appropriate members.

⁵In the original paper [Wong et al., 1998, Wong et al., 2000], this class is called user-oriented rekeying.

⁶The rekeying method for the binary key tree discussed in the previous sections is similar to group-oriented rekeying.

⁷Portions reprinted, with permission of the authors, IEEE and ACM, from C. K. Wong et al., *Secure Group Communications Using Key Graphs* in IEEE/ACM Transactions on Networks, 8(1), 2000, pages: 16–30. © 2002 IEEE/ACM.

Figure 3.9 shows an example⁸ in a ternary key tree ($d = 3$). In this example, the lower figure illustrates the key tree after member m_9 joins the key tree in the upper figure. Correspondingly, the upper figure can be viewed as the key tree after member m_9 leaves the key tree in the lower figure. For both join and leave, the keys from the parent of m_9 to the root need to be changed, i.e., k_{78} to k_{789} and k_{1-8} to k_{1-9} for the join and k_{789} to k_{78} and k_{1-9} to k_{1-8} for the leave. We explain the three rekeying strategies by means of this example. Suppose the height of the d -ary key tree is h , that the original key of a node x is denoted by k_x and its new key by k'_x .

3.1.3.1 Member-Oriented Rekeying

The idea of member-oriented rekeying is that for each member (or set of members), one rekeying message is constructed. Let S be a set of members, with $|S| \geq 1$. The rekeying message contains exactly the new keys needed by the members of S and is encrypted using a key held by the members of S . In other words, multiple new keys can be encrypted together using one key and transmitted to the members of S . The transmission is accomplished by either unicast or subgroup multicast.

For example, the encryptions and rekeying messages when m_9 joins are as follows:

$$\begin{aligned} GC \rightarrow \{m_1, \dots, m_6\} & : \{k_{1-9}\}_{k_{1-8}} \\ GC \rightarrow \{m_7, m_8\} & : \{k_{1-9}, k_{789}\}_{k_{78}} \\ GC \rightarrow m_9 & : \{k_{1-9}, k_{789}\}_{k_9} \end{aligned}$$

The notation “ $GC \rightarrow S : M$ ” stands for “GC sending message M to the members in set S ”, while “ $\{x_1, \dots, x_\ell\}_k$ ” stands for the result of encrypting values $\{x_1, \dots, x_\ell\}$ with key k . Note that the first rekeying message, destined to members m_1, \dots, m_6 , is encrypted with k_{1-8} which is also held by m_7 and m_8 . This will not result in security breach since m_7 and m_8 are eligible to receive k_{1-9} .

The number of rekeying messages is h . However, each rekeying message corresponds to a node x in the path from the parent of the joining member to the root and contains encryption of multiple new keys, i.e., those along the path from node x to the root. These new keys are encrypted by x 's old key k_x . As a result, the total number of encrypting operations is:

$$1 + 2 + \dots + (h - 1) + (h - 1) = \frac{h(h+1)}{2} - 1$$

where 1 counts the encryption of the root key at level 0, 2 counts the encryptions of the node key at level 1 and the root key, \dots , etc. with the first $(h - 1)$

⁸The example is excerpted from [Wong et al., 2000] and member naming begins from m_1 rather than m_0 .

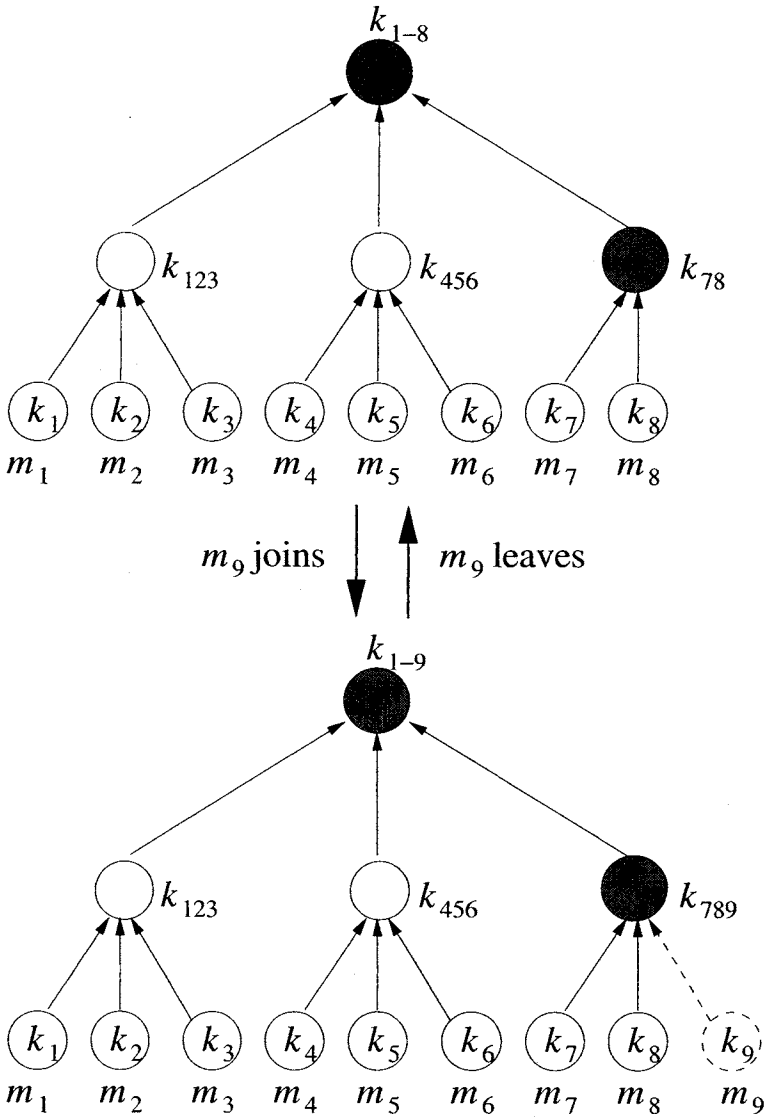


Figure 3.9. 3 – ary key tree and join/leave operation (Source: [Wong et al., 2000], © IEEE/ACM).

counting the encryptions of the node key at level $h - 1$ and all successive keys on the path to the root. The second quantity $(h - 1)$ counts the encryptions of $(h - 1)$ keys destined to the joining member.

Let us examine the member-oriented rekeying process when m_9 leaves the group (see Figure 3.9). The GC sends the following rekeying messages:

$$\begin{aligned} GC \rightarrow \{m_1, m_2, m_3\} & : \{k_{1-8}\}_{k_{123}} \\ GC \rightarrow \{m_4, m_5, m_6\} & : \{k_{1-8}\}_{k_{456}} \\ GC \rightarrow m_7 & : \{k_{1-8}, k_{78}\}_{k_7} \\ GC \rightarrow m_8 & : \{k_{1-8}, k_{78}\}_{k_8} \end{aligned}$$

In this case, for each node x whose key is changed, there are $(d-1)$ rekeying messages. Each of the rekeying messages corresponds to one of x 's unchanged child nodes y , contains the new keys from x up to the root, and is encrypted using the key of node y . Therefore the number of rekeying messages is $(d-1)(h-1)$. Thus, the total number of encrypting operations is:

$$(d-1)(1+2+\dots+(h-1)) = \frac{(d-1)h(h-1)}{2}$$

3.1.3.2 Key-Oriented Rekeying

The idea of member-oriented rekeying is that each changed key is encrypted individually. In case of a member's join, for each node x whose key has been changed, say, from k_x to k'_x , there are two encryptions, one using key k_x and the other using the joining member's key. The first encryption is multicast to the set of members who are under node x , except for the joining member. The second encryption is unicast to the joining member. In case of a member's leave, each changed key is encrypted by its children's keys respectively. Note that encryptions are performed in an order from bottom to the root and a new key of a node x is used in the encryption of the new key of x 's parent.

In the example of Figure 3.9, the rekeying messages for m_9 's join are shown below. Here the number of rekeying messages and the number of encryptions are same, i.e., $2(h-1)$.

$$\begin{aligned} GC \rightarrow \{m_1, \dots, m_8\} & : \{k_{1-9}\}_{k_{1-8}} \\ GC \rightarrow m_9 & : \{k_{1-9}\}_{k_9} \\ GC \rightarrow \{m_7, m_8\} & : \{k_{789}\}_{k_{78}} \\ GC \rightarrow m_9 & : \{k_{789}\}_{k_9} \end{aligned}$$

Compared to member-oriented rekeying, the number of rekeying messages nearly doubles (from h to $2(h-1)$). A way of reducing the number of rekeying messages is to combine all the rekeying messages for a particular member into one message and then send it to the member. The following sequence of rekeying messages shows the method. Hence, the number of rekeying messages is reduced to h .

$$\begin{aligned} GC \rightarrow \{m_1, \dots, m_6\} & : \{k_{1-9}\}_{k_{1-8}} \\ GC \rightarrow \{m_7, m_8\} & : \{k_{1-9}\}_{k_{1-8}}, \{k_{789}\}_{k_{78}} \\ GC \rightarrow m_9 & : \{k_{1-9}, k_{789}\}_{k_9} \end{aligned}$$

The rekeying messages for m_9 's leave are shown in what follows. Note that the new key k_{78} is used in the encryption of the new key k_{1-8} . The number of rekeying messages is $(d-1)(h-1)$ and the number of encryptions $d(h-1)$.

$$\begin{array}{ll}
 GC \rightarrow \{m_1, m_2, m_3\} & : \{k_{1-8}\}_{k_{123}} \\
 GC \rightarrow \{m_4, m_5, m_6\} & : \{k_{1-8}\}_{k_{456}} \\
 GC \rightarrow m_7 & : \{k_{1-8}\}_{k_{78}}, \{k_{78}\}_{k_7} \\
 GC \rightarrow m_8 & : \{k_{1-8}\}_{k_{78}}, \{k_{78}\}_{k_8}
 \end{array}$$

3.1.3.3 Group-Oriented Rekeying

In the group-oriented rekeying strategy, a single rekeying message is constructed. The message contains all the new keys and is multicast to the entire group, where (generic) multicast is used instead of subgroup multicast or directed multicast. Since the number of new keys is bounded above by $O(\log_d(n))$ for group size n and key tree degree d , group-oriented rekeying will not cause serious problems for scalability. On the contrary, the GC's overhead per rekeying message can be reduced, and without duplication of information, the total number of bytes sent by the GC for each join/leave operation is much less than those of the previous two rekeying strategies which duplicate information in multiple rekeying messages.

As illustrated below, in group-oriented rekeying, a single multicast message followed by a single unicast message is needed for m_9 's join. Thus, the number of rekeying messages is 2 and the number of encryptions $2(h-1)$.

$$\begin{array}{ll}
 GC \rightarrow \{m_1, \dots, m_8\} & : \{k_{1-9}\}_{k_{1-8}}, \{k_{789}\}_{k_{78}} \\
 GC \rightarrow m_9 & : \{k_{1-9}, k_{789}\}_{k_9}
 \end{array}$$

Another way to perform group-oriented rekeying for m_9 's join operation is to broadcast a single message which includes all the new keys. The new keys are used in the encryption of the parent's new keys, and the number of encryptions is $d(h-1)$. Note that this is the method used in sections 3.1.1 and 3.1.2 for a join operation. The advantage of this method is that a group-oriented join operation is performed in the same way as a group-oriented leave operation.

$$GC \rightarrow \{m_1, \dots, m_9\} : \{k_{1-9}\}_{k_{123}}, \{k_{1-9}\}_{k_{456}}, \{k_{1-9}\}_{k_{789}}, \{k_{789}\}_{k_7}, \{k_{789}\}_{k_8}, \{k_{789}\}_{k_9}$$

Similarly, the group-oriented rekeying for m_9 's leave operation is a single multicast message which includes $d(h-1)$ encryptions. Again, this is the method used in sections 3.1.1 and 3.1.2 for a leave operation.

$$GC \rightarrow \{m_1, \dots, m_8\} : \{k_{1-8}\}_{k_{123}}, \{k_{1-8}\}_{k_{456}}, \{k_{1-8}\}_{k_{78}}, \{k_{78}\}_{k_7}, \{k_{78}\}_{k_8}$$

The numbers of rekeying messages and encryptions for the three different rekeying strategies are summarized in Table 3.1. It is clear that the member-

oriented rekeying strategy is the worst one. With respect to the number of encryptions, key-oriented rekeying is equivalent to group-oriented rekeying. On the other hand, with respect to the number of rekeying messages, group-oriented rekeying is better than key-oriented rekeying. However, if the underlying network supports subgroup multicast or directed multicast, key-oriented rekeying is preferred, since this choice will reduce the total amount of network traffic.

Table 3.1. Comparison of three rekeying strategies (Source: [Wong et al., 2000], © IEEE/ACM).

Strategy	join/leave	#Rekeying messages	#Encryptions
Member-oriented	join	h	$\frac{h(h+1)}{2} - 1$
	leave	$(d-1)(h-1)$	$\frac{(d-1)h(h-1)}{2}$
Key-oriented	join	h	$2(h-1)$
	leave	$(d-1)(h-1)$	$d(h-1)$
Group-oriented	join	2 (or 1)	$2(h-1)$ (or $d(h-1)$)
	leave	1	$d(h-1)$

Assuming that a request is equally likely to be a join or a leave, the average encryption cost to the GC per request is $(d+2)(h-1)/2$. If the group size is large, reducing the average cost for the GC is important. Suppose the d -ary key tree is always adjusted to be as balanced as possible, then the average cost for a group of size n is $(d+2)(\log_d(n))/2$, which is minimized for $d=4$. That is, the optimal degree of key trees is four.

3.1.3.4 Performance of Bursty Operation in d -ary Key Trees

In section 3.1.2.3, we analyzed the performance of bursty operation in binary key trees. The performance for d -ary key trees is studied in [Li et al., 2000]. We present the results as follows.

Best case performance

$$N_{best} = m - \sum_{i=0}^{h-1} m_i + \sum_{i=1}^k indiv(m, d^i) + (h-k)$$

where $m = m_{h-1} \dots m_0$ is the radix- d representation of m and $indiv(m, l) = 1$ if $m \bmod l \neq 0$, otherwise $indiv(m, l) = 0$.

Worst case performance

$$N_{worst} = m \cdot (h-k-1) + (d^{k+1} - 1)$$

Average case performance

$$N_{average} = \sum_{\ell=0}^{h-1} d^{\ell} \left(1 - \frac{\binom{n-n/d^{\ell}}{m}}{\binom{n}{m}}\right) = (n-1)/(d-1) - \sum_{\ell=0}^{h-1} d^{\ell} \cdot \frac{\binom{n-n/d^{\ell}}{m}}{\binom{n}{m}}$$

3.1.4 One-way Function Tree (OFT)

The *One-way Function Tree* (OFT) scheme was proposed by Sherman et. al in 1997 [Sherman and McGrew, 2003]. The tree structure in OFT is the same as for LKH. The scheme assumes that there is a secure channel between the GC and each group member. The main advantage of OFT over LKH is that the number of broadcast messages is halved from $2\log(n)$ for LKH, to $\log(n)$ messages for OFT.

In OFT, the GC maintains a logical key tree and the group members are placed at leaf nodes. For every non-root node v , let \hat{v} denote the sibling of v and v^{\uparrow} the parent of v . Moreover, if v is a non-leaf node, let v_{ℓ} and v_r denote the respective left and right children of v . Let f and g be two one-way functions⁹ publicly known and computable by all members of the group. There are three values associated with each node v : i) the *node secret* s_v , ii) the *blinded node secret* $b_v = f(s_v)$, and iii) the *node key* $k_v = g(s_v)$. The node secret of the root is the group key k_G . LKH may be considered as a special case of OFT where the node secret and node key are identical.

The node secrets of leaf nodes are randomly selected by the GC, and are securely unicast to the corresponding members. The node secrets of each internal node can be computed by the GC, or individual group members, from the blinded node secrets of the two children of the node. In particular, if v_{ℓ} and v_r are as defined above, then the node secret of v is computed by $s_v = b_{v_{\ell}} \oplus b_{v_r}$ where \oplus denotes bitwise exclusive-or (XOR). The node keys are used to encrypt the blinded node secrets.

The GC computes the node secrets, blinded node secrets and node keys of all nodes in a bottom-up manner, beginning with the leaf nodes, level by level, up to the root.

Then, for every node v , with sibling \hat{v} , the GC encrypts the blinded node secret b_v using the node key $k_{\hat{v}}$ of \hat{v} . Finally the GC broadcasts all encrypted blinded node secrets.

Every member v can compute its own node key $k_v = g(s_v)$ and blinded node secret $b_v = f(s_v)$. After receiving GC's broadcast, v can perform decryption using k_v to obtain its sibling's blinded node secret $b_{\hat{v}}$. If $p = v^{\uparrow}$ is v 's parent, then v can compute p 's node secret s_p from b_v and $b_{\hat{v}}$, and then (by applying f and g) v can obtain p 's blinded node secret b_p and node key k_p . Using

⁹One possible choice for f and g is as follows: Let G be a length-doubling pseudo-random function, so that $G(x) = L(x)R(x)$ where $|L(x)| = |R(x)| = |x|$. Define $f(x) = L(x)$ and $g(x) = R(x)$.

k_p , member v can decrypt \hat{p} 's blinded node secret from the GC's broadcast and then compute the node secret of p 's parent, p^\dagger . Continuing in this fashion the member v will obtain the node secret of the root.

Whenever a member v joins or leaves, the GC changes the node secret of \hat{v} , and computes all blinded node secrets and node keys of all the nodes along the path from the parent of v and \hat{v} to the root, including \hat{v} 's blinded node secret in the case of join. The GC then encrypts these blinded node secrets with their corresponding sibling's node keys, and broadcasts them to the group. After receiving the broadcast, every member is able to reconstruct those (and only those) blinded node secrets he/she needs to know, and can recompute the new group key. In case of a join, the set of previous blinded node secrets, which are siblings to the path from the joining member to the root, is also unicast to the joining member by means of a secure channel between the GC and the member.

3.1.5 One-way Function Chain (OFC)

The *One-way Function Chain* (OFC) scheme, proposed by Canetti et al. [Canetti et al., 1999a] and named by Sherman et al. [Sherman and McGrew, 2003], is a variation of OFT. In OFC, the one-way functions f and g are defined as the left and right halves of G , a length-doubling pseudo-random generator. Blinded node secrets do not occur in OFC, thus, in OFC, to each node are attached the i) a *node secret* and ii) a *node key*. However, there is always a functional relationship among the node secrets in the key tree along the path from the root to the parent of the last member to leave the group. This path is called a *chain*. Suppose node $p = v^\dagger$ is the parent of node v , then the node secret of p is computed from the node secret of v as $s_p = f(s_v)$. Similar to OFT, the node key for v is computed by applying the function g to the node secret, $k_v = g(s_v)$.

It is worth pointing out that the OFC proposal [Canetti et al., 1999a, Canetti et al., 1999a] does not discuss operations for the initial formation of the key tree or member joins. The reason for this may be that the difficulty of group key management for secure group communications comes from member leaves. Therefore, the chained relationship among the node secrets appears when a member leaves but might not exist when a member joins. For example, when a member joins, all the keys affected by this join (i.e., from the parent of the joining member to the root) are passed through a one-way function by individual members and the GC and the GC sends these new keys to the joining member.

We give an example to illustrate how LKH, OFT, OFC work when a member leaves. Suppose there are 8 members in the group with the initial setting as in Figure 3.10. Note that for OFT, only the keys are shown. In fact, for each node v , there is a node secret s_v , a blinded node secret $b_v = f(s_v)$, and node key

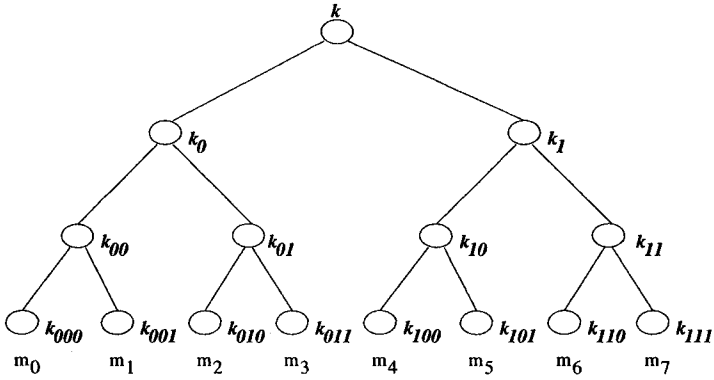


Figure 3.10. Logical Key Hierarchy for 8 members (Source: [Canetti et al., 1999a], © IEEE).

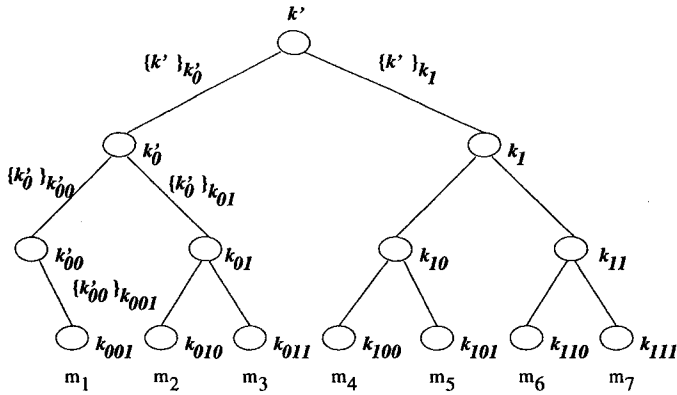


Figure 3.11. LKH operation when m_0 leaves (Source: [Canetti et al., 1999a], © IEEE).

$k_v = g(s_v)$. Moreover, $s_v = b_{v_\ell} \oplus b_{v_r} = f(s_{v_\ell}) \oplus f(s_{v_r})$ where v_ℓ and v_r are the left and right children of v .

Figure 3.11 shows the operation of LKH when m_0 leaves. The GC selects new keys k'_{00} , k'_0 , k' and encrypts each of these with the keys of the corresponding children. Since m_0 has left and $m_1 = \hat{m}_0$, the key k'_{00} is encrypted with the key of m_1 only. On the other hand, for each of k'_0 and k' two encryptions must be performed, one for each (internal node) child.

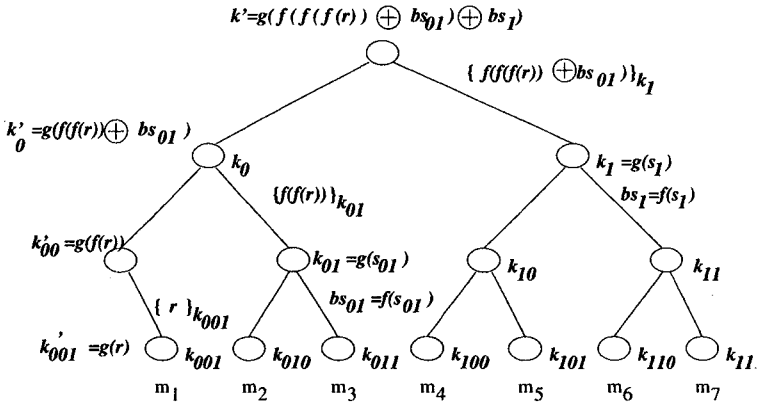


Figure 3.12. OFT operation when m_0 leaves (Source: [Canetti et al., 1999a], © IEEE).

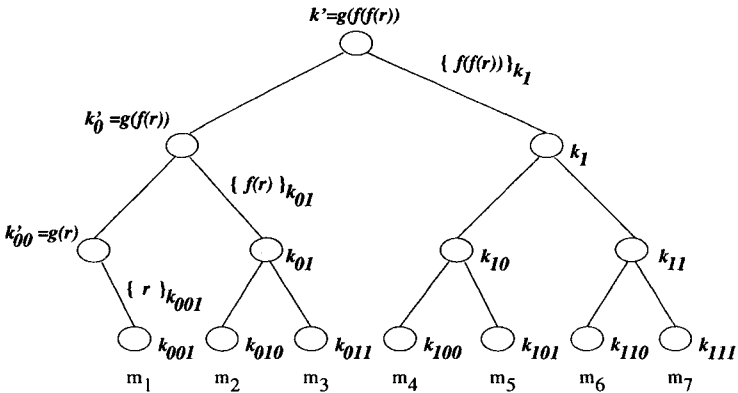


Figure 3.13. OFC operation when m_0 leaves (Source: [Canetti et al., 1999a], © IEEE).

Figure 3.12 shows the operation of OFT when m_0 leaves. The GC selects a random number r as the new node secret for m_1 , and encrypts r with m_1 's node key k_{001} . So m_1 can decrypt his new node secret, computes his new blinded node secret and new node key. Moreover, m_1 has kept the blinded node secrets of all siblings along the path from his leaf node to the root, so m_1 can compute all the new blinded node secrets and new node keys of nodes from his leaf to the root. The GC encrypts all new blinded node secrets of the nodes from m_1 to the root, and broadcasts them. All other members can compute the new node secrets and new node keys. Note that in practice it may be simpler to move m_1

a level up to the node of k_{00} . However, for the purpose of comparison, we have kept the figures for LKH and OFC the same. Figure 3.13 shows the operation of OFC when m_0 leaves. Compared to OFT, OFC is simpler.

3.1.6 Collusion Attacks on OFT and Improvement

OFT was found to be vulnerable to collusion attacks by Horng in 2002, as well as by Ku and Chen in 2003. Suppose a member u leaves the group at time t_1 , and a new member v joins at time t_2 (where $t_2 > t_1$). Then u and v can collude to obtain the group key for the time interval $[t_1, t_2]$, which neither u nor v should know. As a result, the OFT scheme fails to provide forward secrecy against u and backward secrecy against v . We describe several collusion scenarios and an improved OFT scheme proposed in [Ku and Chen, 2003].

Figure 3.14 shows a first possible collusion scenario [Horng, 2002]. Here, the figure exhibits the node secrets. Initially, the root key is $k = g(s) = g(f(s_0) \oplus f(s_1))$ by definition. Suppose Alice, associated with node 010, is evicted at time t_1 . Then, at t_1 the group key is changed to $k_{(t_1)} = g(f(s_{0(t_1)}) \oplus f(s_1))$, where the time subscript (t_1) , is used to indicate the key, node secrets, or node keys, respectively, from that specific time t_1 onward. Note that the blinded node secret $f(s_1)$ is known to Alice, and does not change at time t_1 . Suppose at a later time t_2 , Bob joins the group, and is placed at node 100. Then, at t_2 the group key will be changed to $k_{(t_2)} = g(f(s_{0(t_1)}) \oplus f(s_{1(t_2)}))$. The blinded node secrets $f(s_{0(t_1)})$ and $f(s_{1(t_2)})$ are known to Bob. If Alice and Bob were to collude, they would collectively know the quantities $f(s_{0(t_1)})$ and $f(s_1)$. Thus, they would be able to compute the key $k_{(t_1)}$. If there are no updates to the key $k_{(t_1)}$ during the time interval $[t_1, t_2]$, then Alice and Bob obtain $k_{(t_1)}$, which they should not know.

In the above scenario, the evicted member Alice and joining member Bob are assumed to belong to different branch subtrees under the root. Figure 3.15 shows a second collusion scenario [Ku and Chen, 2003], where the evicted member and the joining member belong to the same branch subtree under the root. In this example, suppose that Alice, associated with node 000, is evicted at time t_1 , and that Bob joins the group, and is placed at node 010 at time $t_2 (> t_1)$. The node secret of node 0 from t_1 onward is $s_{0(t_1)} = f(s_{00(t_1)}) \oplus f(s_{01})$, and the group key from t_1 onward is $k_{(t_1)} = g(f(s_{0(t_1)}) \oplus f(s_1))$. The node secret of node 0 from t_2 onward is $s_{0(t_2)} = f(s_{00(t_1)}) \oplus f(s_{01(t_2)})$, and the group key from t_2 onward is $k_{(t_2)} = g(f(s_{0(t_2)}) \oplus f(s_1))$. Suppose there are no key updates between times t_1 and t_2 . Alice knows $f(s_{01})$ and $f(s_1)$, and Bob knows $f(s_{00(t_1)})$ and $f(s_1)$. From $f(s_{00(t_1)})$ and $f(s_{01})$ Alice and Bob can collude to compute $s_{0(t_1)}$ and then $f(s_{0(t_1)})$. Combined with $f(s_1)$, they can now compute the group key $k_{(t_1)}$, which is used in the time interval $[t_1, t_2]$, and should not be known by either of them.

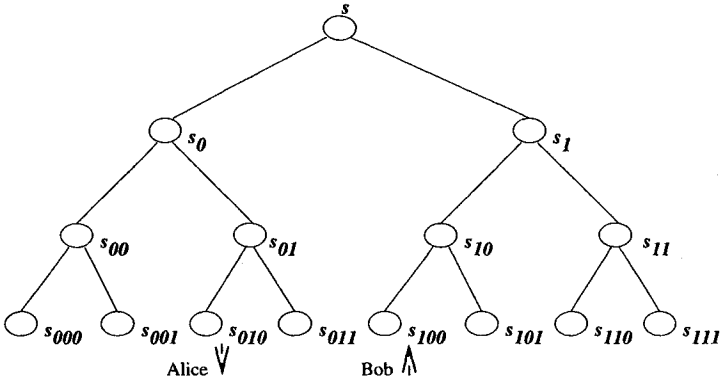


Figure 3.14. OFT collusion scenario 1 (Source: [Ku and Chen, 2003], © IEEE).

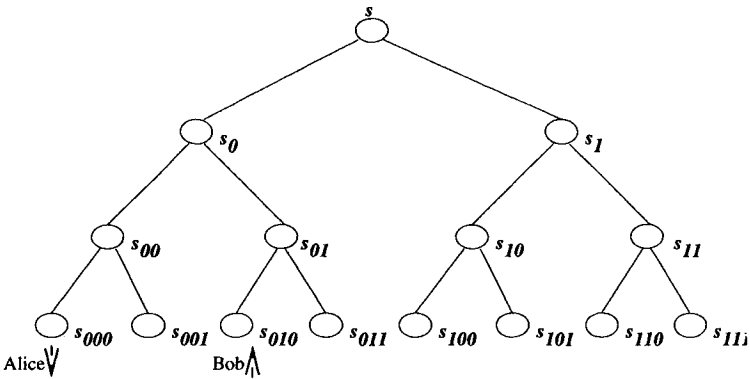


Figure 3.15. OFT collusion scenario 2 (Source: [Ku and Chen, 2003], © IEEE).

In the above two scenarios, we made the assumption that there were no key updates in a time interval $[t_a, t_b]$, from Alice's departure to Bob's arrival. We now see that key information can be computed by colluding participants, even when key updates do occur in some $[t_a, t_b]$. Figure 3.16 shows a third collusion scenario [Ku and Chen, 2003] where there is one key update between t_1 and t_3 , which will be caused by a new, joining member. In this case, suppose that Alice is evicted at time t_1 , Bob is added at time t_2 (where $t_2 > t_1$), and Cindy joins and is placed at node 101 at time $t_3 > t_2$. The group key during $[t_2, t_3]$ is $k_{(t_2)} = g(f(s_{0(t_2)}) \oplus f(s_1))$. Note that $f(s_1)$ does not change before t_3

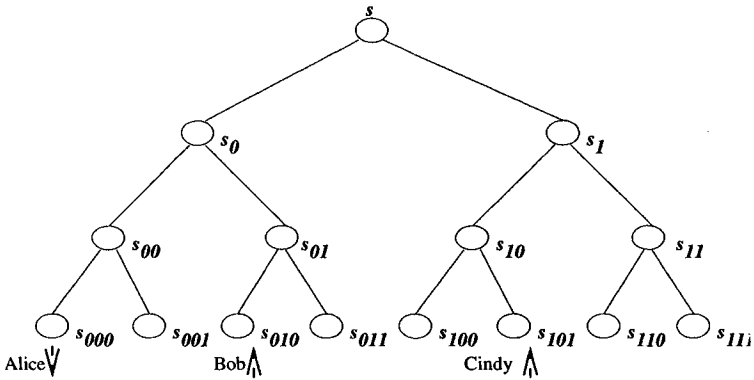


Figure 3.16. OFT collusion scenario 3 (Source: [Ku and Chen, 2003], © IEEE).

and is known to Alice, while $f(s_{0(t_2)})$ is known to Cindy after t_3 . Now, Alice and Cindy can collude to compute the group key $k_{(t_2)}$, which neither of them should know.

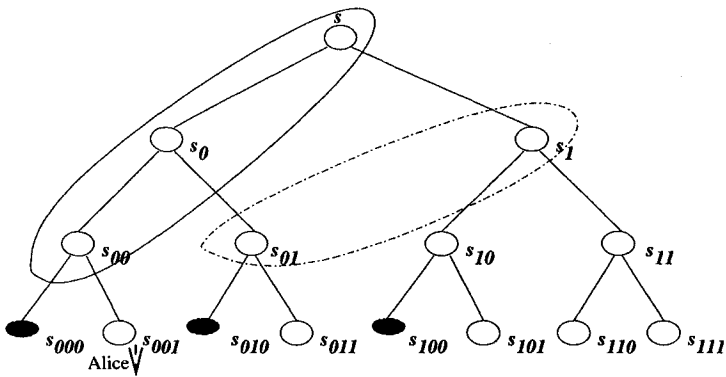


Figure 3.17. OFT improvement (Source: [Ku and Chen, 2003], © IEEE).

From the above discussion, it can be seen that a collusion could succeed because some blinded node secrets known to an evicted member are not changed by the rekeying operation and can be utilized to compute the current group key even after several key updating operations [Ku and Chen, 2003]. If all blinded

node secrets known by an evicted member m are changed when m leaves, a collusion attack such as the ones discussed above becomes impossible. With respect to a leaving or joining member m , there are two classes of such blinded node secrets. The first class consists of all nodes on the path from the parent of m to the root, called the *path nodes* of m . The second class consists of those nodes which are siblings of the nodes in the first class, and are called the *sibling nodes* of m . Figure 3.17 illustrates an example. Suppose Alice leaves from the group. Then, it is not sufficient for the blinded node secrets of the path nodes of Alice (i.e., $f(s_{00})$, $f(s_0)$, $f(s)$) to be changed. It is necessary that the blinded node secrets of the sibling nodes of Alice (i.e., $f(s_{01})$, $f(s_1)$) be changed also. Changing $f(s_{00})$, $f(s_0)$, $f(s)$ can be accomplished by changing node secret s_{000} as previously. Changing $f(s_{01})$ can be accomplished by changing node secret s_{010} , and the change to $f(s_1)$ can be performed by changing node secret s_{100} . Thus the OFT scheme can be improved as follows: *Whenever a member m leaves or is evicted from the group, apart from the original operations which change the path nodes of m , all the blinded node secrets of the sibling nodes of m must also be changed.*

It is worthwhile discussing the *improved OFT* scheme further. The initial purpose of the OFT scheme was to try to reduce the number of encryptions in LKH's rekeying operations from $2\log(n)$ to $\log(n)$. In LKH, when a member m joins or leaves, the keys of the path nodes of m need to be changed. Each changed key needs to be encrypted by means of the keys of its two children. Thus, if the group size is n , then there will be total of $2\log(n)$ encryptions.

In the original OFT scheme, only the blinded node secrets of the path nodes of a leaving member m need to be encrypted, that is a total of $\log(n)$ encryptions. However, for a joining operation, apart from the path nodes, all the blinded node secrets of the sibling nodes of m need to be processed. These blinded node secrets need to be encrypted by the same kind of secure mechanism between the joining member and the GC. This should be counted as another $\log(n)$ encryptions. That is, the joining operation in OFT needs $2\log(n)$ encryptions.

In order to solve the collusion problem in OFT, the *improved OFT* scheme changes all blinded node secrets an evicted member m knows, and this results in an average number of $(\log(n))^2 + \log(n)$ encryptions [Ku and Chen, 2003]. In this sense, the improved OFT scheme loses its intended advantage over LKH.

3.1.7 Group Key Management based on Boolean Function Minimization Technique

In SGC, performing a member-leave operation is much more difficult than performing a join operation. Moreover, the problem of efficiently performing a multiple-member-leave operation is of considerable interest and concern. A

group key management protocol based on a boolean function minimization technique is proposed in [Chang et al., 1999]. The protocol is optimal in terms of the number of rekeying messages when performing bulk leaves in one round (also known as *cumulative member removal*). We discuss this protocol in this section.

Suppose there are n members and let $m = \log(n)$. As in section 3.1.2.2, each member of the group is associated with a unique member ID (MID) which is a binary string of length m . Thus, a MID can be written as $X_{m-1}X_{m-2} \cdots X_0$, where $X_i \in \{0, 1\}$.

There is a common session key, denoted by SK , which is shared by all current members of the group. SK is used to encrypt/decrypt the messages destined to the group. The GC generates randomly $2m$ auxiliary keys, $\mathcal{K} = \{k_0, \bar{k}_0, k_1, \bar{k}_1, \dots, k_{m-1}, \bar{k}_{m-1}\}$ which he holds secret. A member with MID $X_{m-1}X_{m-2} \cdots X_0$ receives from the GC a sequence of m auxiliary keys $K_{m-1}, K_{m-2}, \dots, K_0$, where $K_i = k_i$ if $X_i = 1$ and $K_i = \bar{k}_i$ otherwise. It is important to note here that \bar{k}_i and k_i are not bitwise complements of each other, instead, they are random and totally independent of each other. The assignment of keys follows a key structure which conforms to a key tree, called the *auxiliary key tree* (AKT) (Figure 3.18). Just as in LKH, the members are placed at leaf nodes, and every member is assigned the keys along the path from its leaf to the root. The root key is the group key SK . The difference between AKT and the earlier LKH is that in AKT, exactly two keys are used at each tree level, assigned in an alternating fashion, to the nodes of the level.

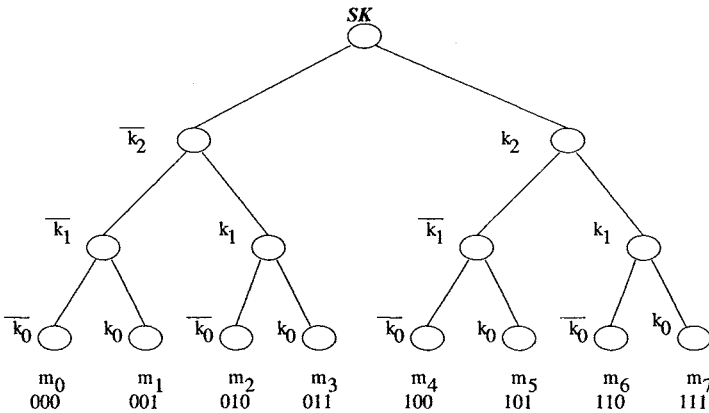


Figure 3.18. Auxiliary Key Tree (Source: [Chang et al., 1999], © IEEE).

When a member with MID $X_{m-1}X_{m-2} \cdots X_0$ leaves the group, the root key SK is changed, encrypted and redistributed to every member except the

one leaving. The new session key SK_{new} is encrypted with the keys which are “complementary” to the ones of the leaving member. For example, if the MID of the leaving member u is 010, u possesses keys $\bar{k}_2, k_1, \bar{k}_0$. The SK_{new} will be encrypted using k_2, \bar{k}_1, k_0 separately. It is clear now that every member except for the leaving one can decrypt the new session keys.

Figure 3.19 gives a visual interpretation for the above example. The solid nodes correspond to the keys possessed by the leaving member.

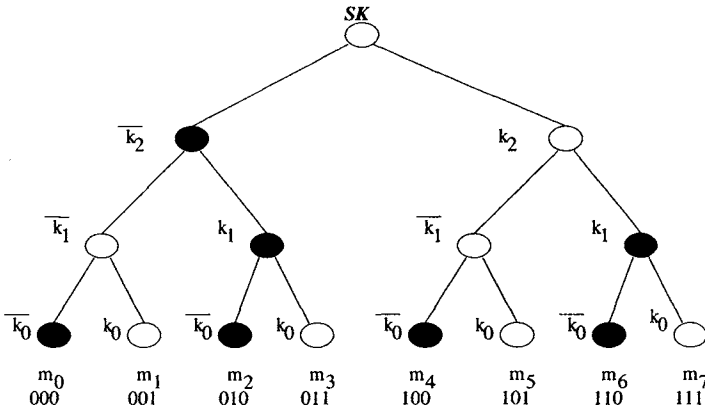


Figure 3.19. m_2 leaves (Source: [Chang et al., 1999], © IEEE).

After the new session key has been obtained by the remaining members, the auxiliary keys also need to be updated. The reason for doing this is to make sure the leaving member cannot use its auxiliary keys to decrypt future session keys. A one-way hash function f is used to update the auxiliary keys as follows $K_i = f(K_i, SK_{new})$.

When multiple members are removed, the new session key needs to be encrypted with keys common to the subsets of the remaining members. Let us see an example first. Suppose two members m_0 and m_4 will be removed from the group. The purpose of rekeying is to provide the new session key to the remaining members $m_1, m_2, m_3, m_5, m_6, m_7$. It is sufficient that the new session key be encrypted with k_0 and k_1 , where k_0 is shared by m_1, m_3, m_5, m_7 and k_1 is shared by m_2, m_3, m_6, m_7 but not possessed by m_0 or m_4 . See Figure 3.20 for a visual interpretation.

From the example, it is easy to see that the general problem of multiple member removal is how to efficiently group the remaining members that share common bits in their MIDs which are different from those of the removed members. This problem is equivalent to one that minimizes the Boolean mem-

bership function $\mu()$, for the subset of members that remain in the group. More precisely, $\mu()$ is determined as follows: (i) Write down the binary labels of the members that remain in the group, (ii) Write down the *disjunction* of boolean monomials corresponding to these binary labels.

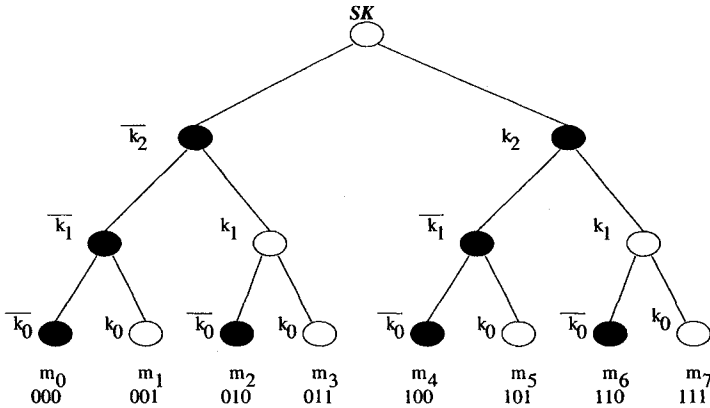


Figure 3.20. m_0 and m_4 leave (Source: [Chang et al., 1999], © IEEE).

For instance, in the above example, the subset of members remaining is $\{m_1, m_2, m_3, m_5, m_6, m_7\}$ and the corresponding set of binary labels is $\{001, 010, 011, 101, 110, 111\}$, so we write $\mu(X_2, X_1, X_0) = \bar{X}_2\bar{X}_1X_0 + \bar{X}_2X_1\bar{X}_0 + \bar{X}_2X_1X_0 + X_2\bar{X}_1X_0 + X_2X_1\bar{X}_0 + X_2X_1X_0$, where + stands for logical OR, and multiplication stands for logical AND.

From the above form, rekeying may be performed in a straightforward way: multicast 6 messages, one for each term of the sum in the form $\mu()$. Each message would be encrypted with a composite key which is derived from the keys corresponding to that term by using some one-way hash function. For example, term $\bar{X}_2X_1\bar{X}_0$ corresponds to a message which is encrypted with a key derived from keys k_2, k_1, \bar{k}_0 . In this way, every encrypted message can be decrypted by exactly one of the remaining members. Obviously, this is inefficient because there is no grouping and no deployment of keys which might be common among different remaining members.

Using some boolean function minimization technique such as the Karnaugh map, the above form can be minimized to $\mu(X_2, X_1, X_0) = x_0 + x_1$, which corresponds to the minimum number of messages: one encrypted with k_0 and the other with k_1 . This result represents the maximum aggregation of keys common among the remaining members.

The problem with this protocol is that when two or more ex-members collude, they will have all the keys, thus they can figure out all the new keys.

3.2 Distributed Key Agreement based on Tree Structures

In this section, we discuss several distributed key agreement protocols.

3.2.1 Tree based Group Diffie-Hellman Key Agreement (TGDH)

Tree based Group Diffie-Hellman key agreement (TGDH) [Kim et al., 2000, Kim et al., 2004] extends the two party Diffie-Hellman key exchange scheme to a tree structure for multi-party communications. Let p be a large prime and g a primitive element in \mathbb{Z}_p . In TGDH, all members maintain an identical virtual binary key tree which may or may not be balanced. The nodes in the key tree are denoted by $\langle l, v \rangle$, where l is the level in the tree and v indicates the sequential index of the node at level l . We will have $0 \leq v \leq 2^l - 1$ since level l holds at most 2^l nodes. The root node is labeled as $\langle 0, 0 \rangle$, and the two children of a parent node $\langle l, v \rangle$ are labeled as $\langle l+1, 2v \rangle$ and $\langle l+1, 2v+1 \rangle$, respectively. Every node $\langle l, v \rangle$ except for the root node is associated with a secret key $k_{\langle l, v \rangle}$ and a blinded key $bk_{\langle l, v \rangle} = f(k_{\langle l, v \rangle})$, where $f(x) = g^x \bmod p$. The key $k_{\langle l, v \rangle}$ of a parent node is the Diffie-Hellman key of its two children, i.e., $k_{\langle l, v \rangle} = g^{k_{\langle l+1, 2v \rangle} k_{\langle l+1, 2v+1 \rangle}} \bmod p = f(k_{\langle l+1, 2v \rangle} k_{\langle l+1, 2v+1 \rangle})$, thus, the secret key $k_{\langle l, v \rangle}$ of a node can be formed from the secret key of one of its two children and the blinded key of the other child by using the Diffie-Hellman key exchange protocol. Every blinded key $bk_{\langle l, v \rangle}$ is publicly broadcast by some member(s) to the group. All group members are hosted on leaf nodes respectively. Every member m_i selects its own Diffie-Hellman private share a_i , and computes the corresponding Diffie-Hellman disguised public share b_i . Suppose a leaf node $\langle l, v \rangle$ hosts member m_i , then the secret key $k_{\langle l, v \rangle}$ for the node is a_i , and the blinded key $bk_{\langle l, v \rangle} = g^{k_{\langle l, v \rangle}}$ is b_i . Member m_i can compute all secret keys from its leaf node $\langle l, v \rangle$ to the root node $\langle 0, 0 \rangle$, and of course, m_i will receive all blinded keys in the key tree. The root node has secret key $k_{\langle 0, 0 \rangle}$, and there is no need for $bk_{\langle 0, 0 \rangle}$. The group data encryption key (DEK) is obtained by passing $k_{\langle 0, 0 \rangle}$ through a one-way function.

Figure 3.21 illustrates a key tree with six members. As shown in the figure, node $\langle 3, 1 \rangle$ hosts member m_2 and is associated with m_2 's private share a_2 and public share $b_2 = g^{a_2}$. Member m_2 can compute $k_{\langle 2, 0 \rangle} = g^{k_{\langle 3, 0 \rangle} k_{\langle 3, 1 \rangle}} = g^{a_1 a_2} \bmod p$ after receiving $b_1 = g^{a_1}$. Next, m_2 computes and broadcasts $bk_{\langle 2, 0 \rangle} = g^{a_1 a_2}$. Next, m_2 proceeds to compute $k_{\langle 1, 0 \rangle} = g^{k_{\langle 2, 0 \rangle} k_{\langle 2, 1 \rangle}} \bmod p = g^{a_3 g^{a_1 a_2}} \bmod p$, after receiving $bk_{\langle 2, 1 \rangle} = g^{a_3}$ from m_3 . Further, m_2 computes and broadcasts $bk_{\langle 1, 0 \rangle} = g^{k_{\langle 1, 0 \rangle}} \bmod p = g^{g^{a_3 g^{a_1 a_2}}} \bmod p$. Finally m_2 computes $k_{\langle 0, 0 \rangle} = g^{g^{a_3 g^{a_1 a_2}} g^{a_6 g^{a_4 a_5}}} \bmod p$ after receiving $bk_{\langle 1, 1 \rangle}$. It is clear now that the root key $k_{\langle 0, 0 \rangle}$ contains the Diffie-Hellman private shares of all mem-

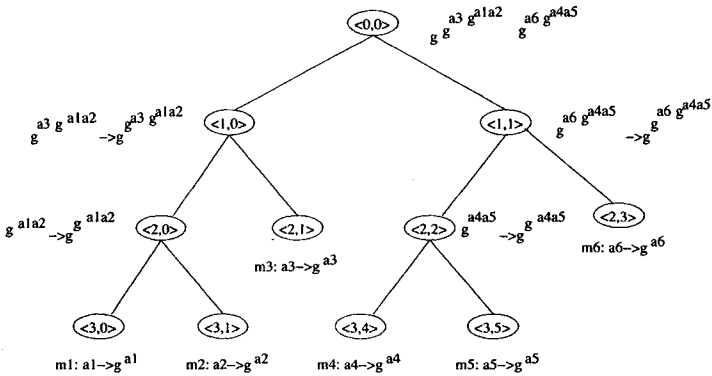


Figure 3.21. Tree based Group Diffie-Hellman key agreement. The notation $k \rightarrow g^k$ means that members compute the secret key k , then compute the blinded key $BK = g^k$ and broadcast BK .

bers. Thus, the group key is obtained by a uniform contribution of all group members.

As indicated in the above example (for m_2), the initial computation of the root key is performed in multiple rounds by all members. In general, each member m hosted on leaf node w , computes recursively the secret key of each node v_i , ascending along the path from w to the root, using the secret key of v_i 's child on the path, and the blinded key of the other child of v_i .

The join and leave operations are summarized below. A *sponsor* is a member defined as follows: (i) The *sponsor* of a *subtree* is the member hosted on the rightmost leaf in the subtree, and (ii) The *sponsor* of a *leaf node* v is the member hosted on the rightmost leaf node (other than itself) of the lowest subtree to which v belongs.

When a new member m requests to join the group, m begins by broadcasting its public share. Next, all pre-existing members determine the insertion location for m on the tree, and determine m 's sponsor. Each member updates the key tree by adding a new (leaf) node for m , a new internal node, and removes all secret keys and blinded keys from the sponsor's leaf node to the root node. Then, the sponsor generates its new private share, computes all secret and blinded keys from its leaf node to the root, and broadcasts all new blinded keys. Every member computes the new root key after receiving the new blinded keys. It can be seen that the root key is computed in one round. Similarly, the root key can be computed in one round for a single leaf. Gen-

erally, for multiple leaves there will be multiple sponsors. The sponsors need to collaborate to compute secret and blinded keys in multiple rounds in a way similar to the initial setting up phase, limited by $O(\log_2 n)$. After all sponsors compute and broadcast all blinded keys, every member can compute the root key. Similarly, multiple joins (and merging) can be performed in multiple rounds. A network partition can be treated as “multiple leaves” from the point of view of any subgroup resulting from the partition.

3.2.2 Block-Free Tree based Group Diffie-Hellman Key Agreement (BF-TGDH)

3.2.2.1 BF-TGDH Principle

A common problem with most group key management schemes (especially distributed ones) is that during the process of rekeying, all members stop data communication until the new group (root) key is distributed (formed) (this is called a *block*). In this section we present a new protocol called *Block-Free Tree based Group Diffie-Hellman* key agreement (BF-TGDH) [Zou and Ramamurthy, 2004], extended from TGDH [Kim et al., 2000]. The BF-TGDH protocol allows group members to continue their communication seamlessly and without interruption during the rekeying process regardless of join, leave, multiple joins (merging) or multiple leaves (partition).

The basic idea behind BF-TGDH is as follows: (1) there are two kinds of keys: *front-end* key and *back-end* keys. The front-end key can be computed by all group members whereas for back-end keys, each member will have one key he/she can not compute. (2) whenever a member leaves¹⁰, the remaining members *switch* to the *back-end* key the leaving member does not have *immediately*. (3) re-computation of all keys is performed in the *background*. There are two meanings of *background* here. One is that computation of keys is performed during the intervals between sending out packets and waiting to receive packets, thus utilizing idle computer time to compute new keys. The other is that the rekeying materials are appended to out-going data packets, thus, reducing communication cost. The idea appears simple but it is indeed a difficult problem to design and implement back-end keys so that i) they are capable of excluding ex-members, and ii) they can be computed efficiently.

Based on TGDH, we propose the following mechanism: (1) The front-end key is the root key in TDGH. (2) The back-end keys are computed in an identical way as the root key, extended as follows: Suppose the root key is denoted as $RK(a_1 \cdots a_n)$ where a_1, \dots, a_n are the private shares and $b_1 = g^{a_1}, \dots, b_n = g^{a_n}$, are the respective public shares of m_1, \dots, m_n . Imagine there are n *dummy members* D_1, \dots, D_n and corresponding n *dummy pri-*

¹⁰The join operation is generally much easier and more efficient to perform than the leave operation.

vate shares d_1, \dots, d_n . The members can compute a back-end key, called a *dummy root key* and denoted by $DRK_i(a_1 \cdots d_i \cdots a_n)$ ($i = 1, \dots, n$), in parallel with the computation of the root key $RK(a_1 \cdots a_i \cdots a_n)$. In DRK_i , the private share a_i is replaced by the dummy private share d_i . Therefore, DRK_i can be computed by all members except for m_i .

As indicated in Section 1.3.3 of Chapter 1, a typical problem with the Diffie-Hellman key exchange protocol is the *Man-in-the-Middle* attack. There are two typical ways to defend against the attack [Kaufman et al., 2002]: authenticated Diffie-Hellman and published Diffie-Hellman shares, i.e., the quantities $(a_1, b_1 = g^{a_1}), (a_2, b_2 = g^{a_2})$ in the Diffie-Hellman key exchange are made public (or to say, permanent). In TGDH, the protocol assumes that each group member has a pair of RSA public and private keys and the members utilize the RSA signature to authenticate each other when performing the Diffie-Hellman key exchange. This seems inefficient, and unnecessary as well. Instead, in BF-TGDH, we assume that every member possesses a permanent pair (private share, disguised public share). As long as the Diffie-Hellman public shares of members become publicly known in a permanent manner, the Man-in-the-Middle attack will be unsuccessful. We believe that this idea is reasonable and useful. Just like in the RSA system, where every individual has a private key and public key (in a permanent manner) which will allow any individual to communicate with any other individual securely whenever both of them want to, the permanent Diffie-Hellman private share and disguised public share will allow any number of individuals to communicate securely whenever all of them want to. Moreover the authentication of message senders (called *message/source authentication (MSA)*) is more crucial in group communications than in two-party communications. The BF-TGDH protocol utilizes the inherent ElGamal signature for MSA, rather than a separate RSA signature, which is used in TGDH. In the following paragraphs, we describe BF-TGDH in greater detail.

Suppose that every member m_i has a permanent Diffie-Hellman pair of shares (a_i, b_i) . Suppose, moreover, that there is an off-line Diffie-Hellman *shares generator*. If the largest possible number of members in the group is n , the *shares generator* computes n Diffie-Hellman pairs $(d_1, e_1), \dots, (d_n, e_n)$ where the e_i are the corresponding public shares of the d_i , i.e. $e_i = g^{d_i} \bmod p$. The e_1, \dots, e_n are made public, and are called the *dummy public shares*. The d_1, \dots, d_n are called the *dummy private shares* and are kept secret from all members, known only by the off-line shares generator. We also suppose that there are n dummy members D_1, \dots, D_n who possess these dummy components $(d_1, e_1), \dots, (d_n, e_n)$, respectively. We further assume there is a public one-way function *POF*.

As in TGDH, each leaf node $\langle l, v \rangle$ is associated with a member m_i . However now, the leaf node $\langle l, v \rangle$ for m_i will also be associated with the corre-

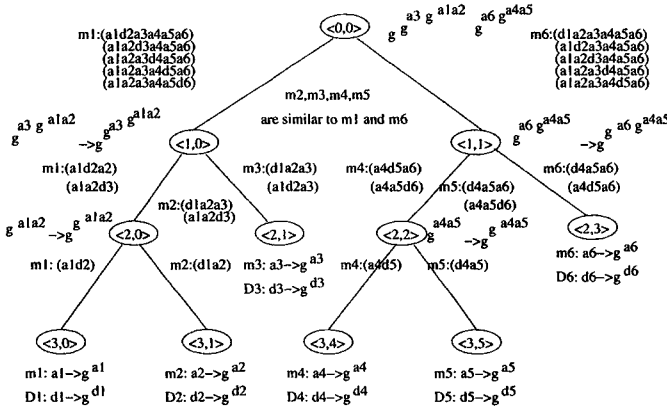


Figure 3.22. Block-Free Tree based Group Diffie-Hellman key agreement for seamless SGC.

sponding dummy member D_i (See Figure 3.22). Like in TGDH, the group members compute the root key $RK(a_1 \cdots a_n)$, where a_1, \dots, a_n are permanent private shares of group members m_1, \dots, m_n . Moreover, every member m_i can compute *dummy root keys* $DRK_j(a_1 \cdots d_j \cdots a_n)$, $j = 1, \dots, i - 1, i + 1, \dots, n$, in parallel with the computation of the root key RK , at some additional computational cost but no extra communication cost. For secure group communications, all group members pass the root key RK through the public one-way function POF to get the Data Encryption Key (DEK), i.e., $DEK = POF(RK(a_1 \cdots a_n))$, and then encrypt messages using DEK . Further, when a member m broadcasts a message, m signs the message using the ElGamal signature scheme. All other members can authenticate the sender.

Figure 3.22 illustrates the operation of BF-TGDH. To simplify notation, let $\gamma(x) = g^x \text{ mod } p$, and $\gamma^k(x) = \gamma(\gamma^{k-1}(x))$, for any positive integer k . The symbol $(a_1 \cdots d_i \cdots a_m)$ represents the computation $k(a_1 \cdots d_i \cdots a_m) \rightarrow \gamma(k(a_1 \cdots d_i \cdots a_m))$ where $k(a_1 \cdots d_i \cdots a_m)$ is a dummy secret key, and $BK(a_1 \cdots d_i \cdots a_m) = \gamma(k(a_1 \cdots d_i \cdots a_m)) = g^{k(a_1 \cdots d_i \cdots a_m)}$ is a dummy blinded key. For example, let us consider m_1 . Member m_1 executes $(a_1 d_2)$, i.e., computes $\gamma(a_1 d_2) \rightarrow \gamma^2(a_1 d_2)$, and broadcasts $BK(a_1 d_2) = \gamma^2(a_1 d_2)$. Then m_1 executes $(a_1 d_2 a_3)$, i.e., $\gamma(a_3 \gamma(a_1 d_2)) \rightarrow \gamma^2(a_3 \gamma(a_1 d_2))$, after receiving $\gamma(a_3)$ from m_3 . Next, m_1 computes $(a_1 a_2 d_3)$ from $\gamma(d_3)$ which is generated by the off-line shares generator. Then, m_1 computes $(a_1 d_2 a_3 a_4 a_5 a_6)$, after receiving $\gamma^2(a_4 \gamma(a_5 a_6))$, $(a_1 a_2 d_3 a_4 a_5 a_6)$ after receiving $\gamma^2(a_4 \gamma(a_5 a_6))$, $(a_1 a_2 a_3 d_4 a_5 a_6)$ after receiving $\gamma^2(d_4 \gamma(a_5 a_6))$, $(a_1 a_2 a_3 a_4 d_5 a_6)$ after receiving $\gamma^2(a_4 \gamma(d_5 a_6))$, and finally $(a_1 a_2 a_3 a_4 a_5 d_6)$ after receiving $\gamma^2(a_4 \gamma(a_5 d_6))$.

3.2.2.2 BF-TGDH Rekeying Operations

When a member joins the group, it broadcasts a joining request. After receiving the request, each group member passes the current DEK through the one-way function to obtain the new DEK (i.e. $POF(DEK)$), and uses the new DEK to encrypt and decrypt messages without any interruption. The sponsor of the joining member is responsible for sending the new DEK to the joining member. Before sending the new DEK, the sponsor encrypts it with the Diffie-Hellman key between him/her and the joining member. As a result, the joining member can participate in group communications immediately. During data communication, the sponsor S of the joining member will compute *in the background*, the secret keys, dummy secret keys, blinded keys, and dummy blinded keys of all nodes on the path from S 's leaf node to the root node, and broadcast the blinded keys and dummy blinded keys to the group by appending them to outgoing messages or by means of separate packets if there are no outgoing messages. All group members will compute the new root key and new dummy root keys after they receive the blinded keys and dummy blinded keys from S . The sponsor also sends blinded keys and dummy blinded keys to the joining member, for all nodes along the path from the joining member to the root, so that the joining member can compute the new root key and new dummy root keys. Once a group member computes the new root key and new dummy root keys, it computes the new $DEK = POF(RK(a_1 \cdots a_n))$, and uses the new DEK for communication. For a while, it is still possible for a member to use the old DEK to decrypt some messages encrypted with the old DEK¹¹).

When a member m_i leaves, a departure request for m_i is broadcast. After receiving the request, all remaining members use the dummy root key DRK_i to compute the new DEK as $DEK = POF(DRK_i)$, and use the new DEK to continue communications without interruption. However, the leaving member m_i cannot decrypt these messages because its share is not in DRK_i . During communication, the sponsor S of the leaving member recomputes the secret keys/dummy secret keys and blinded keys/dummy blinded keys along the path from S to the root, and broadcasts all blinded keys/dummy blinded keys by appending them to outgoing messages.

When multiple members join at the same time, all members pass the current DEK through the one-way function to obtain the new DEK, and use the new DEK to communicate. In general, there are multiple sponsors, and a member may be the sponsor for several joining members. Each sponsor S will encrypt the new DEK using the Diffie-Hellman key between S and one of S 's joining members, and send the DEK to the joining member. The joining members can

¹¹The version number of the DEK used in the encryption may be included in the messages.

participate in the communication immediately thereafter. During communication, the new root key and dummy root keys will be reestablished.

We discuss how things proceed when multiple members leave simultaneously. Suppose that the leaving members are $m_{i_1}, m_{i_2}, \dots, m_{i_k}$. All remaining members will compute the new DEK as follows:

$$DEK = POF\left(\bigoplus_{j=1}^k DRK_{i_j}\right)$$

i.e., they form the XOR (denoted as XDRK) of the corresponding k dummy root keys, and then pass XDRK through the one-way function to obtain the new DEK. Since any leaving member will have one dummy root key missing, he/she cannot compute the new DEK. After the new DEK is computed, group communications continue and the members that have left are excluded. During the process of communication, the sponsors will reestablish the new root key and dummy root keys. We note that if two leaving members collude, they have among them all dummy root keys for computing the new DEK, so BF-TGDH is not resistant to the collusion of simultaneously leaving members. However, this collusion vulnerability persists for a short time. Once the new root key and dummy root keys have been reestablished, members that have left are firmly excluded.

When multiple members join and leave at the same time, the new DEK is computed as in the case of multiple leaves. This DEK is sent to joining members immediately. Therefore, group communication can continue without interruption.

It is worth pointing out that when member(s) join or leave, the (virtual) key tree may need to be adjusted (expanded or compressed), and this will be done simultaneously by all members resulting in an identical key tree for all members.

3.2.2.3 Performance and Security of BF-TGDH

In this section we discuss performance and security issues regarding the BF-TGDH scheme. The number of rounds for rekeying in BF-TGDH is exactly the same as in TGDH. However all the rounds in BF-TGDH are hidden behind data communication. Therefore there is no interruption, no latency, and more robustness against network congestion and link failure. When computational efficiency is considered, every member will compute n root keys (one root key and $n-1$ dummy root keys) instead of one root key. It seems that the computational efficiency reduces a lot. However for network based group communication, the communication efficiency through a network is the main concern rather than computational efficiency within a computer, especially since modern computers have huge storage and very high CPU speeds. As for dummy

blinded keys, since there are a total of n dummy root keys, every member needs to be responsible for computing and broadcasting only one dummy blinded key at each level. This is a very small amount of extra cost compared to the computation of just one blinded key. As for communication cost, since the keys are broadcast by appending them to the out-going messages, there is very little extra cost for communication. In summary, the high cost of communications and computations for n keys can be greatly mitigated by the background property of BF-TGDH.

Regarding the security issues, we have pointed out that when two leaving members collude, they can compute XDRK of dummy root keys, thus getting the new DEK. Fortunately, the success for their collusion lasts just for a short time. Once the new root key is formed the leaving member(s) are excluded completely. Another problem with BF-TGDH is its loss of the *perfect forward security* (PFS) because of the permanent property of Diffie-Hellman shares. One possible solution is that the members generate temporary Diffie-Hellman shares and utilize them for the group key. Moreover the permanent shares are used for mutual authentication in Diffie-Hellman key exchange and for MSA.

3.2.3 DIstributed Scalable sECure Communication (DISEC)

A centralized scheme “OFT” was discussed in section 3.1.4. If the GC is removed from OFT, it becomes a *distributed scheme*. The new scheme, proposed in [Dondeti, 1999, Dondeti et al., 2000], is called a *DIstributed Scalable sECure Communication* (DISEC) scheme. We discuss DISEC below¹².

There are some differences in notation and terminology between OFT and DISEC as presented. The one-way function f in OFT is denoted by E in DISEC. A *node secret* in OFT is called an *unblinded key* in DISEC and a *blinded node secret* in OFT is called *blinded key* in DISEC. Furthermore, DISEC no longer uses a counterpart for the one-way function g and the OFT *node keys* coincide with *node secrets* in DISEC. Let E be a one-way function, and define function *mix* simply by $mix(x, y) = x \oplus y$. A (virtual) key distribution binary tree is formed (see Figure 3.23) so that all group members are located at leaf nodes of the tree, while the internal nodes are called key nodes. Each node will be assigned two keys: an *unblinded key* and a *blinded key*. The unblinded key of a leaf node is randomly selected by the member assigned to the node, and is kept secret. The unblinded key of an internal node is computed by *mixing* the blinded keys of its two children. The blinded key

¹²Portions reprinted, with permission of the authors and the publisher, from L. R. Dondeti et al., *DISEC: a distributed framework for scalable secure many-to-many communication* in Proceedings of the 5th IEEE Symposium on Computers and Communications, July, 2000, pages: 693–698. © 2002 IEEE.

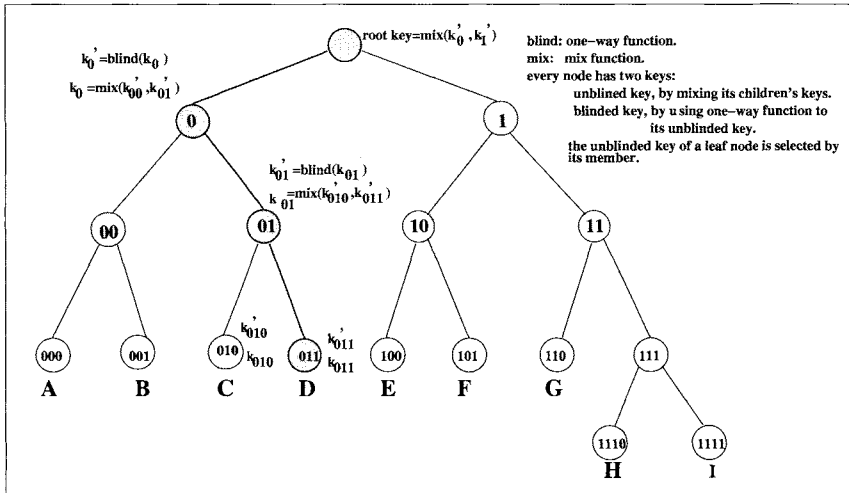


Figure 3.23. One-way function tree in DISEC (Source: [Dondeti et al., 2000], © IEEE).

of each node is computed from the unblinded key of that node by applying the one-way function E .

For a more precise exposition we adopt the following notations. If x is a node in the tree, let \hat{x} denote the binary neighbor of x , and denote by $x \vee \hat{x}$ the parent of x and \hat{x} . Moreover, let k_x denote the unblinded key of node x .

Every member computes the keys along the path from its leaf node to the root as follows:

1. Let x be the current leaf node, and select a random secret number as the unblinded key k_x .
2. Compute the blinded key $E(k_x)$ from k_x . Repeat the following steps until x becomes the root node.
3. Exchange the knowledge of blinded keys between x and \hat{x}
4. Compute the unblinded key of the parent node $x \vee \hat{x}$, i.e.,

$$k_{x \vee \hat{x}} = \text{mix}(E(k_x), E(k_{\hat{x}})) = E(k_x) \oplus E(k_{\hat{x}}).$$
5. Compute the blinded key of the parent node, i.e., $E(k_{x \vee \hat{x}})$.
6. Replace x by $x \vee \hat{x}$.

Suppose the group size is n , then the distributed binary key tree has height $\log(n)$, therefore, every member will take $\log(n)$ steps to compute the root key.

When a member joins or leaves the group, the keys (blinded and unblinded) along the path from the leaf node of the joining or leaving member to the root need to be changed. The rekeying process is activated by the neighbor of the joining or leaving member. It will take $\log(n)$ steps for the new group key to be computed. Next we discuss these operations in detail.

3.2.3.1 Discovery of the Neighbor

During the rekeying task, a procedure is needed for finding the neighbor of a given node. Suppose each member has an ID represented as a binary string. The following algorithm (**Find_Neighbor()**) (see Figure 3.24) takes a binary ID $x = b_h b_{h-1} \cdots b_1$ as its input and returns the binary ID x' of x 's neighbor. Here, $h = \lceil \log(n) \rceil$ is the height of the binary tree and the b_i , $1 \leq i \leq h$ are binary bits.

For example, in Figure 3.23 H(1110)'s neighbor is I(1111), while G(110)'s neighbor is H(1110). If the IDs of two neighbors have the same length, they are called *immediate neighbors* (e.g., H and I), otherwise, they are just called *neighbors* (e.g., G and H). Immediate neighbors mutually exchange blinded keys. Otherwise, the member x with the shorter ID sends its blinded key to x' , and receives the blinded key of x 's sibling (say \hat{x}), from the neighbor x' with the longer ID. Thus, in the example G receives k'_{111} from H. After receiving the new blinded keys, members compute their parent's node secrets.

```

Find_Neighbor ( $X = b_h b_{h-1} \cdots b_1$ )
begin
   $X' = b_h b_{h-1} \cdots \bar{b}_1$ 
  if (leaf_node( $X'$ ) == "true")
    return  $X'$ 
  else if (internal_node( $X'$ ) == "true")
    do
       $X' = X'0$ 
    while (leaf_node( $X'$ ) == "false")
  return  $X'$ 
end

```

Figure 3.24. Find_Neighbor Algorithm.

3.2.3.2 Key Association Groups

The *Key Association Group* (KAG) of a member x is defined as the set of members that supply blinded keys to x . The purpose of the sets $KAG(x)$ is to help delegate the key distribution task evenly among all the members. In

order to compute the root key, a member x with an ID of length h needs h blinded keys. For each bit b_i of x 's ID, there is one member $\kappa_{x,i}$ that supplies the corresponding blinded key. All these members comprise $\text{KAG}(x)$. Thus, $\text{KAG}(x) = \{\kappa_{x,i} : 1 \leq i \leq h\}$. The following algorithm (**Find_Key_Association()**) (see Figure 3.25) takes a binary ID $X = b_h b_{h-1} \cdots b_1$ and a position i as its inputs and returns its KAG member corresponding to bit b_i , i.e. $\kappa_{x,i}$.

```

Find_Key_Association ( $X = b_h b_{h-1} \cdots b_1, i$ )
begin
   $X_i = b_h b_{h-1} \cdots b_{i+1} \bar{b}_i b_{i-1} \cdots b_1$ 
   $k'_i = k_{b_h b_{h-1} \cdots b_{i+1} \bar{b}_i}$ 
  if (leaf_node( $X_i$ ) == "true")
    return ( $X_i, k'_i$ )
  else if (internal_node( $X_i$ ) == "true")
    do
       $X_i = X_i 0$ 
    while (leaf_node( $X_i$ ) == "false")
    return ( $X_i, k'_i$ )
  else
    do
       $X_i = \text{right\_shift}(X_i, 1)$ 
    while (leaf_node( $X_i$ ) == "false")
    return ( $X_i, k'_i$ )
end

```

Figure 3.25. Find_Key_Association Algorithm.

Let us compute the KAG for member H(1110) in Figure 3.23 by using the above algorithm. For position $i = 1$, we get $\kappa_{H,1} = \text{I}(1111)$ after complementing b_1 . So I(1111) is in H's KAG. Next for position $i = 2$, we get the binary string 1100 after complementing b_2 . Since the node with ID 1100 is not in the tree, we right-shift it by one position and get $\kappa_{H,2} = 110$, that is G, so G is in H's KAG. Similarly, for position $i = 3$, and $i = 4$, we get F(101) and D(011). Thus $\text{KAG}(H) = \{ \text{I}, \text{G}, \text{F}, \text{D} \}$, and these elements which supply the blinded keys k'_{1111} , k'_{110} , k'_{10} , and k'_0 , respectively.

3.2.3.3 Join Operation

When a member joins, there should be some way to determine the position in the distribution key tree for the member. In order to increase efficiency, it is desirable to keep the key tree as balanced as possible. However, keeping the

global balance of the key tree is not cost-effective, thus DISEC attempts to locally balance the key tree by choosing local members in the tree that are within an administratively or Time-to-Live (TTL) scoped area and having the new member join at a local member with the smallest ID size. Once the placement position for the joining member x is determined, an existing member (i.e., the neighbor of x) and x collaborate to perform rekeying tasks.

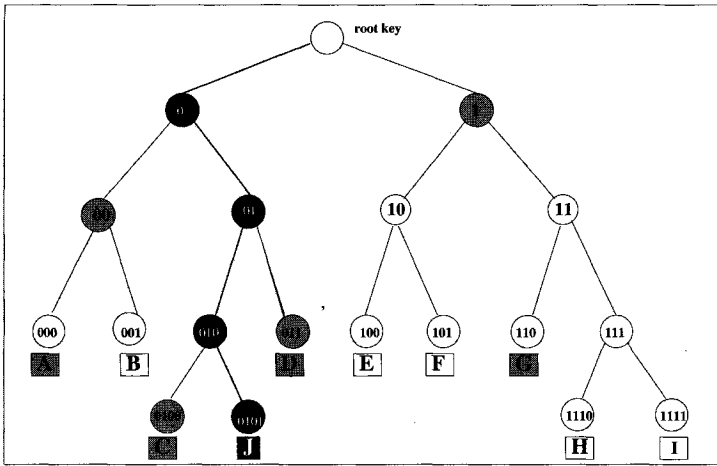


Figure 3.26. Join operation in DISEC (Source: [Dondeti et al., 2000], © IEEE).

Figure 3.26 shows an example where a new member J joins at C, with C initiating the rekeying process. First C splits its ID 010 to 0100 and 0101, keeps 0100 for itself and assigns 0101 to J. C changes its unblinded key and J selects its unblinded key. Note that the keys from J to the root (i.e., the black nodes plus the root node) need to be changed. To finish the task, J needs all the blinded keys of the siblings of the nodes from J to the root (i.e., the gray nodes) and other members need the corresponding blinded keys from J. Next J finds its KAG (i.e., $KAG(J) = \{ C(0100), D(011), A(000), G(110) \}$) by running **Find_Key_Association()**. Finally J and each member in $KAG(J)$ exchange blinded keys and compute the new keys as follows: (1) J and C (i.e., nodes 0101 and 0100) exchange their blinded keys by unicast. C can compute all the keys of nodes 010, 01, 0, and the root key since C keeps all needed blinded keys and J can compute node 010's unblinded key and blinded key. (2) J and D (i.e., nodes 010 and 011) exchange the blinded keys by unicast. D can compute the keys of nodes 01, 0, and the root key and J can compute node 01's unblinded key and blinded key. (3) J and A (i.e., nodes 01 and 00) exchange the blinded keys by unicast. A can compute the keys of node 0 and the root key and J can compute the unblinded key and blinded key of node 0. Moreover,

A locally multicasts the blinded key of node 01, encrypted by the unblinded key of node 00 such that (only) B can decrypt the blinded key and compute the keys of node 0 and the root key. (4) J and G exchange the blinded keys (i.e., nodes 0 and 1) by unicast. G and J can compute the root key. Moreover, G multicasts the blinded key of node 0, encrypted with the unblinded key of node 1, such that E, F, H, I can get the blinded key of node 0 and compute the root key. In all, after $O(\log(n))$ unicast messages and $O(\log(n))$ multicast messages, all the authorized members receive the blinded keys they need and compute the new root key.

3.2.3.4 Leave Operation

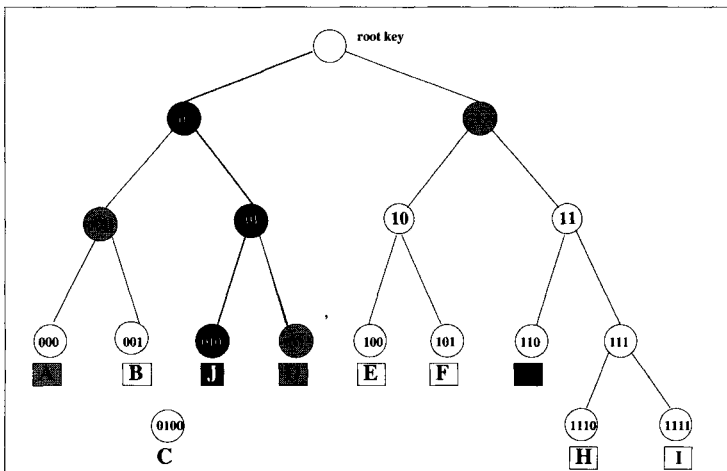


Figure 3.27. Leave operation in DISEC (Source: [Dondeti et al., 2000], © IEEE).

When a member leaves, the rekeying process is initiated by its neighbor, determined by the **Find_Neighbor()** algorithm. The process includes changing member IDs, selecting new unblinded keys, exchanging the blinded keys and computing blinded keys and the root key. If the neighbor is an immediate one (i.e., the sibling of the leaving member), then it assumes its parent's position and changes its ID to its parent's ID. For example, if E (100) leaves, F changes its ID to 10. Otherwise, the descendants of the leaving member's sibling are notified by the neighbor to update their IDs. For example, if G leaves, then H notifies the descendants of the sibling node (111) of G (i.e., I and H itself) to change their IDs. Thus, H changes to 110 and I changes to 111. After the IDs have been changed, the neighbor generates a new unblinded key for itself and is able to recompute all the keys (blinded and unblinded) from its node to

the root since the neighbor has all needed (previous) blinded keys. Then the neighbor finds its KAG and unicasts the corresponding blinded key to each of its KAG members. Its KAG members will be able to compute the keys. Similar to the join operation, the KAG members will be responsible for sharing with their local members the blinded keys received from the neighbor.

Figure 3.27 shows an example where member C leaves. Its immediate neighbor J takes its parent's position and J's ID changes to 010. J changes its unblinded key and computes all keys from node 010 to the root. Then, J determines that $KAG(J) = \{ A, D, G \}$. Next, J sends the blinded keys of nodes 010, 01, and 0 to D, A, and G respectively. A, D, and G then are able to compute the root key. Moreover, A and G multicast the blinded keys of nodes 01 and 0, encrypted with the unblinded keys of nodes 00 and 1 to their local members, i.e., $\{B\}$ and $\{E,F,H,I\}$, respectively, so that these local members can also compute the root key.

Chapter 4

DYNAMIC CONFERENCING SCHEMES

Dynamic conferencing refers to a scenario where there is a group of members and an arbitrary subset (of size at least 2) of members are able to form a privileged subgroup. We call such a subgroup a *conference*. The members in a conference are able to communicate in such a way that a member not in the conference or any outsider is unable to glean any messages addressed to the conference. Moreover, a conferencing scheme must be designed so that communications among different conferences in the group will not interfere with each other.

In previous chapters, we discussed key management schemes for SGC. SGC can be considered as a specific case of dynamic conferencing, i.e., one in which there is a single conference, namely the entire group. Even through considerable research achievements for SGC have been made, research on dynamic conferencing is still quite limited. This is because dynamic conferencing appears to be a very hard problem, much more difficult than SGC. At first, it appears possible and intuitive that every conference be treated independently by using any of the SGC schemes such as the key tree scheme [Caronni et al., 1998, Noubir, 1998, Wong et al., 1998]. However, in a group G of size n , there are too many possible conferences within G , their number bounded above by $2^n - n - 1$. Moreover, members may need to join and/or leave the group dynamically. Every join or leave will cause many conferences to change, possibly an exponential number of them.

There have been several dynamic conferencing schemes proposed in the literature [Chiou and W.T.Chen, 1989, Desmedt and Viswanathan, 1998, Zou et al., 2002a]. Each of these has different properties and performance. In this chapter, we first discuss them one by one, and then, we propose certain criteria for dynamic conferencing and present a comprehensive comparison for the discussed schemes based on these criteria.

4.1 Dynamic Conferencing and a Naive Solution

In [Desmedt and Viswanathan, 1998] the authors propose an unconditionally secure dynamic conference key distribution scheme. It is unconditionally secure because keys for different conferences are completely independent, so knowing a key cannot provide any information about other keys. The scheme is in fact a “naive” solution. We introduce the idea as follows. Suppose there are n members in a group G , then there are $2^n - n - 1$ possible conferences¹. Each member m can join $2^{n-1} - 1$ possible conferences, i.e. any conference consisting of the union of $\{m\}$ with any non-empty subset of $G \setminus \{m\}$. Suppose there is an off-line central trusted server (*group controller*) and a secure channel between the server and every member. The server generates one key for each possible conference, so a total of $2^n - n - 1$ independent keys. Then the server sends each member the keys for the conferences which contain the member, via the secure channel between the server and the member. Hence, each member receives $2^{n-1} - 1$ keys.

For any conference, each member in the conference only needs to pick up the key for the conference from his/her $2^{n-1} - 1$ keys.

The advantage of this scheme is that it is straightforward and easy to implement. Moreover, the length of the initial secrets is minimum [Desmedt and Viswanathan, 1998] among all of the dynamic conferencing schemes. However there are some problems with the scheme: (i) There are too many keys for the server to generate and for the members to store; (ii) The system is not scalable, (iii) The scheme is not dynamic, i.e, the group G is fixed and no join and/or leave operations can be performed.

4.2 Public-Key based Dynamic Conferencing Scheme (PKDC)

In public key based schemes, we assume that every member m_i has his public-private key pair (P_i, S_i) . Whenever a member m_i wants to send a message M to a conference $C = \{m_{i_1}, \dots, m_{i_\ell}\} \cup \{m_i\}$, m_i selects a random session key k , encrypts the message with k , and encrypts k independently with public keys $P_{i_1}, \dots, P_{i_\ell}$ of the respective members $m_{i_1}, \dots, m_{i_\ell}$. Then, m_i broadcasts or multicasts the encrypted message along with the encrypted k , that is $(\{E_{P_{i_1}}(k), \dots, E_{P_{i_\ell}}(k)\}, \{M\}_k)^2$ to the group.

The members in conference C can recover k , using their private keys, and then decrypt the message.

¹By definition a conference has at least two members.

² $E_K(x)$ (or $D_K(x)$) stands for encrypting (or decrypting/signing) x under key K using a public key encryption (or decryption) algorithm E (or D) and $\{x\}_k$ for encrypting x under key k using a secret key encryption algorithm.

The advantage of this scheme is that there is no need to determine any conference in advance. Whenever a member wants to send a message to several members, the member just does it. Another advantage is that the group scope does not exist. The group is determined by the multicast channel. As long as a member joins the multicast channel allocated to the group, the member belongs to the group. Moreover, members can join or leave dynamically.

There are however several problems with the scheme. Even through the message is encrypted once, the session key needs to be encrypted $|C| - 1$ times by the public key encryption algorithm used. Clearly, this is inefficient. Moreover, the multiple encryptions of the session key need to be sent along with the message, which increases the length of packets. At the receiver side, the receiver needs to search for the correct encrypted key (on average, half the number of the multiple key encryptions). If multiple encrypted keys are labeled or sequenced in some way so that the receiver can locate its corresponding encrypted key directly, then some secret information, such as the purpose or the security level of the message might be disclosed to a potential intruder [Chiou and W.T.Chen, 1989]. Even through there is no group size limitation, the scheme is not scalable because of the time and space complexity induced by multiple invocations of the public-key cryptosystem.

4.3 Chinese Remainder Theorem based (Secure Lock) Dynamic Conferencing Scheme

In order to avoid multiple encrypted keys in the above public key based dynamic conferencing scheme, a *secure lock* solution was proposed in [Chiou and W.T.Chen, 1989]. The lock is, in fact, a single value computed from the multiple encrypted keys using the Chinese Remainder Theorem (CRT). The reader is referred to Theorem 1.7 in Chapter 1 for a brief discussion and an example of CRT.

The Secure Lock scheme works as follows. An off-line server determines a sequence of $n = |G|$ pairwise relatively prime natural numbers N_1, \dots, N_n . These numbers are assigned to group members m_1, \dots, m_n . All the N_i are made public. When member m_i wants to send a message M to members $m_{i_1}, \dots, m_{i_\ell}$ he/she first establishes the following congruences:

$$\begin{aligned} u &\equiv E_{P_{i_1}}(k) \pmod{N_{i_1}} \\ &\vdots \\ u &\equiv E_{P_{i_\ell}}(k) \pmod{N_{i_\ell}} \end{aligned}$$

Then, m_i computes u by applying the CRT. Integer u will be the lock for the encrypted keys $E_{P_{i_j}}$, and is sent along the encrypted message. Thus, m_i sends $(u, \{M\}_k)$. When a receiver, such as m_{i_j} , receives the above packet, he/she can compute $E_{P_{i_j}}(k) = u \pmod{N_{i_j}}$, then obtains $k = D_{P_{i_j}}(E_{P_{i_j}}(k))$

using his/her private key, and finally decrypts the message using k , to recover M .

Just as in public key based dynamic conferencing, the secure lock scheme is inefficient and not scalable.

4.4 Symmetric Polynomial based Dynamic Conferencing Scheme

The public-key based dynamic conferencing schemes, mentioned in the previous sections, including secure lock, are computationally secure. However, a conferencing scheme based on symmetric polynomials is *unconditionally secure*. We describe such schemes in this section. We begin by discussing a limited dynamic scheme, that is, one in which the size of conferences is fixed. We then discuss an extended scheme which is not subject to this limitation.

4.4.1 Limited Symmetric Polynomial based Dynamic Conference Scheme

A symmetric polynomial based dynamic conferencing scheme was proposed in [Blundo et al., 1993, Blundo et al., 1998] and is summarized below.

Assume there is a group controller (GC), and let the group of size n be denoted by $G = \{m_1, \dots, m_n\}$. Let p be a prime such that $p \geq n$. The GC selects n distinct random numbers $s_i \in \mathbb{Z}_p$, and distributes s_i to member m_i . The s_i do not need to be secret. The GC constructs a random symmetric polynomial³ in t indeterminates ($t \leq n$) with coefficients from \mathbb{Z}_p in which the degree of any variables is at most w (w is a security parameter).

$$f(x_1, \dots, x_t) = \sum_{i_1=0}^w \cdots \sum_{i_t=0}^w a_{i_1, \dots, i_t} x_1^{i_1} \cdots x_t^{i_t}.$$

Then for $1 \leq i \leq n$, the GC computes a polynomial g_i in the $t-1$ variables x_2, \dots, x_t by setting $x_1 = s_i$ in $f(x_1, \dots, x_t)$. The coefficients of g_i comprise the secret information of m_i , and are transmitted to m_i securely. Finally, for any t subset of members $P = \{m_{i_1}, \dots, m_{i_t}\}$, the key associated with P is $k_P = f(s_{i_1}, \dots, s_{i_t}) \bmod p$. Each member $m_{i_j} \in P$ can compute $k_P = g_{i_j}(s_{i_1}, \dots, s_{i_{j-1}}, s_{i_{j+1}}, \dots, s_{i_t}) \bmod p$.

This scheme allows any subset of size t to form a dynamic conference. Moreover, the scheme is resilient to collusion of up to w members, that is, if fewer than $w+1$ group members collude, they cannot figure out any in-

³Recall that f is called a symmetric polynomial if $a_{i_1, \dots, i_t} = a_{\pi(i_1), \dots, \pi(i_t)}$ for all permutations π of $\{1, \dots, t\}$.

formation about the secret polynomial f . On the other hand, if more than w members collude, they can determine the polynomial f completely.

4.4.2 Extended Symmetric Polynomial based Dynamic Conference Scheme

In order to remedy the fixed- t restriction in the above conferencing scheme, we extended the scheme so that an arbitrary subset of G can form a conference [Zou et al., 2002a]. Moreover, the extended scheme is now able to deal with dynamics, allowing members to join and/or leave the group, especially handling bursty behavior efficiently. We summarize the extended scheme below.

Under the same assumption as for the above limited case, the scheme is extended as follows:

1. Extend the number of members to $2n$, i.e., $G = \{m_1, m_2, \dots, m_n, m'_1, m'_2, \dots, m'_n\}$ where the m'_1, m'_2, \dots, m'_n are called *dummy members*;
2. Set $t = n$ in the symmetric polynomial of the limited scheme, i.e., the GC constructs a symmetric polynomial in n indeterminates

$$f(x_1, \dots, x_n) = \sum_{i_1=0}^w \cdots \sum_{i_n=0}^w a_{i_1, \dots, i_n} x_1^{i_1} \cdots x_n^{i_n}$$

3. The GC selects $2n$ random numbers $s_1, s_2, \dots, s_n, s'_1, s'_2, \dots, s'_n$, all of which are made public. The s'_1, s'_2, \dots, s'_n are called *dummy numbers*.
4. For $i = 1, \dots, n$, the GC computes $g_i = f(x_1, x_2, \dots, x_{i-1}, s_i, x_{i+1}, \dots, x_n)$.
5. If there is a member corresponding to index i currently in the group, the member will be identified with the symbol m_i , and assigned public number s_i and private key g_i . On the other hand, if there is no current member in the group corresponding to index i , the dummy member m'_i and dummy number s'_i are used instead, and in this case, we also call index i a *non-occupied position*.
6. When a member joins the group, the GC finds a non-occupied position i for the member, identifies the member with m_i , and sends g_i to the member through a secure channel. The polynomial g_i becomes the *secret share* of m_i .
7. When the GC (or m_i) computes the group key from polynomial f (or g_i), s_j is used if there is a current group member corresponding to index j , and s'_j is used otherwise.

8. Suppose, without loss of generality, the current group members are m_1, \dots, m_ℓ , then the group key is

$$k = f(s_1, \dots, s_\ell, s'_{\ell+1}, \dots, s'_n)$$

Every group member m_i can compute k as follows:

$$k = g_i(s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_\ell, s'_{\ell+1}, \dots, s'_n).$$

9. Suppose, without loss of generality, m_2, m_4, m_5 want to form a conference, then the conference key is

$$ck = f(s'_1, s_2, s'_3, s_4, s_5, s'_6, \dots, s'_n)$$

Every conference member, for example m_4 , can compute ck as follows:

$$ck = g_4(s'_1, s_2, s'_3, s_5, s'_6, \dots, s'_n).$$

The main problem with the symmetric polynomials based dynamic conferencing scheme is that the space and time complexity for storing the coefficients of polynomial f or g_i and computing the polynomial is very high, in fact exponential in n . Moreover, the group size is fixed.

4.5 Tree based Dynamic Conferencing Scheme

4.5.1 Key Tree based Interval Multicast and Dynamic Conferencing (IDC)

In the key tree scheme, as used for SGC, all members use the root key for encryption and decryption of messages. On the other hand, the key tree scheme can be extended for interval communications or for dynamic conferencing. The authors of [Gouda et al., 2002a] propose an interval multicast scheme based on the key tree scheme and extend the scheme to dynamic conferencing in [Gouda et al., 2002b]. We briefly present results from these two papers here.

An interval $U[i, j]$ is defined as a set of members who are in consecutive positions from leaf i to leaf j (i.e., from m_i to m_j). A *basic* interval is an interval whose members exactly cover a subtree. A smallest (*minimal*) basic interval is an interval containing one only member, such as $U[i, i]$. For each basic interval $U[i, j]$, there is a key k_{i-j} which is the root key for the subtree covering the interval and is shared by all the members in the interval. For example, in Figure 4.1, $U[3, 6]$, $U[2, 7]$, $U[4, 7]$ are intervals, but only $U[4, 7]$ is a basic interval. $U[2, 7]$ consists of two basic intervals $U[2, 3]$ and $U[4, 7]$. The key corresponding to $U[4, 7]$ is k_{4-7} . $U[3, 6]$ consists of three intervals $U[3, 3]$,

$U[4, 5]$, $U[6, 6]$ where $U[3, 3]$ and $U[6, 6]$ are minimal intervals, corresponding to the leaf key k_3 and k_6 .

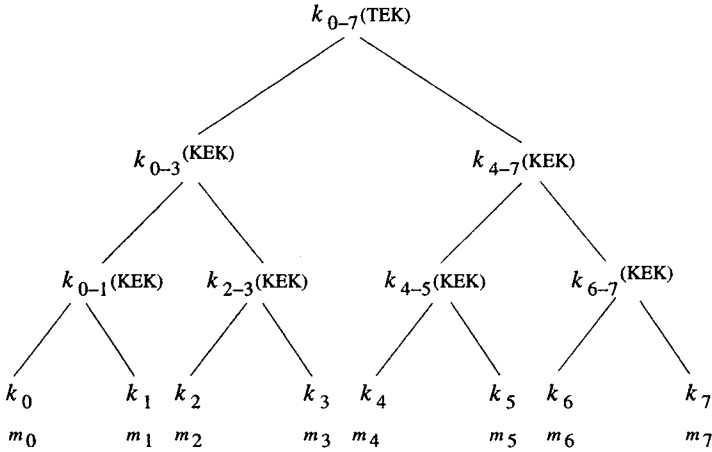


Figure 4.1. Key tree: each member assigned the keys from its leaf to the root.

The interval-based dynamic conferencing scheme works as follows. As in the key tree scheme, we assume there is a Group Controller (GC) who maintains the key tree. When a member m_i wants to send a message to a subset of members (the sender and the members in the subset form a conference), he first sends the message along with the information identifying the subset to the GC. The message is encrypted using m_i 's leaf key k_i before transmission. After decrypting the message, the GC performs the following functions: (1) if the subset of members forms a basic interval, then the GC encrypts the message using the key corresponding to the particular basic interval and broadcasts the message to the group; otherwise, (2) if the subset of members forms an interval, but not a basic interval, the GC splits the interval into the union of a number of disjoint basic intervals, and for every basic interval executes (1); otherwise, (3) the subset is characterized as *random*. In this case the GC splits the random subset into the union of intervals, and for every interval executes (2).

The overall idea is that a random subset of members is split into the union of basic intervals, and every basic interval has a corresponding key. The message will be encrypted using these keys respectively by the GC. Every member in the conference will belong to one (and only one) of these basic intervals, and has the key corresponding to this basic interval, so the member can decrypt the message.

There are several problems with the interval-based dynamic conferencing scheme. First, the GC relays all the messages, that is, the GC performs all computations for determining the multiple basic intervals, decrypts every message (once), encrypts the message (multiple times, once for each basic interval), and broadcasts the message to the group. This amounts to a considerable load for the GC. Secondly, the operations of splitting a random subset of members into intervals and each interval into basic intervals is inefficient and, in fact, unnecessary.

In the next subsection, we discuss a different scheme which avoids all of the above problems.

4.5.2 An Efficient and Scalable Key Tree based Dynamic Conferencing Scheme (KTDC)

We assume that every member has the knowledge of the current group, in particular, all members have the same image of the key tree as the GC. The main properties of this scheme are the following: (i) Conference key management and data communication for conferences are independent. (ii) The GC is only responsible for distributing the conference keys, but is not involved in key computation/generation or relay of data messages, and therefore, the GC will not be a bottle-neck. (iii) There is no computation for identifying basic intervals for conferences. However, we propose an elegant algorithm which can determine directly and efficiently the keys corresponding to the basic intervals.

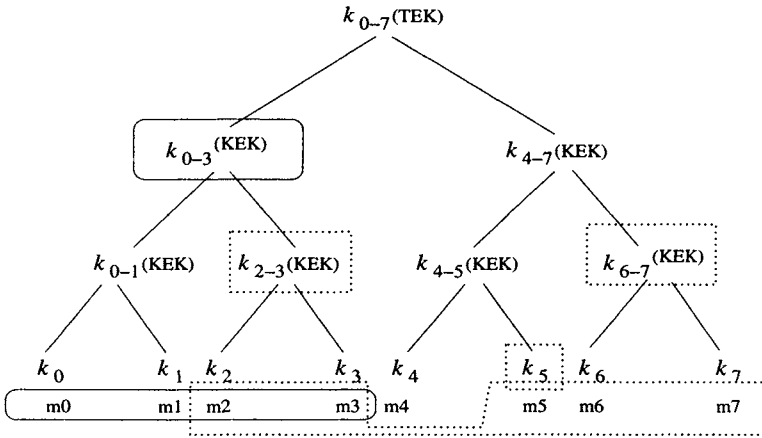


Figure 4.2. Conference members are covered under nodes.

We describe the details of the scheme below. Whenever a member wants to send messages to a conference, the following steps are carried out.

(1) Based on the binary string indexing of keys and members proposed in [Zou et al., 2002b], any member can directly determine the indices of keys needed for encryption in a very efficient way (the algorithm will be discussed below). These keys correspond exactly to the basic intervals discussed above, but we do not use the notion of an *interval* here.

(2) If there is only one key to be determined, that is, all members in this conference happen to form a basic interval, the key will be used as the conference key. Thus, conference communications can be encrypted and decrypted using this key directly. For example, in Figure 4.2, m_0 initiates a conference containing m_0, m_1, m_2, m_3 , which are exactly covered under node k_{0-3} . So, k_{0-3} will be used as their conference key. In this case, there is no involvement of the GC.

(3) Otherwise, there are multiple keys corresponding to multiple basic intervals. Two steps will be performed. First, the member who initiates the conference communication selects a random number R as the conference key CK , encrypts CK with his/her own leaf-node key, and sends the encrypted CK along with the indices of the keys determined above, to the GC. In the second step, the GC decrypts the conference key CK , re-encrypts CK individually with each of the keys whose indices are listed, and broadcasts the new encrypted CK to the group. As a result, all members in the conference obtain CK and can perform conference communications securely using CK . Let us see an example from Figure 4.2 again. Suppose m_2 initiates a conference containing $\{m_2, m_3, m_5, m_6, m_7\}$ (dotted boxes in the figure). The nodes which *exactly* cover these members are k_{2-3}, k_5, k_{6-7} . m_2 will encrypt a randomly selected new conference key CK with its leaf key k_2 and sends it along the key indices (i.e., $(\{CK\}_{k_2}, \{2-3, 5, 6-7\})$) to the GC. When the conference members receive the conference key CK , they can perform conference communication without the involvement of the GC. As we can see, the GC just *relays* conference keys, but is not involved in any computation/generation of conference keys or the relaying of conference messages.

It can be seen that the first task in the scheme is to determine (the indices of) the keys. Based on an efficient implementation for the key tree scheme described in [Zou et al., 2002b], *using binary strings to index keys and members* (also see Section 3.1.2.2 of Chapter 3), we design an algorithm: “**Determine-keys-of-conference-in-key-tree** ($m, i[\]$)”, which can perform this task efficiently (see Figure 4.3).

The basic operation in the algorithm is *neighboring comparison*. Whenever the current member’s ID (after a right shift of 1 bit) is not equal to the next member’s ID (also after a right shift of 1 bit), the current member is not within the same basic interval as the next member, so the corresponding key with index as the current member’s ID will be involved in a basic interval and thus, is stored. Then, the comparison for this round continues. Otherwise, when the

current member's ID (after a 1 bit right shift) is equal to the next member's ID (after a 1 bit right shift), the two members are in the same basic interval and one of them (with ID right-shifted by 1 bit) is kept for the next round comparisons which will search for larger basic intervals. Comparisons for this round continue.

```

Determine-keys-of-conference-in-key-tree ( $m, i[]$ ):
  /* $m$ : conference size;  $i_0, i_1, \dots, i_{m-1}, i_0$ : members' IDs list*/
  /* $i_0$  is appended to simplify comparison between  $i_{m-1}$  and  $i_0$ .*/
  While (current member ID list is not empty) {
    Scan all IDs in current member ID list
      /* $Current\_ID$  is a scanning pointer*/
      if ( $Current\_ID \gg 1 \neq Next\_ID \gg 1$ ) {
        Store the key corresponding to  $Current\_ID$ 
        remove the  $Current\_ID$  from current ID list
        Move  $Current\_ID$  to the next ID
      } else {
        Replace  $Current\_ID$  with  $Current\_ID \gg 1$  in the ID list
        Delete  $Next\_ID$  from current ID list
        Move  $Current\_ID$  to the next ID
      }
  }

```

Figure 4.3. Algorithm to determine the keys of the conference in a key tree.

It is worth discussing the burden incurred by group members when moving certain functions from the GC to members. Let us first consider the burden when group members keep the same image of the key tree as the GC. First, members just need to maintain a tree structure which indicates where every member is placed and they do not perform any operation of key generations, key updates in the tree, and key distribution. Maintaining a (key) tree is an easy task and there exist many elegant data structures to represent a tree and efficient algorithms to implement a tree. Second, the GC does not need to transmit the key tree to group members. We assume that group communication is based on multicast, which implies that the joining and leaving requests will be received by all the group members. As long as group members use the same tree update policies (e.g., which location a new joining member should be placed at, when to expand the tree, when to compress the tree, etc.) as the GC, they will achieve the same key tree. As a result, keeping the key tree is not a big burden on members. Next, let us consider the computation of indices of keys

which is also moved to members from the GC. First, the KTDC protocol uses a more efficient algorithm for the index computation than IDC. Second, the frequency of the index computation is much lower than that of group messages transmissions because only one computation is needed for a conference and each conference will last for a period of time. Thus the computation of key indices is not a big load to members either. In contrast, the GC in IDC needs to relay all messages for all conferences, which is a prohibitive task.

4.6 BF-TGDH based Distributed Dynamic Conferencing (BF-TGDH DC)

In chapter 3, we discussed the BF-TGDH scheme for SGC. BF-TGDH provides the *blocking free* property by introducing *dummy root keys*. Based on dummy root keys, the BF-TGDH scheme can be extended to obtain an elegant distributed dynamic conferencing scheme, i.e., a random conference of any collection of members can be formed, and the conference key can be computed easily from the dummy root keys. The scheme is called BF-TGDH DC, and the idea is as follows: Since every member m_i has all the dummy root keys except its own DRK_i , the *conference key* is computed as the one way function applied to the XOR of all dummy root keys except those whose members are in the conference. That is, if S is the conference and G the full group, so that $S \subset G$,

$$CK_S = POF\left(\bigoplus_{m \in G-S} DRK_m\right).$$

For example, if m_2 , m_4 , and m_7 want to form a conference, then each of them can compute the conference key as $POF(DRK_1 \oplus DRK_3 \oplus DRK_5 \oplus DRK_6 \oplus DRK_8 \oplus \dots \oplus DRK_n)$, i.e., the POF of the XOR of all DRK_i 's ($i \neq 2, 4, 7$). Any other member m_i ($i \neq 2, 4, 7$) cannot compute the key since m_i does not have DRK_i .

4.7 Discussion and Comparisons

We can discuss the properties of the above schemes based on different criteria. We believe the following criteria are appropriate for comparing and categorizing all dynamic conferencing schemes: (i) centralized vs. distributed, (ii) unconditionally secure vs. computationally secure, (iii) fixed vs. variable group size, (iv) static vs. dynamic group, (v) fixed vs. variable conference keys.

In a centralized scheme, there is a central trusted authority, such as a group controller which manages and distributes keys. However, in distributed schemes there is no need for central control. Among the schemes examined thus far,

only the *public-key* based dynamic conferencing scheme is totally distributed. The *naive* and *secure lock* based schemes can be considered as distributed except that an off-line trusted server is needed during the setup phase to initialize the group. By an *unconditionally secure* scheme we mean one in which conference keys are independent and it is impossible to compute other keys from some keys already known. A *computationally secure* scheme is one in which it is impossible to compute any other key, assuming bounded computing resources, when some keys are known. Among the schemes examined, the naive scheme, the key tree based scheme, and the symmetric polynomial based scheme are unconditionally secure. All these are based on secret-key cryptosystems except that the underlying secure channel between the GC and every member may be implemented by means of a public-key cryptosystem.

Table 4.1. Comparison of dynamic conferencing schemes

Scheme	Distributed	Uncond. secure	Group variability	Dynamics	Scalability ¹	VCK ²
Naive	No	Yes	No	No	No	No
PKDC	Yes	No	Yes	Yes	No	Yes
SLDC	Yes ³	No	Yes	Yes	No	Yes
SPDC	No	Yes ⁴	Yes ⁵	Yes	No	No
IDC ⁶	No	Yes	Yes	Yes	No	No
KTDC	No	Yes	Yes	Yes	Yes	Yes
BF-TGDH DC	Yes	No	Yes	Yes	Yes	No

¹ Different schemes have different reasons for non-scalability.

² Variable (changeable) conference keys.

³ SLDC needs an off-line trusted server to determine pairwise relatively prime numbers.

⁴ SPDC is unconditionally secure. However, when more than w members collude, they can obtain the secret polynomial.

⁵ In SPDC, group members are allowed to join and/or leave, however the maximum group size is not allowed to change (i.e., fixed).

⁶ IDC stands for Interval based Dynamic Conferencing.

The naive scheme assumes a predetermined, fixed group which does not allow dynamics. The symmetric polynomial based scheme assumes a group of fixed size but is highly dynamic. All other schemes have no limitation on group size⁴ and allow dynamics. For a number of different reasons none of schemes examined is scalable except the key tree based scheme. The naive scheme needs an exponential number of keys. The public-key based and se-

⁴In the public-key based scheme the notion of *group* is of little importance and one can regard the multicast channel as the group. Also, in the secure lock scheme the notion of group is unimportant except that there are n predetermined pairwise relative prime numbers.

cure lock based schemes are inefficient due to the public-key cryptosystems involved. The symmetric polynomial based scheme is not scalable because there is considerable representational complexity in the polynomials, requiring extensive space and time resources. The interval based scheme is not scalable because the group controller is involved in relaying all conference messages.

As for conference keys, the naive scheme, the symmetric polynomial based scheme, and the interval based scheme use fixed conference keys for each conference. However, each of the other schemes allows a member to select a random number as the key for a conference, while the conference is active, for the messages of the conference sent by this member, or for every message. The variability of conference keys may increase security in some scenarios.

The results are summarized in Table 4.1.

Chapter 5

SECURE GROUP COMMUNICATIONS WITH HIERARCHICAL ACCESS CONTROL

Secure group communication (SGC) with hierarchical access control (HAC) refers to a scenario in which a group of members is divided into a number of subgroups located at different privilege levels and a high-level subgroup can receive and decrypt messages within any of its descendant lower-level subgroups, while the reverse is not allowed. HAC is generally enforced using cryptography based techniques [Birget et al., 2001] i.e., cryptographic keys play a primary role in the control of access rights. If the members of a higher level subgroup possess or can derive the key of a lower level subgroup, they have the right to access all messages within the lower level subgroup. In this chapter, we discuss several typical SGC schemes with HAC.

5.1 Classification

In SGC with HAC, when a member joins or leaves a subgroup S , not only the key of S needs to be changed, but also the keys of all descendant subgroups of S because the joining/leaving member already knows all descendant subgroup keys. Therefore, secure group communication with hierarchical access control is more difficult to treat than secure group communication without HAC. Some work dealing with the hierarchical access control problem has been done [Akl and Taylor, 1983, Chick and Tavares, 1990, Halevi and Pe-trank, 1995, Lin, 1997, Mackinnon et al., 1985, Sandhu, 1988].

Cryptography based hierarchical access control (HAC) schemes can be classified as (i) *unconditionally secure* or (ii) *computationally secure (one-way function based)* schemes. In an unconditionally secure scheme, keys of vertices are totally independent and there is no information which might reflect any relationship between the keys. Here, the uncertainty in any particular vertex key is equal to the a priori uncertainty about keys. Unconditionally secure access control schemes are divided into three sub-classes: (i) *user multiple*

keying schemes, (ii) *resource multiple keying schemes*, and (iii) *mixed keying schemes*. The one-way function based schemes can be divided into two categories. In the first category a subgroup key can be directly derived from the parent key using a particular one-way function. Thus any subgroup key can be derived, directly or indirectly, from any one of its ancestors' keys. Moreover, messages, which are sent by members in a subgroup and encrypted by their subgroup key, can be decrypted by all members in their ancestral subgroups [Akl and Taylor, 1983, Chick and Tavares, 1990, Mackinnon et al., 1985, Sandhu, 1988]. We call this class of schemes *directly dependent key schemes*. The second category [Lin, 1997] is characterized by the fact that all subgroup keys are independent. However, there are some parameters which are computed by the GC using some one-way function and reflect the relationship between the keys of a parent subgroup and a child subgroup. From these parameters, an ancestor can compute the keys of all its descendants, but a descendant cannot compute the key of any of its ancestors. Therefore, when a member in a subgroup sends a message, all members in its ancestor subgroups can decrypt the message, but no member in any of its descendant subgroups can decrypt the message. We call this second class of schemes *indirectly dependent key schemes*. Based on the structure of the hierarchy, some hierarchical access control schemes deal with a general *Directed Acyclic Graph* (DAG), while other schemes deal with a DAG having a unique root vertex. There are further schemes which deal with specific types of DAG's such as a tree structure. In the following sections we discuss (1) Unconditionally secure keying schemes for HAC, (2) One-way function based (computationally secure) keying schemes for HAC, (3) Index based scheme for SGC with HAC, and (4) Chinese Remainder Theorem based HAC scheme for SGC.

We make the following assumptions: (1) The hierarchy is defined by a Directed Acyclic Graph (DAG). (2) There is a group controller (GC) who manages the hierarchy and may compute and distribute keys. (3) Since every vertex in the hierarchy represents a set of users, we also call a vertex a *subgroup*. (4) There is a subgroup controller for each subgroup who is responsible for securely distributing its subgroup key to all members of the subgroup. (5) Between the GC and every subgroup controller, there is a secure channel to guarantee secure communications. (6) There is an underlying secure key management scheme such as "*key tree*" [Molva and Pannetrat, 1999, Noubir, 1998, Wong et al., 1998] (also see Section 3.1.1) used by every subgroup controller to perform subgroup key management.

5.2 Unconditionally Secure Keying Schemes for Hierarchical Access Control

In this section, we discuss a class of keying schemes that assign keys to users and resources so that a user can access a resource if and only if the user

possesses the specific key for the particular resource. The keys are independent from each other forming an unconditionally secure scheme.

Let (V, \preceq) be an access hierarchy (i.e., a DAG). In these keying schemes, every vertex v in the hierarchy has a key k_v chosen from a large key space. Users and resources are mapped onto the vertices of the DAG so that a user occupies a unique vertex of V , and similarly, a resource is attached to a unique vertex of V . Each user at v knows a set of keys $U_v \subseteq \{k_w : w \preceq v\}$ (i.e., the user knows some keys of lower-ranking resources), and each resource at vertex v knows a set of keys $R_v \subseteq \{k_w : v \preceq w\}$ (i.e., the resource knows some keys of higher-ranking users). The sets U_v and R_v must be chosen in such a way that

$$(*) \quad v_i \preceq v_j \text{ if and only if } U_{v_j} \cap R_{v_i} \neq \emptyset.$$

A user u_j at vertex v_j requesting a resource r_i at vertex v_i will present the set U_{v_j} to the resource. The resource then checks whether $U_{v_j} \cap R_{v_i} \neq \emptyset$, which holds if and only if $v_i \preceq v_j$, i.e., if and only if u_j has the right to access r_i .

In this protocol, a resource can get information about the keying material held by users. A user u_j , however, only knows his set of keys U_{v_j} and no other keys. An alternative would be to use (local) trusted third parties, to whom keying material is presented by users and resources. Such a trusted third party does not need to have knowledge of the sets U_v and R_v , it just has to check condition (*) when a user and a resource present their sets.

There are many ways of choosing the sets U_v and R_v so that condition (*) holds. We illustrate this approach in simple special cases, namely ‘user multiple keying’, ‘resource multiple keying’, and ‘mixed keying’.

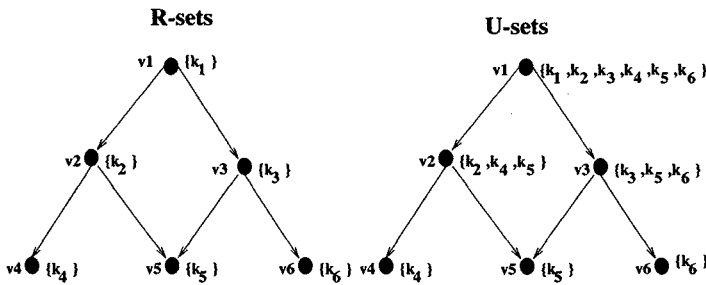


Figure 5.1. User multiple keying.

(a) User multiple keying:

Here, for each vertex v :

$$R_v = \{k_v\}, \text{ and}$$

$$U_v = \{k_{v_j} : v_j \preceq v\}.$$

A user u requesting access to a resource r presents U_u to r . The resource r verifies u 's access right by checking whether $k_r \in U_u$, which is equivalent to condition (*) in this case.

Example: See Figure 5.1, which represents two copies of an access hierarchy. The first copy presents the sets R_v , while the second copy displays the sets U_v . Here, $U_{v_2} = \{k_2, k_4, k_5\}$, $R_{v_2} = \{k_2\}$, $R_{v_4} = \{k_4\}$, and $R_{v_5} = \{k_5\}$. Therefore user u_2 can access resources r_2, r_4 , and r_5 .

(b) Resource multiple keying:

This scheme is similar to User multiple keying, with the roles of user and resources interchanged. Figure 5.2 shows an example of resource multiple keying.

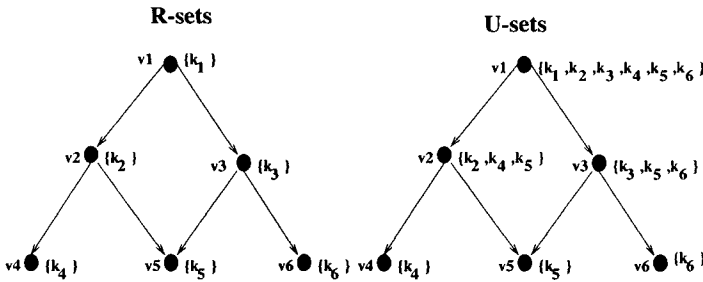


Figure 5.2. Resource multiple keying.

(c) Mixed keying:

This is the general case. There are many possibilities, and we give just one example (see Figure 5.3): Here, $U_{v_1} = \{k_1, k_5\}$, moreover, $R_{v_2}, R_{v_3}, R_{v_4}, R_{v_6}$ contain the key k_1 , so user u_1 at vertex v_1 can access the resources at vertices v_2, v_3, v_4, v_6 . User u_1 can also access the resource at v_5 because U_{v_1} and R_{v_5} share the key k_5 . One can check from the two graphs in Figure 5.3 that condition (*) holds at all the vertices.

5.3 One-way Function based Schemes for Hierarchical Access Control

In this section, we summarize several one-way function based schemes which are computationally secure and are proposed for hierarchical access control.

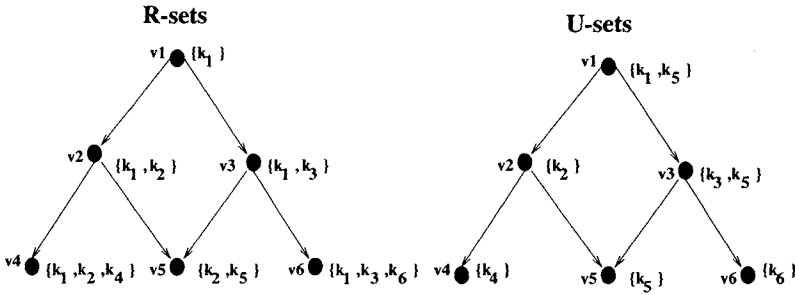


Figure 5.3. Mixed-Keying.

5.3.1 The Akl-Taylor Scheme

The scheme proposed in [Akl and Taylor, 1983] is a *directly dependent key scheme*. The mechanism is as follows. Assume there is a group controller, GC and a root vertex, v_0 , which is the ancestor of all other vertices. The GC selects two large primes p and q , and computes $M = pq$ which is made public. For every vertex v_i , the GC selects a distinct prime p_i and assigns p_i to v_i . Then the GC computes $t_i = \prod_{v_j \leq v_i} p_j$ and assigns t_i to vertex v_i . The root vertex v_0 is assigned integer 1 (see Figure 5.4). As a result, the following condition is satisfied: $v_i \leq v_j$ if and only if $t_j | t_i$ (i.e., t_i is divisible by t_j). All t_i are made public. Finally the GC randomly selects a large random integer k_0 as the key for the root vertex, and computes the key for v_i as $k_i = k_0^{t_i} \text{ mod } M$.

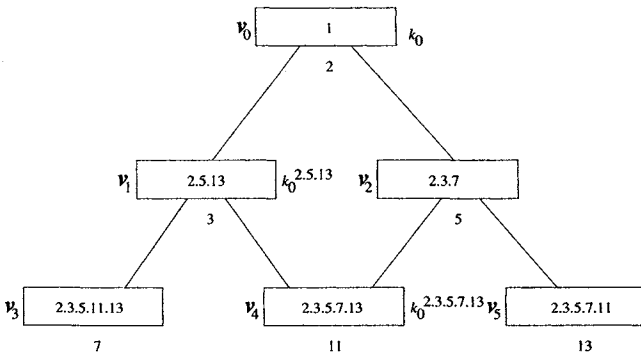


Figure 5.4. Assignment of primes, t_i s, and k_i s (Source: [Akl and Taylor, 1983], © ACM).

Each k_i is securely sent to the respective v_i . As a result, if $v_i \leq v_j$, then v_j can compute v_i 's key as follows: $k_i = k_0^{t_i} = k_0^{t_j \cdot t_i / t_j} = k_j^{t_i / t_j}$. However, v_i can not compute v_j 's key k_j . Thus, hierarchical access control is enforced. It is worth to mention that because of the modulus operation, all the k_i 's have the

same magnitude whether they are computed from k_0 with larger exponent t_i 's or smaller t_i 's.

In [Mackinnon et al., 1985], an improved algorithm for assigning primes to vertices was proposed and can achieve a near-optimal assignment in the sense that it produces the smallest values for the t_i 's. The principle is utilizing the following rules:

- 1 $p_i \neq p_j$ if $v_j \not\leq v_i$ and $v_i \not\leq v_j$.
- 2 $p_i = p_j$ if $v_i \leq v_j$ and this does not conflict with the above rule.
- 3 p_i 's are the smallest allowable primes.

The advantages of the Akl-Taylor scheme are that (i) There is no need for a member to remember the access hierarchy since the hierarchical access relationship is completely implied by the divisibility among the t_i 's, and (ii) There is no need to remember any values other than the member's own key. The main problem with the scheme is that the hierarchy is static; any required key change will result in a change of the root key as well as all other keys.

5.3.2 Flexible Access Control with Master Keys

In a paper by Chick and Tavares, [Chick and Tavares, 1990], a modification to the Akl-Taylor scheme is proposed. The modified scheme is more flexible and has several advantages over the Akl-Taylor scheme. In this section we discuss the modified scheme.

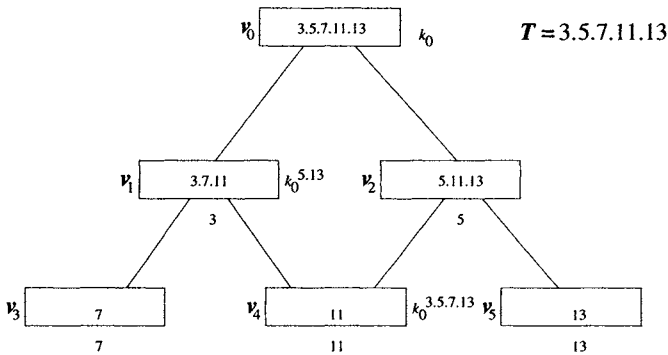


Figure 5.5. Assignment of primes, t_i s, and k_i s in the modified scheme.

Just as in the the Akl-Taylor scheme, all vertices (except for root vertices) are assigned different primes p_1, p_2, \dots, p_n . Let $T = \prod_{i=1}^n p_i$. Each vertex v_j is assigned the value $t_j = \prod_{v_i \leq v_j} p_i$ and secret key $k_j = k_0^{T/t_j} \text{ mod } M$, where the modulus M is same as in the Akl-Taylor scheme. A random, secret

value k_0 can be treated as the root key if there is only one root vertex in the hierarchy. Figure 5.5 shows an example. If $v_i \leq v_j$, then v_j can compute v_i 's key as follows: $k_i = k_0^{T/t_i} = k_0^{(T/t_j) \cdot t_j/t_i} = k_j^{t_j/t_i}$. However v_i can not compute v_j 's key k_j .

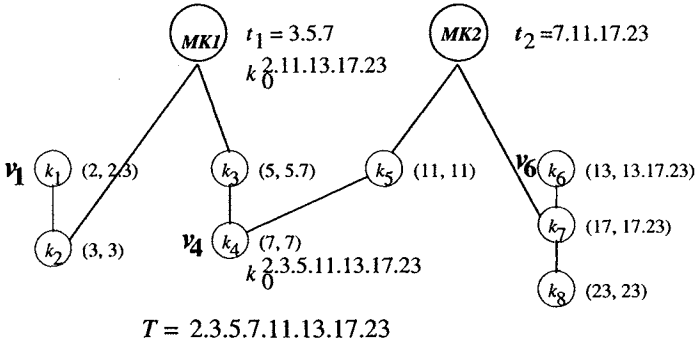


Figure 5.6. Multiple root vertices and master keys.

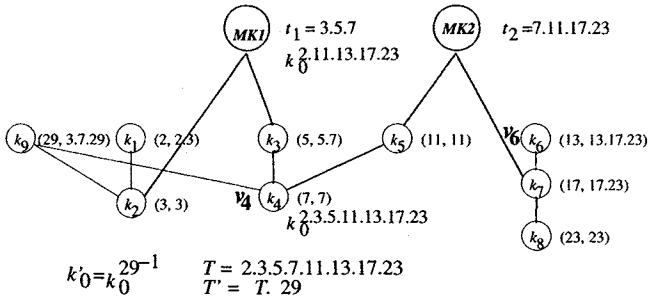


Figure 5.7. Adding a new vertex to the access hierarchy.

The first advantage of the modified Chick-Tavares scheme over the Akl-Taylor scheme is that the modified scheme is applicable to a wider class of order relations, including ones with multiple roots, and order relations in which there are vertices not dominated by any root. Figure 5.6 illustrates such an example with a flexible order relation. Here, each vertex is labeled (on the right) by the tuple (p_i, t_i) . The vertices denoted by MK_1 and MK_2 are root vertices and their keys are called master keys. Some vertices such as v_1 and v_6 are not subordinate to any root vertex. One extra advantage over the Akl-Taylor scheme is that the t_i 's could be much smaller when the number of vertices is large.

The third advantage over the Akl-Taylor scheme is that adding a vertex (sub-group) can be done without affecting other vertices as long as the new vertex is not subordinate to any existing vertex (see Figure 5.7). Suppose a new vertex v_{n+1} is added to the access hierarchy. A new prime p_{n+1} is assigned to v_{n+1} , chosen so that it is distinct from p_1, \dots, p_n and which does not divide $\phi(M)$. Since $\gcd(p_{n+1}, \phi(M)) = 1$, the multiplicative inverse p_{n+1}^{-1} in $\mathbb{Z}_{\phi(M)}$ exists, and can be computed efficiently (using Extended Euclidean Algorithm). Let $T' = T \cdot p_{n+1}$ and $k'_0 = k_0^{p_{n+1}}$. The value t_{n+1} is computed as before, i.e., $t_{n+1} = \prod_{v_i \leq v_{n+1}} p_i$, and k_{n+1} is computed from k'_0 , i.e., $k_{n+1} = (k'_0)^{T'/t_{n+1}}$. For all the previous keys, $k_i = (k_0)^{T/t_i} = (k_0^{p_{n+1}})^{(T \cdot p_{n+1})/t_i} = (k'_0)^{T'/t_i}$. Thus, after the introduction of p_{n+1} , the previous keys are not affected by the substitution of T' for T and k'_0 for k_0 .

5.3.3 Lin's Scheme

The scheme proposed in [Lin, 1997] is an *indirectly dependent key scheme*.

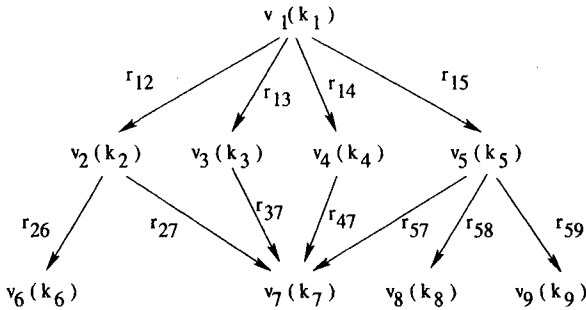


Figure 5.8. Independent keys and the corresponding r_{ji} s.

We present the scheme here, with a small generalization (see Figure 5.8). Every vertex v_i selects its own independent key k_i which is sent to the GC securely. For all $v_i \leq v_j$, the GC computes an r_{ji} as follows: $r_{ji} = F(k_j, ID_i) \oplus k_i$ where F is a one-way function¹ and ID_i is the identification of v_i . All the r_{ji} are made public. Using r_{ji} , v_j can compute the key k_i of v_i as follows: $k_i = F(k_j, ID_i) \oplus r_{ji}$. However v_i cannot compute k_j efficiently because of the one-way property of F . Thus, the hierarchical access control is enforced.

The advantages of the Lin's scheme are that every vertex can change its own key independently, and inserting and deleting a vertex is easy. However, apart

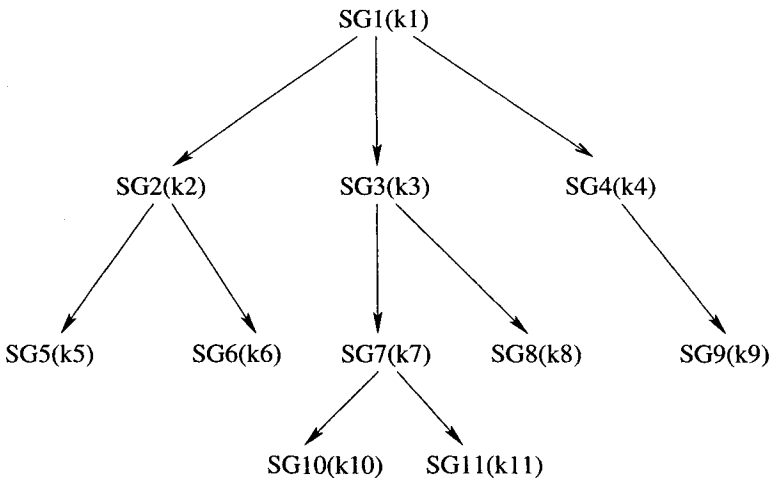
¹In Lin's scheme [Lin, 1997], $F(k_j, ID_i) = Z^{k_j \oplus ID_i} \text{ mod } P$ where P is a large prime number and Z is a primitive element of $GF(P)$.

from its own key, a member needs to remember the parameters r_{ji} or to get the r_{ji} from some public medium whenever the member needs them. Moreover, a member needs to remember the hierarchy information (or, at least, all of its own descendants.)

The above two schemes accept a generic hierarchical structure, that is, a DAG. When the hierarchy is reduced to a tree structure, there are more efficient schemes. In the sections that follow, we describe two tree-based HAC schemes.

5.3.4 Sandhu's Scheme

In [Sandhu, 1988], it is assumed that the hierarchical structure is a tree. We use S_i to denote a subgroup and $name(S_i)$ its name. The scheme is based on one-way functions, and works as follows. The GC selects a random key k_1 as the root key, computes the keys of all subgroups as described below, and sends each key to the corresponding subgroup securely. Suppose S_j is a child of S_i with key k_i , the key of S_j can be computed as $k_j = E_{k_i}(name(S_j))$ where $E_k(x)$ is a one-way function (see Section 5.4.4 for a discussion on this kind of one-way function) (see Figure 5.9). Now assume S_i has a descendant S_j with intermediate subgroups $S_{i_1}, S_{i_2}, \dots, S_{i_k}$ along the path from S_i to S_j .



Example: $k_7 = E_{k_3}(name(SG7)) = E_{E_{k_1}(name(SG3))}(name(SG7))$

Figure 5.9. Keys computed from their ancestors' keys in Sandhu's scheme (Source: [Sandhu, 1988], © Elsevier).

Then S_i can compute the key of S_j as follows:

$$\begin{aligned} k_j &= E_{k_{i_k}}(\text{name}(S_j)) = \\ &E_{E_{k_{i_{k-1}}}(\text{name}(S_{i_k}))}(\text{name}(S_j)) = \\ &\dots \\ &= E_{E_{\dots E_{k_{i_1}}(\text{name}(S_{i_1}))\dots(\text{name}(S_{i_k}))}(\text{name}(S_j))} \end{aligned}$$

However, S_j cannot compute k_i from its key k_j because of the one way property of the function E .

The advantages of Sandhu's scheme are that each member only needs to remember its own key and is allowed to insert or delete a vertex from the hierarchy. However a member needs to remember the hierarchy information (at least all its descendants.) The limitation is that the scheme can be used with a tree structure only.

5.4 Index based Scheme for SGC with HAC

The *index based* hierarchical access control scheme (IHACS) for a tree hierarchical structure is simpler than Sandhu's scheme and is discussed in this section.

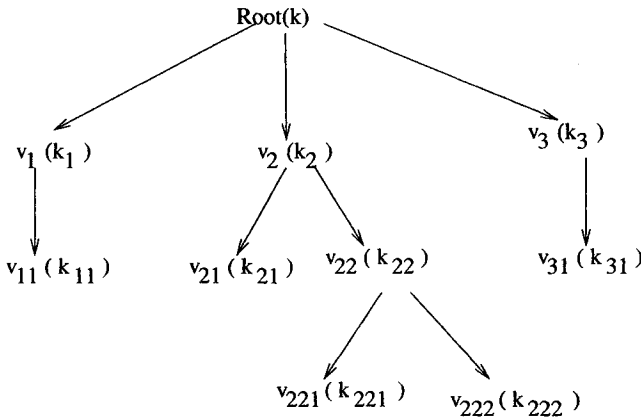
5.4.1 Principle

Let $F(x, c)$ be a one-way function, that is, it is easy to compute $y = F(x, c)$ if we know x and c , however, it is computationally infeasible to compute x from y and c .

Every subgroup is assigned a unique but dynamic *index* with the exception that the root subgroup has a fixed *index* 0. The *index* of a subgroup will indicate its level and the relationship with its parent, its children, as well as its siblings. For example, if a subgroup has *index* 231, then it is at level 3 (number of decimal digits), its parent's *index* is 23, its children's *indices* will be 2311, 2312, 2313, ..., and its siblings' *indices* will be 232, 233, ... (See Figure 5.10.)

Assume the root subgroup key is k , a random number selected by GC. All other subgroup-keys will be computed using the one-way function $F(x, c)$ as follows.

- The keys of the subgroups at level 1 are:
 $k_1 = F(k, 1), k_2 = F(k, 2), \dots, k_i = F(k, i)$
- The keys of the subgroups at level 2 are:
 $k_{11} = F(k_1, 1) = F(F(k, 1), 1)$
 $k_{12} = F(k_1, 2) = F(F(k, 1), 2)$
 \dots
 $k_{ij} = F(k_i, j) = F(F(k, i), j)$



Example: $k_{221} = F(k_{22}, 1) = F(F(k_2, 2), 1) = F(F(F(k, 2), 2), 1)$

Figure 5.10. Hierarchical key management using indices.

- ...
- The keys of the subgroups at level m are:

$$\begin{aligned}
 k_{i_1 i_2 \dots i_m} &= F(k_{i_1 \dots i_{m-1}}, i_m) = \dots \\
 &= F(\dots F(F(k, i_1), i_2), \dots, i_m).
 \end{aligned}$$

From the above computation formulas, it is easy to see that a subgroup-key is computed from its parent subgroup-key and its index, in particular, the last digit in its index. Thus, recursively, a subgroup-key for subgroup v is computed from the root subgroup-key and v 's index. Once the root subgroup key is fixed, the key of a subgroup is determined by its index. Change and management of keys now become change and management of indices. Suppose a subgroup with index $i_1 i_2 \dots i_k$ has key $k_{i_1 i_2 \dots i_k}$, then it can compute the key of its descendant subgroup with index $i_1 i_2 \dots i_k \dots i_m$ by:

$$\begin{aligned}
 k_{i_1 i_2 \dots i_m} &= F(k_{i_1 \dots i_{m-1}}, i_m) = \dots \\
 &= F(F(\dots F(k_{i_1 i_2 \dots i_k}, i_{k+1}), \dots, i_{m-1}), i_m)
 \end{aligned}$$

However, since F is a one-way function, a subgroup cannot compute its ancestors' keys efficiently, thus, the requirement of hierarchical access control is achieved.

Now the following questions arise: (i) How are we to distribute the keys to subgroups securely? (ii) How are we to perform subgroup dynamics and member dynamics in an efficient way? In what follows we discuss these problems.

5.4.2 Key Management

5.4.2.1 Key Distribution

Whether at the initial setup stage or at a time when a subgroup key is changed, the GC needs to distribute the keys to the members of subgroups (perhaps just one, or possibly all subgroups). The key distribution is done in two steps. The GC first sends the subgroup keys to the subgroup controllers of the respective subgroups using secure unicast. Then the subgroup controllers broadcast/multicasts their subgroup-keys to members of their own subgroup using a predetermined underlying secure communication scheme (e.g., those discussed in Chapters 2 and 3). The advantages of this two-level management include freeing the GC from taking care of too many members, increasing scalability, and enhancing security. When in the following sections, we say “the GC sends the keys to the members of a subgroup”, we mean, in fact, that the above two steps take place.

5.4.2.2 Subgroup Dynamics

By *subgroup dynamics*, we mean the process of *adding* (inserting) a subgroup into, *deleting* a subgroup from, *merging* two subgroups in, or *splitting* a subgroup in the hierarchy. We discuss each of these below.

When a new subgroup S is inserted into the hierarchy, the GC first determines its position in the hierarchy, finds an available *index* for the subgroup, assigns the *index* to it, computes its key, and sends it to the subgroup. If the operation is insertion, the GC also changes all the *indices* of the descendant subgroups of S and computes their keys.

When a subgroup S on node v is removed/deleted from the hierarchy, there are two possibilities: one is that the sub-tree of the deleted subgroup is also removed, and the other possibility is that the root of the sub-tree of v will move to the parent node of v (i.e., all v 's children become the children of v 's parent). In the latter case, the GC assigns new *indices* to the descendant subgroups of S , and recomputes and distributes all these keys.

When two subgroups need to be merged, the GC assigns an available *index* to the new fused subgroup and computes its key. At the same time, the GC reassigns the *indices* to all the subgroups which were originally descendants of the two fused subgroups and recomputes the keys.

When a subgroup splits, the GC finds two available *indices*, assigns them to the respective components and computes the keys. At the same time, the GC splits all subgroups which were originally descendants of the subgroup into two parts and assigns each of the two parts to the two new subgroups respectively. Then, the GC assigns the new *indices* to these subgroups and computes the new keys.

5.4.2.3 Member Dynamics

As has been pointed out earlier, the subgroup *index* plays an important role in rekeying. The *index* of a subgroup changes dynamically along with requests for key changes. Generally speaking, when a member joins or leaves a subgroup, the subgroup *index* will be changed. The new subgroup key will be the value computed by using this new *index*. For example, if there are three sibling subgroups: $SG21(k_{21})$, $SG22(k_{22})$, and $SG23(k_{23})$ and the key k_{22} of $SG22$ needs to be changed, the *index* 22 will be changed to the next available *index*, that is to 24, so that $SG22$ is changed to $SG24$. Then its new key is computed as $k_{24} = F(k_2, 4)$. As a result, the three siblings are now labeled as $SG21$, $SG23$, and $SG24$. If at this time $SG21$ needs to change its key, it will be changed to $SG25$, and the new key will be $k_{25} = F(k_2, 5)$.

Obviously, if we only use decimal digits 1 through 9 in *indices*, problems will arise. The first problem is that if a subgroup has more than 9 children, the 9 digit values are not enough to number all subgroups. The second problem is that the available *indices* may soon be exhausted during rekeying operations. In order to solve these problems, we may increase the range of digit values in *indices*. More generally, indices may be thought of as positive integers represented with respect to some selected *radix* R , or even with respect to a mixed radix (R_1, R_2, \dots, R_h) where the R_i are selected large enough to accommodate the largest hierarchical structure possible subject to the parameters of the particular application. An equivalent approach would be to use a set of the form $\mathbb{Z}_{r_1} \times \mathbb{Z}_{r_2} \times \dots \times \mathbb{Z}_{r_h}$ for the indices of subgroups. If (x_1, x_2, \dots, x_h) represents a node at level i , this is indicated by setting $x_{i+1} = x_{i+2} = \dots = x_h = 0$.

In summary, if a subgroup S requires to change its subgroup key because a member joins or leaves the subgroup, the GC will check whether there is an available *index*. There are several cases:

- 1 If an available *index* exists, the GC changes the *index* of S and recomputes the new key. At the same time, the GC changes the *indices* of the children and descendants of S and recomputes the keys.
- 2 Otherwise, the GC checks the parent P of S to see whether there is an available *index* for P . If there exists an available *index*, the GC first changes the *index* of P and recomputes P 's key; then the GC changes all the *indices* of the children and descendants of P , including S , and recomputes the new keys.
- 3 Otherwise, the GC continues to check the parent of P, \dots , until the GC finds one of the ancestors of S which has an available *index*. From there, the GC changes all descendant *indices* and recomputes the keys.
- 4 In the worst case, the GC needs to trace all the way to the root and change the root key k , thus to change all subgroup-keys.

For security considerations, the GC will change the root key k and all subgroup keys periodically. Doing this will also increase the availability of digit values as *indices*. For example, using radix $R = 256$, assume that three sibling subgroups are still $SG21(k_{21})$, $SG22(k_{22})$, and $SG23(k_{23})$. If there have already been 250 key change requests from these three subgroups, there are $255 - 250 = 5$ *indices* available for new key changes without tracing up to their parent. If the GC changes the root subgroup-key k , and thus all subgroup-keys at this time, then the available number of *indices* for new key changes will go back to $255 - 3 = 252$.

5.4.3 Performance Analysis

We consider the time and space complexity of operations for the GC, subgroup controllers, and other members separately.

For any member in the group, the space complexity is $O(1)$. Since the *indices* indicate the structure of the hierarchy, a member does not need to store the structure of the hierarchy. So, the only thing any member needs to store is its subgroup key and its *index*. As to time complexity, sending a message has no overhead except for encrypting the message. When a member receives a message coming from one of its descendant subgroups which is ℓ levels below it, the member will take ℓ steps to compute its descendant's subgroup-key. Therefore, the time complexity is $O(L)$ where L is the maximum value of levels in the hierarchy.

The GC will store the entire hierarchy and thus, the space complexity is $O(N)$ where N is the number of subgroups in the hierarchy. When a key for subgroup v is changed, the *indices* and the keys of the descendant subgroups as well as itself need to be changed. The change of *indices* is very easy, i.e., for every subgroup in question, the GC just finds the next available *index* and assigns the *index* to the subgroup. So the time complexity is dependent on the computation of the keys. Suppose the subgroup has n descendant subgroups, the GC will take $n + 1$ steps to compute the subgroup-key and all descendant keys. In the worst case, a change to a subgroup-key will cause the change of the root subgroup-key, thus causing the change of all subgroup-keys. Therefore, the time complexity for the GC is $O(N)$.

Subgroup controllers will manage their own subgroups, and the complexities depend on the underlying secure group communication scheme. Suppose the subgroup controllers use a *key-tree* scheme [Molva and Pannetrat, 1999, Noubir, 1998, Wong et al., 1998] (also see Section 3.1.1 of Chapter 3), then the space complexity is $O(L)$ where L is the maximum possible number of members in a subgroup. The time complexity will be $O(\log L)$.

There is another kind of complexity called communication complexity, i.e., the cost of transmitting rekeying messages through the network. The GC needs to send the keys to the subgroup controllers, and the subgroup controllers will

send the keys to the respective subgroup members through the network. The communication complexity between the GC and all subgroup controllers is $O(N)$, and $O(\log(L))$ between a subgroup controller and its members.

5.4.4 Security Issues

A necessary condition for the security of an IHACS scheme is that F be a one-way function. However, this is not a sufficient condition to guarantee the security of the scheme. The following example explains the reason. Let us consider function $F(x, c) = x^c \bmod N$, where $N = pq$ and p , and q are large primes. Given x and c , *modular exponentiation* (also called *square-multiply*) [Menezes et al., 1996, Stinson, 1995] will efficiently compute $y = F(x, c) = x^c \bmod N$. However it is difficult to compute x from y and c . Therefore $F(x, c) = x^c \bmod N$ is a one-way function. But this function can not avoid an attack resulting from the collusion of two members from two different subgroups. For example, suppose the root key is k , the keys of subgroups² 2 and 3 are $k_2 = F(k, 2) = k^2$ and $k_3 = F(k, 3) = k^3$. From k_2 and k_3 , k can be computed as follows: $k_2^{-1}k_3 = k^{-2}k^3 = k$. Therefore, two members from subgroup 2 and subgroup 3 can collude to compute the root key. This kind of attack is related to a more general type of attack, called a *GCD* attack (Greatest Common Divisor attack) in which information is obtained by computing GCD's of observables.

It is generally accepted that a good cryptosystem can be used to implement a one-way function [Sandhu, 1988]. Suppose $E_k(x)$ is an encryption function where k is the encryption key, we can get a one-way function as follows: $F(x, c) = E_x(c)$. Computing x from $F(x, c)$ and c means to compute the key from a plaintext and its ciphertext, which is called a known plaintext attack [Sandhu, 1988, Stinson, 1995]. Good cryptosystems are expected to be immune to known plaintext attacks, therefore, can be used as one-way functions. Besides, in order to prevent the collusion of members from different subgroups, there are some other requirements that a cryptosystem needs or is supposed to have when it is used as a one-way function. We discuss these requirements bellow.

- 1 Siblings collude to attack their parent and/or ancestors. Suppose siblings³ i_1, \dots, i_n come together to try to attack their parent. This means that they know $\{(E_k(i_1), i_1), \dots, (E_k(i_n), i_n)\}$ and try to compute the encryption key k , which is their parent's key. This is still a known plaintext attack, from which a good cryptosystem is supposed to be immune. Moreover, it

²Suppose there is no subgroup labeled as "1" since $k^1 = k$.

³Suppose their parent has *index* I , then they have *indices* Ii_1, \dots, Ii_n . For simplicity, we omit the prefix I from their *indices* and just use i_1, \dots, i_n whenever their meaning is clear from the context.

is reasonable to assume that it is more difficult for siblings to attack their grandparent or other ancestors than to attack their parent. Therefore, such a collusion is unsuccessful.

- 2 Siblings collude to attack another sibling. Suppose siblings i_1, \dots, i_{m-1} want to attack their sibling i_m . This means to compute $E_k(i_m)$ from $\{(E_k(i_1), i_1), \dots, (E_k(i_{m-1}), i_{m-1})\}$ and i_m . From the point above, they can not compute $E_k(i_m)$ by first computing their parent's k and then computing $E_k(i_m)$ from k and i_m . If we assume that $E_k(i)$ and $E_k(j)$ are independent for any i and j ($i \neq j$), then computing $E_k(i_m)$ from $\{(E_k(i_1), i_1), \dots, (E_k(i_{m-1}), i_{m-1})\}$ and i_m is computationally infeasible. On the other hand, a good cryptosystem is a random number generator [Magliveras et al., 1985, Menezes et al., 1996], which means that $E_k(1), E_k(2), \dots, E_k(n), \dots$ are a sequence of random numbers and can be considered to be independent of each other. Therefore, such a collusion is also unsuccessful.
- 3 Suppose i and j are siblings, and i has a child ij (in fact with *index* ij), and j has a child ji (in fact with *index* ji). It is required that $E_{E_k(i)}(j) \neq E_{E_k(j)}(i)$ and that $E_{E_k(i)}(j)$ and $E_{E_k(j)}(i)$ be independent. Again, we can assume that a good cryptosystem satisfies this requirement.
- 4 In order to prevent all other kinds of collusions, we need to assume that for any $k \neq k', E_k(i)$ and $E_{k'}(j)$ are independent for any i and j . It is reasonable to assume that a good cryptosystem satisfies this requirement because for two different keys k and k' , two cryptosystems $E_k(x)$ along with its corresponding decryption function $D_k(x)$, and $E_{k'}(x)$ along with its corresponding $D_{k'}(x)$ can be used as two independent systems. Under this assumption, it is easy to show that all other kinds of collusions are unsuccessful. We describe an example below.

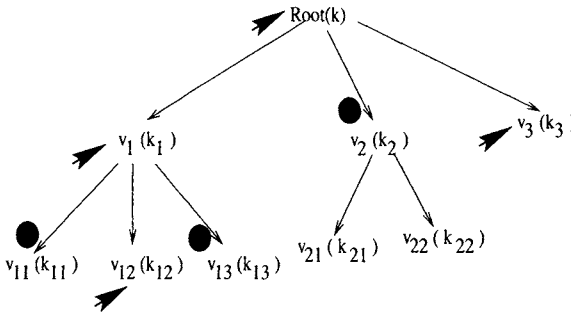


Figure 5.11. Members from subgroup 2, 11, and 13 collude.

Suppose the root subgroup-key is k and the root has three children 1, 2, and 3 (see Figure 5.11) which have their own children. Assume three members from the nodes in $S = \{2, 11, 13\}$ collude to try to compute other keys. They know $\{E_k(2), E_{E_k(1)}(1), E_{E_k(1)}(3)\}$ and all the *indices*. First, suppose by way of contradiction, that members from nodes in S can jointly compute the root key k . Since $E_{E_k(1)}(1)$ and $E_{E_k(1)}(3)$ can be computed from $E_k(1)$, we would then have that the root key k can be computed from $\{E_k(1), E_k(2)\}$. This however contradicts a fact we have already proved that a parent key can not be deduced from its children keys. Therefore, members of S can not collude to compute the root key. Secondly, they can not compute $E_k(1)$. For, according to the assumption in the above case 2, $E_k(1)$ is independent of $E_k(2)$. It is generally correct to assume that $k \neq E_k(1)$. Thus $E_k(1)$ is independent of both $E_{E_k(1)}(1)$, and $E_{E_k(1)}(3)$ by the previous assumption. As a result, $E_k(1)$ is independent of $E_k(2)$, $E_{E_k(1)}(1)$ and $E_{E_k(1)}(3)$. So $E_k(1)$ can not be computed by only knowing $E_k(2)$, $E_{E_k(1)}(1)$ and $E_{E_k(1)}(3)$. Thirdly, members from nodes in S can not compute $E_{E_k(1)}(2)$ because $E_{E_k(1)}(2)$ is independent of all of $\{E_k(2), E_{E_k(1)}(1), E_{E_k(1)}(3)\}$. Fourth, members from nodes in S can not compute $E_k(3)$. For, if they could, this would mean that $E_k(3)$ can also be computed from $E_k(1)$ and $E_k(2)$, which is a contradiction to the result in case 2 where siblings can not collude to deduce the key of another sibling.

Above all, the one-way property alone is not enough to guarantee the security of the scheme. It is also required that several assumptions discussed above be satisfied to prevent various kinds of collusions. It is believed that a good cryptosystem would satisfy these assumptions and can be used. For example, the encryption function $E_{\alpha,\beta}$ in PGM (Permutation Group Mapping) [Magliveras, 1986] can be selected as the function $F(x, c)$. The only extension to the PGM secret-key cryptosystem suggested is to select α to be a wild logarithmic signature. Here $E_{\alpha,\beta}$ is made public and $F(x, c) = E_{\alpha,\beta}(x + c)$. Because PGM has an excellent performance as a random number generator [Magliveras and Memon, 1992, Magliveras et al., 1985], it is difficult to guess $E_{\alpha,\beta}(x + 1)$ from $E_{\alpha,\beta}(x)$. Therefore, using $E_{\alpha,\beta}$ as $F(x, c)$ can guarantee the security of the scheme.

5.4.5 Properties

From the above discussion, we can observe that the IHACS scheme has some special properties:

- The structure of the hierarchy and the keys of subgroups are all dependent on the *indices* of subgroups. Since the *indices* are easy to change and manage, the management of the structure and the generation and modification of keys can be efficiently performed.

- Every subgroup only needs minimum space to store the necessary information, i.e. its subgroup key.
- Both *subgroup dynamics* and *member dynamics* can be efficiently performed by assigning new *indices* to subgroups.
- Under certain independence assumptions, the scheme is computationally secure and can prevent various kinds of collusions.

It is worth pointing out that from a theoretical point of view, it may be possible that two subgroups are assigned the same key. One solution is that whenever the GC computes a subgroup key, it compares the key with all other subgroup-keys. If there is a coincidence, the GC selects a new root key. However, the probability of generating the same key for two subgroups is extremely small and can be ignored in practice.

5.5 Chinese Remainder Theorem Based Scheme for SGC with HAC

In the previous schemes, it was generally assumed that the hierarchy is known by all participants. In this section, we present a scheme which has the property of hiding the hierarchy. The scheme is based on the *Chinese Remainder Theorem* (CRT) and is called the *Chinese Remainder Theorem Based Hierarchical Access Control Scheme* (CRTHACS) [Zou et al., 2001]. This scheme can be applied to any general hierarchy. We present the CRTHACS scheme in this section.

The Chinese Remainder Theorem is central to the scheme, its statement, Theorem 1.7, can be found in Chapter 1. Apart from the CRTHACS, a second CRT based scheme can be found in section 4.3 of Chapter 4.

5.5.1 CRTHACS Components and Initialization

There is a Group Controller (GC) in CRTHACS. The entire group is divided into subgroups, and the subgroups are located at different nodes of a hierarchy. Every subgroup has a subgroup controller which is responsible for managing all members in its subgroup and communicating with the GC. We do not consider here how subgroup controllers manage their subgroups, however, we remark that any group key management protocol such as the key tree scheme [Noubir, 1998],[Wong et al., 1998] can be used (also see Section 3.1.1 of Chapter 3). We denote the subgroups in the system by G_1, G_2, \dots, G_m . For simplicity, we also use G_1, G_2, \dots, G_m to denote the subgroup controllers. The ancestors of subgroup G_i are denoted by G_{i_1}, \dots, G_{i_k} .

The GC has a pair (P_{GC}, S_{GC}) of keys public and private respectively with P_{GC} being made public. The GC performs the following tasks. (i) It maintains the entire structure of the group; (ii) It generates a random set of pair-

wise relatively prime numbers $N_0, N_1, N_2, \dots, N_m$; (iii) It makes N_0 public and sends N_i to subgroup controller G_i securely, i.e., N_i is encrypted by G_i 's public key P_i ; (iv) It computes CRT_i (see Equation (5.1)) using the CRT algorithm and sends CRT_i to subgroup controller G_i securely.

Every subgroup controller G_i is associated with the following five elements $(P_i, S_i, k_i, N_i, CRT_i)$, where P_i, S_i and k_i are generated by subgroup controller G_i , whereas N_i and CRT_i are generated by the GC. P_i is the public key of subgroup controller G_i and is made public. However the other four elements are kept secret. S_i is the private key of subgroup controller G_i corresponding to P_i . P_i and S_i are used to encrypt and decrypt the other three elements. k_i is the data key of subgroup G_i and is used to encrypt data messages by participants in G_i . N_i is the positive integer received from the GC and will be used by G_i 's participants in obtaining the data key k_j of G_i 's descendant subgroup G_j . CRT_i , a positive integer, is called a CRT key and is computed by the GC from k_i using the CRT algorithm (see Equation (5.1)). All ancestral subgroups of G_i can use CRT_i to compute key k_i using the CRT algorithm too.

Every participant j in a subgroup G_i has five elements $(p_j, s_j, k_i, N_i, CRT_i)$ where (p_j, s_j) are j 's public and private keys. Participant j receives k_i, N_i , and CRT_i from its subgroup controller G_i securely.

By convention, in what follows, $E_k(x)$ denotes public key encryption or signature under key k and $\{x\}_k$ secret key encryption under key k ⁴.

The GC and subgroup controllers collaborate to compute the CRT keys as follows. Every subgroup controller G_i selects its own subgroup data key k_i . After signing and encrypting the key⁵ G_i sends $E_{P_{GC}}(E_{S_i}(k_i))$ to the GC.

The GC decrypts the key k_i , determines all the ancestors $G_{i_1}, G_{i_2}, \dots, G_{i_k}$ of G_i and thus, all data keys k_{i_j} and CRT numbers N_{i_j} of these ancestors ($1 \leq j \leq k$). The GC establishes the system of $k + 1$ congruences (5.1) and then computes CRT_i using the CRT algorithm. Each of the first k congruences is for one ancestral subgroup and thus, the ancestral subgroup can compute k_i from CRT_i directly. The last congruence is for verifying the key k_i and determining whether the CRT_i was modified during transmission.

$$\begin{aligned}
 CRT_i &\equiv \{k_i\}_{k_{i_1}} \pmod{N_{i_1}} \\
 CRT_i &\equiv \{k_i\}_{k_{i_2}} \pmod{N_{i_2}} \\
 &\vdots \\
 CRT_i &\equiv \{k_i\}_{k_{i_k}} \pmod{N_{i_k}} \\
 CRT_i &\equiv \{k_i\}_{k_i} \pmod{N_0}
 \end{aligned} \tag{5.1}$$

⁴For simplicity, when E is used on a private key, the result represents a signature.

⁵In order to verify the signature, the verification information should be included in this message. We omit it for simplicity.

Then the GC signs and encrypts (N_i, CRT_i) , and sends the result $E_{P_i}(E_{S_{GC}}(N_i, CRT_i))$ to subgroup controller G_i . The subgroup controller G_i decrypts them and then sends (multicasts) them along with k_i to its subgroup securely.

Remarks: The quantity CRT_i contains the information of k_{i_j} and N_{i_j} of all ancestral subgroups of G_i . The subgroup controller G_i does not know who its ancestors are. However, any ancestral subgroup can compute G_i 's key k_i directly from CRT_i . As a result, hierarchical access control is achieved with the hierarchy being hidden. It is worth mentioning that a subgroup S will know which participants are in its descendant subgroups but S cannot figure out how far down any of these descendant subgroups is.

5.5.2 Data Communication

Whenever participant $l \in G_i$, with identity ID_l , sends a message M , it encrypts the message, computes the *MAC* (Message Authentication Code), signs the *MAC*, and then sends $(ID_l, CRT_i, Signed_MAC, \{M\}_{k_i})$, where $Signed_MAC = E_{s_l}(MAC_{k_i}(\{M\}_{k_i}))$.

Assume as above that sender $l \in G_i$. When a receiver m receives $(ID_l, CRT_i, Signed_MAC, \{M\}_{k_i})$, it proceeds as follows:

- 1) If $m \in G_i$, then m has the same subgroup key k_i as the participant l . If $m \in G_j$ where G_j is an ancestral subgroup of G_i , m first computes $t = CRT_i \bmod N_j$ (i.e., $\{k_i\}_{k_j}$) and decrypts t to get k_i ,
- 2) m computes $x = \{k_i\}_{k_i}$ and $y = CRT_i \bmod N_0$,
- 3) m compares x and y :
 - 3.1) If $x \neq y$, the verification of the key fails. In this case there are two possibilities: i) the CRT_i value was modified during transmission or ii) the receiver is not in the sender's subgroup or any of the ancestral subgroups. In either case the receiver discards the message and stops.
 - 3.2) Otherwise, $x = y$, in which case, the key is correct and the message is intended for m ,
- 4) m decrypts the *Signed_MAC* using l 's public key to get $MAC_{k_i}(\{M\}_{k_i}) = E_{p_l}^{-1}(Signed_MAC)$, where E^{-1} stands for the decryption algorithm corresponding to E ,
- 5) m computes $MAC_{k_i}(\{M\}_{k_i})$ using k_i which already passed the verification in (3),

- 6) m compares the above two MAC s. If the two MAC s are equal, then both the sender and the message are authenticated. The receiver decrypts the message using k_i . Otherwise, the message was modified during transmission, and the receiver discards the message.

5.5.3 Dynamic Key Management

In SGC with HAC, there are two levels of dynamics: i) *low level* dynamics by which we mean that a member may join/leave a subgroup and which is operated by subgroup controllers and is dependent on the subgroup key management protocol (also called member dynamics), and ii) *high level* dynamics which include the following operations (also called subgroup dynamics): adding/inserting a new subgroup, removing an existing subgroup, merging two subgroups, splitting a subgroup and modifying an existing subgroup key, all of which are easily done in CRTHACS. For example, when a new subgroup G_i is added into the hierarchy, the GC computes G_i 's CRT_i by equation (1) and sends CRT_i and N_i to G_i . If G_i has descendant subgroups (i.e., G_i is inserted into the hierarchy), the GC also needs to recompute the CRT values for all descendant subgroups of G_i so that these CRT include the information of G_i 's data key k_i and the corresponding N_i . All other subgroups are not affected.

5.5.4 Security and Performance Analysis

The security of the CRTHACS scheme is based on several factors. The public-key cryptosystem provides a secure channel among prospective parties such as the GC and subgroup controllers, the subgroup controllers and subgroup members and provides for entity authentication such as signatures. The independence among subgroup data keys guarantees that a subgroup is unable to obtain other subgroup keys except that an ancestral subgroup is always able to compute its descendant subgroup keys. The CRT value enforces the hierarchical access control structure while keeping the hierarchy hidden. The subgroup controller utilizes a specific key management protocol to manage its subgroup and achieves secure and efficient multicast in its subgroup. The MAC ensures message integrity. One good security property of the CRTHACS scheme is that it will defeat GCD attacks, a typical threat to CRT-based systems.

As for the performance of the CRTHACS scheme, there are three complexities to be considered: *space*, *time*, and *communication* complexity. By *communication* complexity we mean the size of key-related materials, including the CRT parameters, communicated between the GC and the subgroup controllers, or between subgroup controllers and subgroup members. There are three classes of entities: The Group Controller (GC), Subgroup Controllers (G_i) and participants (p_j). The complexities are summarized in Table 5.1.

Table 5.1. Performance characteristics in CRTHACS

	Space ¹	Time ²	Communication ³
GC	$O(mHL)$	$O(mM(HL)\log(H)) + O(mHM(L)\log(L))$	$O(HL)$ (GC and G_i/p_j)
G_i	$O(HL)$	Independent of m and H	
p_j	$O(HL)$	$O(M(2L)) + O(2M(L)\log(L))$	

Note: H : the maximum number of ancestors a subgroup may have; L : the length of a large integer in bits; m : the number of subgroups; $M(n)$: the time to multiply two n -bit integers in bit operations

¹ counts the space for representing P_i , S_i , k_i , N_i , and CRT_i , which require large integers, possibly 1024-bit integers but ignores the space for representing the access control structure or membership, which need small integers.

² counts the complexity of the CRT algorithm, i.e.,

$O(M(kL)\log(k)) + O(kM(L)\log(L))$ [Chiou and W.T.Chen, 1989],[Stinson, 1995], where k is the number of moduli. However, the time, which is consumed on key generation, encryption and decryption and is dependent on the special algorithms selected, is ignored.

³ the key materials between subgroup controllers and subgroup members depend on the subgroup key management protocol selected and are ignored here.

From the above presentation, it can be seen that the CRTHACS scheme possesses the following valuable properties: (i) correctly enforces hierarchical access control; (ii) hides the hierarchy and receivers, (iii) authenticates both messages and the origins of the messages (i.e., senders).

Chapter 6

SGC CHALLENGES AND SGC FOR WIRELESS NETWORKS

In this chapter, we highlight some critical factors, other than group key management, which are required to make SGC work in the real world. We then discuss some challenging problems for SGC and introduce a few solutions when SGC is used in wireless environments.

6.1 Factors Enabling SGC Functionality

Apart from efficient key management in SGC to provide dynamic joining and leaving as well as efficient bursty operations, there are other problems which need to be solved so that a practical secure group communication system can be implemented. Moreover, there are certain factors which can strengthen the security of group communications. In what follows, we describe these challenging issues for future research.

6.1.1 Admission Control and Membership Management

Group key management alone is all but useless. *Admission control* and group *membership management* are indispensable for SGC. These factors have been overlooked in the past.

Admission control is the process by means of which a user who requests to join a group is allowed to join or refused membership. There are many issues regarding admission control.

Who performs the task of admission control? There are several possible ways to do this: (i) A central authority could perform the task on-line (*centralized admission control*), (ii) Certain or all members of a group perform the task jointly (*distributed admission control*), or (iii) A trusted entity, such as the group initiator, performs the task (off-line).

What is admission control policy? Admission control policy is application-dependent and there is no single policy relevant to all applications. Thus, the question becomes: *what is a representation mechanism which can specify different policies in a uniform way.* A simple way to represent admission control policy is to use an *access control list (ACL)*. A *grant-ACL* lists the users who are allowed to join a group and a *deny-ACL* lists the users who are denied group admission.

A further question is: *How to perform the admission control efficiently.* In particular, for distributed admission control, how do members conduct votes to determine whether a user is allowed to join.

By *membership management*, we mean how and who keeps the current state of group members and change-sequences of the group. As before, (i) a central authority can keep track of group changes, or (ii) every group member keeps (the same) image of group membership, or (iii) each member remembers part of the group and certain subsets of members can reconstruct the entire current state of the group. Another issue is how to evict a member in case of member misbehavior.

In many systems, especially distributed key agreement protocols, there is a need for the members to be enumerated in some order. Methods of enumerating or ordering members still need to be devised. Numbering members is neither efficient nor flexible. Is there a protocol that does away with the need for member numbering? Moreover, how should we integrate admission control management, membership management and key management ?

[Kim et al., 2002], first attempted to address and provide a framework for the admission control problem in SGC. In this scheme, admission control has three components: (i) An off-line central authority (CA), (ii) A group charter which is a document codifying admission policy, and (iii) A group authority (GAUTH), which can certify (or vouch for) group admission, and group members. Admission control consists of three steps: (1) The CA sets up the group charter and delivers it to GAUTH (via an off-line channel), (2) A user obtains the group charter from group members when the user asks to join the group, and (3) The user sends its group charter to GAUTH. After GAUTH verifies the user, the GAUTH issues a Group Membership Certificate (GMC) to the user, and the user joins the group.

6.1.2 Message/Packet Source Authentication

In group communications, we can distinguish three types of authentication: (i) user authentication, (ii) message authentication, and (iii) sender authentication. User authentication is a process to verify a user's credentials before permitting him/her to join the group. This is in fact what we discussed in the previous section, *admission control*. Message authentication is a process to verify that a message received is indeed what was originally sent. If there

are any alterations whatsoever to the original message during transmission, the event should be detected and the received message should be flagged as not authentic. Message authentication can be implemented by means of a cryptographic hash function and/or Message Authentication Code (MAC). For increased security, it is recommended that the key used for encrypting/decrypting group messages and the key used for authenticating group messages be different. Sender authentication is a process to verify the origin of messages. Incorporating sender authentication is important for preventing malicious behavior of outsiders or rogue group members. Without the enforcement of sender authentication the SGC mechanism may collapse. Sender authentication is also called *Message/Packet Source Authentication* (MSA) [Moyer et al., 1999]. MSA schemes can be classified into three levels [Moyer et al., 1999], according to increased security strength – (i) A registered receiver can verify that a packet was sent by a registered group member (either a registered sender or a registered receiver), (ii) A registered receiver can verify that a packet was sent by a registered sender, and (iii) Registered receivers can precisely identify the registered sender who sent each received packet. Ways to perform these MSAs in an efficient manner need to be studied further.

Apart from MSA, the criterion of *non-repudiation* is an important security requirement in some SGC applications, especially in hostile SGC environments where the group members may belong to hostile communities. Studying techniques, designing protocols and implementing APIs for non-repudiation in SGC are brand-new challenging tasks facing researchers in this area.

6.1.3 Coordination: Timing and Versioning in SGC

Most SGC key management protocols assume implicitly that group members are synchronized in time and that the transmission of data and rekeying messages is reliable. In particular, some distributed group key agreement protocols work in rounds and group members co-execute protocols in synchrony, side-by-side in each round. Based on these assumptions, the effect is that all group members will obtain the same group key(s) at specific time points and communicate using the same group key(s).

In the real world however, network conditions vary widely and group members may be located in geographically distant areas. There are variable delays (time differences) among different group members. The rekeying messages may get lost. A member may be holding an old key because the new key has not reached the member yet or is lost, so the member encrypts messages with this old key. In order to solve problems arising from the loss of rekeying messages, asynchrony of members, and mis-match of group keys, there must be some mechanism to coordinate the actions among all group members.

One way to solve the lost rekeying message problem is for a member to request rekeying materials from the group controller or from other group mem-

bers, if he has not received rekeying messages for a certain period of time. As an alternative, the group controller could periodically broadcast rekeying messages. To solve mis-match of group keys, *versioning* of keys is needed. Every new group key is given a new version tag. Group members keep several keys of different versions. A member always encrypts the message with its most recent key (which may not be the newest key) and includes the version of the key in the encrypted messages. The receivers can decrypt the messages with the correct version key. Above all, efficient and scalable coordination mechanisms need to be studied and integrated into rekeying and data transmission.

6.1.4 Broadcast Authentication

Broadcast authentication is a process in which the sender of a broadcast message can be verified by all receivers of the message. Broadcast authentication is specific to broadcast applications where there is one sender (information center or distributor) and multiple receivers (subscribers). The importance of broadcast authentication is that it can prevent a malicious member from impersonating a bona fide sender.

Two new broadcast authentication schemes, called TESLA (Timed Efficient Stream Loss-tolerant Authentication) [Perrig et al., 2001a, Perrig and Tygar, 2002] and BiBa (Bin Ball) [Perrig, 2001, Perrig and Tygar, 2002] have been proposed recently.

Broadcast authentication can be considered as a specific case of SGC message/source authentication. In the other words, SGC message/source authentication can be achieved by implementing any broadcast authentication scheme at each group member because he/she is a sender of group (broadcast) messages. However, although the sender is much more powerful than the receivers in the TESLA and BiBa schemes, in the general SGC case, all members are assumed to have equal capabilities except for the Group Controller (if any). New MSA-capable schemes for SGC need to be investigated.

6.2 SGC in Wireless Environments

Wireless networks are being widely deployed and utilized in today's world. More and more people have mobile devices and communicate with their peers when they move around. Imagine two scenarios. In the first, a group of friends come to a convention place and they want to form an ad hoc network on the fly to communicate among themselves. The ad hoc network consists of their mobile devices and (maybe) other people's mobile devices. Consider another scenario in which a few people are traveling separately to different places. They want to communicate using their mobile devices via the underlying wireless/wired infrastructures (such as cellular communication systems). The important issue of concern in any of the above situations is whether their commu-

nications can be held *private*, comprehensible only by the specified members and no other individuals. These are typical SGC scenarios in wireless/mobile environments.

Wireless networks are more susceptible to intrusion and unauthorized access because any intruder can tune his/her wireless device to a specific radio frequency to capture the information transmitted on the frequency without being detected. Furthermore, compared to desktop computers, mobile devices are less powerful and energy-limited. As a result, the SGC techniques for wired networks are difficult to adopt to wireless environments because the former techniques require time-consuming and complicated computations. SGC for wireless environments is a very challenging task.

Because wireless/mobile devices possess less computational power (less memory, low-power processor, limited battery capacity etc.), it is worthwhile considering so-called “lightweight” SGC key management. The intuitive solution is to use symmetric key systems instead of asymmetric ones because symmetric key systems are much faster. However, key management protocols can be built on asymmetric key systems with a reduction in the size of cryptographic keys and other parameters, provided that the adversary is also limited to a mobile device of comparable computational power as the ordinary users.

The basic approach for SGC in wireless environments is to match the key management protocols with the structure of wireless networks. In a wireless LAN with an access point (base station), the access point is a central trusted authority and has more computational power, so a centralized key management protocol may be used. In an ad hoc network, a distributed key agreement protocol may be more appropriate. In a cellular based wireless network, mobile stations in a cell are considered as one level and the base stations in cells are considered as a second level. Therefore two level hierarchical key management protocols may be deployed. Examples of hierarchical key management matching hierarchical structures of wireless networks are found in [DeCleene et al., 2001, Sun et al., 2002]. According to [NIST, 2003], a wireless network can be classified as a wireless WAN, a wireless LAN, a wireless PAN (Personal Area Network), or a combination of them. Accordingly, the SGC key management protocols should be studied and deployed based on the structures of the different wireless networks. In a similar way, this strategy is adopted in two-party secure communications for ad hoc networks and cellular based networks [Zhou and Haas, 1999].

In the following sections, we discuss two key management protocols. The first is for cellular based wireless networks and the second for administrative-domain-based hierarchical group key management.

6.2.1 Topology Matching Key Management for SGC in Cellular Networks

In [Sun et al., 2002], a group key management scheme for cellular networks is proposed¹. The basic idea is to adapt the traditional key tree (i.e., LKH) to cellular networks and to match the key tree to a two level topological structure. Thus the protocol is called *Topology Matching Key Management (TMKM)*. In contrast, the traditional key tree protocol, which is independent of the network structure, is called *Topology Independent Key Management (TIKM)*.

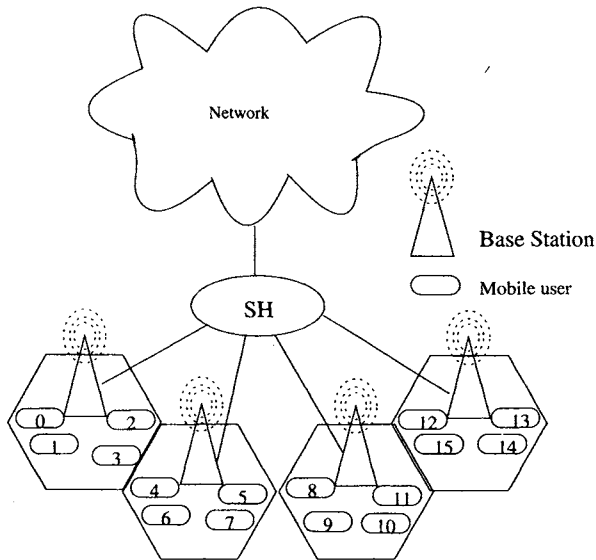


Figure 6.1. A cellular network model (Source: [Sun et al., 2002], © IEEE).

Suppose the cellular network consists of *mobile users*, *base stations (BS)* (access points), and a *supervisor host (SH)* (see Figure 6.1). The SH, being part of the wired network, performs the following tasks: handles most of the routing and protocol details for mobile users, controls the base stations, and manages the keys necessary to protect group communication. Each BS is responsible for multicasting messages to users within its scope; these users can all be viewed as a subgroup. It is assumed that a BS knows whether a rekeying message is

¹Portions reprinted, with permission of the authors and the publisher, from Y. Sun et al., *An efficient key management scheme for secure wireless multicast* in Proceedings of IEEE International Conference on Communications, Vol. 2, 2002, pages: 1236–1240. © 2002 IEEE.

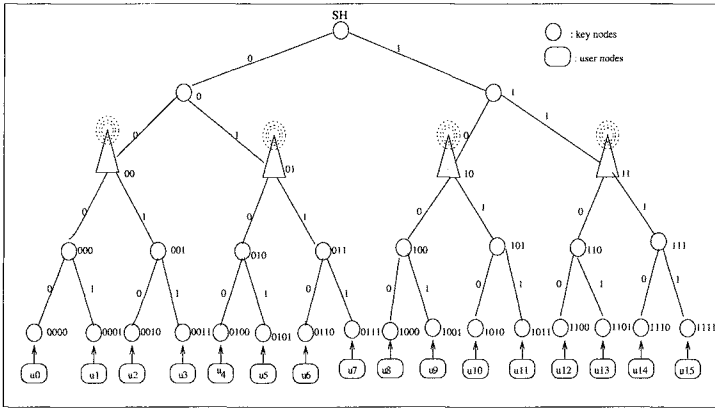


Figure 6.2. Topology matching key tree (Source: [Sun et al., 2002], © IEEE).

useful to its subgroup. A rekeying message is distributed in two steps: (i) it is multicast to all BSs by the SH through the wired networks, and (ii) each receiving BS broadcasts the rekeying message to its subgroup, if it finds it useful to the subgroup.

Based on the above two-level hierarchy of the cellular network, the key-tree based key management also consists of two levels: multiple *user-subtrees* at the lower level (one for each BS), and one *BS-subtree* at the upper level. Each user-subtree is configured by a BS and its users and is used by the BS to broadcast rekeying messages to its subgroup. The BS-subtree is designed by the SH and the BS's and is used by the SH to multicast rekeying messages to all BS's. The two-level subtrees are combined to form a complete key tree which is used to perform the key management among the SH and all mobile users. Figure 6.2 shows a key management tree which matches the hierarchical cellular network of Figure 6.1.

6.2.1.1 Key Management for TMKM

Rekeying occurs in three situations: i) the arrival of a new member, ii) the departure of an existing member, and iii) the movement of a member from one cell to another. The first two situations result in normal rekeying operations as a member joins and leaves. We discuss the rekeying operations caused by the third situation in this section.

In a cellular network, when a user moves from one cell to another, a *hand-off* process takes place. During this process, the user stops receiving messages from the old cell and begins to receive messages in the new cell. In TMKM the hand-off involves not only data flow, but also the flow of rekeying messages

since TMKM is network topology-dependent. In TMKM when a user u moves from one cell c to another cell c' , it is necessary to move u from one branch of the key-tree to another. Thus, a naive solution for a *hand-off* operation in TMKM is to process it as a leave from c and a join to c' as follows:

1. perform a regular member leave operation, i.e., remove u from the subtree of the BS c and update the keys for user u .
2. perform a regular member join operation, i.e., choose a non-occupied leaf of the subtree under BS c' , place user u there and update the keys from the leaf to the root.

This solution is clearly not efficient as frequent hand-offs in the cellular network will cause high communication overhead for the rekeying operation. An efficient solution is to allow a user to have keys for multiple cells as the user moves from one cell to another, and update all these keys when the user leaves the multicast service. In order to keep track of user movements, there is a *wait-to-be-removed* (WTBR) list for each BS. The WTBR of a cell records the users (and their locations in the subtree of the cell) who move out of their cell from the time of the last leave operation performed on the key tree. All WTBR lists are stored at the SH. A concrete solution is described as follows.

When user u leaves the multicast service from cell c , the SH searches through all the WTBR lists. Cell c and all the cells whose WTBR lists contain u need to update the corresponding keys. These keys can be updated in bulk, which is more efficient than individually (see Section 3.1.2 of Chapter 3). Then the SH checks c 's WTBR list and finds all the users previously located at the same place as u , and removes them from the WTBR list at no extra cost. Finally the SH checks all other WTBR lists containing u , finds all users located at the same place as u , and removes them (along with u) from the lists at no extra cost.

When user u moves from cell c to c' , the SH puts u in c 's WTBR list. Then the SH finds a non-occupied leaf of the subtree under cell c' , which was most recently updated at time t due to other users' leaving. Suppose the time at which u joined the multicast service is t' . If $t' > t$ (i.e., t' is later than t), then the keys from the leaf to the root need to be updated using a regular join operation;², otherwise no update is needed³.

²The reason for this is that u could find the root key(s) for the time interval $[t, t']$ if u had intercepted and stored the rekeying messages before his join, while the keys from the leaf to the node next to the root are given to u without change. This violates the *backward secrecy* criterion of SGC.

³In paper [Sun et al., 2002], it is stated in the opposite way without explanation or reason, i.e., to update the keys if t' is earlier than t and not update if t' is later than t . This might be a typo.

In order to minimize the rekeying message size when a user joins/leaves, an (a, L, x) -logic tree, called *ALX tree*, is introduced for user-subtrees in addition to the BS-subtree. The ALX tree can maintain the tree structure at all times and be optimized based on the number of users under the following assumptions: 1) Users arrival times satisfy a *Poisson* distribution; 2) The service times, i.e., the periods of time users stay in the multicast service are distributed according to an exponential distribution; and 3) The users join/leave behavior is independent of each other. The (a, L, x) -logic tree is defined as follows. 1) It has $L + 1$ levels; 2) The upper L levels form a fixed subtree with degree a and are fixed during the multicast service; and 3) Users are attached at the $(L + 1)^{th}$ level, which has no fixed degree and changes when users join and/or leave. A vector $x = [x_1, \dots, x_i, \dots, x_{aL}]$ is used to represent this level where x_i is the number of users attached to the i^{th} node at level L . For example, in Figure 6.3, $x = [4, 2, 3, 3, 2, 4, 3, 3, 3]$.

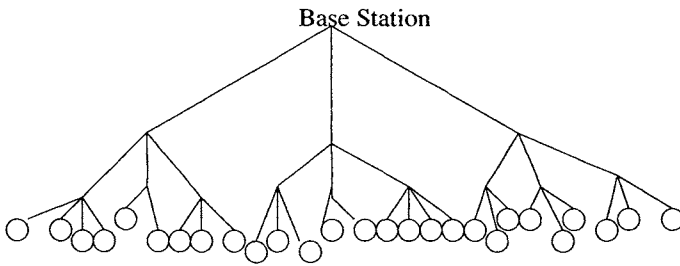


Figure 6.3. ALX tree (Source: [Sun et al., 2002], © IEEE).

6.2.2 Administration-scoped Hierarchy based Key Management for SGC in Wireless Networks

In [DeCleene et al., 2001], group key management for GSC is considered for a highly-mobile wireless networking environment in which command and control nodes move along with individual users⁴. A typical scenario of the environments occurs in modern military action where multinational coalition forces collaborate to perform certain tasks and maintain and administer separate networks⁵. In this scenario, data must be securely multicast from one

⁴Portions reprinted, with permission of the authors and the publisher, from B. DeCleene et al., *Secure group communications for wireless networks* in Proceedings of MILCOM, Vol. 1, 2001, pages: 113–117. © 2001 IEEE.

⁵Networks may be small (such as a fine-grained ad hoc network) or large (such as a satellite broadcast network)

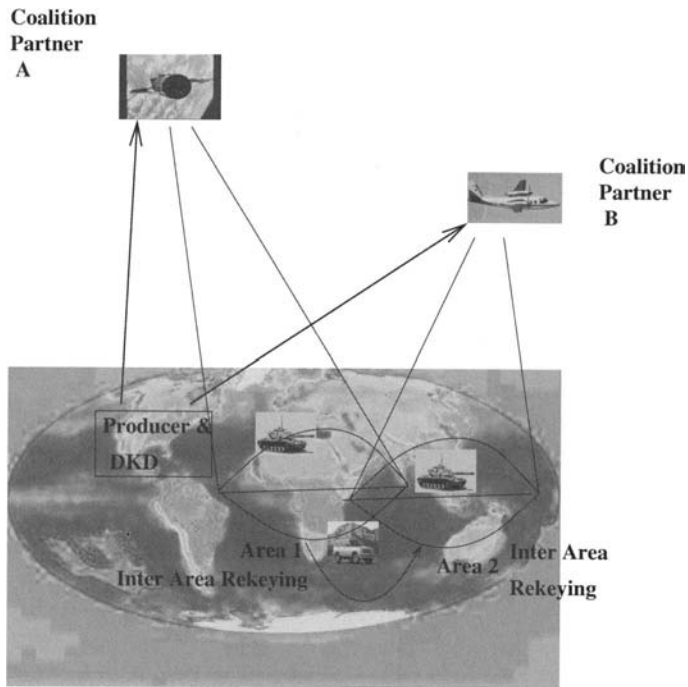


Figure 6.4. Two level rekeying (Source: [DeCleene et al., 2001], © IEEE).

source in one force to (many) users in multiple forces, requiring that users be properly keyed. Key management in this scenario is made complicated by mobility where users not only leave or join a session but also transfer between networks while remaining in the session. Two factors need to be considered for key management in such a system: performance (or network latency) caused by users' join, leave and transfer operations and level of trust caused by users' transfer operations since a mobile user may collect information about the local security services for each of the areas visited.

To increase scalability and match the administrative architecture, the proposed key management protocol adopts a hierarchical methodology: Dividing the key management domain into smaller administratively scoped areas. Throughout the domain, the highest level in the hierarchy, a Domain Key Distributor (DKD) generates and distributes data encryption keys (DEK). In order to guarantee both backward secrecy and forward secrecy, a new DEK must be generated and distributed whenever a new user joins or an existing user leaves a communication session.

The domain is further divided into distinct areas based on administrative scopes. Each area is unique in the sense that there is no need for rekeying when users move within the area and the rekeying cost is considered “reasonable” when a join/leave occurs within the same area. An Area Key Distributor (AKD) is associated with each area and is responsible for distributing DEK to the users within that area.

As a result of the above two-level architecture, the key management protocol consists of two levels (see Figure 6.4): (i) *intra-area rekeying* and (ii) *inter-area rekeying*. The new DEK will be securely broadcast to all AKDs from the DKD via an inter-area rekeying algorithm. AKDs will then broadcast the DEK to the users in their areas via an intra-area rekeying algorithm. Many approaches, such as Public Key Infrastructure (PKI), secure multicast, and Logical Key Hierarchy (LKH), can be chosen for intra-area rekeying. There are four approaches for inter-area rekeying: *baseline rekeying*, *immediate rekeying*, *delayed rekeying* and *periodic rekeying*. These are discussed below.

Baseline Rekeying

In this approach, movement across areas is treated directly as a leave from the old area followed by a join to the new area. Figure 6.5 shows the operations of the approach. When a user u moves from area j to area k , the following actions occur: (1) u notifies AKD_j of his/her leaving, and AKD_j terminates the current data transmission. (2) AKD_j updates its area key following its intra-area key management protocol. (3) A new DEK is generated by DKD and distributed to all areas. (4) At this point, u is excluded from the multicast group and data transmission resumes. Note that the above steps consist of a leave operation without distinguishing between a departure and a movement. Similarly, the following steps consist of a join operation without distinguishing between a join and a movement. (5) u notifies AKD_k of its intent to join area k and AKD_k terminates current data transmission. (6) AKD_k updates its area key, which is also sent to u . At this point, u joins area k successfully. Note this step is to guarantee backward secrecy when a new member joins the multicast group and is purely wasted on movement. (7). A new DEK is generated and distributed to all areas. (8) Data transmission continues.

This approach has the obvious shortcoming that data transmission is unnecessarily interrupted twice during movement among areas.

Immediate Rekeying

This approach utilizes the same operations for joins of new users and departures of existing users as baseline rekeying with extension by adding explicit semantics for transfer between areas. When a user moves between areas, the user notifies the two affected areas. Both areas update their area keys. How-

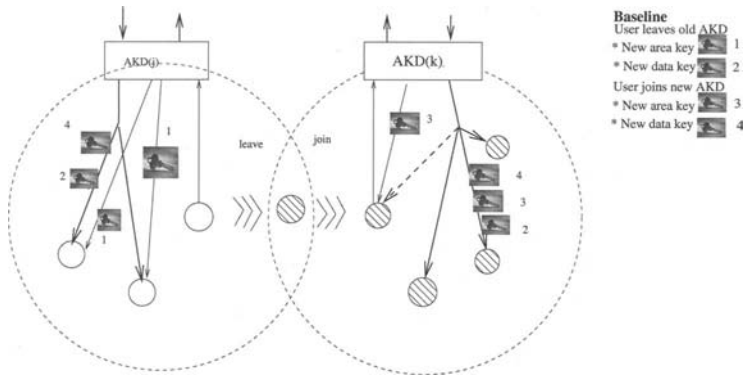


Figure 6.5. Inter-area rekeying: Baseline Rekeying (Source: [DeCleene et al., 2001], © IEEE).

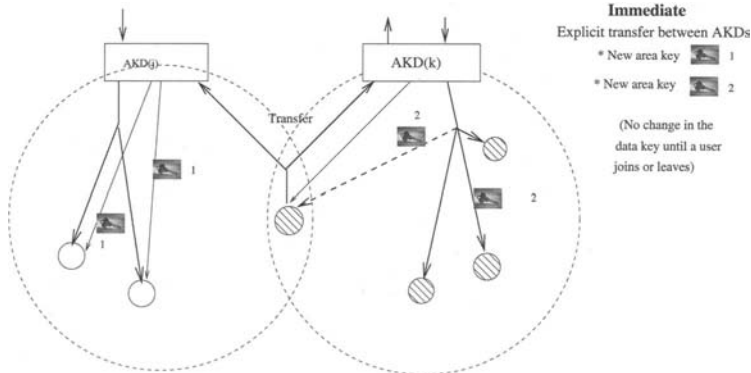


Figure 6.6. Inter-area rekeying: Immediate Rekeying (Source: [DeCleene et al., 2001], © IEEE).

ever, no new DEK is generated and data transmission continues without interruption. Figure 6.6 shows this rekeying approach.

Immediate rekeying, as well as baseline rekeying, rekeys area keys as soon as a member transfers. The disadvantage of this is that a member moving rapidly between areas may cause area keys to be changed repeatedly.

Delayed Rekeying

In delayed rekeying, whenever a movement occurs, local rekeying may not be performed immediately. Instead, rekeying is postponed until a particular

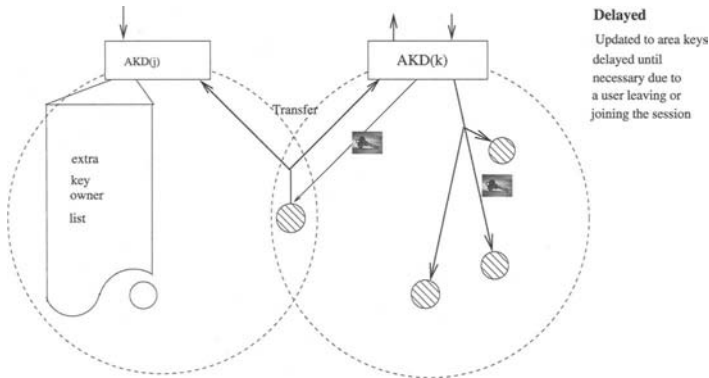


Figure 6.7. Inter-area rekeying: Delayed Rekeying (Source: [DeCleene et al., 2001], © IEEE).

criterion is satisfied. Here we discuss two delayed rekeying methods: (i) *pure delayed rekeying* and (ii) *threshold rekeying*. In pure delayed rekeying, each AKD maintains a list called an Extra Key Owner List (EKOL). The EKOL is similar to WTBR in TMKM (see Section 6.2.1.1) and records the users who have moved out of the area but who still hold valid keys for the area. When a user u moves from area j to area k , AKD_j does not perform local rekeying but instead adds u to its EKOL. However, AKD_k performs its local rekeying. The reason for this is to guarantee backward secrecy, i.e., to prevent a user from falsely transferring into an area to get access to the old keys. Before local rekeying, AKD_k checks its EKOL local rekeying will be conducted if u is on the list, i.e., u returns to this area. Figure 6.7 shows this scenario.

Threshold rekeying extends pure delayed rekeying by performing local rekeying whenever the number of users exceeds a predefined threshold. There are two threshold criteria here: (i) *area threshold* and (ii) *user threshold*. In the first case, an area will be rekeyed when its EKOL has more than a particular number of users. Under the second case, if a user collects more than a predetermined number of area keys, local rekeying (in multiple areas) occurs. This will prevent a user from accumulating all the area keys by visiting all of the areas.

Periodic Rekeying

Periodic rekeying updates area keys periodically. The periodic interval depends on security level and performance. In general, periodic rekeying can be used along with other inter-area rekeying approaches.

6.2.3 Secure Group Communications over Ad Hoc Networks

As indicated above, “*ad hoc networks*” constitute another family of wireless networks, characterized by a dynamic formation of the network, multi-hop routes, no fixed, wired infrastructure, and no central control node. Moreover, the nodes are generally resource constrained, so symmetric key cryptography is preferred. For example, the authors in [Basagni et al., 2001] propose a symmetric key based key management protocol for SGC over an ad hoc network which consists of a large number of *small* nodes. *Small* here means that the physical size of the nodes is small and computational power and resources at each node are limited. In this section, we describe a key management protocol for SGC over an ad hoc network, based on symmetric keys and proposed in [Basagni et al., 2001]. Then, we discuss the desired properties any SGC protocol over ad hoc networks should possess.

6.2.3.1 Securing Ad Hoc Networks with a Large Number of Nodes

The paper in [Basagni et al., 2001] proposes a symmetric key based SGC protocol⁶ for an ad hoc network consisting of a large number of *small* nodes. The nodes in such a network are small in size and their computational power and resources limited, thus they are called *pebbles* [Basagni et al., 2001]. Correspondingly, such an ad hoc network is called a *pebble network* or *pebblenet* for short. Figure 6.8 shows an example of a pebblenet.

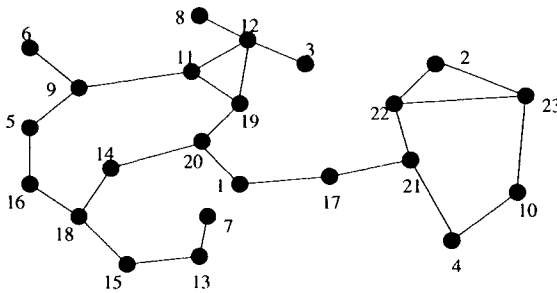


Figure 6.8. A pebblenet (Source: [Basagni et al., 2001], © ACM).

⁶S. Basagni et al., *Secure pebblenets*, Proceedings of ACM Symposium on Mobile Ad Hoc Networking & Computing (MobiHOC), October 2001, pages: 156–163, © 2001 ACM, Inc. Reprinted by permission.

There are several assumptions about a pebblenet: 1) Each node (pebble) has an identity, which uniquely identifies the node. 2) All pebbles are honest, which means that no pebble will behave maliciously. 3) All nodes (i.e., mobile devices) are equipped with *tamper-resistant* capability. This property has two implications: First, “tamper-resistance protects the network from active adversaries who may try to insert malicious nodes in the network after capturing honest ones.” If an adversary tries to gain access to the group key by physically extracting it from a mobile device, the mobile device would self-destruct or “definitely be damaged, thus thwarting the attempt” [Basagni et al., 2001]. Secondly, “tamper-resistance provides for a *safe leave*” operation in the sense that a mobile device which is no longer part of the network will be unable to access any traffic exchanged after the device has left the network. [Basagni et al., 2001]. 4) There is a global key K_{GI} , which is used to identify the group and is called a group identity key. Each pebble is equipped with K_{GI} . The possession of K_{GI} , rather than individual identities of pebbles, guarantees the group membership of pebbles. Such a restriction improves system efficiency since group membership can be verified using a single key for the entire group.

There are five types of keys involved in the protocol: K_{GI} , Traffic Encryption Key (TEK), K_H^i , K_B^i , and K_c^i s. The protocol will be executed periodically or whenever a new node joins the group. Because of assumption 2) above, no action is needed when a node leaves the group. Every execution of the protocol is called a *round* and aims to update TEK . K_{GI} is the unique group identity key described above and is independent of rounds. TEK is used for the encryption of group communications (i.e., data traffic), and is updated and distributed to all pebbles in each round and is denoted by t^i for round i . K_H^i , one for each round, is called a HELLO message encryption key. K_B^i , one for each round, is the backbone key. For each round i , there are multiple K_c^i s, one for each cluster c , called cluster keys. Initially, $K_B^0 = K_B = K_{GI}$. Let f be an appropriately chosen one-way function. Then for each round i , $K_H^i = f(K_B^{i-1})$ and $K_B^i = f(K_H^i)$. If p is any pebble in the network, by the *neighborhood* N_p of p we mean all pebbles in the network that would receive a message from p in a *single hop*.

Each round of the protocol consists of two phases, with two steps each. During the *first phase* the pebblenet is partitioned into *clusters*, and this involves: step 1) the selection of *clusterheads* & the formation of clusters and step 2) superimposing the backbone. The *second phase* is key management, which involves: step 1) the generation of (possibly multiple) keys, and step 2) the distribution of a unique key, agreed upon by all clusterheads. To increase security, all messages in the above steps are encrypted during transmission. These steps are executed as follows.

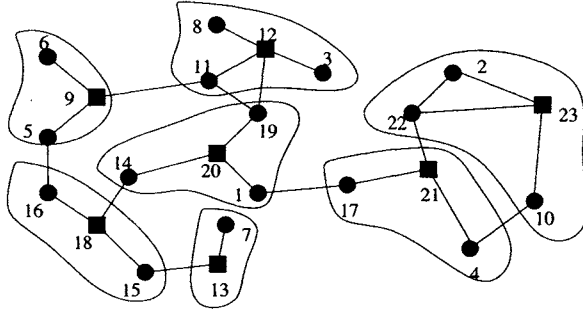


Figure 6.9. Pebble clusters in the above pebblenet (Source: [Basagni et al., 2001], © ACM).

- 1 Phase 1, step 1 Clusterhead selection and cluster formation: All pebbles are randomly assigned dynamic weights and the pebbles with the highest weight among their neighbors are selected as heads. Once a pebble p is selected as head, p along with its neighbors N_p form a cluster (see Figure 6.9). In this step there are two kinds of broadcast messages: *HELLO* and *ROLE*:

$$HELLO : p \Rightarrow N_p : E_{K_H^i}(w_p|ID_p|MAC(K_{GI}, \langle w_p|ID_p \rangle))$$

$$ROLE : p \Rightarrow N_p : E_{K_H^i}(w_p|ID_p|R|MAC(K_{GI}, \langle w_p|ID_p \rangle R))$$

where “ $m \Rightarrow S : M$ ” stands for entity m sending message M to the members of set S . For pebble p , N_p is as earlier defined the set of p 's neighboring pebbles, while w_p and ID_p are p 's weight and identity respectively. $MAC(k, \langle m \rangle)$ is a keyed message authentication code applied to message m , using key k . $E_k(m)$ denotes the encryption of m using k . The symbol $|$ indicates the concatenation of two messages. The value R in the *ROLE* message indicates the potential role ($R = CH$ as cluster head or $R = v$ as ordinary affiliated with v) a pebble p believes it could be.

The *HELLO* messages are used to find neighbors and determine the relative weights of pebbles. The *ROLE* messages are used to determine the pebbles' role (head or ordinary). Both types of messages are encrypted with K_H^i .

The set of clusterheads is denoted as \mathcal{C} . Each clusterhead c in \mathcal{C} can discover its neighboring clusterheads. Moreover each $c \in \mathcal{C}$ generates a cluster key K_c^i for its own cluster and communicates the key to its members M_c via the

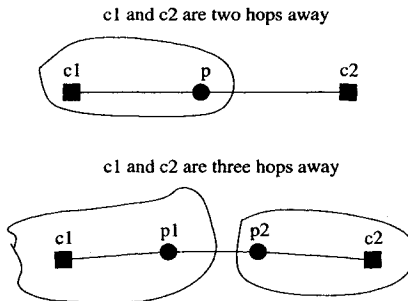


Figure 6.10. Neighboring clusterhead discovery (Source: [Basagni et al., 2001], © ACM).

following messages encrypted by K_H^i :

$$c \Rightarrow M_c : E_{K_H^i}(ID_c | K_c^i | MAC(K_{GI}, \langle ID_c | K_c^i \rangle))$$

The K_c^i s are used in two ways. First, clusterheads use them to encrypt and transmit secure information, such as TEK , to their clusters. Secondly, they are used in the *discovery of neighboring clusterheads*.

Once clusterheads are selected and clusters are formed, there is a need for clusterheads to recognize each other, that is, finding their neighboring clusterheads. This is done via ordinary pebbles. The ordinary pebbles communicate to their clusterheads information about other clusterheads that are at most three hops away. Figure 6.10 shows two cases of neighboring clusterheads. In the first case, p receives messages with $R = CH$ from both c_1 and c_2 . Since p belongs to c_1 , p sends a message, to c_1 , encrypted with $K_{c_1}^i$, which states that c_2 is a clusterhead. Moreover, c_2 has previously received a message from p with $R = c_1$ and knows that c_1 is two hops away. In the second case, p_1 (or p_2) receives a message from p_2 (or p_1) and knows that p_2 (or p_1) has joined the cluster of c_2 (or c_1). p_1 (or p_2) sends an encrypted message to c_1 (or c_2) to state this fact. Thus c_1 (or c_2) is aware of the presence of c_2 (or c_1) three hops away.

- 2 Phase 1, step 2 Backbone Superimposing: The selected clusterheads, already having knowledge of their neighboring clusterheads, begin to communicate with each other to build a *backbone of clusterheads* CB . The communication among neighboring clusterheads will be encryption-protected using key K_B^i . The backbone is obtained by connecting each clusterhead

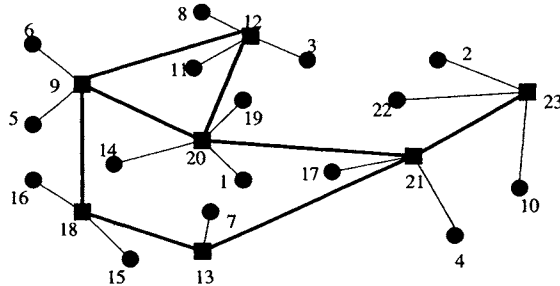


Figure 6.11. Backbone in the above pebble clusters (Source: [Basagni et al., 2001], © ACM).

with all its neighboring clusterheads. Figure 6.11 shows a backbone formed by the clusterheads in Figure 6.9.

- 3 Phase 2, step 1 Key Generation: It is necessary for an entity, called a *key manager*, to initiate the key generation phase. Due to lack of global knowledge as to which clusterhead has the largest weight among all clusterheads, the protocol is designed so that all clusterheads whose weight is larger than the weights of their neighboring clusterheads become *potential key managers* (PKMs). All PKMs are eligible to initiate the generation and distribution of a new *TEK*. Now, each pebble is equipped with an exponential delay parameter, similar to that in the Ethernet exponential back-off algorithm. By means of an analogous variable back-off, some PKMs will not generate a *TEK*, having already received *TEK* messages from other PKMs. Sometimes collisions are generated, i.e., more than one PKM generates and distributes *TEKs*. Eventually, by using weights contained in the rekeying messages for the new keys (see below), setting and resetting of timers, all clusterheads, including PKMs, will finally agree to accept the key distributed by one particular PKM c^* and this key will be chosen as the new *TEK*.

A typical message containing a *TEK* candidate for t^i , generated by a PKM c , encrypted by K_B^i , and distributed to the backbone (prior to the final selection of t^i) is:

$$c \Rightarrow CB : E_{K_B^i}(ID_c|w_c|t^i|MAC(K_{GI}, \langle ID_c|w_c|t^i \rangle))$$

- 4 Phase 2, step 2 Key Distribution: Once the new *TEK* t^i has been selected uniquely and unambiguously by all clusterheads in \mathcal{C} , each clusterhead $c \in$

C encrypts by K_c^i and multicasts securely the key to the pebbles in its own cluster as follows:

$$c \Rightarrow M_c : E_{K_c^i}(ID_c | t^i | MAC(K_{GI}, \langle ID_c | t^i \rangle))$$

After the TEK is received by all pebbles, they can perform secure group communication protected by the TEK , independent of the backbone.

It is worthwhile discussing the above protocol which tries to give a solution to the key management problem in large ad hoc networks. The protocol assumes that all pebbles are equipped with “tamper-resistant” capacity and an ex-member automatically stops receiving communications after its separation from the group. This assumption makes group key management almost unnecessary because the real difficulty in key management comes from the departure events. As well, the assumption about the preshared group identity key K_{GI} is not practical in ad hoc environments. Moreover, the keys K_B^i and K_H^i are dependent on each other and on the one-way function. If any one of the keys is compromised, all future keys including the K_c^i are exposed to an attacker. Finally, when a clusterhead c distributes its cluster key K_c^i to its cluster, it encrypts the message using key K_H^i . Since all pebbles in all the clusters know K_H^i , the goal that only pebbles in c 's cluster should obtain K_c^i is not achieved.

6.2.3.2 Desired Properties of Any SGC Protocol over Ad Hoc Networks

As indicated above, most assumptions in securing the pebblenet protocol are not practical in ad hoc networks because compared with other kinds of networks, ad hoc networks have their own special features and group key management for SGC over such networks needs to satisfy these particular features.

In what follows, we identify some of the important features of an ad hoc network with respect to SGC and propose desired properties of SGC protocols for such networks.

Ad Hoc Network Properties

- No Pre-shared secret - Due to the ‘ad hoc’ nature of group communications, participating members cannot be assumed to share a pre-shared secret. Consider a scenario, where delegates to a summit wish to form a secure group to share information. Since the participating members are not known beforehand, it is not possible to assume that the members share some secret information before forming the group. A number of protocols have been proposed that are based on the key pre-distribution concept [Eschenauer and Gligor, 2002, Du et al., 2003, Liu and Ning, 2003, Zhu et al., 2003a, Chan et al., 2003, Zhu et al., 2003b]. It should also be mentioned

here, that these schemes are targeted for sensor networks with thousands of low-power nodes, where any key-agreement scheme presents too much of a computational overhead.

- No Centralized Trusted Authority (TA) - Security services are usually provided by a central authority which is trusted by all the nodes. In the case of wireless networks this is provided by a base station. Perrig et al. in [Perrig et al., 2001b] discuss security protocols for sensor networks which are based on such base station support. In ad hoc networks, the assumption of a central authority is invalid since the network is formed by all the participating nodes themselves without any infrastructure. Some of the other general problems with a central authority are performance and availability issues. With increasing number of group members, there will be a performance bottleneck at the central authority. Furthermore, if the TA is not available then further group communication is not possible (single-point-of-failure).
- No Group Controller(GC) - In the absence of a centralized trusted authority, some protocols [Li et al., 2002, Yasinsac et al., 2002] use the services of a Group Controller. A Group Controller is usually one of the members in the group who has the responsibility of playing the lead role in setting up the conference and key distribution. Consequently, the group controller does more work compared to the other members in the group. In ad hoc networks, since all the nodes in the network are 'equal', having a GC which does more work and has more responsibility is not desirable. Since the function of a GC is similar to that of a TA, it also suffers from some of the same drawbacks of a TA namely, performance bottleneck and availability. Furthermore, in ad hoc networks, the GC node may run out of battery power or even leave the secure group. Another important challenge is that of selecting a member who will perform the duties of a GC since this member does more computation and communication than the rest of the group.
- Battery power - This is the primary concern for a mobile node. Any application or service should be power-optimized to run in an ad hoc environment since ad hoc nodes are energy starved devices. Every time a wireless node transmits or receives information it has to expend some battery power. An adversary instead of attacking a wireless node directly, can launch a different type of attack which drains the node's batteries [Stajano and Anderson, 1999]. Hence, an algorithm for SGC should be efficient in both computation and communication. For the same reason, protocols which extensively use the computational intensive PKI (public key cryptosystems) are not preferred.
- Equal work load - The proper operation of an ad hoc network depends upon the contribution of every mobile node in the network. Hence, there should

be a sense of 'fairness' wherein a particular node does not perform more work compared to another node. Therefore, members in a group should perform equal amount of computation and communication to set up the conference.

- Mobility - The nodes in an ad hoc environment are highly mobile. Due to this high degree of mobility, a mobile node may fall in and out of range from any of the other nodes and thus from the ad hoc network. With respect to an SGC protocol, a high level of mobility means that any SGC scheme should efficiently support user join and leave operations.
- Non-Serialization - Some key agreement protocols such as [Steiner et al., 2000, Ingemarsson et al., 1982, Steer et al., 1990] depend on serial data transfer operations to achieve the group key. In such schemes, all the members in the group are serialized and information is sent from one member to the other in a pre-determined order. Such serialization is not efficient in a highly dynamic environment such as an ad hoc network. Also, in such protocols there is the additional overhead of serializing the members and the operations.

Properties of An Ideal SGC Protocol for Ad Hoc Networks

- Contributory protocol - An SGC scheme is said to be contributory if the key is made up of contributions from all the group members. Using such a contributory protocol ensures that all the group members in the ad hoc network play an active role in the computation of the group key.
- Key Agreement - This is defined as a key establishment protocol whose secret key is a function of information contributed by all the participants in the group, so that no member can predetermine the value of the key. It is a method for negotiating a key value without actually transferring the keys, even in encrypted form. The best example for key agreement is the Diffie-Hellman (DH) key exchange protocol [Diffie and Hellman, 1976a]. As discussed previously, since the existence of either a centralized trusted authority (TA), group controller (GC) or a pre-shared secret among all the mobile nodes is not assumed, the SGC scheme should include a key agreement protocol.
- Good user dynamics - This means that the SGC scheme should be able to support member join/leave operations efficiently. This is a very important feature in ad hoc networks due to its highly dynamic topology and user mobility.

- Efficiency - Any scheme for SGC in ad hoc networks should be efficient in both computation and communication and should scale well due to the constrained computation power and resources of mobile devices.

Chapter 7

CONCLUDING REMARKS

In this book, we have provided a comprehensive discussion of group key management for secure group communications, dynamic conferencing, and hierarchical access control approaches in SGC. Most of the group key management schemes are generic in the sense that they are not subject to any assumptions about the underlying networks. However, there are several schemes which are specific to wireless networks. We have also highlighted membership management, admission control, message/source authentication, and other issues related to correct and practical implementation of SGC schemes. In this chapter, we summarize what we have discussed in the book, and present several exemplary applications which call for SGC technology and highlight the importance and bright future of secure group communication.

7.1 Summary of Book Contents

Chapter 1 includes a brief introduction to SGC and to few cryptographic primitives used further in the book. Chapter 2 discusses *group key management* for SGC in its various forms. Chapter 3 covers a specific class of group management called *tree-based* schemes. Chapter 4 discusses *group/conference key management* for dynamic conferencing. Chapter 5 deals with *hierarchical access control* schemes in SGC. Finally, Chapter 6 covers SGC group key management for *wireless* environments.

One issue regarding key management is whether it is implemented in a centralized or distributed manner. In this book, we have discussed several distributed key management schemes, including various *n-party Diffie-Hellman key exchange* schemes in section 2.5, TGDH in section 3.2.1, BF-TGDH in section 3.2.2, DISEC in section 3.2.3, PKDC in section 4.2, SLDC in section 4.3 and BF-TGDH DC in section 4.6.

7.2 Exemplary Applications

SGC has broad application paradigms. Typical applications include teleconferencing, tele-medicine, collaborative work, interactive games, distance teaching-learning, information distribution, duplication of servers, virtual private networks, and emerging grid computing. In this section, we briefly discuss four typical applications.

7.2.1 Secure Teleconferencing

Teleconferencing has been in practice for many years. Multiple individuals at geographically distant locations around the world can attend a teleconference session using the telephone network or the Internet. Increasingly the use of Internet is being preferred for teleconferencing through the *Access Grid* technology. A typical teleconference scenario is a *graduate student's thesis defense* held when some of his/her supervisory committee members are away from the university. The committee members use teleconferencing to participate in the defense. Even though a thesis defense may not require any security protocols, security is needed for teleconferencing among the leaders of participating countries in a war to prevent an eavesdropping enemy from harvesting any information in transit. In this case, SGC protocols are needed. Since the teleconferencing in the above example is of a small size, any SGC key management protocol would be sufficient. However, in a teleconference involving tens of thousands of participants, scalable SGC solutions such as a *key tree* scheme or *hierarchical structure* may be required.

7.2.2 Virtual Private Networks (VPN)

The *Virtual Private Network* (VPN) has attracted the attention of many organizations looking both to expanding their networking capabilities and reducing their costs.

VPNs can be found in the workplace but also in the home, where they allow employees to safely log into company networks. Telecommuters and those who travel often find VPNs to be a more convenient way to stay "plugged in" to the corporate intranet.

VPNs supply network connectivity over a possibly long physical distance. In this respect, a VPN is a form of a Wide Area Network (WAN).

The key feature of a VPN, however, is its ability to use public networks such as the Internet rather than rely on private leased lines. VPN technologies implement restricted-access networks that utilize the same cabling and routers as a public network, and they do so without sacrificing features or basic security.

All nodes (systems, computers) in a VPN are connected by public networks. However, there are shared secrets (keys) among these VPN nodes. When a VPN node transmits messages to other VPN nodes, the node sending encrypts

the messages first before transmission. The encrypted messages travel through the public networks and the other VPN nodes decrypt the messages after them. These messages are unintelligible to an attacker who is not a group member since he/she does not possess the shared secrets. VPN entities in a VPN network may be added or removed dynamically. The shared secrets need to be changed accordingly. This is fundamental to key management in SGC. As a result, various SGC key management protocols can be deployed in VPNs.

7.2.3 Secure Grid Computing

Grid computing has become one of the latest buzzwords in the IT industry. This is an innovative approach that leverages existing IT infrastructure to optimize computational resources and manage data and computing workloads. In grid computing, a *grid* is a collection of resources owned by multiple organizations that are shared and coordinated. The key point in grid computing is "flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources" [Chang, 2004]. Because grid computing is based on networks (e.g., the Internet), the participating individuals in grid computing communicate with each other. Therefore, grid computing needs group communication. Moreover, apart from many security issues, such as *user authentication* and *access authorization*, many grid computing applications require that the communication messages be confidential, integrity-protected, and source authenticated. All these security issues may be naturally solved when grid computing is implemented by means of SGC protocols.

7.2.4 Secure Collaborative Work

The previous three applications are essentially organization-bound group communication tools. Due to the large-scale deployment of the Internet, scientists, researchers, and practitioners are free to perform collaborative work with their colleagues at other institutions worldwide. If this collaborative work is new and mission-critical, the collaborators will require their communications to be kept secret and intact. Consequently, SGC protocols are needed for their work. Moreover, since their work is conducted on the basis of collaboration, the participants are generally at the same hierarchical level. Therefore, distributed contributory key agreement protocols are more suitable for such an application.

7.3 Conclusion

Secure Group Communication is an emerging and important application and research area. Even though there have been exciting achievements in SGC key management, much work still remains to be done by researchers and designers. Detailed and comprehensive analyses of complexity and performance need to

be conducted. The implementation of flexible, configurable, and scalable secure group communication systems which can be used in many applications in the real world is certainly required in the near future. Finally, since wireless communication is becoming ubiquitous, SGC in wireless environments is a significant field to be investigated.

We hope that more researchers and professionals will benefit from, and join the area of SGC through, this book. The future of secure group communication is very promising.

References

- [Agrawal et al., 2002] Agrawal, M., Kayal, N., and Saxena, N. (2002). A deterministic poly-time primality testing algorithm. *IIT Kanpur, India*, available at <http://www.cse.iitk.ac.in/users/manindra/primality.ps>, accessed in June 2004.
- [Akl and Taylor, 1983] Akl, S. G. and Taylor, P. D. (1983). Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems*, 1(3):239–247.
- [Amir et al., 2003] Amir, Y., Nita-Rotaru, C., Schultz, J., and Stanton, J. (2003). Securespread. <http://www.cnds.jhu.edu/securespread/>, accessed in June 2004.
- [Bakkardie, 1996] Bakkardie, A. (1996). Scalable multicast key distribution. *RFC 1949*.
- [Banerjee and Bhattacharjee, 2002] Banerjee, S. and Bhattacharjee, B. (2002). Scalable secure group communication over IP multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1151–1527.
- [Basagni et al., 2001] Basagni, S., Herrin, K., Bruschi, D., and Rosti, E. (2001). Secure pebblenets. *MobiHOC 2001*, pages 156–163.
- [Becker and Wille, 1998] Becker, K. and Wille, U. (1998). Communication complexity of group key distribution. *ACM conference on computer and communication security*, pages 1–6.
- [Beimel and Chor, 1994] Beimel, A. and Chor, B. (1994). Interaction in key distribution schemes. *Advances in Cryptology - CRYPTO'93, LNCS, Springer, Berlin*, 773:444–457.
- [Beimel and Chor, 1996] Beimel, A. and Chor, B. (1996). Communications in key distribution schemes. *IEEE Transactions on Information Theory*, 42:19–28.
- [Birget et al., 2001] Birget, J.-C., Zou, X., Noubir, G., and Ramamurthy, B. (2001). Hierarchical access control in distributed environments. *IEEE International Conference on Communications (ICC)*, pages 101–140.
- [Blom, 1985] Blom, R. (1985). An optimal class of symmetric key generation systems. *Advances in Cryptology - EUROCRYPT'84, LNCS, Springer, Berlin*, 209:335–338.

- [Blundo and Cresti, 1995] Blundo, C. and Cresti, A. (1995). Space requirements for broadcast encryption. *Advances in Cryptology - EUROCRYPT'94, LNCS, Springer, Berlin*, 950:287–298.
- [Blundo et al., 1998] Blundo, C., Mattos, L. A. F., and Stinson, D. R. (1998). Generalized Beigel-Chor scheme for broadcast encryption and interactive key distribution. *Theoretical Computer Science*, 200(1-2):313–334.
- [Blundo et al., 1993] Blundo, C., Santis, A. D., Herzberg, A., Kutten, S., Vaccaro, U., and Yung, M. (1993). Perfect secure key distribution for dynamic conferences. *Advances in Cryptology - CRYPTO'92, LNCS, Springer, Berlin*, 740:471–486.
- [Burmester and Desmedt, 1995] Burmester, M. and Desmedt, Y. (1995). A secure and efficient conference key distribution system. *Advances in Cryptology - EUROCRYPT'94, LNCS, Springer, Berlin*, 950:275–286.
- [Burmester and Desmedt, 1996] Burmester, M. and Desmedt, Y. (1996). Efficient and secure conference-key distribution. *Security Protocols Workshop*, pages 119–129.
- [Cain et al., 2001] Cain, B., Speakman, T., and Towsley, D. (2001). Generic router assist (GRA) building block motivation and architecture. *Internet Draft: Internet Engineering Task Force*.
- [Canetti et al., 1999a] Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M., and Pinkas, B. (1999a). Multicast security: a taxonomy and some efficient constructions. *Proceedings of INFOCOM'99: Conference on Computer Communications*, 2:708–716.
- [Canetti et al., 1999b] Canetti, R., Malkin, T., and Nissim, K. (1999b). Efficient communication-storage tradeoffs for multicast encryption. *Lecture Notes in Computer Science (Advances in Cryptology-EUROCRYPT'99)*, 1592:459–470.
- [Caronni et al., 1998] Caronni, G., Waldvogel, K., Sun, D., and Plattner, B. (1998). Efficient security for large and dynamic multicast groups. *Proceedings of the Seventh IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '98) (Cat. No.98TB100253)*, pages 376–383.
- [Chan et al., 2003] Chan, H., Perrig, A., and Song, D. (2003). Random key predistribution schemes for sensor networks. In *In Proc. of the IEEE Security and Privacy Symposium*, pages 197–215.
- [Chang, 2004] Chang, H. (2004). Wireless grid computing
. <http://www.eecs.tufts.edu/~hchang/Projects2003/Mark-Grant-Senior-Project-Presentation1-1.ppt>, accessed in June 2004.
- [Chang et al., 1999] Chang, I., Engel, R., Kandlur, D., Pendarakis, D., and Saha, D. (1999). Key management for secure internet multicast using boolean function minimization techniques. *Proceedings of INFOCOM'99: Conference on Computer Communications*, 2:689–698.
- [Chick and Tavares, 1990] Chick, G. C. and Tavares, S. E. (1990). Flexible access control with master keys. *Advances in Cryptology: CRYPTO '89 LNCS*, 435:316–322.
- [Chiou and W.T.Chen, 1989] Chiou, G. H. and W.T.Chen (1989). Secure broadcasting using the Secure Lock. *IEEE Transactions on Software Engineering*, 15(8):929–934.

- [DeCleene et al., 2001] DeCleene, B., Dondeti, L., Griffin, S., Hardjono, T., Kiwior, D., Kurose, J., Towsley, D., Vasudevan, S., and Zhang, C. (2001). Secure group communications for wireless networks. *Proceedings Communications for Network-Centric Operations: Creating the Information Force (2001 MILCOM)*, 1:113–117.
- [Desmedt and Frankel, 1989] Desmedt, Y. and Frankel, Y. (1989). Threshold cryptosystems. *Proceeding on Advances in Cryptology*, pages 307–315.
- [Desmedt and Viswanathan, 1998] Desmedt, Y. and Viswanathan, V. (1998). Unconditionally secure dynamic conference key distribution. *Proceedings of the IEEE International Symposium on Information Theory*, pages 383–383.
- [Diffie and Hellman, 1976a] Diffie, W. and Hellman, M. (1976a). New directions in cryptography. In *IEEE transactions on Information Theory*, pages 644–654.
- [Diffie and Hellman, 1976b] Diffie, W. and Hellman, M. E. (1976b). Multiuser cryptographic techniques. *AFIPS conference proceedings*, 45:109–112.
- [Dondeti, 1999] Dondeti, L. R. (1999). Efficient private group communication over public networks. *Phd. Dissertation, CSE UNL*.
- [Dondeti et al., 1999] Dondeti, L. R., Mukherjee, S., and Samal, A. (1999). A dual encryption protocol for scalable secure multicasting. In *Fourth IEEE Symposium on Computers and Communications*, pages 2–8.
- [Dondeti et al., 2000] Dondeti, L. R., Mukherjee, S., and Samal, A. (2000). DISEC: a distributed framework for scalable secure many-to-many communication. In *Proceedings of Fifth IEEE Symposium on Computers and Communications (ISCC 2000)*, pages 693–698.
- [Du et al., 1999] Du, F., Ni, L. M., and Esfahanian, A. H. (1999). Towards solving multicast key management problem. *ICCCN'99 Eighth International Conference on Computer Communications and Networks*, pages 232–236.
- [Du et al., 2003] Du, W., J. Deng, Y. H., and Varshney, P. (2003). A pairwise key pre-distribution scheme for wireless sensor networks. In *Proc. of the 10th ACM Conference on Computer and Communication Security*, pages 42–51.
- [ElGamal, 1985] ElGamal, T. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–471.
- [Eschenauer and Gligor, 2002] Eschenauer, L. and Gligor, V. (2002). A key-management scheme for distributed sensor networks. In *In Proc. of 9th ACM Conference on Computer and Communication Security*, pages 41–47.
- [Fiat and Naor, 1994] Fiat, A. and Naor, M. (1994). Broadcast encryption. *Advances in Cryptology - CRYPTO'93, LNCS, Springer, Berlin*, 773:480–491.
- [Gouda et al., 2002a] Gouda, M. G., Huang, C.-T., and Elnozahy, E. N. (2002a). Key trees and the security of interval multicast. *Proceedings 22nd International Conference on Distributed Computing Systems*, pages 467–468.
- [Gouda et al., 2002b] Gouda, M. G., Huang, C.-T., and Elnozahy, E. N. (2002b). Key trees and the security of interval multicast. *Technical Report TR-02-18, Department of Computer Sciences, The University of Texas at Austin, Austin, Texas*.

- [Halevi and Petrank, 1995] Halevi, S. and Petrank, E. (1995). Storing classified files. *MIT Technical Report*.
- [Hamey and Muckenhim, 1997a] Hamey, H. and Muckenhim, C. (1997a). Group Key Management Protocol (GKMP) Architecture. *RCF 2094*.
- [Hamey and Muckenhim, 1997b] Hamey, H. and Muckenhim, C. (1997b). Group Key Management Protocol (GKMP) Specification. *RCF 2093*.
- [Harney and Harder, 1999] Harney, H. and Harder, E. (1999). Logical key hierarchy protocol. *Internet Draft (work in progress), draft-harney-sparta-lkhp-sec-00.txt, Internet Engineering Task Force*.
- [Horng, 2002] Horng, G. (2002). Cryptanalysis of a key management scheme for secure multicast communications. *IEICE Transactions on Communications*, E85-B(5):1050–1051.
- [Ingemarsson et al., 1982] Ingemarsson, I., Tang, D., and Wong, C. (1982). A conference key distribution system. *IEEE Transactions on Information Theory*, 28(5):714–720.
- [Kaufman et al., 2002] Kaufman, C., Perlman, R., and Speciner, M. (2002). *Network security: private communication in a public world*. Prentice Hall, Upper Saddle River, NJ, USA.
- [Kihlstrom et al., 1998] Kihlstrom, K. P., Moser, L. E., and Melliar-Smith, P. M. (1998). The SecureRing protocols for securing group communication. *Thirty-First Annual Hawaii International Conference on System Sciences*, 3:317–326.
- [Kim et al., 2000] Kim, Y., Perrig, A., and Tsudik, G. (2000). Simple and fault-tolerant key agreement for dynamic collaborative groups. In *Proceedings of the 7th ACM Conference on Computer and Communications Security (ACM CCS 2000)*, pages 235–244.
- [Kim et al., 2001] Kim, Y., Perrig, A., and Tsudik, G. (2001). Communication-efficient group key agreement. In *Information System Security, Proceedings of the 17th International Information Security Conference IFIP SEC'01*, pages 229–244.
- [Kim et al., 2002] Kim, Y., Perrig, A., and Tsudik, G. (2002). Admission control in peer groups. *Manuscript*.
- [Kim et al., 2004] Kim, Y., Perrig, A., and Tsudik, G. (2004). Tree-based group key agreement. *ACM Transactions on Information Systems Security*, 7(1):60–96.
- [Koblitz, 1994] Koblitz, N. (1994). *A Course in Number Theory and Cryptography*. Springer, Verlag, NY, USA.
- [Ku and Chen, 2003] Ku, W.-C. and Chen, S.-M. (2003). An improved key management scheme for large dynamic groups using one-way function trees. *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 391–396.
- [Levine and Garcia-Luna-Aceves, 1997] Levine, B. N. and Garcia-Luna-Aceves, J. (1997). Improving internet multicast with routing labels. in *Proc. IEEE Int. Conf. on Network Protocols*, pages 241–250.
- [Li et al., 2002] Li, X., Wang, Y., and Frieder, O. (2002). Efficient hybrid key agreement protocol for wireless ad-hoc networks. In *IEEE 11th International Conference on Computer, Communication and Networks*, pages 404–409.

- [Li et al., 2000] Li, X., Yang, Y., Gouda, M., and Lam, S. S. (2000). Batch updates of key trees. *Technical Report TR-00-22, The University of Texas, September 2000*, <http://www.cs.utexas.edu/ftp/pub/techreports/tr00-22.ps.Z>.
- [Lin, 1997] Lin, C. H. (1997). Dynamic key management schemes for access control in a hierarchy. *Computer Communications*, 20:1381–1385.
- [Liu and Ning, 2003] Liu, D. and Ning, P. (2003). Establishing pairwise keys in distributed sensor networks. In *Proc. of the 10th ACM Conference on Computer and Communication Security*, pages 52–61.
- [Mackinnon et al., 1985] Mackinnon, S. T., Taylor, P. D., Meijer, H., and Akl, S. G. (1985). An optimal algorithm for assigning cryptographic keys to control access in a hierarchy. *IEEE Transactions on Computers*, 34(9):797–802.
- [Magliveras, 1986] Magliveras, S. (1986). A cryptosystem from logarithmic signatures of finite groups. *Proceedings of the 29th Midwest Symposium on Circuit and Systems*, pages 972–975.
- [Magliveras and Memon, 1992] Magliveras, S. and Memon, N. D. (1992). Algebraic properties of cryptosystem PGM. *J. Cryptology*, 5:167–183.
- [Magliveras et al., 1985] Magliveras, S. S., Oberg, B. A., and Surkan, A. J. (1985). A new random number generator from permutation groups. *Red. Del Sem Matemat. Di Milano*, LIV:203–223.
- [Menezes et al., 1996] Menezes, A., Oorschot, P. V., and Vanstone, S., editors (1996). *Handbook of applied cryptography*. CRC Press, Inc., Boca Raton, Florida, USA.
- [Mitra, 1997] Mitra, S. (1997). Iolus: A framework for scalable secure multicasting. *Journal of Computer Communication Reviews*, 27(4):277–288.
- [Molva and Pannetrat, 1999] Molva, R. and Pannetrat, A. (1999). Scalable multicast security in dynamic groups. *6th ACM Conference on Computer and Communications Security (ACM CCS 1999), Singapore*, pages 101–112.
- [Moyer et al., 1999] Moyer, M. J., Rao, J. R., and Rohatgi, P. (1999). A survey of security issues in multicast communications. *IEEE Network*, pages 12–23.
- [NIST, 2003] NIST (2003). Draft wireless network security: IEEE 802.11, Bluetooth and handheld devices. [Http://src.nist.gov/publications/drafts/draft-sp800-48.pdf](http://src.nist.gov/publications/drafts/draft-sp800-48.pdf), accessed in November 2003.
- [Noubir, 1998] Noubir, G. (1998). Multicast security. *European Space Agency, Project: Performance Optimization of Internet Protocol Via Satellite*.
- [Perrig, 2001] Perrig, A. (2001). The BiBa one-time signature and broadcast authentication protocol. *Proceedings of the ACM Conference on Computer and Communications Security (CCS 2001)*, pages 28–37.
- [Perrig et al., 2001a] Perrig, A., Canetti, R., Song, D., and Tygar, D. (2001a). Efficient and secure source authentication for multicast. *Proceedings of Network and Distributed System Security Symposium (NDSS 2001)*, pages 35–46.

- [Perrig et al., 2001b] Perrig, A., Szewczyk, R., Wen, V., Culler, D., and Tygar, J. (2001b). SPINS: Security protocols for sensor networks. In *Proc. of 7th ACM Mobicom*, pages 521–534.
- [Perrig and Tygar, 2002] Perrig, A. and Tygar, J., editors (2002). *Secure Broadcast Communication in Wired and Wireless Networks*. Kluwer Academic Publishers, Boston, MA, USA.
- [Reiter, 1994] Reiter, M. K. (1994). Secure agreement protocols: reliable and atomic group multicast in rampart. *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80.
- [Rodeh et al., 2001] Rodeh, O., Birman, K., and Dolev, D. (2001). The architecture and performance of security protocols in the ensemble group communication system. *ACM Transactions on Information and System Security*, 4(3):289–319.
- [Rodeh et al., 1998] Rodeh, O., Birman, K., Hayden, M., Xiao, Z., and Dolev (1998). Horus/ensemble: Ensemble security. *Tech. Rep. TR98-1703, Cornell University, Department of Computer Science*.
- [Sandhu, 1988] Sandhu, R. S. (1988). Cryptographic implementation of a tree hierarchy for access control. *Information Processing Letters*, 27:95–98.
- [Shamir, 1979] Shamir, A. (1979). How to share a secret. *Communication of ACM*, 22:612–613.
- [Sherman and McGrew, 2003] Sherman, A. T. and McGrew, D. A. (2003). Key establishment in large dynamic groups using one-way function trees. *IEEE transactions on Software Engineering*, 29(5):444–458.
- [Speakman et al., 2000] Speakman, T. et al. (2000). PGM reliable transport protocol. *Internet Draft: Internet Engineering Task Force*.
- [Stajano and Anderson, 1999] Stajano, F. and Anderson, R. (1999). The resurrecting duckling: Security issues for ad-hoc wireless networks. In *7th International Workshop in Security Protocols*, pages 22–26.
- [Steer et al., 1990] Steer, D., Strawczynski, L., Diffie, W., and Wiener, M. (1990). A secure audio teleconference system. *Advances in Cryptology-CRYPTO'88, LNCS, Springer-Verlag*, 403:520–528.
- [Steiner et al., 1996] Steiner, M., Tsudik, G., and Waidner, M. (1996). Diffie-Hellman key distribution extended to group communication. *ACM Conference on Computer and Communications Security (ACM CCS 1996)*, pages 31–37.
- [Steiner et al., 1997] Steiner, M., Tsudik, G., and Waidner, M. (1997). CLIQUES: A new approach to group key agreement. *IEEE International Conference on Distributed Computing Systems (ICDCS 1997)*, pages 380–387.
- [Steiner et al., 2000] Steiner, M., Tsudik, G., and Waidner, M. (2000). Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780.
- [Stinson, 1995] Stinson, D. R., editor (1995). *Cryptography: Theory and Practice*. CRC Press, Inc., Boca Raton, Florida, USA.

- [Stinson, 1997] Stinson, D. R. (1997). On some methods for unconditionally secure key distribution and broadcast encryption. *Design, Codes and Cryptography*, 12:215–243.
- [Sun et al., 2002] Sun, Y., Trappe, W., and Liu, K. J. R. (2002). An efficient key management scheme for secure wireless multicast. *Proceedings of the IEEE International Conference on Communications (ICC)*, 2:1236–1240.
- [Wallner et al., 1998] Wallner, D., Harder, E., and Agee, R. (1998). Key management for multicast: Issues and architectures. *Internet Draft (work in progress), draft-wallner-key-arch-01.txt, Internet Eng. Task Force*.
- [Wong et al., 1998] Wong, C. K., Gouda, M., and Lam, S. S. (1998). Secure group communications using key graphs. *SIGCOMM '98, Also University of Texas at Austin, Computer Science Technical report TR 97-23*, pages 68–79.
- [Wong et al., 2000] Wong, C. K., Gouda, M., and Lam, S. S. (2000). Secure group communications using key graphs. *IEEE/ACM Transactions on Networks*, 8(1):16–30.
- [Yasinsac et al., 2002] Yasinsac, A., Thakur, V., Carter, S., and I.Cubukcu (2002). A family of protocols for group key generation in ad hoc networks. *Proceedings of the IASTED International Conference on Communications and Computer Networks (CCN02)*, pages 183–187.
- [Zhang et al., 2001] Zhang, X. B., Lam, S. S., Lee, D.-Y., and Yang, Y. R. (2001). Protocol design for scalable and reliable group rekeying. *Proceedings SPIE Conference on Scalability and Traffic Control in IP Networks*, pages 87–108.
- [Zhou and Haas, 1999] Zhou, L. and Haas, Z. J. (1999). Securing ad hoc networks. *IEEE Networks*, 13(6):24–30.
- [Zhu et al., 2003a] Zhu, S., Setia, S., and Jajodia, S. (2003a). LEAP: efficient security mechanisms for large-scale distributed sensor networks. In *Proc. of the 10th ACM Conference on Computer and Communication Security*, pages 62–72.
- [Zhu et al., 2003b] Zhu, S., Xu, S., Setia, S., and Jajodia, S. (2003b). Establishing pair-wise keys for secure communication in ad-hoc networks: A probabilistic approach. In *IEEE international Conference on Network Protocols*, pages 326–335.
- [Zou et al., 2002a] Zou, X., Magliveras, S., and Ramamurthy, B. (2002a). A dynamic conference scheme extension with efficient burst operation. *Congressus Numerantium*, 158:83–92.
- [Zou and Ramamurthy, 2004] Zou, X. and Ramamurthy, B. (2004). A block-free tree-based group Diffie-Hellman key agreement for secure group communications. *Proceedings of International Conference on Parallel and Distributed Computing and Networks, Innsbruck, Austria*, pages 288–293.
- [Zou et al., 2001] Zou, X., Ramamurthy, B., and Magliveras, S. (2001). Chinese Remainder Theorem based hierarchical access control for secure group communications. *Lecture Notes in Computer Science (LNCS), Springer-Verlag (International Conference on Information and Communication Security)*, 2229:381–385.
- [Zou et al., 2002b] Zou, X., Ramamurthy, B., and Magliveras, S. (2002b). Efficient key management for secure group communication with bursty behavior. *Proceedings of International Conference on Communication, Internet, and Information Technology (CIIT)*, pages 148–153.

- [Zou et al., 2003] Zou, X., Ramamurthy, B., Vinodchandran, N. V., and Balachandran, R. K. (2003). Algorithms for unified hierarchy based access control. *Proceedings of International Conference on Communications, Internet, and Information Technology (CIIT 2003)*, Scottsdale, AZ, USA, pages 31–36.

About the Authors



Xukai Zou received his B.S. degree in Computer Science from Zhengzhou University, Zhengzhou (China) in 1983, his M.S. degree in Computer Science and Engineering from Huazhong University of Science and Technology, Wuhan (China) in 1986 and his Ph.D. degree in Computer Science from the University of Nebraska-Lincoln in 2000. Before pursuing his Ph.D. degree, Dr. Zou served as an Associate Professor (1994–1998), Assistant Professor (1988–1993) and Lecturer (1986–1987) at Zhengzhou University. Currently Dr. Xukai Zou is an Assistant Professor with Purdue University School of Science at Indianapolis, Indiana, USA. His research interests include applied cryptography and network security, in particular, secure group communication/secure dynamic conferencing, Web technology and Internet engineering, wireless networks, and analysis and design of computer algorithms. Dr. Zou has authored two books and published over ten security-related papers on group/conference key management for secure group communications and secure dynamic conferencing. He has served as a member of a number of technical program committees, member of editorial boards, and a reviewer for many international organizations, international conferences and international journals. He is a recipient of two U.S. National Science Foundation grants. His email address is xkzou@cs.iupui.edu.



Byrav Ramamurthy received his B.Tech. degree in Computer Science and Engineering from the Indian Institute of Technology, Madras (India) in 1993. He received his M.S. and Ph.D. degrees in Computer Science from the University of California (UC), Davis in 1995 and 1998, respectively. Currently Dr. Ramamurthy is an Associate Professor in the Department of Computer Science and Engineering at the University of Nebraska-Lincoln (UNL). At UNL, Dr. Ramamurthy leads the Networking Research Group in ongoing projects on topics such as optical networks, network security, wireless networks, grid computing, and telecommunications. He is the co-director of the UNL Academic Program Priority Initiative in the areas of Simulation & Computing Engineering (SCE) and Information Technology & Telecommunications (ITT). He is the founding co-director of the Advanced Networking and Distributed Experimental Systems (ANDES) Laboratory at UNL. Dr. Ramamurthy is the author of the textbook "Design of Optical WDM Networks - LAN, MAN and WAN Architectures" published by Kluwer Academic Publishers in 2000. He was the Feature Editor on Theses for the *Optical Networks* magazine. He was a guest co-editor for a special issue of IEEE Network magazine on Optical Communication Networks. He has served as a member of the technical program committees for the IEEE INFOCOM, IEEE GLOBECOM, Opticomm, ICC and ICCCN conferences. From 2001-2003, he served as the founding secretary of the IEEE ComSoc Optical Networking Technical Committee (ONTC). Dr. Ramamurthy was a recipient of the Indian National Talent Search scholarship and was a Fellow of the *Professors for the Future* program at UC Davis. He was the recipient of the UNL College of Engineering and Technology Faculty Research Award for 2000 and the UNL CSE Dept. Students Choice Award for the Best Graduate Professor for 2002-2003. Dr. Ramamurthy's research is supported by the U.S. National Science Foundation, Agilent Tech., and OPNET Inc. His email address is byrav@cse.unl.edu.



Spyros Magliveras received his Bachelor's Degree in Electrical Engineering in 1961 and his Master's degree in Mathematics in 1963 both from the University of Florida. He studied for his Ph.D. degree at the University of Michigan under Professor Donald Livingstone from 1964 to 1968. In 1968 he followed his Professor to Birmingham, England, and completed his Ph.D. in Mathematics from the University of Birmingham in 1970. During his studies he was influenced by Donald Livingstone, John H. Conway, Donald G. Higman, Roger C. Lyndon, Marshall Hall, Jr. and Tom Storer. Currently Dr. Magliveras is Professor of Mathematical Sciences and Director of the *Center for Cryptology and Information Security* at Florida Atlantic University. Prior to his present position, Dr. Magliveras held the *Distinguished Henson Chair for Communication and Information Theory* at the University of Nebraska-Lincoln (1991-2000) and earlier academic positions at the University of Nebraska - Lincoln (1978-1991) and the State University of New York (1970-1978). His research interests include cryptology, network security, data compression, finite groups, combinatorics, the design & complexity of algorithms, and finite geometry. He has been working on *group-theoretic cryptography* for over two decades and is becoming increasingly interested in approaches based on combinatorial group theory. He has served on many professional committees and boards, has organized several International Conferences and served on several editorial boards. He has received numerous awards and sponsored research grants and holds a U.S. patent for a cryptosystem. He was awarded 8 prizes and honors including The ICA *Euler Gold Medal Award* for research in combinatorial mathematics. He has presented over 100 papers in International Conferences, has published over 80 papers in International Journals and the Proceedings of International Conferences and has edited four technical volumes. His email address is spyros@fau.edu.

The authors welcome your comments and suggestions about this book. Please send them to the following addresses:

Prof. Xukai Zou
Department of Computer and Information Science
Purdue University School of Science at Indianapolis
723 W. Michigan ST. SL280E
Indianapolis, IN 46202, U.S.A.
Phone: (317) 278-8576
Fax: (317) 274-9742
Email: xkzou@cs.iupui.edu
Web: www.cs.iupui.edu/~xkzou/

Prof. Byrav Ramamurthy
Department of Computer Science and Engineering
University of Nebraska-Lincoln
256 Avery Hall
Lincoln, NE 68588-0115, U.S.A.
Phone: (402) 472-7791
Fax: (402) 472-7767
Email: byrav@cse.unl.edu
Web: www.cse.unl.edu/~byrav/

Prof. Spyros Magliveras
Department of Mathematics Sciences
Florida Atlantic University
Boca Raton, FL 33431, U.S.A.
Phone: (561) 297-0274
Fax: (561) 297-2436
Email: spyros@fau.edu
Web: www.zeus.math.fau.edu/spyros/

Index

- AKD, 137, 139
- AKT, 73
- ALX Tree, 135
- Access Control, 3
- Access Grid, 150
- Access Point, 131–132
- Ad Hoc Network, 131, 135, 140, 145–148
- Admission Control, 3, 24, 149
- Aggregate Operation, 14, 52–55
- Akl-Taylor Scheme, 109–112
- Area Key Distributor
 - See* AKD
- Area Key, 137–139
- Area Threshold, 139
- Asymmetric Cryptosystem, 9, 11
- Authenticated Diffie-Hellman, 12, 79
- Authentication, 1–2, 125–126
- Auxiliary Key Tree
 - See* AKT
- Auxiliary Key, 73–74
- BD Protocol, 39
- BF-TGDH DC, 101, 149
- BF-TGDH, 50, 78–80, 82, 101
 - Back-End Key, 78–79
 - Dummy Blinded Key, 80
 - Dummy Component, 79
 - Dummy Member, 79
 - Dummy Private Share, 79
 - Dummy Public Share, 79
 - Dummy Root Key, 79–80
 - Dummy Secret Key, 80
 - Front-End Key, 78
- BS, 131–133, 146
- BS-Subtree, 133, 135
- Back-End Key, 78
- Backbone Key, 141
- Backbone, 141, 143–145
- Background, 78, 81
- Backward Secrecy, 69, 134, 136–137, 139
- Base Station
 - See* BS
- Baseline Rekeying, 137–138
- Basic Interval, 96–100
- Battery Power, 146
- BiBa, 130
- Bijection, 10
- Bin Ball, 130
- Binary Key Tree, 59
- Bit-string, 54
- Blinded Key, 81, 83, 85–89
- Blinded Node Secret, 65–66, 68, 71–72, 83
- Block-Free Tree Based Group Diffie-Hellman Scheme
 - See* BF-TGDH

- Boolean Function Minimization, 73, 75
- Boolean Membership Function, 75
- Boolean Monomial, 75
- Bottleneck, 146
- Broadcast Authentication, 130
- Broadcast, 1, 8
- Bulk Leave, 73
- Bulk Operation, 13–14, 49, 52, 134
- Bursty Behavior, 13, 49, 52, 95
- Bursty Operation, 13–14, 49–50, 52
- CA, 11
- CBT, 23–24
- CLIQUES, 2
- CRT, 3, 6, 93, 106, 122–123, 125–126
- CRTHACS, 122, 125–126
- Cellular Network, 131–132
- Central Authority, 146
- Central Trusted Server, 92
- Certificate Authority
 - See* CA
- Certificate, 11
- Chinese Remainder Theorem Based Hierarchical Access Control Scheme
 - See* CRTHACS
- Chinese Remainder Theorem
 - See* CRT
- Ciphertext Space, 7
- Ciphertext, 7, 119
- Cluster Key, 29, 31–33, 141–142, 145
- Cluster Leader, 28, 30–33
- Cluster, 28
- Clusterhead, 142–145
- Clustering Protocol, 28–31, 33, 36
- Coefficient, 59, 94, 96
- Collaborative Work, 1
- Collision Resistant, 4
- Communicant, 2, 11
- Complexity, 4, 93, 96, 103, 118, 125
 - Communication Complexity, 118–119, 125
 - Space Complexity, 4, 93, 96, 103, 118, 125
 - Time Complexity, 4, 93, 96, 103, 118, 125
- Computational Security, 4
- Computationally Infeasible, 10, 20, 114, 120
- Computationally Secure, 4, 19, 94, 101–102, 105–106, 108
- Conference Key, 99, 102–103
- Conference, 3, 91, 98
- Congruence, 6
- Contributory Group Key, 37
- Contributory Protocol, 147
- Coordinator, 45–46
- Core Based Tree
 - See* CBT
- Core Tree, 23
- Cryptographic Hash Function, 3
- Cryptology, 1
- Cryptosystems, 8
- Cumulative Member Removal, 73
- Cyclic Multiplicative Group, 5
- D-Ary Key Tree, 59
- DAG, 106–107, 113
- DEK, 22–23, 26–28, 76, 80–81, 136–137
- DEP, 25–27
 - Key-Subgroup Key, 26
 - Key-Subgroup, 26
 - Subgroup Key, 26
 - Subgroup, 26
- DH Key, 12
- DHP, 12
- DISEC, 50, 83–84, 87–88
 - Blinded Key, 83
 - Key Distribution Binary Tree, 83
 - Key Node, 83
 - Node Secret, 83
 - Unblinded Key, 83

- Distributed Scalable SEcure Communication
 - See DISEC
- DKD, 136–137
- DLP, 3, 5–6, 11–12, 20
- Data Encryption Key
 - See DEK
- Decryption Function, 120
- Decryption Rule, 7
- Decryption Transformation, 10
- Delayed Rekeying, 137–139
- Deterministic Algorithm, 7
- Diffie-Hellman Disguised Public Share, 12, 76
- Diffie-Hellman Key Exchange, 5, 11–13, 37, 76, 79
- Diffie-Hellman Key, 76, 81
 - See DH Key
- Diffie-Hellman Private Share, 12, 76
- Diffie-Hellman Problem
 - See DHP
- Diffie-Hellman Share Generator, 79
- Diffie-Hellman, 30
- Directed Acyclic Graph
 - See DAG
- Directed Multicast, 9, 59, 63–64
- Directly Dependent Key Scheme, 106, 109
- Discrete Logarithm Problem, 5
 - See DLP
- Disjunction, 75
- Distributed Binary Key Tree, 84
- Distributed Interactive Simulation, 1
- Domain Key Distributor, 136
 - See DKD
- Domain, 136–137
- Dual Encryption Protocol
 - See DEP
- Dummy Blinded Key, 80–81
- Dummy Component, 79
- Dummy Member, 78–80, 95
- Dummy Number, 95
- Dummy Private Share, 79
- Dummy Public Share, 79
- Dummy Root Key, 79–80, 82
- Dummy Secret Key, 80–81
- Dynamic Conferencing, 2–3, 15, 91–94, 96–98, 101–102, 149
- EKOL, 139
- ElGamal Public-Key Cryptosystem, 5
- ElGamal Signature Scheme, 5, 80
- ElGamal Signature, 79
- Elliptic Curve, 5
- Encryption Algorithm, 93
- Encryption Function, 119, 121
- Encryption Rule, 7
- Encryption Transformation, 10
- Encryption, 2
- Ethernet, 144
- Euclidean Algorithm, 5
- Euler's Φ Function, 7
- Exponential Back-Off Algorithm, 144
- Exponential Delay Parameter, 144
- Extended Euclidean Algorithm, 21, 112
- Extra Key Owner List
 - See EKOL
- Factorization, 7
- Finite Field, 5, 20
- Forward Secrecy, 69, 136
- Front-End Key, 78
- GC, 17, 29, 50, 52, 65–66, 68, 73, 83, 92, 94–95, 97–99, 101, 103, 106, 109, 112, 116–119, 122–123, 125, 130, 146
- GCD Attack, 119, 125
- GCD, 119
- GDH, 39
 - GDH.1, 39
 - GDH.2, 41
 - GDH.3, 42
- GI, 23
- GSA, 25

- GSC, 25, 135
- GSEC, 1
- GSI, 25
- Generator, 5, 11, 20, 37
- Greatest Common Divisor Attack
 - See* GCD Attack
- Greatest Common Divisor
 - See* GCD
- Group Communication, 1–2, 18
 - Broadcast Communication, 18
 - Few-To-Many Communication, 18
 - Many-To-Many Communication, 18, 20, 49
 - Multicast Communication, 18, 20–21
 - One-To-Many Communication, 18, 20, 22, 26, 28, 49
- Group Controller
 - See* GC
- Group Diffie-Hellman
 - See* GDH
- Group Dynamics, 2, 13, 95
- Group Identity Key, 141
- Group Initiator
 - See* GI
- Group Key Management, 2–3, 13–14, 17, 49, 66, 73, 127, 132, 135, 145, 149
 - Centralized Key Distribution, 18, 20, 49–50
 - Contributory Key Agreement, 18, 20, 49
 - Distributed Key Agreement, 18, 20, 49–50, 76
 - Public-Key Based Key Management, 20, 23
 - Secret-Key Based Key Management, 23
- Group Key, 2, 17
- Group Merge, 14
- Group Partition, 14
- Group SECURITY
 - See* GSEC
- Group Security Agent
 - See* GSA
- Group Security Controller
 - See* GSC
- Group Security Intermediate
 - See* GSI
- Group Splitting, 14
- Group-Oriented Rekeying, 59, 63–64
- Group-Oriented, 63
 - HAC, 3–4, 105–106, 108–109, 112–113, 115, 124–126
- HELLO Message Encryption Key, 141
- Hand-Off, 134
- Hash Function, 3, 46, 74–75, 129
- Hierarchical Access Control, 149
 - See* HAC
- Horus/Ensemble, 2
- IDC, 101
- IETF, 1, 9
- IHACS, 114, 119, 121
- ING Protocol, 37
- IP Multicast, 8
- IP, 1
- IRTF, 1
- ISP, 25
- Immediate Rekeying, 137–138
- Index Based Hierarchical Access Control Scheme
 - See* IHACS
- Indirectly Dependent Key Scheme, 106, 112
- Integrity, 2
- Inter-Area Rekeying, 137
- Interactive Game, 1
- Internet Engineering Task Force
 - See* IETF
- Internet Protocol
 - See* IP
- Internet Research Task Force
 - See* IRTF

- Internet Service Provider
 - See* ISP
- Interruption, 78, 81–82
- Interval Based Dynamic Conferencing, 97–98
- Interval Communication, 96
- Interval Multicast, 96
- Interval, 96–97, 99, 103
- Intra-Area Rekeying, 137
- Intractability, 5
- Intractable, 11
- Iolus, 24–26
- Isomorphic, 5
- K-Resilient Security, 4, 19
- KAG, 85–87, 89
- KDC, 23
- KEK, 23, 27–28, 50
- KTDC, 101
- Karnaugh Map, 75
- Key Agreement, 2, 147
- Key Association Group
 - See* KAG
- Key Distribution Binary Tree, 83
- Key Distribution Center
 - See* KDC
- Key Distribution, 92, 146
- Key Encryption Key
 - See* KEK
- Key Graph, 49
- Key Management, 2–3, 15, 91
- Key Manager, 144
- Key Node, 83
- Key Server, 29–30, 32
- Key Tree, 19, 49–50, 54, 57, 65–66, 73, 76–77, 82, 84, 91, 96–99, 102, 106, 122, 132, 150
- Key-Oriented Rekeying, 59, 64
- Known Plaintext Attack, 119
- LAN, 8
- LKH, 49–50, 65–67, 69, 72–73, 132, 137
- LSK, 26–28
- Lagrange Interpolation, 5
- Layer Key, 29–33
- Leader, 30
- Lin's Scheme, 112
- Local Area Network
 - See* LAN
- Local Subgroup Key
 - See* LSK
- Logarithmic Signature, 121
- Logical Key Hierarchy
 - See* LKH
- Logical Key Tree
 - See* LKH
- MAC, 124–125, 142
- MID, 73
- MSA, 3, 79, 130, 149
- MSEC, 1
- Man-In-The-Middle, 11–13, 79
- Master Keys, 111
- Matching Resistant, 3
- Member Discovery Protocol, 33–34
- Member Dynamics, 2, 115, 117, 122, 125
- Member Exclusion List, 23
- Member ID
 - See* MID
- Member Inclusion List, 23
- Member Overlay Tree, 33–34, 36
- Member Serialization, 37
- Member-Oriented Rekeying, 59–60, 62, 64
- Member-Oriented, 62
- Membership Control, 23
- Membership Management, 3, 149
- Message Authentication Code
 - See* MAC
- Message/Source Authentication
 - See* MSA
- Mixed Keying, 106–108
- Mobile Station, 131
- Mobility, 147
- Modern Cryptology, 1

- Modular Exponentiation, 119
- Modulo, 6
- Multicast Delivery Tree, 9
- Multicast SECURITY Working Group
 - See* MSEC
- Multicast Tree, 9, 21, 23, 33
- Multicast, 1, 8
 - Directed Multicast, 9
 - Multiple Multicast, 9
 - Scoped Multicast, 9
 - Subgroup Multicast, 9
- Multiple Multicast, 9
- Multiplicative Group, 7, 11
- Multiplicative Inverse, 6, 21, 112
- N-Party Diffie-Hellman Key Exchange, 18, 37, 149
- Neighboring Comparison, 56, 99
- Neighboring clusterhead discovery, 143
- Node Key, 65–66, 68, 83
- Node Secret, 65–66, 68, 83, 85
- Non-Occupied Position, 95
- OFC, 49–50, 66, 68–69
 - Node Secret, 66
- OFT, 49–50, 65–66, 68–72, 83
 - Blinded Node Secret, 65
 - Blinded Node Secret, 65–66
 - Node Key, 65–66
 - Node Secret, 65–66
- Off-Line, 79–80, 92–93, 102
- One-Way Function Chain
 - See* OFC
- One-Way Function Tree
 - See* OFT
- One-Way Function, 3–4, 6, 15, 20, 49, 65–66, 76, 79–84, 105–106, 108, 112–115, 119, 141
- One-Way Hash Function, 3
- One-Way-Function Tree, 49
- Outsider, 1
- PGM, 121
- PKDC, 149
- PKI, 137
- PKM, 144
- PKMs, 144
- POF, 79–80
- Pairwise Key, 29
- Participant, 1–2
- Pebble Network
 - See* Pebblenet
- Pebble, 140
- Pebblenet, 140, 142
- Periodic Rekeying, 14, 137, 139
- Permanent Private Share, 80
- Permutation Group Mapping
 - See* PGM
- Permutation, 8
- Plaintext Space, 7
- Plaintext, 7, 119
- Poisson, 135
- Potential Key Manager, 144
- Primitive Element, 5
- Privacy, 1
- Private Key, 7
- Private Share, 12, 45, 76–79
- Privilege, 105
- Probabilistic Algorithm, 7
- Public Directory, 10
- Public Key Based Scheme, 92
- Public Key Certificate, 11, 13
- Public Key Infrastructure
 - See* PKI
- Public Key, 7, 23
- Public One-Way Function
 - See* POF
- Public Share Certificate, 13
- Public Share, 12, 45, 76–79
- Public-Key Based Scheme, 18
- Public-Key Based System, 4
- Public-Key Cryptosystem, 7, 9–11, 17–20, 22, 93, 102–103, 125
- Published Diffie-Hellman, 12–13, 79
- Pure Delayed Rekeying, 139
- RPS, 20–22

- RSA Cryptosystem, 7
- RSA Signature, 79
- RSA, 7, 21
- Rampart, 2
- Random Number Generator, 120–121
- Real-Time Information Service, 1
- Rekeying, 24
- Relaying Message, 24
- Resource Multiple Keying, 106–108
- Reversible Parametric Sequence
 - See RPS
- Root Key, 82
- SGC with HAC, 2–3, 15, 105
- SGC with Hierarchical Access Control
 - See SGC with HAC
- SGC, 1, 3–4, 6, 8, 13–15, 17, 37, 49, 66, 72, 91, 105–106, 140, 145, 149
 - Lightweight, 131
- SGCS, 4
- SGM, 19, 25–26
 - Member SGM, 26–27
 - Participant SGM, 26–28
- SH, 132–134
- SK, 22, 73–74, 92–93
- SLDC, 149
- SMuG, 1
- SPREAD, 2
- STB, 20, 22–23
- STPC, 1–2
- STR Protocol, 45
- Scoped Multicast, 9
- Secret Key, 10–12, 81
- Secret Share, 95
- Secret Sharing, 7
- Secret-Key Based Scheme, 19
- Secret-Key Based System, 4
- Secret-Key Cryptosystem, 9–11, 19, 23, 102, 121
- Secure Group Communication Scheme
 - See SGCS
- Secure Group Communication
 - See SGC
- Secure Lock, 6, 93–94, 102–103
- Secure Multicast Research Group
 - See SMuG
- Secure SPREAD, 2
- Secure Transmission Backbone
 - See STB
- Secure Two-Party Communication
 - See STPC
- SecureRing, 2
- Security Requirement, 14
 - Backward Secrecy, 14–15
 - Forward Secrecy, 14
- Sensor Network, 146
- Serialization, 45, 147
- Session Key
 - See SK
- Shadow, 4
- Shadowholder, 4
- Shareholder, 5
- Shares Generator, 79
- Signatures, 2
- Single-Point-Of-Failure, 146
- Source Heartbeat Message, 34
- Split Operation, 52
- Sponsor, 77–78, 81–82
- Square-Multiply, 119
- Steer Protocol, 43
- SubGroup Manager
 - See SGM
- Subgroup Controller, 106, 118, 123, 125
- Subgroup Dynamics, 115–116, 122, 125
- Subgroup Key, 19
- Subgroup Multicast, 9, 59–60, 63–64
- Subgroup, 19, 24, 105, 113, 122
- Supervisor Host
 - See SH

- Symmetric Cryptosystem, 9–10, 19, 23
- Symmetric Polynomial, 3, 7–8, 94–96, 102–103
- TA, 17
- TEK, 50, 141, 143–144
- TESLA, 130
- TGDH, 50, 76, 78–80, 149
 - Blinded Key, 76–78
 - Secret Key, 76–77
- TIKM, 132
- TMKM, 132, 134, 139
- TTL, 87
 - TTL-Scoped Heartbeat Message, 34
 - TTL-Scoped Message, 34, 36
- Tamper-Resistant, 141, 145
- Tele-Medicine, 1
- Teleconferencing, 1
- Threshold Cryptosystem, 4–5
- Threshold Rekeying, 139
- Time-To-Live
 - See* TTL
- Timed Efficient Stream Loss-Tolerant Authentication, 130
- Topology Independent Key Management
 - See* TIKM
- Topology Matching Key Management
 - See* TMKM
- Traffic Encryption Key
 - See* TEK
- Transformation, 10
- Tree Based Group Diffie-Hellman Scheme
 - See* TGDH
- Tree-Based Key Management, 17
- Tree-Based SGC Key Management, 15
- Trusted Authority
 - See* TA
- Two Party Diffie-Hellman Key Exchange, 76
- Two-Party Communication, 2
- Unblinded Key, 83–84, 87–89
- Unconditional Security, 4
- Unconditionally Secure, 4, 19, 92, 94, 101–102, 105–107
- User Multiple Keying, 106–107
- User Threshold, 139
- User-Oriented Rekeying, 59
- User-Subtree, 133, 135
- VPN, 1, 150
- Virtual Private Network
 - See* VPN
- WAN, 150
- WLAN, 131
- WPAN, 131
- WTBR, 134, 139
- WWAN, 131
- Wait-To-Be-Removed List
 - See* WTBR
- Weight, 142, 144
- Wide Area Network
 - See* WAN
- Wireless Local Area Network
 - See* WLAN
- Wireless Network, 130–131
- Wireless Personal Area Network
 - See* WPAN
- Wireless Wide Area Network
 - See* WWAN