Contributions to group key distribution schemes

Hartono Kurnio

University of Wollongong

Follow this and additional works at: https://ro.uow.edu.au/theses

Recommended Citation

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au
Contributions to Group Key Distribution Schemes

A thesis submitted in fulfillment of the requirements for the award of the degree

Doctor of Philosophy

from

UNIVERSITY OF WOLLONGONG

by

Hartono Kurnio

School of Information Technology and Computer Science
January 2005
© Copyright 2005
by
Hartono Kurnio
All Rights Reserved
Dedicated to
my parents, Victor and Varia
Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

______________________________
Hartono Kurnio
21 January 2005
Group-oriented communication has grown considerably with the wide use of broadcasting and multicasting of media content. In most group-oriented applications, access to the communicated content must be restricted to authorised users. These applications include news feeds, Pay-TV and private teleconferencing systems.

A commonly used solution for controlling access in a group communication is to encrypt the content using a group key (session key). The group key is only known to the users in the authorised group. A group is called dynamic if the set of authorised group members changes in each session. The group key must be updated in each session to ensure only authorised users of the session can access the content. A group key distribution scheme provides algorithms to establish and maintain the group key.

The challenge is to design secure and efficient group key distribution schemes. Security means that the collusion of unauthorised users cannot obtain the group key. Efficiency is measured in terms of the required secure storage, communication bandwidth and computation effort to update the group key. Diverse group applications pose new challenges and designing group key distribution schemes that are tailored to specific group communication scenarios is of high importance.

In this thesis, we propose methods of constructing secure and efficient group key distribution schemes with several properties of high interest. We consider group key distribution schemes for completely decentralised environments, and propose secure and efficient constructions for group key distribution schemes where group management operations can be performed by either any group member or a collaboration of several group members. Both these settings have many applications in modern group communication systems. We show correctness of the proposed constructions, prove their security and assess their performance.
I give thanks for this work. I am most grateful to my supervisor, Professor Reihaneh Safavi-Naini, for guiding and encouraging me throughout this thesis. Without her inspiration and support, this work would have never been completed. I also thank my co-supervisor, Dr. Huaxiong Wang, who had been assisting me during his appointment with the University of Wollongong. Thanks also go to Dr. Luke McAven for his interest and collaboration during my research. I would also like to thank other members of Centre for Information Security and the management of School of Information Technology and Computer Science for their friendship and support. I am thankful to my family for their patience and understanding during my PhD, and motivating me all the way to reach this point.


## Contents

Abstract v

Acknowledgments vi

Thesis Related Publications vii

1 Introduction 1

1.1 Communication Security 1

1.2 Group Communication and Group Key Distribution 3

1.2.1 Benchmarks of Group Key Distribution Schemes 7

1.2.2 Other Security Objectives for Group Communication 9

1.3 An Overview of Existing Group Key Distribution Schemes 10

1.3.1 Trivial Solutions 10

1.3.2 Broadcast Encryption System 11

1.3.3 Key Predistribution System 14

1.3.4 Multicast Key Distribution System 16

1.3.5 Conference Key Distribution System 17

1.3.6 Secret Sharing Key Distribution System 20

1.4 Preliminary Cryptography 21

1.4.1 Hard Problems and Assumptions 21

1.4.2 Threshold Secret Sharing 22

1.4.3 Cumulative Scheme 23

1.4.4 Key Distribution Pattern 23

1.4.5 Erasure Code 25

1.4.6 Set Cover Problem 25

1.5 Thesis Inside 26

1.5.1 Aims and Objectives 26

1.5.2 Contributions and Structure 26
2 A Model of Group Key Distribution Systems

2.1 Introduction .......................................................... 29
2.2 The System Model ...................................................... 29
  2.2.1 Membership Events ................................................. 30
  2.2.2 Group Key Distribution Schemes ................................ 32
  2.2.3 Security Requirements ............................................. 33
  2.2.4 Efficiency Requirements .......................................... 36
  2.2.5 Aspects of Scheme ................................................. 37
2.3 A Summary of Notations .............................................. 38
2.4 Conclusion ............................................................ 39

3 A Secure Re-keying Scheme with Key Recovery Property

3.1 Introduction .......................................................... 40
  3.1.1 A Summary of Our Contribution ................................. 41
  3.1.2 Protocols .......................................................... 42
  3.1.3 Key Recovery Model ................................................. 43
3.2 An LKH Re-keying Scheme ............................................ 44
  3.2.1 Single-$K(T)$ ......................................................... 45
  3.2.2 Multi-$K(T)$ ......................................................... 47
3.3 Group Operations ...................................................... 50
  3.3.1 System Initialisation ................................................. 51
  3.3.2 User Eviction ....................................................... 51
  3.3.3 User Join .......................................................... 53
3.4 Key Recovery .......................................................... 56
3.5 Further Discussion and Conclusion .................................. 59

4 A Collaborative Re-keying Scheme

4.1 Introduction .......................................................... 60
  4.1.1 A Summary of Our Contribution ................................. 60
  4.1.2 Protocols .......................................................... 62
4.2 A Collaborative LKH Construction .................................. 63
  4.2.1 Updating Internal Keys ............................................. 66
4.3 Group Operations ...................................................... 71
  4.3.1 System Initialisation ................................................. 71
  4.3.2 User Eviction ....................................................... 73
  4.3.3 User Join .......................................................... 75
## 5 A Dynamic and Stateless Revocation Scheme

### 5.1 Introduction

- **5.1.1 A Summary of Our Contribution**
- **5.1.2 Protocol**

### 5.2 The Dynamic Scheme

- **5.2.1 System Initialisation**
- **5.2.2 User Revocation**
- **5.2.3 An Algorithm for Finding $T$**
- **5.2.4 Authentication**
- **5.2.5 A Centralised Protocol**

### 5.3 A Variant of Key Generation and Allocation

- **5.3.1 System Initialisation**
- **5.3.2 User Revocation**

### 5.4 Further Discussion and Conclusion

## 6 Group Key Distribution Schemes with Decentralised User Join – An Algebraic Approach

### 6.1 Introduction

- **6.1.1 A Summary of Our Contribution**
- **6.1.2 Protocols**

### 6.2 A Decentralised GKDS

- **6.2.1 System Initialisation**
- **6.2.2 Subgroup Event**
- **6.2.3 Join Event**
- **6.2.4 Evict Event**
- **6.2.5 Traceability**

### 6.3 The Extended Scheme

- **6.3.1 System Initialisation**
- **6.3.2 Subgroup Event**
- **6.3.3 Join Event**
- **6.3.4 Adding New Access Sets**

### 6.4 A Variant of the Extended Scheme

- **6.4.1 System Initialisation**
- **6.4.2 Subgroup Event**
6.4.3 Join Event ................................................. 125
6.5 Further Discussion and Conclusions ................................. 127

7 A Group Key Distribution Scheme with Decentralised User Join – A KDP Approach 129
  7.1 Introduction .................................................. 129
  7.1.1 A Summary of Our Contribution ............................. 129
  7.2 A KDP Construction ........................................... 131
     7.2.1 System Initialisation ....................................... 131
     7.2.2 Subgroup Event ............................................ 132
     7.2.3 Improving Communication Efficiency ....................... 136
     7.2.4 Join Event ................................................. 138
  7.3 Conclusion ..................................................... 142

8 Conclusion 143
  8.1 Fulfilment of Aims and Objectives ................................. 143
  8.2 Further Work .................................................. 145

A On the Security of Anzai et al. Scheme (Asiacrypt ’99) 146
  A.1 Introduction ...................................................... 146
  A.2 Security of Anzai et al. Scheme ................................ 146
     A.2.1 How to Repair Anzai et al. Scheme ......................... 147

Bibliography 149
# List of Tables

1.1 Efficiency comparison of the broadcast encryption schemes and the trivial schemes ............................................. 12
1.2 Cumulative array for $\Gamma^- = U_1U_2 + U_2U_3 + U_3U_4$ ................................................................. 23

2.1 Subgroup event ................................................................................................................................. 31
2.2 Join event ........................................................................................................................................ 31
2.3 Evict event ..................................................................................................................................... 32
2.4 Refresh event .................................................................................................................................. 32
2.5 A summary of main notations ......................................................................................................... 38

3.1 Efficiency comparison of the proposed scheme and the basic logical key hierarchy schemes ......................................................... 42
3.2 Join protocol ..................................................................................................................................... 43
3.3 Evict protocol .................................................................................................................................. 43
3.4 The re-keying algorithm to generate $M_{rkey}$ ............................................................................. 47
3.5 The re-keying algorithm to update user keys ............................................................................... 48

4.1 Efficiency comparison of the proposed scheme and several relevant schemes .............................................. 61
4.2 Join protocol ..................................................................................................................................... 62
4.3 Evict protocol .................................................................................................................................. 62
4.4 Procedures for updating internal keys ............................................................................................ 67
4.5 Case I, case II and case III protocols ............................................................................................. 68

5.1 Efficiency comparison of the proposed schemes and several relevant schemes 81
5.2 Subgroup protocol ......................................................................................................................... 81
5.3 An algorithm to find $T$ .................................................................................................................. 87

6.1 Join protocol ..................................................................................................................................... 101
6.2 Evict protocol .................................................................................................................................. 101
6.3 The subgroup protocol .................................................................................................................... 104
6.4 The join protocol ..................................................... 109
6.5 The join protocol of extended scheme .......................... 120
6.6 The join protocol of variant scheme ............................ 126

7.1 Efficiency comparison of the subgroup protocols and the join protocols
    proposed in this chapter and Chapter 6 .......................... 131
7.2 The OR protocol .................................................... 133
7.3 The AND protocol .................................................. 135
7.4 The join protocol ................................................... 139
# List of Figures

1.1 Two-party communication using encryption to provide message secrecy 2
1.2 Secure group communication using symmetric-key encryption to provide message secrecy 5
1.3 Events in dynamic group communication 6
1.4 An instance of Subset Difference revocation scheme 15
1.5 An instance of LKH key distribution scheme 18

2.1 A possible sequence of events in the system 33

3.1 An example of a tree structure for 8 users 45
3.2 Updated tree $T'_s$ when evicting users $U_3, U_5$ and $U_6$ 53
3.3 Updated tree $T'_s$ when joining new users $U_9$ and $U_{10}$ 55
3.4 The sets $P$ and $F$ in sessions $S_{s-1}, S_s$, and $S_{s+1}$ 57

4.1 An example of a tree structure for 8 users 64
4.2 Updated tree $T'_s$ for group $U_s \setminus R_s$ 75
4.3 Updated tree $T'_s$ for group $U_s \cup J_s$ 78

5.1 A tree structure of degree 2 for 8 users 83
5.2 A logical key hierarchy for Figure 5.1 83
5.3 The sets $I(U_s \setminus G_s)$ and $T$ when revoking users $U_3$ and $U_4$ 90
5.4 A tree structure with $d = 3$ and $h = 3$ for 9 users 92
1.1 Communication Security

The traditional model of communication is point-to-point and involves two parties communicating over a unicast channel. Protecting information transmitted over the channel has become critical as networks in general are open to attack. The most widely known security service for networks is confidentiality (secrecy). Confidentiality ensures that the message flowing between the sender and the receiver is unintelligible to outsiders. Encryption is the cryptographic operation required to provide secrecy. Encryption takes a message (plaintext) and transforms it into a cryptogram (ciphertext) using a secret cryptographic key (encryption key). Decryption is the reverse operation to encryption. The receiver who holds the correct secret key (decryption key) can recover the message from the cryptogram. An illustration is given in Figure 1.1 where the sender encrypts a message $m$ using the encryption key $K_e$. The sender sends the cryptogram $c = E_{K_e}(m)$, where $E_K()$ is the encryption algorithm and $K$ is the encryption key. The receiver uses the corresponding decryption key $K_d$ to decrypt the cryptogram $c$ to obtain the message $m = D_{K_d}(c)$, where $D()$ is the corresponding decryption algorithm. In computationally secure systems, it is computationally infeasible for the adversary to recover the plaintext $m$ from the ciphertext $c$ without knowing the decryption key $K_d$.

There are two categories of algorithms: symmetric-key and asymmetric-key algorithms. Symmetric-key algorithms are algorithms in which the encryption key $K_e$ can be easily determined from the decryption key $K_d$ and vice versa. (In most symmetric-key algorithms, $K_e = K_d$.) This setting requires the sender and the receiver to agree on a key before they can communicate securely, and so one of the major issues in this setting is to find an efficient method to establish the key. This problem is referred to as the key distribution problem. The security of a symmetric-key algorithm relies on
1.1. Communication Security

Figure 1.1: Two-party communication using encryption to provide message secrecy

the secrecy of the key, and revealing the key allows anyone to decrypt the ciphertexts. There are two types of symmetric-key algorithms: stream ciphers and block ciphers. Stream ciphers operate on a single symbol at a time while block ciphers operate on a group of symbols at a time. Most well-known symmetric-key algorithms, such as AES [65] and DES [33], are block cipher algorithms.

In asymmetric-key algorithms, the key $K_e$ used for encryption is different from the key $K_d$ used for decryption, and it is computationally infeasible to calculate $K_d$ from $K_e$. The encryption key $K_e$ can be made public while the decryption key $K_d$ remains secret. Anyone can use $K_e$ to encrypt a message which is decryptable by a person with the corresponding decryption key $K_d$. In this setting, the encryption key is often called the public key, and the decryption key is called the private key. There are several public-key encryption algorithms currently available, for examples RSA [74] and ElGamal [30].

Other communication security services are authenticity and non-repudiation. Authenticity may be considered for source or messages. Source authenticity, also called data origin authenticity, enables the receivers of messages to determine the true identity of the sender and guards messages against impersonation, substitution or spoofing. Message authenticity, also called message integrity, allows receivers to verify whether the received messages have been tampered with, and ensures that any modification of the received stream such as changing the order of transmitted messages, or deleting parts of messages are detected. Non-repudiation protects against the sender of a message claiming that he has not sent the message.

A common way to provide authenticity in symmetric-key systems is to use message authentication codes (MACs). MAC algorithms take a message and a secret key, and produce a fixed-size message digest (MAC). It must be infeasible to produce a
valid MAC without the knowledge of the secret key. If the originator and the receiver share the secret key, the receiver can calculate the same MAC and verify if it matches the MAC accompanying the message. If they are identical, the receiver can be certain about both the integrity of the message and the sender’s identity. Enabling authenticity in public-key systems is by using digital signatures. A signature scheme consists of two algorithms: a signing algorithm and a verification algorithm. The signing algorithm produces a signature for a message using a private key such that it is computationally infeasible for anyone without access to the private key, to produce the signature. The verification algorithm uses the corresponding public key and is used to verify authenticity of a signed message. If the output of the verification algorithm is true, the verifier can be sure about both the integrity and the origin of the message.

MACs are shared key primitives and cannot make distinction between the parties that share the key; and so do not provide non-repudiation. This is because sender and receiver can both generate a MAC using the shared key. Digital signatures, however, allow the distinction to be made and so can provide non-repudiation.

1.2 Group Communication and Group Key Distribution

Group communication has grown rapidly with the development of diverse group applications. Group applications include Pay-TV, multiparty teleconferencing, online video games and distance education. Data distribution applications such as news feeds, stock quotes, distributed databases, chat rooms, shared white boards and software updates have recently found wide popularity.

In traditional point-to-point communication only two users communicate. However in group communication the group size may vary from tens or hundreds in teleconferencing, to thousands in distance education systems and up to hundreds of thousands in Pay-TV systems. Communication in the group can originate from a single sender, or can be from several users communicating with one another. Communicating a message through a unicast channel requires the sender to send an individual copy of the message to each member of the group, while using broadcast or multicast channel, the sender can send the message to a group of users at the same time. In broadcast communication, the message reaches all users in the system and in multicast systems, multicast-enabled routers forward the message to all users who have subscribed to a
multicast group, hence reducing the number of message copies that traverse the network. Multicast communication can target the message to a specific group and results in a more efficient usage of bandwidth. Hence multicasting is the preferred mode of communication for most group communication services. The sender is not necessarily a group member and can simply direct the message to the group address.

The transmission medium in group communication is open and unauthorised users (adversaries) can eavesdrop the communication and learn the content of the transmitted message. In many group applications access to the content must be restricted to authorised users. Examples include Pay-TV and Pay-per-View systems where only subscribers of particular channels and programs, respectively, are allowed to receive the broadcast content. Similarly in distance education systems it is needed to ensure that only enrolled students can access lectures, and in private teleconferences such as board meeting and scientific discussions access to exchanged information must be restricted to participants. Other examples are access to online databases for banks and travel agencies, news feeds and stock quotes, and Pay-per-Use multiparty games.

The straightforward solution to restrict access is by using public-key encryption. The sender can send a copy of the message to each authorised user, encrypted using the user’s public key. This will have high communication cost if the size of the authorised group is large, and so is not suitable for secure group communication in large groups.

A commonly used technique to control access is to use a symmetric-key encryption algorithm to encrypt the content and then establish a shared group key (or a session key) among members of the authorised group that is unknown to outsiders. An illustration is given in Figure 1.2 where user $U_7$ sends an encrypted message $E_{GK}(m)$ to users $U_1, \cdots, U_6$ such that only receivers that possess the group key $GK$ can recover the message $m$.

For a secure encryption algorithm secrecy of the communication relies on the security of the group key. So the problem of securing group communication is equivalent to the problem of securely establishing a group key, that is, securely establishing a group key among members of an authorised group. This is a more challenging problem than securing point-to-point communication (one-to-one) because it involves multiple receivers and must take into account the dynamic nature of the group.

The change in the authorised group is by using the following basic events illustrated in Figure 1.3. It is assumed that the group has users $U_1, U_2, \cdots, U_7$.

\footnote{If the sender is a trusted authority TA (also named center, group controller or group manager in literature), it also needs to know the common key.}
1.2. Group Communication and Group Key Distribution

Subgroup Secure communication is for a subset of users. The subgroup members establish a group key which is unknown to users outside the subgroup. Other users can be viewed as if they are revoked from the group, and the revocation is temporary. Figure 1.3 shows a subgroup of $U_1, U_2, U_3$ users. This scenario occurs for example, in Pay-TV system where different channels have different subsets of subscribers and subsets vary from time to time, or in a virtual meeting of multiple organisations where members of each organisation may want to have secret discussion before making a decision.

Join New users join the group and so the group membership changes by the additional members. Secure communication in the enlarged group is made possible by forming a new group key known by all members of the new group. The new users can participate in future events but are not allowed to learn any communication in the system prior to their admission. Figure 1.3 shows users $U_8$ and $U_9$ join the group. This event happens in applications where the number of customers or clients grows. For example, new subscribers in Pay-TV system should be able to access the content, but only after subscription.

Evict Users are evicted from the group. Secure communication in the new group is by establishing a new group key among authorised users. The evicted users are revoked and will not be able to participate in future events. Figure 1.3 shows users

Figure 1.2: Secure group communication using symmetric-key encryption to provide message secrecy
1.2. Group Communication and Group Key Distribution

$U_6$ and $U_7$ permanently leaving the group. Scenarios reflecting this event include, Pay-TV systems where subscribers that contribute to the making of pirate copies of the content or pirate decoders must be completely expelled from the system, in battlefields where compromised/corrupted devices must be excluded from the system, and finally in virtual education systems where graduated students should not be in the system.

**Refresh** In many cases, there is a need to change a group key without revoking or adding users. Figure 1.3 shows users $U_1, U_2, \ldots, U_7$ sharing a new group key. This event is needed when the usage of a group key has a time limit, or is limited by the amount of data encrypted using the key. This requires a group key refreshing function. Scenarios that require this event include, when a group member accidentally reveals the group key and an immediate key update for the same group is needed, and granting a temporary membership to a potential user before his join to the group, and giving him the group key. The user’s access can later be ended by simply changing the group key without changing group membership.

Dynamic group communication requires the group key to be updated each time an event occurs, to ensure only authorised users can access the content. A group key distribution scheme provides algorithms to establish and maintain the group key in dynamic environments, and consists of the following algorithms.
1. An algorithm to initialise the system whereby parameters are chosen, and secret information (a set of keys) is generated and distributed to users in the initial group. Each user securely stores the individual secret information that will be used to participate in future group operations.

2. An algorithm to securely and efficiently update the group key for each event. For user admission, this includes a method of giving individual secret information (a set of keys) to new users to allow them to participate in group operations. For user eviction, this includes a method of disabling individual secret information (a set of keys) belonging to evicted users that will prevent them from participating in group operations after their evictions.

A group key distribution scheme is said to have $c$-resilience if up to $c$ colluders are unable to find the group key. It is desirable to provide security for an arbitrary size groups against large collusions.

Efficiency of group key distribution schemes is determined by several parameters as follows: (i) communication overhead, which is the length of the message transmitted in key update, (ii) storage overhead, which is the size of secure memory required by users to store their secret information, and (iii) computation overhead, which is the computing effort required by a user to determine the group key. Minimizing the overheads is crucial. The message transmission should consume low bandwidth and take the advantage of efficient broadcast or multicast channel to scale well for large groups. The required secure storage and computing effort should be low to accommodate devices that have limited secure storage, such as smartcards (used in Pay-TV system), and limited computing power. Minimizing the overheads also provides faster group key update that is required in real-time applications, such as Pay-TV, private teleconferencing, and military command and control in which group operations must not be disrupted. Fast key update is also important for group applications in ad hoc environment where connections among devices typically last for a short time period. Group key distribution systems usually have tradeoff among various overheads.

### 1.2.1 Benchmarks of Group Key Distribution Schemes

Group applications are diverse and group key distribution must be tuned for the applications. Solutions can be generally divided into two classes: centralised and decentralised.
1.2. Group Communication and Group Key Distribution

Centralised Setting

There is a single trusted entity that decides and manages the system, including system setup and key update. This setting is mostly suitable for one-to-many group applications such as file distribution where there is a single sender and a large number of receivers. The sender could act as the group manager, or a trusted third party can be used to control access in the group.

The trusted entity is typically a top-end machine with abundant resources. The recipients are typically low-end heterogeneous machines with constrained resources. Accordingly, group key distribution solutions should have optimum efficiency at the recipient side, while overheads of the trusted entity must remain feasible.

An important concern of this model is potential performance bottleneck. Possible drawbacks include (i) a single point of failure where the entity could be unavailable or unreachable (because of congestion or overload), and (ii) a single point of attack where compromising the trusted entity means divulging all the system secrets. These drawbacks will bring the entire system down.

Decentralised Setting

The system management role is distributed to several trusted entities and security task may be performed by a single entity, or a collaboration of a subset of the trusted entities. This setting is suitable for many-to-many group applications with multiple senders. For instance, in interactive applications such as virtual conferencing and collaboration work, any group member may wish to securely send data to a subgroup of his choice.

This setting alleviates the problems of the centralised setting and provides a suitable solution for ad hoc wireless networks wherein no entity can be assumed to be present all the time. Decentralised models allow ad hoc systems to provide continuous security for mobile users despite rapidly changing network topology.

In this setting nodes often have roughly similar resources. Group key distribution systems in this setting tend to be more complex compared to those of centralised setting, and it is important that the solution be sufficiently efficient for the involved entities.
1.2.2 Other Security Objectives for Group Communication

The main security goal in group communication is controlling access and ensuring that only authorised users can access the content. Access control requires secrecy of communication. Communication secrecy in dynamic environments is obtained by establishing a group key that is shared by the authorised users. Other desired security goals are authenticity and traceability described below.

Authenticity

Group authenticity allows each group member to recognise whether a message was sent by a group member, however, the source of the message may not be known to the other members. With source authenticity, it is possible to identify the particular sender within the group, and the origin of messages when the originator is not a group member.

A group key shared by group members can be used to generate MAC that is used for group authenticity, but is inadequate for source authenticity since a shared key cannot be used to differentiate potential senders in the group. There are two basic approaches to providing source and message authenticity: using public key signatures or using message authentication codes. Signing every message is costly as signatures are typically long and computing and verifying a signature results in significant computational overhead. Signature amortization improves efficiency of the system. In signature amortization a single digital signature is used for authentication of multiple messages (see [34, 68, 88] for examples). Graph-based authentication generalises the idea of signature amortization in such a way as to tolerate message losses (see [60, 70] for examples). An alternative to public key signatures is message authentication codes (see [14, 17] for instances). MACs are generated using individual key sets of users instead of the group key. This approach is usually more efficient than digital signature approach.

Traceability

Traceability can be motivated by looking at the Pay-TV systems. In these systems, a subscriber possesses a decryption box. The broadcasting center encrypts digital content and broadcasts it to all subscribers that will use their decoders to decrypt the content. It is possible that some subscribers collude to produce a pirate decoder. The pirate decoder, that is not registered with the center, can decrypt the encrypted digital
content. Given a pirate decoder, traitor tracing provides a method for the center to recover the identity of a subscriber that participated in the construction of the decoder (traitor). The traceability would discourage piracy in the system.

It is assumed that each subscriber can recover the decryption key stored in his decoder, and a subset of subscribers can combine their keys to construct a pirate decoder. An open box traitor tracing scheme allows the center, given the content of a pirate decoder, to recover one of the traitors’ keys. A traitor tracing is $t$-collusion resistant if in any collusion of up to $t$ colluders at least one of them can be identified.

In black box tracing, the tracer does not have access to the keys inside the pirate box and can only query the box and observe the response. A lot of research has been done to construct traitor tracing schemes with various properties, see for examples [15, 20, 46, 62, 63, 64].

1.3 An Overview of Existing Group Key Distribution Schemes

Group key distribution schemes have been studied in the context of (i) broadcast encryption system, (ii) key predistribution system, (iii) multicast key distribution system, (iv) conference key distribution system, and (v) secret sharing key distribution system. In this section we briefly describe these systems.

1.3.1 Trivial Solutions

The group key establishment problem can be trivially solved by the following methods.

Consider a system wherein each user shares a key with a trusted authority. To establish a new group key for a selected subset of users (subgroup), the trusted authority randomly generates the group key, encrypts it using the keys shared with all subgroup members and individually sends each encrypted version to the corresponding member. This solution only requires a small amount of storage as each user holds just a single key. However, it requires a large amount of communication as the number of transmissions is equal to the number of subgroup members. Also, the response time may be unacceptable since it is necessary to repeatedly send the encrypted group key, one to each subgroup member, which consumes a considerable amount of time. This solution can be applied to a decentralised model that assumes a shared key exists between each pair of users in the group. In this case, any user can take the initiative to generate and
distribute the group key.

Consider another system wherein every possible subset of users is assigned a key, and each user holds the keys for all subsets in which he is a member of. To establish a new group key for a selected subset of users (subgroup), the trusted authority randomly generates the group key, encrypts it using the key associated with the subset and broadcasts the encrypted version to the group. This solution only requires one transmission and so the communication length is short. However, it requires a large amount of storage as each user holds all keys associated with the subsets to which he belongs. This solution can be performed in a decentralised fashion where any user can assume the role of the trusted authority.

Consider the event of a new user admitting to the group. To establish a new group key for the enlarged group, a trusted authority randomly generates the group key, gives it directly to the new user and broadcasts the new group key encrypted with the old group key. In this way the new group key is accessible to all authorised users, however, the trusted authority will be required to directly give every future group key to the new user. This is because the new user does not have the individual secret information that is required to participate in group key update algorithms. If the number of new users is large, this solution is essentially the same as the straightforward solution of individually giving a group key to authorised users that incurs in high communication cost.

1.3.2 Broadcast Encryption System

In broadcast encryption system, there are a collection $G \subseteq 2^U$ of authorised subsets and a collection $H \subseteq 2^U$ of subsets of colluders. A broadcast encryption scheme enables a center to securely broadcast data to $G \in G$ while preventing any collusion $H \in H$, where $G \cap H = \emptyset$, from finding the data. Broadcast encryption system is introduced by Fiat and Naor [31]. Their proposed schemes enable a single source to securely broadcast to an arbitrary subset of users from a group $U$, and the system is secure against any collusion of at most $t$ unauthorised users. The system can be used to send a group key to a subset of users (subgroup) and so provides a solution to group key distributions. The communication overhead of the centre [31], assuming the existence of one way function, is $O(t^2 \log^2 t \log n)$ keys and each user has to store $O(t \log t \log n)$ keys, where $n = |U|$. Broadcast encryption schemes are further studied by Blundo et al. [10, 11] and Stinson et al. [84] who consider unconditionally secure model and give lower and upper bounds on communication overhead and key storage of the system. Luby and
1.3. An Overview of Existing Group Key Distribution Schemes

Staddon [55] use combinatorial methods to show the tradeoff between the number of keys held by each user, and the number of transmissions required to establish a new group key. Other works on broadcast encryption include [1, 48, 50, 51, 75].

Recently, Naor et al. [62] proposed two subset-cover revocation schemes: Complete Subtree (CS) method and Subset Difference (SD) method. Both methods put users at the leaves of a full binary tree and utilise the tree structure and pseudo-random sequence generators to generate individual key sets for users. Both schemes can be used to revoke an arbitrary number of users and collusion of all revoked users cannot compromise the system secrecy. The notion of stateless receivers is introduced in [62], that is, users considered in both schemes are not required to change their individual key sets, and the keys given at the initial stage are used for the entire system lifetime. To revoke \( r \) users, the communication and storage overheads in CS method are \( r \log\left(\frac{n}{r}\right) \) and \( \log n + 1 \) keys, respectively, while in SD method are \( 2r - 1 \) and \( \frac{1}{2} \log^2 n + \frac{1}{2} \log n + 1 \) keys, respectively. Halevi et al. [37] proposed Layered Subset Difference (LSD) method to improve the storage overhead of SD method to \( O(\log^{1+\epsilon} n) \) keys for small \( \epsilon > 0 \) while the communication overhead is equivalent to \( O(\frac{r}{n}) \) keys. Other variants of CS, SD and LSD methods can be found in [4, 5, 7, 8, 27, 39].

Table 1.1 compares the efficiency of the broadcast encryption schemes and the trivial solutions described in Section 1.3.1. The computation cost is measured in terms of the number of decryptions.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>User Storage</th>
<th>Communication</th>
<th>Computation</th>
<th>Collusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS [62]</td>
<td>( \log n + 1 )</td>
<td>( r \log\left(\frac{n}{r}\right) )</td>
<td>1</td>
<td>( n )</td>
</tr>
<tr>
<td>SD [62]</td>
<td>( \frac{1}{2} \log^2 n + \frac{1}{2} \log n + 1 )</td>
<td>( 2r - 1 )</td>
<td>1</td>
<td>( n )</td>
</tr>
<tr>
<td>LSD [37]</td>
<td>( O(\log^{1+\epsilon} n) )</td>
<td>( O\left(\frac{r}{n}\right) )</td>
<td>1</td>
<td>( n )</td>
</tr>
<tr>
<td>First Trivial</td>
<td>1</td>
<td>( n - r )</td>
<td>1</td>
<td>( n )</td>
</tr>
<tr>
<td>Second Trivial</td>
<td>2( n - 1 )</td>
<td>1</td>
<td>1</td>
<td>( n )</td>
</tr>
</tbody>
</table>

Broadcast encryption is found in many applications including Pay-TV system and distribution of copyrighted materials. Below we briefly describe the broadcast encryption scheme in [62].
1.3. An Overview of Existing Group Key Distribution Schemes

Subset-Cover Revocation Scheme [62]

We describe the SD method of subset-cover revocation scheme. We assume there are
\( n \) users (assume \( n \) is a power of 2), \( \{U_1, \ldots, U_n\} \), in the group. The tree consists of \( n \) leaves and \( 2n - 1 \) nodes (leaves plus internal nodes). To initialise the system, a trusted
authority defines a collection of user subsets, generates secret keys for the subsets and
gives a portion of secret keys to each user through a secure channel as follows.

**Subset description:** A subset \( S_{\ell,\omega} \) is specified by two nodes (\( \ell, \omega \)), and defined as
\[ S_{\ell,\omega} = S_\ell \setminus S_\omega, \]
where \( S_\ell \) and \( S_\omega \) are the sets of users associated with the leaves of the
subtrees rooted at nodes \( \ell \) and \( \omega \), respectively. The collection consists of subsets \( S_{\ell,\omega} \)
for all internal nodes \( \ell \) in the tree and for all descendants \( \omega \) of node \( \ell \). Each subset
\( S_{\ell,\omega} \) is assigned a key \( K_{\ell,\omega} \) and each user knows the keys corresponding to the subsets
that contain him. Observe that each user belongs to \( \mathcal{O}(n) \) subsets.

**Key generation:** The method uses a publicly known pseudo-random sequence gen-
erator \( G : \{0,1\}^b \rightarrow \{0,1\}^{3b} \) whose output length is three times the length of the input,
and the concept of label. Let \( G_L(r), G_M(r) \) and \( G_R(r) \) respectively denote the left,
middle and right third of the output of \( G \) on seed \( r \). Also, let \( P(\ell), LC(\ell) \) and \( RC(\ell) \)
respectively denote the parent, left child and right child node of a node \( \ell \). The trusted
authority does as follows.

1. For each internal node \( \ell \) in the tree, chooses a random element \( r_\ell \in \{0,1\}^b \) as
the label \( LABEL_\ell \) for node \( \ell \). The label for subset \( S_{\ell,LC(\ell)} \) is
\( LABEL_{\ell,LC(\ell)} = G_L(LABEL_\ell) \) and the label for subset \( S_{\ell,RC(\ell)} \) is
\( LABEL_{\ell,RC(\ell)} = G_R(LABEL_\ell) \). Labels for subsets \( S_{\ell,LC(\omega)} \) and \( S_{\ell,RC(\omega)} \) for each descendant \( \omega \) of node \( \ell \) are
generated by the same way: the label for subset \( S_{\ell,LC(\omega)} \) is
\( LABEL_{\ell,LC(\omega)} = G_L(LABEL_{\ell,\omega}) \); the label for subset \( S_{\ell,RC(\omega)} \) is
\( LABEL_{\ell,RC(\omega)} = G_R(LABEL_{\ell,\omega}) \).
The key for each subset \( S_{\ell,\omega} \) is \( K_{\ell,\omega} = G_M(LABEL_{\ell,\omega}) \).

2. Sends to each user \( U_i \) the labels \( LABEL_{\ell,\omega} \) for every node \( \ell \) along the path from
the root to \( U_i \)'s leaf and for every descendant \( \omega \) of node \( \ell \) just hanging off the
path over a secure channel. User \( U_i \) securely stores the labels that allow him
to derive all other labels for the subsets that contain him. (Note that a label
\( LABEL_{\ell,\omega} \) can be used to derive labels \( LABEL_{\ell,v} \), for all descendants \( v \) of node
\( \omega \).) Subsequently, the user can determine all secret keys corresponding to subsets
to which he belongs.

The trusted authority does the following to temporarily revoke users \( U_{i_1}, \ldots, U_{i_m} \),
\( m \leq n \), from the group, and to establish a group key \( K \) for the \( n - m \) authorised users.
1.3. An Overview of Existing Group Key Distribution Schemes

1. Partitions the authorised users into a collection of disjoint subsets, called the cover. The algorithm for finding the cover is as follows. Let \( ST(U_{i_1}, \ldots, U_{i_m}) \) be the (directed) Steiner Tree \([21]\) induced by leaves of \( U_{i_1}, \ldots, U_{i_m} \) and the root, and initially let a tree \( T = ST(U_{i_1}, \ldots, U_{i_m}) \).

(a) Finds two leaves \( \ell \) and \( \omega \) in \( T \) such that the least-common-ancestor \( v \) of \( \ell \) and \( \omega \) does not contain any other leaf of \( T \) in its subtree. Suppose \( v_1 \) and \( v_2 \) are the two children of \( v \) such that \( \ell \) is a descendant of \( v_1 \) and \( \omega \) is a descendant of \( v_2 \). (If there is only one leaf left, make \( \ell = \omega \) to the leaf, \( v \) to be the root of \( T \) and \( v_1 = v_2 = v \).)

(b) If \( v_1 \neq \ell \) then adds the subset \( S_{v_1, \ell} \) to the cover. Also, if \( v_2 \neq \omega \) then adds the subset \( S_{v_2, \omega} \) to the cover.

(c) Updates \( T \) by removing all descendants of \( v \) from \( T \) and making \( v \) as a leaf.

(d) If \( T \) consists of just a single node then stops, otherwise repeats the steps.

2. Randomly chooses the group key \( K \), encrypts \( K \) with keys \( K_{\ell, \omega} \) corresponding to all subsets \( K_{\ell, \omega} \) in the cover and broadcasts the ciphertexts to the group.

Each authorised user \( U_i \) first finds the subset \( S_{\ell, \omega} \) where \( U_i \in S_{\ell, \omega} \) and \( S_{\ell, \omega} \) is in the cover, computes the secret key \( K_{\ell, \omega} \) and decrypts the corresponding ciphertext to obtain \( K \). Figure 1.4 gives an instance of the system.

**Left Figure:** A tree structure for a group of 8 users. The subset \( S_{1,5} = S_1 \setminus S_5 = \{U_1, \ldots, U_8\} \setminus \{U_3, U_4\} = \{U_1, U_2, U_5, \ldots, U_8\} \). Internal node 1 is assigned label \( LABEL_1 = r_1 \). The label for \( S_{1,5} \) is derived from \( r_1 \) as follows: \( LABEL_{1,5} = G_R(LABEL_{1,2}) = G_R(G_L(r_1)) \). The secret key for \( S_{1,5} \) is \( K_{1,5} = G_M(LABEL_{1,5}) \). After defining all subsets and their labels, for example, user \( U_1 \) receives labels \( \{LABEL_{1,3}, LABEL_{1,5}, LABEL_{1,9}, LABEL_{2,5}, LABEL_{2,9}, LABEL_{4,9}\} \).

**Right Figure:** The cover when revoking \( U_3, U_4 \) and \( U_7 \) is \( (S_{2,5}, S_{3,14}) \). The group key \( K \) is encrypted with keys \( K_{2,5} \) and \( K_{3,14} \). Observe that all authorised users can recover \( K \).

1.3.3 Key Predistribution System

A key predistribution scheme, such as \([57, 83]\), allows a trusted authority to distribute individual key information to a set of users in such a way that in a later time each
user of an authorised subset (subgroup) can compute a common group key without any interaction. In a \((t, n)\) key predistribution scheme, any subgroup of \(t\) out of \(n\) users can compute a common key. A \((t, n)\) key predistribution scheme is called \(k\)-secure if group keys are secure against collusion of up to \(k\) users, that is, even if \(k\) users pool their individual key information they cannot compute anything about the group key of any \(t\)-subset of other users. Existing constructions of key predistribution systems either follow Blom’s construction using symmetric polynomials over finite fields [9, 12] and can lead to information-theoretically optimal schemes, or use key distribution patterns (KDP) [61]. The KDP approach is a combinatorial in nature and can be used for any cryptographic keys. Other constructions using key predistribution schemes can be found in [83, 84]. Below we briefly describe the key predistribution scheme in [9].

**Blom Key Predistribution [9]**

We describe 1-secure \((2, n)\) of Blom’s scheme. Let \(p\) be a large prime number and publicly known. Suppose the group has \(n\) users, \(\{U_1, \ldots, U_n\}\). A trusted authority initialises the system as follows.

1. Chooses three random numbers \(a, b, c \in \mathbb{Z}_p\) (the set of integers modulo \(p\)), and forms the polynomial \(F(x, y) = a + b(x + y) + cxy \mod p\).

2. For each user \(U_i\), chooses a unique number \(r_i \in \mathbb{Z}_p\) and publishes \(r_i\), computes a polynomial \(G_i(x) = F(x, r_i)\) and sends \(G_i(x)\) to \(U_i\) over a secure channel.

Later on, any two users \(U_i\) and \(U_j\) in the system establish a common key \(K\) as follows.

1. \(U_i\) computes \(K = G_i(r_j) = F(r_j, r_i)\).
2. $U_j$ computes $K = G_j(r_i) = F(r_i, r_j)$.

### 1.3.4 Multicast Key Distribution System

Multicast key distribution schemes, independently introduced by Wallner et al. [85] and Wong et al. [87], allow a group controller to form a new group by evicting all the users that should not be in the new group, and establish a new group key for the authorised users. The system employs a logical key hierarchy (LKH) in which each node in a tree structure corresponds to a key and each leaf corresponds to a user. The LKH is used to allocate keys to users and the tree root corresponds to the group key. To evict a user, the group controller multicasts a message such that all authorised users can update their keys that are also known by the evicted one (called re-keying). Each re-keying is stateful that requires all authorised users to change their key sets. When being applied to a group of $n$ users, the schemes in [85, 87] require a user to store $\log_d n + 1$ keys and eviction of a single user has communication cost equivalent to $d \log_d n + 1$ keys, where $d$ is the degree of the tree (scheme [85] uses $d = 2$, i.e., binary tree).

The schemes are refined in [17, 58] that use one-way function and pseudo-random generators, respectively, to reduce the communication overhead to $(d-1) \log_d n + 1$ keys. The tradeoff between communication overhead and user storage is studied in [18]. The schemes are primarily designed to support eviction of a single user and so to form an arbitrary new group, the eviction procedure may be repeatedly invoked. This system is secure against arbitrary number of colluders. The schemes in [58, 85, 87] also consider user admission. When a new user joins the group, a secure channel is employed to deliver the secret information to the new user, and a similar re-keying method as that of user eviction is used to establish a new group key for the new group. Other works on multicast key distribution system can be found in [23, 42, 52, 71, 76]. Below we briefly describe the multicast key distribution scheme in [85].

**Wallner-Harder-Agee Key Distribution [85]**

Let $\{U_1, \ldots, U_n\}$ be $n$ users in the multicast group. Without loss of generality, we assume $n$ is a power of 2. A trusted authority initialises the system as follows.

1. Generates a rooted full binary tree structure with $n$ leaves, associates every user to a leaf, and generates a key $K_{\ell}$ for every node $\ell$ in the tree.

2. Sends to every user the keys associated to the nodes along the path connecting the user to the root through a secure channel. Those keys are securely stored by
the user. The root key $K_0$, which is known to all users, is the common key for the group.

At some stage, the trusted authority wishes to evict a user $U_j$ from the group and allow the other members to share a new group key $K = K'_0$. The trusted authority does as follows.

1. Generates a new key $K'_\ell$ for every node $\ell$ along the path connecting $U_j$ to the root (except the leaf associated to $U_j$).

2. Encrypts new key $K'_{P(U_j)}$ with key $K_{S(U_j)}$, where $P(U_j)$ and $S(U_j)$ respectively denote the parent and the sibling of $U_j$.

3. For every other node $\ell$ along the path connecting $U_j$ to the root (except the root), encrypts new key $K'_{P(\ell)}$ with keys $K'_\ell$ and $K_{S(\ell)}$.

4. Multicasts all encrypted new keys.

It is obvious that each user in the new group can decrypt only the keys he is authorised to receive. The user will update his keys that are also known by $U_j$ with the new keys. The scheme can be applied repeatedly to evict multiple users. An instance of the system is given in Figure 1.5.

1.3.5 Conference Key Distribution System

A conference key distribution system, such as [16, 40], allows an arbitrary group of users to establish a common conference key. Contributions from all group members are required to form a conference key. In several rounds, each group member needs to generate partial key information and send it over a public channel. The conference key is determined as a function of the partial information provided by all group members.
There has been a lot of effort to extend the conference key distribution schemes, for example [43, 44], but most of them are for static environments where membership changes requires a new run of the system setup to establish a new conference key. The work in [80, 81] first addresses dynamic membership issues in group key agreement system. They propose a family of Group Diffie-Hellman (GDH) or Clique protocols, based on straightforward extensions of the two-party Diffie-Hellman key agreement protocol, that can be used to update the conference key in a nontrivial way when membership changes. Most of conference key distribution schemes are suitable for groups with small sizes, otherwise it typically involves excessive communication and computation overheads. The constructions that are efficient for large groups are independently proposed in [47] and Chapter 4. Conference key distribution system is an important primitive for secure collaborative and distributed applications such as teleconferences and replicated servers where many users may send data within the group. Below we briefly describe the conference key distribution schemes in [16] and in [81].

**Burmester-Desmedt Key Agreement** [16]

Let $p$ be a large prime, $g$ be a generator for $\mathbb{Z}_p^*$, the multiplicative group of integers modulo $p$, and $p$ and $g$ be public. Suppose the group consists of $n$ users, $\{U_0, \ldots, U_{n-1}\}$, and all identifiers $i$ of $U_i$ are taken modulo $n$. The system initialisation is as follows.

1. Each $U_i$ randomly generates an integer $r_i$, $1 \leq r_i \leq p - 2$, computes $z_i = g^{r_i} \mod p$, and broadcasts $z_i$ to the group.

2. Each $U_i$, after receiving $z_{i-1}$ and $z_{i+1}$, computes $V_i = (\frac{z_{i+1}}{z_{i-1}})^{r_i} \mod p$, and broadcasts $V_i$ to the group.

3. Each $U_i$, after receiving $V_j, 1 \leq j \leq n, j \neq i$, computes the group key $K = K_i$ as

$$K_i = (z_{i-1})^{nr_i} \times V_i^{n-1} \times V_{i+1}^{n-2} \times \cdots \times V_{i+(n-3)}^2 \times V_{i+(n-2)}^1 \mod p$$
\[ K_i = g^{\sum r_i + r_i^2 + r_i^3 + \ldots + r_i^{n-1}} \mod p. \]

When the group membership changes, a new conference key is established by repeating the steps described above.

**Clique Key Agreement** [81]

Let \( p \) be a large prime, \( g \) be a generator for \( \mathbb{Z}_p^* \) (the multiplicative group of integers modulo \( p \)) and \( p, g \) be publicly known. Suppose the group consists of \( n \) users, \( \{U_1, \ldots, U_n\} \). We describe system initialisation using the Initial Key Agreement 1 (IKA.1) protocol of Cliques as follows.

1. For \( i = 1, \ldots, n-1 \), \( U_i \) randomly generates an integer \( r_i \), \( 1 \leq r_i \leq p-2 \), computes 
   \[ z_i = g^{\prod_{j=1}^{i-1} r_j} \mod p, \quad z_{i,k} = g^{\prod_{j=1, j \neq k}^{i-1} r_j} \mod p \]
   for \( 1 \leq k \leq i \), and sends \( z_i, z_{i,k} \) for \( 1 \leq k \leq i \) to \( U_{i+1} \).

2. \( U_n \) randomly generates an integer \( r_n \), \( 1 \leq r_n \leq p-2 \), computes 
   \[ z_{n,k} = g^{\prod_{j=1, j \neq k}^{n-1} r_j} \mod p \]
   for \( 1 \leq k \leq n \), and broadcasts \( z_{n,k} \) for \( 1 \leq k \leq n \) to the group.

3. Each \( U_i \) computes the group key \( K = z_n \) as 
   \[ z_n = g^{\prod_{j=1}^{i-1} r_j} \mod p. \]

Suppose a member \( U_d \) is evicted from \( \{U_1, \ldots, U_n\} \). Any member \( U_c, c \neq d \), that remembers the most recent broadcast key information (in this case, \( z_{n,k} \) for \( 1 \leq k \leq n \)) can update the conference key \( K \) as follows: randomly generates an integer \( r'_d \), \( 1 \leq r'_d \leq p-2 \), computes 
   \[ z'_{n,k} = g^{r'_d \prod_{j=1, j \neq k}^{n-1} r_j} \mod p \]
   for \( 1 \leq k \leq n \), and sends \( z'_n, z'_{n,k} \) for \( 1 \leq k \leq n \) to \( U_{n+1} \). Each member \( U_i \) of the new group computes the new conference key \( K' \) as 
   \[ z'_n = g^{r'_d \prod_{j=1}^{i-1} r_j} \mod p. \]

Suppose a new member \( U_{n+1} \) joins \( \{U_1, \ldots, U_n\} \). Any member \( U_c \) that remembers the most recent broadcast key information (in this case, \( z_{n,k} \) for \( 1 \leq k \leq n \)) does as follows: randomly generates an integer \( r'_c \), \( 1 \leq r'_c \leq p-2 \), computes 
   \[ z'_{n,k} = g^{r'_c \prod_{j=1, j \neq k}^{n-1} r_j} \mod p \]
   for \( 1 \leq k \leq n \), and sends \( z'_n, z'_{n,k} \) for \( 1 \leq k \leq n \) to \( U_{n+1} \). The new member \( U_{n+1} \) does as follows: randomly generates an integer \( r_{n+1} \), \( 1 \leq r_{n+1} \leq p-2 \), computes 
   \[ z'_{n+1,k} = g^{r'_{n+1} \prod_{j=1, j \neq k}^{n} r_j} \mod p \]
   for \( 1 \leq k \leq n+1 \), and broadcasts \( z'_{n+1,k} \) for \( 1 \leq k \leq n+1 \) to the group. Each member \( U_i \) of the enlarged group computes the new conference key \( K' \) as 
   \[ z'_{n+1} = g^{r'_{n+1} \prod_{j=1}^{i} r_j} \mod p. \]
1.3.6 Secret Sharing Key Distribution System

Secret sharing key distribution schemes are independently introduced in [2, 64]. The schemes stem from the combination of threshold secret sharing schemes and Diffie-Hellman type key exchange protocols. For a group of $n$ users, the schemes allow revocation of up to a threshold number, say $t$, users and so the formed subgroup has size at least $n - t$ users. The secret sharing scheme\footnote{In a $(t, w)$ secret sharing scheme, a secret is divided into $w$ shares such that the secret can be recovered from any $t + 1$ shares and less than $t + 1$ shares give no information about the secret (see Section 1.4.2).} is used to divide the system secret into $w$ shares, where $w \geq n + t$, and each user receives a share of the system secret. To form a group key for a user subgroup, shares held by all revoked users and some extra shares are used as parts of a broadcast message. Each authorised user can combine his share with the broadcast to compute the group key, while collusion of all revoked users does not have enough shares to obtain any information about the group key. Extensions of the schemes can be found in [24, 28, 49, 53]. Below we briefly describe the secret sharing based scheme in [2].

Anzai-Matsuzaki-Matsumoto Key Distribution [2]

Let $p, q$ be large primes such that $q \mid (p - 1)$ and let $g$ be a generator of a multiplicative subgroup of $\mathbb{Z}_p^*$ with order $q$. The system is bounded by a parameter $t$ as the threshold of users to be revoked. Suppose the group consists of $\{U_1, \ldots, U_n\}$ users. A trusted authority, who is responsible for system initialisation, performs the following steps.

1. Chooses a system secret key $S \in \mathbb{Z}_q$, $S \neq 0$, and constructs the polynomial,
   \[
   F(x) = S + \sum_{j=1}^t a_j x^j \mod q,
   \]
   where $a_j$ are randomly chosen from $\mathbb{Z}_q$.

2. Computes shares $s_i = F(i)$ and gives $s_i$ to users $U_i$, for $1 \leq i \leq n$, through a secure channel ($i$ and $s_i$ are the user’s identifier and secret information, respectively). The trusted authority also computes shares $s_i = F(i)$, for $n + 1 \leq i \leq n + t$, and keep them as auxiliary shares.

3. Publishes the public keys $y_i = g^{s_i} \mod p$, for $1 \leq i \leq n + t$.

Any user can revoke other $m$, $m \leq t$, users to form a subgroup of $n - m$ remaining users. Without loss of generality, suppose $U_{i_1}, \ldots, U_{i_m}$ is the set of $m$ revoked users. The user will do the following steps.

1. Randomly chooses an element $r \in \mathbb{Z}_q$ and computes $G = g^r \mod p$. 

2. In a $(t, w)$ secret sharing scheme, a secret is divided into $w$ shares such that the secret can be recovered from any $t + 1$ shares and less than $t + 1$ shares give no information about the secret (see Section 1.4.2).
2. Computes $G_j = g_j^r \mod p$ for all $j \in V$, where $V = \{i_1, \cdots, i_m\} \cup \{j_1, \cdots, j_{t-m}\}$ and $\{j_1, \cdots, j_{t-m}\}$ is a $(t-m)$-subset of $\{n + 1, \cdots, n + t\}$.

3. Broadcasts $(G, G_j \| j : j \in V)$, where $\|$ denotes concatenation of data.

Each user $U_i$ in the subgroup, upon receiving the broadcast, computes the group key $K$ using Lagrange interpolation at the exponent as follows.

$$K = (G^{F(i)})^{\psi(V \cup \{i\}, i)} \times \prod_{j \in V} G_j^{\psi(V \cup \{i\}, j)}$$

$$= G^S$$

$$= g^{rs} \mod p,$$

where $\psi(\Phi, k) = \prod_{e \in \Phi, e \neq k} \frac{r}{e-k} \mod q$.

To join a new user, the trusted authority assigns a new identifier $d$ chosen from the set $\{n + t + 1, \cdots, q - 1\} \subseteq \mathbb{Z}_q$, where $d$ is different from the identifiers of existing users. The trusted authority calculates a share $s_d = F(d)$, sends $s_d$ to the new user $U_d$ using a secure channel, and publishes the public key $y_d = g^{s_d} \mod p$. The join operation does not affect the existing users. However, as we will show in Appendix A, the join operation does not maintain system secrecy as the new joined user can learn all past communication in the system.

1.4 Preliminary Cryptography

We describe some security primitives that will be used to construct group key distribution schemes proposed in the thesis.

1.4.1 Hard Problems and Assumptions

Discrete Logarithm (DL) Problem

Given a finite cyclic group $P$, a generator $g$ of $P$, and an element $X \in P$, find the integer $a$ such that $X = g^a$. DH problem is believed to be hard [30].

Computational Diffie-Hellman (CDH)

**CDH Problem:** Given a finite cyclic group $P$, a generator $g$ of $P$, and two elements $X = g^a, Y = g^b \in P$, where $a$ and $b$ are unknown, compute $Z = g^{ab} \in P$.

**CDH Assumption:** Any probabilistic polynomial time algorithm solves the CDH problem only with negligible probability [26].
1.4. Preliminary Cryptography

Decisional Diffie-Hellman (DDH)

DDH Problem: Given a finite cyclic group $P$, a generator $g$ of $P$, and three elements $X, Y, Z \in P$, decide whether there exist integers $a, b$ such that $X = g^a, Y = g^b$, and $Z = g^{ab}$.

DDH Assumption: Given a finite cyclic group $P$ and a generator $g$ of $P$, there is no efficient algorithm that can distinguish between the two distributions $(g^a, g^b, g^{ab})$ and $(g^a, g^b, g^c)$ where $a, b, c$ are randomly chosen in $[1, |P|]$.

An example of the finite cyclic group is $P = \mathbb{Z}_p^*$, the multiplicative group of integers modulo a large prime $p$. The problems and assumptions also hold if $g$ is a generator of a multiplicative subgroup of $\mathbb{Z}_p^*$ with order $q$, where $q$ is large prime and $p = 2q + 1$. In this case, $a, b, c \in [1, q - 1]$.

1.4.2 Threshold Secret Sharing

Shamir threshold secret sharing [77] can be described as follows. Let $q$ be a prime number, $K_0 \in \mathbb{Z}_q$ be the secret to be shared, $n$ be the number of users ($q \geq n + 1$), and $t + 1$ be the threshold for reconstructing the secret.\(^3\) Each user $U_i$ corresponds to a unique public value $y_i, y_i \neq 0$, and $U_i$ holds a share $K_i = F(y_i)$ (a value of $F(x)$ at $x = y_i$), where $F(x)$ is a random polynomial of degree $t$ over $\mathbb{Z}_q$ such that $F(0) = K_0$. Assuming $y_i = i$ and letting the set $\Phi$ consist of $t + 1$ users, the following properties hold for the scheme.

- The secret $K_0$ can be reconstructed by any $t+1$ users pooling together their shares and using polynomial interpolation to calculate the secret as $K_0 = \sum_{i \in \Phi} K_i \times \psi(\Phi, i) \mod q$, where $\psi(\Phi, i) = \prod_{e \in \Phi, e \neq i} \frac{e - i}{e - 1} \mod q$.
- The polynomial $F(x)$ can be reconstructed by any $t+1$ users as $F(x) = \sum_{i \in \Phi} K_i \times \Psi(\Phi, i) \mod q$, where $\Psi(\Phi, i) = \prod_{e \in \Phi, e \neq i} \frac{x - e}{i - e} \mod q$.

The idea of $(v, v)$-threshold scheme [45] can be described as follows. The scheme is basically a simplified version of Shamir’s threshold scheme for the case $n = v$ (it is not necessary that $q$ is prime and $q \geq n+1$). For each user $U_i, 1 \leq i \leq v - 1$, the share $K_i$ is a randomly chosen element of $\mathbb{Z}_q$. The share for user $U_v$ is $K_v = K_0 - \sum_{i=1}^{v-1} K_i \mod q$. Observe that the $v$ users can calculate the secret as $K_0 = \sum_{i=1}^{v} K_i \mod q$.

\(^3\)This scheme can be applied for any finite field of cardinality greater than $n$. 
1.4.3 Cumulative Scheme

Cumulative schemes, first defined in [78], are used in [19, 41] to construct secret sharing schemes for arbitrary access structures. Let $\Gamma$ be a monotone access structure, as defined in [82], on a set $U$. A cumulative scheme for $\Gamma$ is a map $\alpha: U \to 2^F$, where $F$ is a finite set, such that for any $A \subseteq U$,

$$\bigcup_{U_i \in A} \alpha_i = F \iff A \in \Gamma,$$

where $\alpha_i = \alpha(U_i)$. We can represent the scheme as a $|U| \times |F|$ cumulative array $C(\Gamma) = [c_{ij}]$, where row $i$ is indexed by $U_i \in U$ and column $j$ is indexed by an element $f_j \in F$, and where each entry $c_{ij}$ is either 0 or 1 such that

$$c_{ij} = 1 \iff f_j \text{ is given to } U_i.$$

The cumulative scheme of [78] is as follows. Let $\Gamma^- = A_1 + \cdots + A_m$ be the minimal form of the monotone access structure $\Gamma$. That is, $\forall A_{k_1}, A_{k_2} \in \Gamma^-$ and $A_{k_1} \neq A_{k_2}$, $A_{k_1} \not\subseteq A_{k_2}$. The dual access structure $\Gamma^* = B_1 + \cdots + B_n$ is obtained by interchanging sum and product in the boolean expression for $\Gamma^-$. Let the finite set be $F = \{f_1, \cdots, f_n\}$. Then $\alpha: U \to 2^F$ determines $\alpha_i = \{f_j : U_i \in B_j\}$.

**An example:** Let the minimal access structure be $\Gamma^- = U_1U_2 + U_2U_3 + U_3U_4$. The dual access structure is $\Gamma^* = U_1U_3 + U_2U_3 + U_2U_4$. Since $|\Gamma^*| = 3$, let the finite set be $F = \{f_1, f_2, f_3\}$. The cumulative array $C(\Gamma^-)$ is in Table 1.2, where $\alpha_1 = \{f_1\}, \alpha_2 = \{f_2, f_3\}, \alpha_3 = \{f_1, f_2\}$ and $\alpha_4 = \{f_3\}$. Observe that $\bigcup_{U_i \in A} \alpha_i = F, \forall A \in \Gamma^-.$

<table>
<thead>
<tr>
<th></th>
<th>$U_1U_3$</th>
<th>$U_2U_3$</th>
<th>$U_2U_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U_1$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$U_2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$U_3$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$U_4$</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

1.4.4 Key Distribution Pattern

Key distribution patterns (KDP) [61] are finite incidence structures that were originally designed to distribute keys between pairs of participants in a network and in the absence of an online key distribution centre. A KDP is used to allocate a collection of keys to
users in a system such that any pair of users can compute a common key by finding an appropriate combination of their keys. We recall the definition of a key distribution pattern.

**Definition 1.1** Let $\mathcal{X} = \{x_1, \ldots, x_m\}$ and $\mathcal{B} = \{\mathcal{B}_1, \ldots, \mathcal{B}_N\}$ be a family of subsets of $\mathcal{X}$. The pair $(\mathcal{X}, \mathcal{B})$ is called an $(m, N, t)$-key distribution pattern ($(m, N, t)$-KDP) if

$$|(\mathcal{B}_i \cap \mathcal{B}_j) \setminus \left( \bigcup_{e=1}^{t} \mathcal{B}_{k_e} \right)| \geq 1$$

for any $(t + 2)$-subset $\{i, j, k_1, \ldots, k_t\}$ of $\{1, 2, \ldots, N\}$.

The KDP guarantees that for any two subsets $\{\mathcal{B}_i, \mathcal{B}_j\}$ and any $t$ subsets $\{\mathcal{B}_{k_1}, \ldots, \mathcal{B}_{k_t}\}$ where $\{\mathcal{B}_i, \mathcal{B}_j\} \cap \{\mathcal{B}_{k_1}, \ldots, \mathcal{B}_{k_t}\} = \emptyset$, there exists at least an element $x$ that belongs to the two subsets, but does not belong to the $t$ subsets.

KDP can be constructed using several techniques. We give an example of KDP construction using perfect hash families (PHF) [75]. First we give the description of PHF [6]. Let $v$ and $w$ be integers such that $2 \leq w \leq v$. Let $\mathcal{V} = \{1, 2, \ldots, v\}$ and $\mathcal{W} = \{1, 2, \ldots, w\}$. Let $F: \mathcal{V} \to \mathcal{W}$ be a hash function from $\mathcal{V}$ to $\mathcal{W}$. The hash function $F$ is perfect on a subset $\mathcal{D} \subseteq \mathcal{V}$ if $F$ is injective when being applied to $\mathcal{D}$. Let $t$ be an integer such that $2 \leq t \leq w$ and let $\mathcal{F} \subseteq \{F: \mathcal{V} \to \mathcal{W}\}$. $\mathcal{F}$ is said to be an $(v, w, t)$-perfect hash family if for any $\mathcal{D} \subseteq \mathcal{V}$ where $|\mathcal{D}| = t$ there exists at least one element $F \in \mathcal{F}$ such that $F$ is perfect on $\mathcal{D}$. Let $\text{PHF}(S; v, w, t)$ denote an $(v, w, t)$-perfect hash family with $|\mathcal{F}| = S$.

The KDP construction is as follows. Let $\mathcal{F} = \{F_1, \ldots, F_S\}$ be a $\text{PHF}(S; v, w, t + 2)$ from $\mathcal{V}$ to $\mathcal{W}$. Suppose $\mathcal{X}$ consists of $S$ symmetric $w \times w$ matrices,

$$T^1 = [x_{a,b}^1]_{1 \leq a, b \leq w}, \ldots, T^S = [x_{a,b}^S]_{1 \leq a, b \leq w},$$

where $x_{a,b}^\ell = x_{b,a}^\ell$ for all $\ell, a, b$. (Observe that the size of $\mathcal{X}$ is $\frac{Sw(w+1)}{2}$ entries.) For each $1 \leq i \leq v$, suppose $\mathcal{B}_i$ consists of $F_i(i)$th row of matrix $T^\ell$ for all $1 \leq \ell \leq S$, that is,

$$\mathcal{B}_i = \{B^1_i, \ldots, B^S_i\},$$

where $B^\ell_i = \{x_{F_\ell(i),1}^\ell, \ldots, x_{F_\ell(i),w}^\ell\}$ for all $1 \leq \ell \leq S$. (Observe that the size of $\mathcal{B}_i$ is $Sw$ entries.) We show that the pair $(\mathcal{X}, \mathcal{B})$, where $\mathcal{B} = \{\mathcal{B}_1, \ldots, \mathcal{B}_v\}$, is an $(\frac{Sw(w+1)}{2}, v, t)$-KDP. We need to show that any two subsets $\{\mathcal{B}_i, \mathcal{B}_j\}$ have an entry $x$ that does not belong to any $t$ other subsets $\{\mathcal{B}_{k_1}, \ldots, \mathcal{B}_{k_t}\}$. Let $\mathcal{D} = \{i, j, k_1, \ldots, k_t\}$. Since $\mathcal{F}$ is a $\text{PHF}(S; v, w, t + 2)$, there exists a function $F_a \in \mathcal{F}$ such that $F_a$ is perfect on $\mathcal{D}$, which
means $F_a(i), F_a(j), F_a(k_1), \ldots, F_a(k_t)$ are all distinct. It follows that $x_{F_a(i),F_a(j)}^a \in B_i \setminus (B_{k_1} \cup \cdots \cup B_{k_t})$. Observe that $B_j$ has entries of the $F_a(j)$th row of the matrix $T^a$ which means $x_{F_a(j),F_a(i)}^a \in B_j$. Since the matrix $T^a$ is symmetric, it follows that $x_{F_a(i),F_a(j)}^a = x_{F_a(j),F_a(i)}^a$ and the entry is not in $B_{k_1} \cup \cdots \cup B_{k_t}$. So if there exists a PHF $(S; v, w, t + 2)$, then there exists a $(S_{w(w+1)}^t, v, t)$-KDP.

1.4.5 Erasure Code

Erasure codes are a special class of error-correcting codes that allow recovery of a message if part of it is corrupted or erased during the transmission.

**Definition 1.2 ([48])** An $[n,k,m]_q$ erasure code (over the finite field $F_q$) is a polynomial-time function $C : F_q^k \rightarrow F_q^n$ such that there exists a polynomial-time function $D : F_q^n \rightarrow F_q^k$, where $F_q = F_q \cup \{\perp\}$, such that: For all $v \in F_q^k$, if $u \in F_q^n$ is such that $u$ agrees with $C(v)$ in at least $m$ places, and is $\perp$ elsewhere, then $D(u) = v$.

Given an $[n,k,m]_q$ erasure code, one can encode a message $v \in F_q^k$ to obtain a codeword $C(v) \in F_q^n$. The message $v$ can be reconstructed even if up to $n - m$ positions of $C(v)$ are damaged or erased. Erasure codes can be constructed using error-correcting codes, such as Reed-Solomon codes. Given a message vector $v = (v_0, v_1, \ldots, v_{k-1}) \in F_q^k$, we construct the polynomial $p(x) = v_0 + v_1 x + \cdots + v_{k-1} x^{k-1}$. Let $e_1, e_2, \ldots, e_n$ be $n$ distinct elements in $F_q$. The encoding is defined by $C(v) = (p_v(e_1), p_v(e_2), \ldots, p_v(e_n))$, and the decoding $D$ uses $k$ pairs $(e_i, p_v(e_i))$ to interpolate the polynomial and reconstruct the coefficients of $p_v(x)$ and obtain the source message $v$.

1.4.6 Set Cover Problem

Set cover problem [22] is defined as follows. Consider $(\mathcal{X}, \mathcal{B})$, where $\mathcal{X} = \{x_1, \ldots, x_m\}$ is a finite set and $\mathcal{B} = \{B_1, \ldots, B_N\}$ is a family of subsets of $\mathcal{X}$, such that every element of $\mathcal{X}$ belongs to at least one set of $\mathcal{B}$. Consider a subset $C \subseteq \mathcal{B}$. We say that $C$ covers $\mathcal{X}$ if every element of $\mathcal{X}$ is in some sets of $C$, that is

$$\mathcal{X} = \bigcup_{B_i \in C} B_i.$$  

The set cover problem is to find the minimum-sized subset of $C$ of $\mathcal{B}$ that covers $\mathcal{X}$.

The set cover problem is a very significant optimisation problem and has been applied in numerous scenarios. An example is setting up security cameras to cover a
large art gallery. From each possible camera position, we can see a certain subset of paintings. Each subset of paintings is a set in our system and we would like to put up the fewest cameras to see all the paintings.

1.5 Thesis Inside

1.5.1 Aims and Objectives

The main concerns of group key distribution schemes are security and efficiency, and there has been a lot of research for secure and efficient constructions. The diversity of group applications also poses a research challenge. Finding group key distribution schemes for various group communication settings is important to accommodate diverse group applications.

The aim of the thesis is to find methods for constructing secure and efficient group key distribution schemes. We also consider group key distribution system where either single group members or collaboration of several group members can perform security operations. We construct secure and efficient schemes that cater for these settings, and consider properties that are desirable in group communication.

The proposed schemes focus on the problem of providing group secrecy in dynamic environments. All communication channels are assumed to be authentic or an adversary is assumed to be passive (may eavesdrop on arbitrary communication but may not interfere with it in any way). An adversary can be an outsider or a quasi-insider. An outsider is a passive adversary not participating in the group. A quasi-insider is a group member who wants to passively discover group keys used outside his membership interval.

1.5.2 Contributions and Structure

The aims are achieved by the following contributions, that are structured into several chapters. Chapter 1 introduces the topic, overviews the study on group key distribution problem, introduces some security primitives that are used in the thesis, and presents aims of the thesis and a short description of each chapter along with its main contribution.

Chapter 2 describes a generalised model of group key distribution systems that will be used as the system model for proposed group key distribution solutions in the rest of the thesis. A list of main notations used throughout the thesis is also provided in
Chapter 3 studies group key distribution schemes in the context of multicast key distribution. In particular, a secure re-keying scheme with key recovery property is proposed. The scheme is constructed using a one-way hash chain in conjunction with a logical key hierarchy. As noted in Section 1.3.4, previous schemes such as [17, 18, 85, 87] require multiple runs of the eviction procedure to form an arbitrary new group, which is inefficient. The proposed scheme provides an efficient and nontrivial method for eviction or join of an arbitrary number of users, and it is secure against an arbitrary number of colluders. When being applied for a single user eviction, the scheme improves the communication overhead of [85, 87]. The proposed scheme also provides key recovery that allows the key for one session to be recovered from the key of the future sessions combined with those of the past sessions. This property ensures a reliable re-keying even if the re-keying information is lost.

Chapter 4 considers decentralised setting of re-keying scheme, and a construction based on two-party Diffie-Hellman key exchange protocol and logical key hierarchy is given in the chapter. A group key in the proposed scheme is a function of partial key information from all group members and updating the group key requires contributions from several group members, and so it can be regarded as a conference key distribution scheme in dynamic environment. Contrary to other collaborative schemes such as [16, 40, 80, 81] efficient only for small groups, the proposed collaborative scheme is feasible for large groups. It can be used to form a new group with an arbitrary size by user eviction or admission, and it is resilient to an arbitrary number of colluders as long as the DL and CDH problems are intractable.

Most literature on stateless revocation schemes [3, 31, 37, 39, 55, 62, 64, 84] concentrates on efficient methods of forming authorised groups by a fixed group controller. In Chapter 5, the notion of dynamic stateless revocation system is introduced. It allows the group controller to be dynamic and so any group member can assume the role of forming a subgroup, while maintaining security of the system. This setting is desired in numerous applications such as dynamic teleconferences. A dynamic and stateless revocation scheme is also proposed in the chapter, and it is superior to [2, 75] in terms of communication overhead and collusion size. The construction uses the ideas of logical key hierarchy, Shamir secret sharing scheme and Diffie-Hellman key agreement, and so its security relies on the hardness of CDH problem. Several variants, authentication and user eviction for the proposed scheme are also discussed in the sequel.
Most existing group key distribution schemes are concerned with only user revocation (either temporary or permanent); many of them simplify the user join problem by assuming either the group is static [31, 51, 55, 84] or there is a fixed group controller that securely delivers the new group key for the enlarged group [17, 18, 50, 75]. Chapter 6 focuses on both user revocation and user join problems. The chapter introduces a novel scenario for dynamic group key distribution system that allows any group member to (i) form a subgroup of existing users $U$, and (ii) sponsor new users. User admission has two types: sponsorship and full join. In the former type, a sponsored member can participate in groups that are initiated by his sponsor but remains outside $U$. In the latter type, a new user will join $U$ after receiving enough sponsorships. Two secure and efficient group key distribution schemes for above scenario, that are built using algebraic techniques, are also proposed in the chapter. The first scheme provides sponsorship and full join with a threshold admission structure, while the second scheme has an arbitrary admission structure. The security of both schemes is based on DDH assumption with a threshold number of colluders. The chapter also describes a variant, traceability and user eviction for the proposed schemes.

Chapter 7 addresses the problems of allowing any group member to securely form a subgroup, and enabling several group members to securely enroll a new user to the group. In the chapter, a $t$-resilience group key distribution solution that works over key distribution patterns (KDP) is proposed, where $t$ is a system parameter. The storage and communication overheads required in subgroup protocol of the proposed solution can simultaneously achieve $O(\log n)$ keys, where $n$ is the number of users.

Chapter 8 concludes the thesis and suggests future directions.
Chapter 2

A Model of Group Key Distribution Systems

2.1 Introduction

In this chapter we present a model for group key distribution systems. The model is generalised to cope with various scenarios and concerns in group applications, and in turn it will be used as the model for the proposed group key distribution schemes in the rest of the thesis. Characteristics and attributes of the proposed schemes will be given in their corresponding chapters.

2.2 The System Model

Let the set $\mathcal{N}$ denote the universe of all users, and let $\mathcal{K}$ denote the set of all possible secret keys (secret information) in the system. There is a trusted entity, called the group controller (GC), that initialises the system, and an initial group of users $\mathcal{U}_0 = \{U_1, U_2, \cdots, U_{n_0}\} \subseteq \mathcal{N}$. Each user $U_i \in \mathcal{U}_0$ holds a set of secret keys (secret information) $\mathcal{K}(U_i) \subset \mathcal{K}$. The index $i$ is a unique identifier for user $U_i$. In the system initialisation, GC generates system parameters, publishes necessary information, and gives secret information to each initial user through secure and authentic unicast channels. We assume such channels exist between the GC and each user in the system.

After the initialisation phase, the system is dynamic and its lifetime consists of consecutive sessions,

$$\mathcal{S} = (S_1, S_2, \cdots, S_s, \cdots, S_M) .$$

In session $S_s$, there is a user group $\mathcal{U}_s = \{U_1, U_2, \cdots, U_{n_s}\} \subseteq \mathcal{N}$, and each user $U_i \in \mathcal{U}_s$ holds an individual secret key (secret information) set $\mathcal{K}(U_i) \subset \mathcal{K}$. Indeed, the membership of a user in $\mathcal{U}_s$ is indicated by having the appropriate secret key set that are required to participate in group operations.
2.2. The System Model

The system lifetime also consists of consecutive membership events,

\[ \mathcal{E} = (E_1, E_2, \cdots, E_s, \cdots, E_M), \]

that require operations in the groups. One membership event occurs in each session (see next section for the event details). In each event \( E_s \), there is a set of entities that have the privilege of collaboratively performing the membership event. This set is called privileged set \( \mathcal{P}_s \) and contributions from all privileged entities in the set are required for a successful event \( E_s \). A privileged entity could be the GC (centralised setting) and/or a user in \( \mathcal{N} \) (decentralised setting). Every event may have a different privileged set.

In each session \( S_s \), the event \( E_s \) results in a group key (session key) \( GK_s \in \mathcal{K} \) shared by some users in \( \mathcal{N} \) (note that \( \mathcal{U}_s \subseteq \mathcal{N} \)) while unauthorised users do not share that key. Therefore, there is a collection of group keys,

\[ \{GK_1, GK_2, \cdots, GK_s, \cdots, GK_M\} \subset \mathcal{K} \]

in the system lifetime, and every group key is a new key.

2.2.1 Membership Events

There are four elementary membership events in the system: **Subgroup**, **Join**, **Evict**, and **Refresh**. In session \( S_s \), a membership event \( E_s \in \{\text{Subgroup, Join, Evict, Refresh}\} \) is invoked by corresponding privileged entities. The membership events are shown in Tables 2.1, 2.2, 2.3, and 2.4 for subgroup, join, evict, and refresh events, respectively.

**Subgroup Event**

The purpose of this event is to allow secure communication to or within a subset \( \mathcal{G}_s \) in \( \mathcal{U}_s \). A privileged set \( \mathcal{P}_s = \{\text{GC}\} \) or \( \mathcal{P}_s \subset \mathcal{N} \) invoke this event whereby a group key \( GK_s \) is formed and shared by users in \( \mathcal{G}_s \). A privileged entity in \( \mathcal{P}_s \) is called a group initiator in this context. All communications within \( \mathcal{G}_s \) are encrypted using the group key and a secure symmetric encryption algorithm \( E() \).

Users in \( \mathcal{U}_s \setminus \mathcal{G}_s \) are considered to be temporarily revoked from session \( S_s \) and they might become authorised users in the future sessions. Therefore, the users for the next session, \( \mathcal{U}_{s+1} \), are those in \( \mathcal{U}_s \).
Table 2.1: Subgroup event

<table>
<thead>
<tr>
<th>If $E_s = $ Subgroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: $U_s, G_s \subseteq U_s, P_s = {GC}$ or $P_s \subseteq N$</td>
</tr>
<tr>
<td>Process: a subgroup protocol</td>
</tr>
<tr>
<td>Output: - all $U_1 \in G_s$ share a group key $GK_s$</td>
</tr>
<tr>
<td>- $U_{s+1} = U_s$</td>
</tr>
</tbody>
</table>

**Join Event**

In this event a subset of new users $J_s$ in $N \setminus U_s$ join $U_s$, requiring two accomplishments: (i) Giving appropriate secret key sets to the new users such that they are eligible for future membership events, and (ii) Establishing a common key $GK_s$ for the enlarged group $U_s \cup J_s$.

A privileged set $P_s = \{GC\}$, or $P_s \subseteq N$, invokes the admission operation. A privileged entity in the set is called a sponsor in this context. The new joined users can be part of any future event in the group. The user group for the next session, $U_{s+1}$, are $U_s \cup J_s$.

Table 2.2: Join event

<table>
<thead>
<tr>
<th>If $E_s = $ Join</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: $U_s, J_s \subseteq N \setminus U_s, P_s = {GC}$ or $P_s \subseteq N$</td>
</tr>
<tr>
<td>Process: a join protocol</td>
</tr>
<tr>
<td>Output: - all $U_i \in J_s$ obtain secret information $K(U_i)$</td>
</tr>
<tr>
<td>- all $U_i \in U_s \cup J_s$ share a group key $GK_s$</td>
</tr>
<tr>
<td>- $U_{s+1} = U_s \cup J_s$</td>
</tr>
</tbody>
</table>

**Evict Event**

This event is opposite to the join event, and involves a subset $R_s$ in $U_s$ being evicted from $U_s$. This requires two accomplishments: (i) Disabling secret key sets of the evicted users such that they cannot be involved in future group operations, and (ii) Establishing a secure common key $GK_s$ shared by the user subset $U_s \setminus R_s$.

This operation is performed by a privileged set $P_s = \{GC\}$, or $P_s \subseteq N$, and a privileged entity in the set is called an evictor. Eviction from $U_s$ prevents the users
2.2. The System Model

in $\mathcal{R}_s$ from being part of any future authorised groups and taking part in any future privileged sets.

Unlike the subgroup event, user revocation to form new groups in this event is permanent. Users in $\mathcal{R}_s$ lose their memberships in $\mathcal{U}_s$ and all subsequent groups. The users for the next session, $\mathcal{U}_{s+1}$, are those in $\mathcal{U}_s \setminus \mathcal{R}_s$. Note that users in $\mathcal{R}_s$ will have their memberships back in a future session if a join event involving the users takes place.

Table 2.3: Evict event

<table>
<thead>
<tr>
<th>Event</th>
<th>Input: $\mathcal{U}_s, \mathcal{R}_s \subseteq \mathcal{U}_s, \mathcal{P}_s = {GC}$ or $\mathcal{P}_s \subset \mathcal{N}_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Process: an evict protocol</td>
</tr>
<tr>
<td></td>
<td>Output: - all $U_i \in \mathcal{U}_s \setminus \mathcal{R}_s$ share a group key $GK_s$</td>
</tr>
<tr>
<td></td>
<td>- all $U_i \in \mathcal{R}_s$ have unusable secret information $K(U_i)$</td>
</tr>
<tr>
<td></td>
<td>- $\mathcal{U}_{s+1} = \mathcal{U}_s \setminus \mathcal{R}_s$</td>
</tr>
</tbody>
</table>

Refresh Event

There is no user addition or user revocation in this event. A group key $GK_s$ is established for $\mathcal{U}_s$ by a privileged set $\mathcal{P}_s = \{GC\}$, or $\mathcal{P}_s \subset \mathcal{N}$, without changing group memberships. This event will renew the group key shared by users in the group.

Table 2.4: Refresh event

<table>
<thead>
<tr>
<th>Event</th>
<th>Input: $\mathcal{U}_s, \mathcal{P}_s = {GC}$ or $\mathcal{P}_s \subset \mathcal{N}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Process: a refresh protocol</td>
</tr>
<tr>
<td></td>
<td>Output: - all $U_i \in \mathcal{U}_s$ share a group key $GK_s$</td>
</tr>
<tr>
<td></td>
<td>- $\mathcal{U}_{s+1} = \mathcal{U}_s$</td>
</tr>
</tbody>
</table>

A possible sequence of events is given in Figure 2.1, with events ($\cdots$, Join, Subgroup, Subgroup, Evict, Refresh, $\cdots$).

2.2.2 Group Key Distribution Schemes

A group key distribution scheme provides algorithms to perform membership events in a group key distribution system. A group key distribution scheme consists of the
2.2. The System Model

Figure 2.1: A possible sequence of events in the system

following components:

1. A protocol for the GC to initialise the system,

2. Protocols for privileged entities to securely and efficiently execute membership events in the system. The scheme might support protocols for one or more membership events. An event protocol can be divided into two basic consecutive steps.

(a) Transmission – Each privileged entity in $P_s$ performs operations on the public information and his secret information to construct an event message $M$. Transmission of the message to target users is through either a secure unicast channel or a broadcast (multicast) channel, depending on the event protocol. We will denote a secure unicast channel by “$\rightarrow$” and a broadcast (multicast) channel by “$\rightarrow$” when describing the proposed schemes.

(b) Computation – Each target user $U_i$ performs computations on the public information, available secret keys in $K(U_i)$ as well as received event message $M$ to achieve the purpose of the membership event.

2.2.3 Security Requirements

The main requirements of a group key distribution system are correctness and security. When new users join or existing users leave the group, the system must ensure that current memberships change. The system has to guarantee that only authorised users of a session share the common key for the session.

We consider security of the system against the attack by a collusion $C$ of passive and computationally bounded adversaries with access to secret keys (secret information)
2.2. The System Model

\[ \mathcal{K}(C) = \bigcup_{U \in C} \mathcal{K}(U) \] and all broadcast (multicast) messages in the system. We say a group key distribution scheme is \( c \)-resilient if any \( c \) colluders \( U_{i_1}, \ldots, U_{i_c} \), belonging to sessions \( (S_{s_1}, \ldots, S_{s_c}) \), \( c' \leq c \), cannot discover the group key for any session none of the colluders belong to.

**Definition 2.1** A group key distribution scheme is \( c \)-resilient if, for any \( C \subseteq \mathcal{N} \) and \( |C| \leq c \), users in \( C \), even if they collude, cannot find group keys \( GK_s \) for all \( s \) such that \( U_s \cap C = \emptyset \).

The colluders may belong to different sessions and there may be more than one colluder in a single session. We note that the security definition is a restricted form of key confidentiality as it requires the colluders to find the group keys to break the system. In the formal definition of key confidentiality [62], the system is considered broken if the adversary can learn something about the key. That is, the adversary can distinguish the group keys from a random string given all the information known by the colluders.

The notion of \( c \)-resilient defined above can be broken into four security requirements for group key distribution schemes.

**Definition 2.2** *Subgroup Secrecy* – For any session \( S_s \) where \( S_1 \leq S_s \leq S_M \) and \( E_s = \text{Subgroup} \), a collusion \( C \subseteq U_s \setminus G_s \) cannot find the group key \( GK_s \).\(^1\)

**Definition 2.3** *Evict Secrecy* (also called *Forward Secrecy*) – For any session \( S_s \) where \( S_1 \leq S_s \leq S_M \), a collusion \( C \subseteq \bigcup \mathcal{R}_a, S_1 \leq S_a \leq S_s, E_a = \text{Evict} \), cannot find the group keys \( GK_b, S_s \leq S_b \leq S_M \).

**Definition 2.4** *Join Secrecy* (also called *Backward Secrecy*) – For any session \( S_s \) where \( S_1 \leq S_s \leq S_M \), a collusion \( C \subseteq \bigcup \mathcal{J}_a, S_s \leq S_a \leq S_M, E_a = \text{Join} \), cannot find the group keys \( GK_b, S_s \leq S_b \).

**Definition 2.5** *Evict-Join Secrecy* (also called *Forward-Backward Secrecy*) – For any sessions \( S_{s_1}, S_{s_2} \) where \( S_1 \leq S_{s_1} < S_{s_2} \leq S_M \), a collusion \( C \subseteq (\bigcup \mathcal{R}_{a_1}) \cup (\bigcup \mathcal{J}_{a_2}), S_1 \leq S_{a_1} \leq S_{s_1}, E_{a_1} = \text{Evict}, S_{s_2} \leq S_{a_2} \leq S_M, E_{a_2} = \text{Join} \), cannot find the group keys \( GK_b, S_{s_1} \leq S_b \).

In the above definitions,

\(^1\)The definition of subgroup secrecy also implies security for a refresh event.
(i) Subgroup Secrecy requires that a collusion of users in $U_s$ but not in the subgroup $G_s$, cannot obtain the group key for the subgroup $G_s$.

(ii) Evict Secrecy requires that a collusion of users evicted from session $S_s$ and all sessions preceding $S_s$, cannot obtain the group key of session $S_s$ or any subsequent group key.

(iii) Join Secrecy requires that a collusion of new users in session $S_s$ and subsequent sessions, cannot learn any preceding group key.

(iv) Evict-Join Secrecy requires that a collusion of users consisting of both Evict Secrecy and Join Secrecy cases, cannot discover keys of groups where they are not a member of. We observe that Evict-Join Secrecy subsumes both Evict Secrecy and Join Secrecy.

The above security definitions include attacks from outsiders who only have access to the broadcast (multicast) messages. We will show that our proposed schemes are correct and prove that they are secure under the above definitions.

We require a form of authentication for authorised users to assure that the broadcast (multicast) information for updating the group key, i.e., event messages are originated from a privileged entity and the messages are unmodified during transmission (source and message authenticity).

The common approach for providing source and message authentication is using symmetric-key cryptography. The sender and receiver agree on a secret key which is used in conjunction with a message authentication code (MAC) algorithm to generate a MAC for a message. The sender attaches the MAC to the corresponding message and the receiver verifies both origin and integrity of the received message by using the MAC verification algorithm. This approach although efficient, but cannot be naively used in a group communication system (multiple receivers) because any receiver holding the secret key can impersonate the sender and forge messages on his behalf. Several variations were proposed to overcome the problem. For examples, the scheme in [17] uses several secret keys to authenticate every message so that a collusion of a number of receivers cannot break the authentication system. The scheme in [70] embeds a MAC in every message and discloses the corresponding secret key to the receiver to enable verification after certain time elapsed so that the secret key can no longer be used to forge the message.

We observe that using MACs to authenticate event messages in a dynamic group
communication system is impractical. This is because authenticating the event messages when forming a new group requires a new secret key to be shared by members of the new group beforehand. The pre-existing secret key is used for generating MACs of the event messages. This raises an issue of how to distribute the pre-existing secret key to the new group as using a group key distribution scheme to establish the secret key will require another pre-existing secret key, assuming the authenticity property is preserved.

A different method which provides source and message authentication, and accommodates the dynamic group behaviour involves the use of a public-key signature system. The sender registers a public signing key with a certificate authority, signs every event message with the corresponding private key, and appends the signature to the message. Recipients authenticate both message source and content by executing the signature verification algorithm with the senders public key. This solution is more costly as signatures are typically long, and computing and verifying each signature requires a significant computational overhead. Several schemes were proposed to mitigate this overheads by amortizing a single signature over several messages such as [34, 68, 88]. The sender signs only one message and the consecutive messages are linked to that message in a way that allows recipients to verify that they were sent by the signer.

We do not consider authentication systems in the thesis and assume that source and message authenticity is preserved for all communications (an exception is in Chapter 5 where we give an authentication method for the proposed scheme). We note that implementing the public-key authentication system will incur some additional storage, communication and computation costs to our proposed group key distribution schemes. This will affect the efficiency comparisons made in the subsequent chapters where the scheme with longer event messages will have more additional costs that make it less efficient in terms of storage, communication and computation.

### 2.2.4 Efficiency Requirements

A major requirement of a group key distribution system is to be efficient in group operations. The efficiency of group key distribution schemes is measured in terms of the following parameters.

1. **Storage** – The amount of secure storage that the GC and users use to store $K$ and $K(U)$, respectively, during the system lifetime. This is measured in bits.
2.2. The System Model

2. Transmission – The total amount of data transmitted by all entities during an event protocol. This bandwidth is also measured in bits.

3. Computation – The number of operations that users need to perform during an event protocol. Only the number of encryptions/decryptions and the number of exponentiations are taken into account. We assume basic arithmetic operations are negligible compared to the above costs.

2.2.5 Aspects of Scheme

A group key distribution scheme has a number of aspects and a system must be evaluated with respect to all these aspects. Performance of a system is measured by a number of parameters that are grouped according to these aspects. We divide the parameters into three categories as follows.

1. Availability – These relate to the cardinality and structure of the user subset $G_s$, the new user subset $J_s$ and the evicted user subset $R_s$. It is desirable that schemes allow arbitrary values for the sizes and structures of these sets. This category also includes the type of entities (GC or users or both) in the privileged set $P_s$ for each membership event protocol. If the privileged entities are users, a further parameter is the structure of privileged users in $P_s$.

2. Security – These relate to the size of the collusion $C$ and the underlying security problems of each event protocol. Resilience against arbitrary collusion size and provable security are required.

3. Efficiency – These include the storage, transmission (communication) and computation costs required by each event protocol. A more efficient protocol has lower costs. Other parameters in this category are,

   (a) Transmission methods – In the case of multiple recipients, a transmission over broadcast (multicast) channel (one-to-many) consumes less bandwidth than multiple transmissions over unicast channel (one-to-one).

   (b) Transmission rounds – The number of broadcasts (multicasts) rounds that are required for the completion of an event. It is desirable to have less rounds.

   (c) Storage states – The state of the system and the users’ individual secret information during the system lifetime. This information might be static
(fixed from the initialisation time) or dynamic (changed/updated during sessions). The former and latter status are called stateless and stateful, respectively.

Another parameter falling in this category is the response time, that is, the time elapsed between starting and completing execution of each event protocol (time required for the protocol to become effective). It is desirable to keep the response time small. We note that lower overheads in other efficiency parameters implies smaller response time.

### 2.3 A Summary of Notations

Below, we list the main notations (Table 2.5) used in the thesis. Additional notations will be employed when necessary, and will be defined when describing the proposed schemes.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{N} )</td>
<td>Universe of all users</td>
</tr>
<tr>
<td>( \mathcal{K} )</td>
<td>Set of all possible secrets</td>
</tr>
<tr>
<td>( \mathcal{U}_0 )</td>
<td>Initial user group</td>
</tr>
<tr>
<td>( \mathcal{U} )</td>
<td>User group</td>
</tr>
<tr>
<td>( \mathcal{G} )</td>
<td>User subgroup</td>
</tr>
<tr>
<td>( \mathcal{R} )</td>
<td>Evicted user subgroup</td>
</tr>
<tr>
<td>( \mathcal{J} )</td>
<td>New user subgroup</td>
</tr>
<tr>
<td>( \mathcal{P} )</td>
<td>Privileged set</td>
</tr>
<tr>
<td>( \mathcal{C} )</td>
<td>Collusion of adversaries</td>
</tr>
<tr>
<td>( K(U) )</td>
<td>User ( U )'s secret key set</td>
</tr>
<tr>
<td>( M )</td>
<td>Event message</td>
</tr>
<tr>
<td>( S )</td>
<td>Session</td>
</tr>
<tr>
<td>( E )</td>
<td>Membership event</td>
</tr>
<tr>
<td>( K )</td>
<td>Secret key</td>
</tr>
<tr>
<td>( GK )</td>
<td>Group key</td>
</tr>
<tr>
<td>( GC )</td>
<td>Group controller</td>
</tr>
<tr>
<td>( U )</td>
<td>User</td>
</tr>
<tr>
<td>( n_0 )</td>
<td>Cardinality of ( \mathcal{U}_0 )</td>
</tr>
<tr>
<td>( n )</td>
<td>Cardinality of ( \mathcal{U} )</td>
</tr>
<tr>
<td>( s )</td>
<td>Session index</td>
</tr>
<tr>
<td>( i )</td>
<td>User index or identifier</td>
</tr>
<tr>
<td>( | )</td>
<td>Concatenation of data</td>
</tr>
<tr>
<td>( \oplus )</td>
<td>Exclusive-OR operation</td>
</tr>
<tr>
<td>( E() )</td>
<td>Symmetric encryption</td>
</tr>
<tr>
<td>( D() )</td>
<td>Symmetric decryption</td>
</tr>
<tr>
<td>( \text{LKH} )</td>
<td>Logical Key Hierarchy</td>
</tr>
<tr>
<td>( \text{KDP} )</td>
<td>Key Distribution Pattern</td>
</tr>
</tbody>
</table>
2.4 Conclusion

We gave a general framework for the study of group key distribution system. This framework will be used to describe and analyse the group key distribution systems proposed in the thesis. We described components and operations that can occur in the system and revised the basic structure of group key distribution schemes, along with their security and efficiency requirements.
3.1 Introduction

In this chapter we study group key distribution schemes from the perspective of multicast key distribution and consider the re-keying problem in evict and join events. Previous schemes [85, 87] use re-keying schemes to establish a new group key for a session of either user eviction or user join. For a user join session, these schemes exclusively use secure unicast channels to distribute secret keys to new users. The schemes are designed to support eviction or admission of a single user. However they can also be repeatedly used to evict or join multiple users.

Using single user eviction and join for removing or joining a group is inefficient because of redundancy in successive re-keying operations, where a new group key is formed only to be thrown away thereafter. If \( w \) users are to be evicted or joined by this method, the cost in terms of the number of rounds, communication bandwidth and user computation, will be \( w \) times the cost of a single run of re-keying. This overhead is unacceptable for large size groups with frequent membership change. It is desirable to have re-keying schemes that efficiently perform group eviction or group join into a single round. Such schemes would be useful when group operation is performed at certain intervals, or for applications such as Pay-Per-View where users join or leave the system in groups (simultaneously or within a very short time interval) and efficiency is of high importance. The schemes can also be applied to scenarios, such as multi-party games, where membership changes are distributed over time for performance reasons.

Another important problem in multicast communication is reliability. Multicasting is an unreliable mode of communication and delivery of all packets is not guaranteed by the system. Lost packets may also occur because of temporary disconnection from the network. If the packet that carries the re-keying information is lost, legitimate
users will be unable to compute the group key. This may influence re-keying and so the re-keying system must be resilient against packet loss. A naive solution is to re-send the re-keying message. However this not only does not guarantee that re-keying packets will be delivered, but also results in high communication overhead.

### 3.1.1 A Summary of Our Contribution

We propose an efficient re-keying scheme that provides multiple user eviction and user join. The scheme has lower overhead compared to the trivial method of repeatedly using a single eviction (or join) system, and requires only a single re-keying message transmitted over a multicast channel per session. The message contains information not only for authorised users to establish a new group key, but also for new users to determine their secret key sets in a join event. This substantially reduces the number of transmissions over unicast channels that is required in trivial case.

The scheme is constructed using a one-way hash chain in conjunction with a logical key hierarchy. One-way hash chains have been previously used in security systems including micropayment and high-speed signature systems [69, 73]. We use the hash chain to generate the keys of a user from some seed values (the keys propagate up the tree), hence reducing the storage cost of the user. This also reduces the communication cost because it requires less re-keying information, and some computation previously performed by the group controller is distributed to the users. The logical tree in our scheme does not need to be full or balanced and so it can be any tree structure.

The group controller (GC) is the privileged entity for both membership events. There is no limit on the number of users who can be evicted from, or join the group, nor is the collusion size limited. We prove that our scheme satisfies the requirements of strong security, that is, forward secrecy, backward secrecy, and forward-backward secrecy as defined in Chapter 2. For a single user re-keying, the scheme improves on Wallner et al.’s scheme [85] and Wong et al.’s scheme [87] by reducing the cost of communication by factors 2 and $d^2 - 1$, respectively. A user’s cost of computation is equivalent to $u$ symmetric key decryptions together with $v$ hashing where $u + v \leq \log_d n$. The required storage for the GC is $\frac{d(n-1)}{d-1}$ keys and for a user is $\log_d n + 1$ keys, assuming a tree of degree $d$ which is full and balanced. The protocol is stateful and key storage needs to be updated in each session.

We compare the efficiency of the proposed scheme and basic logical key hierarchy schemes [85, 87] for both single user re-keying and multiple user re-keying in Table 3.1. We use the key length as the unit to measure the user storage and communication
costs. The computation cost is measured in terms of the number of decryptions \( (u) \) and the number of hashing operations \( (v) \). We note that schemes \([85, 87]\) use the trivial method of repeatedly applying single user re-keying to achieve multiple user re-keying.

Table 3.1: Efficiency comparison of the proposed scheme and the basic logical key hierarchy schemes

<table>
<thead>
<tr>
<th></th>
<th>Proposed Scheme</th>
<th>WHA [85]</th>
<th>WGL [87]</th>
</tr>
</thead>
<tbody>
<tr>
<td>GC Storage</td>
<td>( \frac{dn-1}{d-1} )</td>
<td>( 2n-1 )</td>
<td>( \frac{dn-1}{d-1} )</td>
</tr>
<tr>
<td>User Storage</td>
<td>( \ell + 1 )</td>
<td>( \log_2 n + 1 )</td>
<td>( \ell + 1 )</td>
</tr>
<tr>
<td>Collusion Resilience</td>
<td></td>
<td>( n )</td>
<td>( n )</td>
</tr>
</tbody>
</table>

**Single User**

| Communication       | \( \leq (d-1)\ell + 1 \) | \( \leq 2\log_2 n + 1 \) | \( \leq d\ell + 1 \) |
| Computation*        | \( u + v \leq \ell \)   | \( u \leq \log_2 n \)   | \( u \leq \ell \)   |
| Rounds              | 1                          | 1                          | 1                          |

**Multiple (\( \beta \)) Users**

| Communication       | \( \leq n + \sum_{e=1}^{\ell-1} d_e \left( 1 - \frac{1}{\beta} \right) \) | \( \leq \frac{n}{\beta} \left( 2\log_2 n + 1 \right) \) | \( \leq \frac{n}{\beta} \left( d\ell + 1 \right) \) |
| Computation         | \( u + v \leq \ell \)       | \( u \leq \beta \log_2 n \) | \( u \leq \beta \ell \) |
| Rounds              | 1                          | \( \beta \)                | \( \beta \)                |

\( n \) is the total number of users after user eviction or user join

*\( u = 1 \) for the proposed scheme

\( \ell = \log_d n \)

The scheme also provides key recovery, that is it allows the key for one session to be recovered from the key of the future sessions combined with those of the past sessions. This property increases the reliability of the system and ensures that the re-keying will succeed, even if the re-keying packet is lost. Key recovery are also studied in [54, 79] in term of self-healing and their model is similar to ours.

### 3.1.2 Protocols

The proposed scheme includes protocols for user join and user eviction. The protocols are described in Tables 3.2 and 3.3.

For a join operation, the GC sends to each new user in \( J_s \) a message \( M \) that contains a portion of the secret information using secure unicast channels. The GC multicasts a single re-keying message \( M_{rkey} \) containing extra information allowing the new users \( U_i \in J_s \) to determine their secret key sets \( \mathcal{K}(U_i) \). The message also includes information required to determine the group key \( GK_s \). Users \( U_i \in U_s \cup J_s \) use \( M_{rkey} \) and \( \mathcal{K}(U_i) \) to obtain the group key.

For an evict operation, the GC multicasts a single re-keying message \( M_{rkey} \) to evict
3.1. Introduction

Table 3.2: Join protocol

If $E_s = \text{Join}$

| Input: $\mathcal{U}_s, \mathcal{J}_s \subseteq \mathcal{N} \setminus \mathcal{U}_s, \mathcal{P}_s = \{\text{GC}\}$ |
| Process: $\xrightarrow{\mathcal{M}}$ all $U_i \in \mathcal{J}_s$ |
| $\xrightarrow{\mathcal{M}_{\text{rkey}}}$ all $U_i \in \mathcal{U}_s \cup \mathcal{J}_s$ |
| all $U_i \in \mathcal{J}_s$ perform computation on $\mathcal{M}$ and $\mathcal{M}_{\text{rkey}}$ |
| all $U_i \in \mathcal{U}_s$ perform computation on $\mathcal{M}_{\text{rkey}}$ and $\mathcal{K}(U_i)$ |
| Output: all $U_i \in \mathcal{J}_s$ obtain secret information $\mathcal{K}(U_i)$ |
| all $U_i \in \mathcal{U}_s \cup \mathcal{J}_s$ share a group key $GK_s$ |
| $\mathcal{U}_{s+1} = \mathcal{U}_s \cup \mathcal{J}_s$ |

Table 3.3: Evict protocol

If $E_s = \text{Evict}$

| Input: $\mathcal{U}_s, \mathcal{R}_s \subseteq \mathcal{U}_s, \mathcal{P}_s = \{\text{GC}\}$ |
| Process: $\xrightarrow{\mathcal{M}_{\text{rkey}}}$ $U_i \in \mathcal{U}_s \setminus \mathcal{R}_s$ |
| all $U_i \in \mathcal{U}_s \setminus \mathcal{R}_s$ perform computation on $\mathcal{M}_{\text{rkey}}$ and $\mathcal{K}(U_i)$ |
| Output: all $U_i \in \mathcal{U}_s \setminus \mathcal{R}_s$ share a group key $GK_s$ |
| all $U_i \in \mathcal{R}_s$ have unusable secret information $\mathcal{K}(U_i)$ |
| $\mathcal{U}_{s+1} = \mathcal{U}_s \setminus \mathcal{R}_s$ |

users in $\mathcal{R}_s$. Authorised users $U_i \in \mathcal{U}_s \setminus \mathcal{R}_s$ use this message and their secret key sets $\mathcal{K}(U_i)$ to obtain the new group key $GK_s$.

3.1.3 Key Recovery Model

Key recovery allows a legitimate user to recover a lost session key in a subsequent session. That is, a session key $GK_s$ can be recovered in any of the future sessions $S_{s+1}, \cdots, S_{s+k}$ ($k$ is a pre-defined system parameter) provided the user is a member of all these sessions. To provide key recovery, there is an additional recovery message containing information that allows legitimate users to recover the session key. Key recovery should not breach the security properties of re-keying given in definitions 2.3, 2.4, 2.5. This is captured in the following definition.

**Definition 3.1**  A key recovery scheme provides (i) forward secrecy, (ii) backward secrecy, or (iii) forward-backward secrecy if it satisfies definitions 2.3, 2.4, or 2.5, respectively. In (iii), it is required that $S_{s_2} \geq S_{s_1+2k}$. 

3.2. An LKH Re-keying Scheme

**Organisation of this chapter.** In Section 3.2 we give a new re-keying scheme based on logical key hierarchy, and prove its correctness and security. In Section 3.3 we apply the re-keying scheme to multicast group operations. Section 3.4 contains a description of a key recovery method that can be included in the re-keying system. Section 3.5 contains further discussions and conclusion of this chapter.

The main results in this chapter appeared in the *Proceedings of The Seventh Australasian Conference on Information Security and Privacy – ACISP 2002* [52].

### 3.2 An LKH Re-keying Scheme

In this section we propose an LKH re-keying scheme that uses one-way hash chains to evict or join users, and to establish a group key. The scheme provides an algorithm for the GC to construct the re-keying message, and an algorithm for users to use the multicast message to compute the group key. The algorithms can be applied for both user join and user eviction. We will show the details of this application in Section 3.3. The logical key hierarchy (LKH) can be defined as follows.

**Definition 3.2** A logical key hierarchy (LKH) is a tree where each node corresponds to a key and each leaf corresponds to a user. A user knows the keys of nodes along the path from the user’s leaf to the root.

Let \( \mathcal{U} = \{U_1, \ldots, U_n\} \) be the set of users and \( T \) be a tree with \( n \) leaves. Nodes of the tree are divided into internal nodes and leaves.\(^1\) Each node is given a label \( I_w^{(l)} \) and a key \( K_w^{(l)} \), called the node key. \( l \) is the level of the node and \( w \) is a unique number identifying the node. Node keys are divided into internal keys and leaf keys. Node labels are public while node keys are secret. Each leaf corresponds to a user \( U \). Let \( t_U \) be the level of the leaf attached to user \( U \). For a user \( U \), let \( \mathcal{I}(U) = \{I_w^{(0)}, \ldots, I_w^{(t_U)}\} \)\(^2\) be the set of nodes along the path from his corresponding leaf to the root. The user holds the set \( \mathcal{K}(U) = \{K_w^{(0)}, \ldots, K_w^{(t_U)}\} \) of node keys along the path to the root, i.e., \( \mathcal{K}(U) = \{K_w^{(l)} : I_w^{(l)} \in \mathcal{I}(U)\} \). All users have a common internal key called the root key \( K_w^{(0)} \). Figure 3.1 shows an example of a tree structure for 8 users, where, for example, the secret keys of \( U_1 \) are \( \mathcal{K}(U_1) = \{K_0^{(0)}, K_1^{(1)}, K_3^{(2)}, K_7^{(3)}\} \).

A re-keying scheme based on a logical key hierarchy updates the root key \( K_w^{(0)} \) by updating a subset \( \mathcal{K}(T) \) of internal keys, including the root key \( K_w^{(0)} \in \mathcal{K}(T) \), to a

\(^1\)An internal node has one or more child nodes while a leaf has no child.

\(^2\)The node identifiers \( w \) are not the same values.
3.2. An LKH Re-keying Scheme

subset $\mathcal{K}'(T)$ of new keys. The basic method of updating the internal keys using LKH can be found in Wallner et al. [85]. Our method employs a one-way hash chain to increase the efficiency of re-keying.

**Definition 3.3** Let $h()$ be a one-way hash function and $v$ denote a positive integer. $h^v(x)$ is a one-way hash chain where the function $h$ is obtained by $v$ times application of $h()$, and $h^0(x) = x$. Thus $h^v(x) = h(h(\ldots(h(x))\ldots))^{v \text{ times}}$.

We assume the function $h()$ is cryptographically secure. For example, we can use the one-way hash function in [32, 89].

We first describe the re-keying protocol single-$\mathcal{K}(T)$ for a single user eviction or join. Later we consider multi-$\mathcal{K}(T)$ for multiple user eviction or join.

### 3.2.1 Single-$\mathcal{K}(T)$

First we consider the special case where $\mathcal{K}(T) = \{K^{(0)}_w, \ldots, K^{(t)}_w\}$ for all levels $t$, $t < \max\{t_U : U \in \mathcal{U}\}$. The set $\mathcal{K}(T)$ contains only one node key at each level. Let $\mathcal{I}(T) = \{I^{(l)}_w : K^{(l)}_w \in \mathcal{K}(T)\}$ be the set of nodes corresponding to keys in $\mathcal{K}(T)$ and let $\mathcal{I}(T)^{(l)}$ be the set of nodes at level $l$ of $\mathcal{I}(T)$. Note that $\mathcal{I}(T)^{(l)}$ contains only a single node. Let $\mathcal{D}(I^{(l)}_w)$ be the set of nodes having parent $I^{(l)}_w$ (the nodes are at level $l + 1$). For a node $I^{(l)}_w \in \mathcal{I}(T)$, the set $\mathcal{D}(I^{(l)}_w)$ can be partitioned into two subsets $\mathcal{D}_X(I^{(l)}_w)$ and $\mathcal{D}_Y(I^{(l)}_w)$ where $\mathcal{D}_X(I^{(l)}_w) = \mathcal{D}(I^{(l)}_w) \cap \mathcal{I}(T)^{(l+1)}$ consists of nodes in $\mathcal{D}(I^{(l)}_w)$ that their corresponding keys must be changed, while $\mathcal{D}_Y(I^{(l)}_w) = \mathcal{D}(I^{(l)}_w) \setminus \mathcal{D}_X(I^{(l)}_w)$ contains the rest of the nodes in $\mathcal{D}(I^{(l)}_w)$. Note that $\mathcal{D}_X(I^{(l)}_w)$ contains only a single node.

**Transmission:** The Re-keying Algorithm for the GC
3.2. An LKH Re-keying Scheme

1. The GC chooses a random number \( r \in \mathbb{Z}_q \) for a large number \( q \).

2. For level \( l = t, \ldots, 0 \) and a node \( I^{(l)}_w \in \mathcal{I}(T)^{(l)} \),
   - (a) The GC updates the key of the node \( K^{(l)}_w \) to \( K^{(l)}_w = h^{l-1}(r) \) where \( h^{l-1}() \) is a one-way hash chain.
   - (b) The GC encrypts \( K^{(l)}_w \) with keys of all nodes in \( \mathcal{D}_Y(I^{(l)}_w) \).

The re-keying message is

\[
\mathcal{M}_{rkey} = \left\{ E_{K^{(l+1)}_w}(K^{(l)}_w) \mid I^{(l+1)}_w \in \mathcal{I}(T), I^{(l+1)}_w \in \mathcal{D}_Y(I^{(l)}_w) \right\},
\]

where \( E() \) denotes the encryption algorithm. Note that labels of node keys that are used to encrypt are included in the re-keying message.\(^3\)

3. The GC multicasts \( \mathcal{M}_{rkey} \) to the system.

**Computation: The Re-keying Algorithm for Users** \( U \in \mathcal{U} \)

1. User \( U \) finds the nodes that are both in \( \mathcal{I}(U) \) and \( \mathcal{M}_{rkey} \), in this case, a single node. Without loss of generality, let \( I^{(y+1)}_w \) be the node in \( \mathcal{I}(U) \cap \mathcal{M}_{rkey} \).

2. User \( U \) decrypts \( E_{K^{(y+1)}_w}(K^{(y)}_w) \in \mathcal{M}_{rkey} \) using his node key \( K^{(y+1)}_w \in \mathcal{K}(U) \) to update \( K^{(y)}_w \in \mathcal{K}(U) \). That is, \( K^{(y)}_w = D_{K^{(y+1)}_w}(E_{K^{(y+1)}_w}(K^{(y)}_w)) \) where \( D() \) denotes the decryption algorithm.

3. User \( U \) needs to update keys of node \( I^{(y)}_w \) and all its ancestors, i.e., \( I^{(y-1)}_w, \ldots, I^{(0)}_w \).

   Thus for level \( l = y - 1, \ldots, 0 \) and a node \( I^{(l)}_w \in \mathcal{I}(U) \), user \( U \) updates the node key \( K^{(l)}_w \) to \( K^{(l)}_w = h^{y-l}(K^{(y)}_w) \) where \( h^{y-l}() \) is a one way hash chain.

**Theorem 3.1** If the tree \( T \) is a full and balanced tree of degree \( d \), then updating single-\( \mathcal{K}(T) \) requires (i) a communication cost equivalent to at most \( 2((d - 1) \log_d n + 1) \log_2 q \) bits\(^4\) and (ii) a user computation cost equal to a single decryption plus at most \( \log_d n - 1 \) hashing operations.

The basic algorithm described above can be extended to evict or join multiple users. We use \( \text{multi-} \mathcal{K}(T) \) to denote the re-keying operation used in this case.

---

\(^3\)We assume the output of the hash function \( h() \) has \( \log_2 q \) bits. The number \( q \) should be large enough such that the encryption algorithm \( E() \) is secure against brute force attack.

\(^4\)We assume the output of the encryption algorithm is of size \( \log_2 q \) bits.
3.2.2 Multi-$\mathcal{K}(\mathcal{T})$

We can generalise the re-keying algorithm to update multi-$\mathcal{K}(\mathcal{T})$. The set $\mathcal{K}(\mathcal{T})$ consists of node keys where for a node key $K_w^{(l)} \in \mathcal{K}(\mathcal{T})$, ancestors of $K_w^{(l)}$ are also in $\mathcal{K}(\mathcal{T})$. The root key $K_w^{(0)}$ is always in $\mathcal{K}(\mathcal{T})$. Recall that $\mathcal{I}(\mathcal{T}) = \{ I_w^{(l)} : K_w^{(l)} \in \mathcal{K}(\mathcal{T}) \}$ and $|\mathcal{I}(\mathcal{T})^{(l)}| \geq 1$ in this case. Let $t = \max\{ l : I_w^{(l)} \in \mathcal{I}(\mathcal{T}) \}$ be the level of the deepest node in $\mathcal{I}(\mathcal{T})$ such that $\mathcal{I}(\mathcal{T}) = \bigcup \mathcal{I}(\mathcal{T})^{(l)} : l = t, \ldots, 0$. We summarise the re-keying process in Table 3.4, giving an algorithm for the GC to generate the re-keying message $\mathcal{M}_{rkey}$, and in Table 3.5, showing how users $U \in \mathcal{U}$ update their keys.

**Transmission:** The Re-keying Algorithm for the GC

Table 3.4: The re-keying algorithm to generate $\mathcal{M}_{rkey}$

<table>
<thead>
<tr>
<th>Step</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>$\mathcal{M}_{rkey} = \emptyset$</td>
</tr>
<tr>
<td>(2)</td>
<td>for $l = t$ to 0 {</td>
</tr>
<tr>
<td>(3)</td>
<td>for all $I_w^{(l)} \in \mathcal{I}(\mathcal{T})^{(l)}$ {</td>
</tr>
<tr>
<td>(4)</td>
<td>if $D_X(I_w^{(l)}) = \emptyset$ {</td>
</tr>
<tr>
<td>(5)</td>
<td>$K_w^{(l)} \in \mathbb{Z}_q$</td>
</tr>
<tr>
<td>(6)</td>
<td>$\mathcal{M}<em>{tmp} = \left{ E</em>{K_w^{(l)}}(K_w^{(l)}) \parallel I_w^{(l+1)} : I_w^{(l+1)} \in D_Y(I_w^{(l)}) \right}$</td>
</tr>
<tr>
<td>(7)</td>
<td>}</td>
</tr>
<tr>
<td>(8)</td>
<td>else {</td>
</tr>
<tr>
<td>(9)</td>
<td>$K_w^{(l)} = h(K_w^{(l+1)})$ where $I_w^{(l+1)} \in \mathbb{Z}_q D_X(I_w^{(l)})$</td>
</tr>
<tr>
<td>(10)</td>
<td>$\mathcal{M}<em>{tmp} = \left{ E</em>{K_w^{(l+1)}}(K_w^{(l)}) \parallel I_w^{(l+1)} : I_w^{(l+1)} \in D_X(I_w^{(l)}) \setminus { I_w^{(l+1)} } \right} \cup \left{ E_{K_w^{(l+1)}}(K_w^{(l)}) \parallel I_w^{(l+1)} : I_w^{(l+1)} \in D_Y(I_w^{(l)}) \right}$</td>
</tr>
<tr>
<td>(11)</td>
<td>}</td>
</tr>
<tr>
<td>(12)</td>
<td>$\mathcal{M}<em>{rkey} = \mathcal{M}</em>{rkey} \cup \mathcal{M}_{tmp}$</td>
</tr>
<tr>
<td>(13)</td>
<td>}</td>
</tr>
<tr>
<td>(14)</td>
<td>}</td>
</tr>
</tbody>
</table>

The algorithm in Table 3.4 works as follows. It starts at level $t$ and visits nodes in each level one by one, before moving to the next level down. At each level $l$ it examines every node in $\mathcal{I}(\mathcal{T})^{(l)}$, updates the keys of the nodes and produces the re-keying message $\mathcal{M}_{rkey}$.

- Step (1) initialises $\mathcal{M}_{rkey} = \emptyset$.
- Steps (2) and (3) together take every node $I_w^{(l)}$ in $\mathcal{I}(\mathcal{T})^{(l)}$, starting from level $l = t$ and going to level $l = 0$, and examine the nodes as in steps (4) to (12).
• Step (4) tests if at least a child node of \( I_w^{(l)} \) is in \( \mathcal{I}(T)^{(l+1)} \).
  
  – If it is not, step (5) updates the node key \( K_w^{(l)} \) to a random value and step (6) encrypts the updated node key \( K_w^{(l)}' \) with keys of all nodes in \( \mathcal{D}_Y(I_w^{(l)}) \) and stores the result in \( \mathcal{M}_{tmp} \).
  
  – If it is, step (9) updates the node key \( K_w^{(l)} \) to the hash value of a seed, where the seed is an updated key \( K_0^{(l+1)} \) of a randomly chosen node \( I_w^{(l+1)} \) in \( \mathcal{D}_X(I_w^{(l)}) \). Observe that updating a node key \( K_w^{(l)} \) implies keys of all nodes in \( \mathcal{D}_X(I_w^{(l)}) \) have been updated. Step (10) encrypts the updated node key \( K_0^{(l+1)} \) with keys of all nodes in \( \mathcal{D}_Y(I_w^{(l)}) \) and the updated keys of all nodes in \( \mathcal{D}_X(I_w^{(l)}) \) except the updated key used as the seed, and stores the result in \( \mathcal{M}_{tmp} \).

• Step (12) adds \( \mathcal{M}_{tmp} \) to \( \mathcal{M}_{rkey} \).

After executing the algorithm, the GC multicasts \( \mathcal{M}_{rkey} \).

**Computation: The Re-keying Algorithm for Users \( U \in \mathcal{U} \)**

Table 3.5: The re-keying algorithm to update user keys

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>for ( l = t_U - 1 ) to 0 {</td>
</tr>
<tr>
<td>(2)</td>
<td>for a pair ( I_w^{(l)}, I_w^{(l+1)} \in \mathcal{I}(U) ) {</td>
</tr>
<tr>
<td>(3)</td>
<td>if ( I_w^{(l+1)} \notin \mathcal{M}_{rkey} ) and ( K_w^{(l+1)} \rightarrow K_w^{(l+1)} )</td>
</tr>
<tr>
<td>(4)</td>
<td>( K_w^{(l)} = D_{K_w^{(l+1)}}(E_{K_w^{(l+1)}}(K_w^{(l)})) ) where ( E_{K_w^{(l+1)}}(K_w^{(l)}) \in \mathcal{M}_{rkey} )</td>
</tr>
<tr>
<td>(5)</td>
<td>else if ( I_w^{(l+1)} \in \mathcal{M}_{rkey} ) and ( K_w^{(l+1)} \neq K_w^{(l+1)} )</td>
</tr>
<tr>
<td>(6)</td>
<td>( K_w^{(l)} = D_{K_w^{(l+1)}}(E_{K_w^{(l+1)}}(K_w^{(l)})) ) where ( E_{K_w^{(l+1)}}(K_w^{(l)}) \in \mathcal{M}_{rkey} )</td>
</tr>
<tr>
<td>(7)</td>
<td>else if ( I_w^{(l+1)} \notin \mathcal{M}_{rkey} ) and ( K_w^{(l+1)} \rightarrow K_w^{(l+1)} )</td>
</tr>
<tr>
<td>(8)</td>
<td>( K_w^{(l)} = h(K_w^{(l+1)}) )</td>
</tr>
<tr>
<td>(9)</td>
<td>}</td>
</tr>
<tr>
<td>(10)</td>
<td>}</td>
</tr>
</tbody>
</table>

The algorithm in Table 3.5 works as follows. It starts from the deepest internal node and visits a node at each level before moving to the next level down. At each level, it examines the node and its child to update the node key (if necessary).

• Steps (1) and (2) together take a pair of nodes \( I_w^{(l)}, I_w^{(l+1)} \in \mathcal{I}(U) \) starting from level \( l = t_U - 1 \) to level \( l = 0 \), and examine the pair in steps (3) to (8).
3.2. An LKH Re-keying Scheme

Step (3) verifies if $I_w^{(l+1)}$ is in $M_{rkey}$ and the node key $K_w^{(l+1)}$ has been updated. If the conditions hold, the updated value for the node key $K_w^{(l)}$ is in $M_{rkey}$ and step (4) decrypts it using the updated node key $K_w^{n(l+1)}$ and assigns it to $K_w^{(l)}$.

- If $I_w^{(l+1)}$ is in $M_{rkey}$ and the node key $K_w^{(l+1)}$ has not been updated, the updated value for the node key $K_w^{(l)}$ is in $M_{rkey}$ and step (6) uses the node key $K_w^{(l+1)}$ to decrypt it and assigns it to $K_w^{(l)}$.

- If $I_w^{(l+1)}$ is not in $M_{rkey}$ and the node key $K_w^{(l+1)}$ has been updated, step (8) updates the node key $K_w^{(l)}$ to the hash value of $K_w^{r(l+1)}$.

- Otherwise, it is unnecessary to update the node key $K_w^{(l)}$.

- Observe that if a node key is updated, the keys of all its ancestors have to be updated also.

**Theorem 3.2** If $T$ is a full and balanced tree of degree $d$, updating multi-$K(T)$ requires, (i) communication of at most $2(n + \sum_{e=1}^{\log_d n-1} d^e(1 - \frac{1}{d})) \log_2 q$ bits\(^5\) and (ii) users’ computation cost of $u$ decryption operations and $v$ hashing operations, where $u + v \leq \log_d n$.

Note that the communication cost of $2(n + \sum_{e=1}^{\log_d n-1} d^e(1 - \frac{1}{d})) \log_2 q$ bits is for the worst case where all internal keys have to be updated. The following shows that even in the worst case, the length of the multicast message in our scheme is less than the length of the multicast message in the trivial method described in Section 3.1. This is because using the trivial method requires $\frac{n}{d}$ executions of single-$K(T)$ algorithm and so it needs total $\frac{n}{d}((d - 1) \log_d n + 1) \log_2 q$ bits. Hence,

$$2(n + \sum_{e=1}^{\log_d n-1} d^e(1 - \frac{1}{d})) < \frac{2n}{d}((d - 1) \log_d n + 1)$$

$$n + \sum_{e=1}^{\log_d n-1} d^e(1 - \frac{1}{d}) < n + \frac{n}{d}(d - 1)(\log_d n - 1)$$

$$\sum_{e=1}^{\log_d n-1} d^e < n(\log_d n - 1)$$

**Theorem 3.3** The re-keying scheme satisfies the following properties (assuming all re-keying messages are accessible).

\(^5\)Again, we assume the cipher of the encryption algorithm has $\log_2 q$ bits.
3.3. Group Operations

(i) All users can calculate the same new root key.

(ii) A user $U$ can only find the new keys for $\mathcal{K}(U) \cap \mathcal{K}(T)$ and not any other new key.

(iii) A passive adversary who knows the keys in $\mathcal{K}(T)$ cannot discover any new key in $\mathcal{K}'(T)$.

(iv) A passive adversary who knows the new keys in $\mathcal{K}'(T)$ cannot discover any key in $\mathcal{K}(T)$.

Proof: (sketch) (i) Referring to Table 3.4, observe that the new root key is a one-way hash chain of a random number $r \in_R \mathbb{Z}_q$. Steps (2) to (5) indicate that there may be more than one $r$. Let $r_1, r_2, \ldots, r_m$ be the random numbers. Steps (2), (3), (8) and (9) guarantee that only one of the random numbers is used for the new root key. Let the chosen one be $r \in \{r_1, r_2, \ldots, r_m\}$ and let $p$, where $0 \leq p \leq t$, be the level of $r$. The new root key is calculated as $h^p(r)$. Step (6) shows that some users have $r$ and step (10) ensures that all other users get one of the $h^1(r), \ldots, h^p(r)$. Together with the steps in Table 3.5 we see that once a user gets one of the $r, h^1(r), \ldots, h^p(r)$, he uses it as a seed to update node keys from the level of the seed to level 0, and so updates the root key.

(ii) Steps (6) and (10) in Table 3.4 show that each new node key $K_{u(l)}^{0} \in \mathcal{K}'(T)$ is encrypted using keys of its children and no other keys. This guarantees that a user only discovers the new keys along the path from his leaf to the root, unless he can break the one-way hash function $h()$.

(iii) Steps (5) and (9) in Table 3.4 show that the new keys in $\mathcal{K}'(T)$ are randomly generated, or are calculated using the one-way hash chains of random numbers. Steps (6) and (10) ensure that the new keys in $\mathcal{K}'(T)$ are never encrypted with keys in $\mathcal{K}(T)$, and so a passive adversary cannot derive any information about the new key in $\mathcal{K}'(T)$ unless he can break the encryption algorithm $E()$.

(iv) Similar to (iii).

3.3 Group Operations

The initial operation in a multicast group is setting up the system. After that, the membership of the group changes as a result of the execution of user eviction and user join operations.
The set of all users of a multicast group corresponds to the set of all leaves in a tree structure. Each node corresponds to a key and users use the root key as the group key. When group membership changes, the tree is updated such that only authorised users remain in the tree. Also some node keys, including the root key, are updated so that a new group key for the authorised users be established.

### 3.3.1 System Initialisation

A multicast group is initialised by the group controller (GC) as follows. Recall that $U_0$ is the initial group of $n_0$ users.

1. The GC constructs a tree structure with $n_0$ leaves, to each node it gives a unique label $I_w^{(l)}$ and attaches a randomly generated key $K_w^{(l)} \in R \ Z_q$, $K_w^{(l)} \neq 0$, and assigns each leaf to a user $U \in U_0$. The GC publishes the tree structure (node labels and users’ positions) in a public bulletin board and keeps all node keys secret.

2. The GC sends to each user $U \in U_0$, a set $K(U)$ of node keys along the path from $U$’s leaf to the root over a secure unicast channel. User $U$ holds the set $K(U)$ as his secret keys. The cardinality of $K(U)$ is the height of $U$’s leaf.

The constructed tree can be any tree structure and should be designed to suit the type of application and the requiring behavior. But to make the re-keying efficient, the tree structure should be balanced. In a full and balanced tree of degree $d$, the total number of nodes is $\frac{dn}{d-1}$ and the height of a leaf is $\log_d n + 1$.

**Theorem 3.4** Assuming a full and balanced tree of degree $d$, a group controller has to store $\frac{dn-1}{d-1} \log_2 q$ bits, and a user has to store $(\log_d n + 1) \log_2 q$ bits.

### 3.3.2 User Eviction

Suppose in session $S_s$ a subset $R_s \subseteq U_s$ of users are evicted from $U_s$, and a group key $GK_s$ is established for authorised users in $U_s \setminus R_s$. The evicted users must not obtain $GK_s$ or any subsequent group keys. Let the logical key hierarchy be $T_s$.

**Group Controller.** The GC does the following.

1. Updates the tree structure by pruning leaves corresponding to users in $R_s$. Observe that the resulting tree may have some redundant internal nodes. To increase efficiency, the GC may remove the redundant nodes by,
(a) pruning the internal nodes having only one child by replacing the parent with the child, and
(b) pruning the internal nodes having no child.

As a result, the GC also removes keys of redundant nodes from his storage and rearranges the levels of affected nodes and keys in the updated tree. Let the updated tree be denoted by $T'_s$. The GC publishes $T'_s$ in a public bulletin board. Observe that only users in $U_s \setminus R_s$ are in $T'_s$.

2. Evicts $R_s$ and establishes the group key $GK_s$ (updating the root key). This requires all internal keys belonging to the evicted users to be updated. In the updated tree $T'_s$, let $K(T'_s) = \bigcup_{U \in R_s} K(U) \setminus \{K^{(t)}_w\}$ be the set of internal keys that need to be updated. The GC performs the re-keying scheme in Section 3.2 with $T = T'_s$ and $K(T) = K(R_s)$.

Users. Users in $U_s \setminus R_s$ do the following.

1. Each affected user $U \in U_s \setminus R_s$ removes the redundant nodes and keys from $\mathcal{I}(U)$ and $\mathcal{K}(U)$, respectively, and rearranges the levels of the affected nodes and keys.

2. Each user $U \in U_s \setminus R_s$ receives the re-keying message $\mathcal{M}_{rkey}$ and performs the re-keying scheme in Section 3.2 to update the keys in $K(U) \cap K(R_s)$ and obtain the group key $GK_s$.

We note that the logical key hierarchy for the next session $S_{s+1}$ will be $T_{s+1} = T'_s$ with user group $U_{s+1} = U_s \setminus R_s$.

Example 3.1 Suppose $R_s = \{U_3, U_5, U_6\}$ are evicted from the group in Figure 3.1.

1. The GC updates the tree structure to that shown in Figure 3.2, where nodes $I^{(1)}_2, I^{(2)}_4, I^{(2)}_5, I^{(3)}_9, I^{(3)}_{11}$ have been pruned. Nodes in dashed line are the affected nodes and they have been rearranged to new levels. Keys of the grey coloured nodes are required to be updated, that is, $K(T) = \{K^{(2)}_3, K^{(1)}_1, K^{(0)}_0\}$. The result of re-keying is $K^{(r)}_3 = r, K^{(1)}_1 = h^1(r), K^{(0)}_0 = h^2(r)$ and $\mathcal{M}_{rkey} = \{E_{K^{(3)}_7}(K^{(2)}_3) \parallel I^{(3)}_7, E_{K^{(3)}_8}(K^{(1)}_1) \parallel I^{(2)}_8, E_{K^{(3)}_{10}}(K^{(0)}_0) \parallel I^{(1)}_{10}\}$.

2. After decryption, users $U_1$ and $U_2$ will have $K^{(r)}_3$ from $\mathcal{M}_{rkey}$ and calculate $K^{(1)}_1 = h^1(K^{(2)}_3), K^{(0)}_0 = h^2(K^{(3)}_3)$. User $U_4$ will have $K^{(1)}_1$ from $\mathcal{M}_{rkey}$ and calculate $K^{(0)}_0 = h^1(K^{(1)}_1)$. Users $U_7$ and $U_8$ will have $K^{(0)}_0$ from $\mathcal{M}_{rkey}$. The group key is $GK_s = K^{(0)}_0 = h^2(r)$.
Theorem 3.5 User eviction provides evict secrecy (forward secrecy). That is, for any session $S_s$ where $S_1 \leq S_s \leq S_M$, a collusion of arbitrary size $C = \bigcup \mathcal{R}_a, S_1 \leq S_a \leq S_s, E_a = \text{Evict}$, cannot distinguish the group key $GK_b, S_s \leq S_b \leq S_M$, from a random string.

Proof: (sketch) First we prove security for a single session $S_s$. That is, we show that the collusion of users $C = \mathcal{R}_s$ cannot obtain the session key $GK_s$. Users in $\mathcal{R}_s$ know $K(\mathcal{R}_s)$ but do not have access to any other internal keys. The proof of security in this case is similar to the proof of Theorem 3.3, part (iii), since $K(T) = K(\mathcal{R}_s)$ and $GK_s \in K'(T)$. To prove security for multiple sessions, we only need to prove that $K(\mathcal{R}_a) : S_1 \leq S_a \leq S_s$ gives no information about $GK_b : S_s \leq S_b \leq S_M$. This can also be proved using an argument similar to that of Theorem 3.3, part (iii). 

Observe that instead of keys in $K(\mathcal{R}_a)$, the updated version of the keys are used for further group operations. By referring to Theorem 3.3, part (iii), users in $\mathcal{R}_s$ only know $K(\mathcal{R}_s)$ but not any updated version of the keys. Then we have the following.

Corollary 3.1 Secret keys belonging to users in $\mathcal{R}_s$ are disabled for future group operations.

### 3.3.3 User Join

Suppose in session $S_s$, a subset $J_s \subseteq \mathcal{N} \setminus \mathcal{U}_s$ of new users join $\mathcal{U}_s$. A set of secret keys $K(U)$ has to be delivered to each new user $U \in J_s$, and a group key $GK_s$ needs to be established for the group $\mathcal{U}_s \cup J_s$ such that new users in $J_s$ cannot obtain any previous group keys. Let the logical key hierarchy be $T_s$. 

Group Controller. The GC does the following.
3.3. Group Operations

1. Updates the tree structure by adding $|J_s|$ new leaves to the tree.

   (a) The new leaves can be simply attached to the existing internal nodes, or
   (b) New internal nodes may be created in order to attach the new leaves. A new internal node replaces an existing node (either an internal node or a leaf) and the existing node (and its descendants if any) becomes the child (and the descendants if any) of the new internal node.

The GC then rearranges the levels of the affected nodes and keys in the updated tree $T'_s$, and publishes $T'_s$ in a public bulletin board. For efficient re-keying, the new leaves should be added in such a way that $T'_s$ is as balanced as possible.

2. Produces a randomly chosen key for each new leaf and associates each new user $U \in J_s$ to a new leaf. The GC then securely sends the key associated with the user to him over a secure unicast channel. The leaf key of the user $U$ is $K_{w_{lc}}(U)$. Observe that all users in $U_s \cup J_s$ are in $T'_s$.

3. In the updated tree $T'_s$, let $I(J_s) = \bigcup_{U \in J_s} I(U) \setminus \{I_{w_{lc}}(U)\}$ be the set of internal nodes belonging to the new users. Joining $J_s$ and establishing the group key $GK_s$ (updating the root key) requires the internal keys of all nodes in $I(J_s)$ to be updated. The GC performs the re-keying scheme in Section 3.2 with $T = T'_s$ and $K(T) = \{K_{w_{lc}}(I): I \in I(J_s)\}$.

**Users.** Users in $U_s \cup J_s$ do the following.

1. Before re-keying, each new user $U \in J_s$ knows only $I(U)$, but not $K(U) \setminus \{K_{w_{lc}}(U)\}$. User $U$ performs the re-keying operation described in Section 3.2 to obtain the updated keys in $K(U)$ and the group key $GK_s$.

2. Each affected user $U \in U_s$ adds the new internal nodes (new ancestors of $U$’s leaf) to $I(U)$ and rearranges the levels of his affected nodes and keys. Note that the user does not know the keys of the new internal nodes. The user performs the re-keying operation described in Section 3.2 to update the keys in $K(U) \cap K(J_s)$ (to obtain the updated keys of the new internal nodes) and obtain the group key $GK_s$.

3. The rest of the users in $U_s$ perform the re-keying operation of Section 3.2 to update the keys in $K(U) \cap K(J_s)$ and obtain the group key $GK_s$. 
Note that unlike other schemes where to join a new user \( U \) the GC must send all the keys in \( K(U) \) to \( U \) through a secure unicast channel, the join operation in our scheme requires the GC to send only the leaf key to \( U \) through a secure unicast channel. The new user obtains other keys through the re-keying (multicast channel).

We note that the logical key hierarchy for the next session \( S_{s+1} \) will be \( T_{s+1} = T_s' \) with user group \( \mathcal{U}_{s+1} = \mathcal{U}_s \cup \mathcal{J}_s \).

**Example 3.2** Suppose \( \mathcal{J}_s = \{U_9, U_{10}\} \) join the group in Figure 3.1.

1. The GC updates the tree structure to Figure 3.3 by adding two new leaves \( I_{15}^{(3)} \), \( I_{16}^{(3)} \) and a new internal node \( I_{14}^{(2)} \) (nodes with bold line). The node in dashed line is the affected node and has been rearranged to the new level. Keys of the grey coloured nodes must be updated, that is, \( K(T) = \{K_{14}^{(2)}, K_6^{(2)}, K_2^{(1)}, K_0^{(0)}\} \). The result of re-keying is \( K_{14}^{(2)} = r_1, K_6^{(2)} = r_2, K_2^{(1)} = h^1(r_1), K_0^{(0)} = h^2(r_1) \) and so \( M_{rkey} = \{E_{K_5^{(3)}}(K_{14}^{(2)}), I_{15}^{(3)}, E_{K_{15}^{(3)}}(K_{14}^{(2)}), I_{15}^{(3)}, E_{K_{12}^{(3)}}(K_6^{(2)}), I_{12}^{(3)}, E_{K_{13}^{(3)}}(K_6^{(2)}), I_{13}^{(3)}, E_{K_{16}^{(3)}}(K_6^{(2)}), I_{16}^{(3)}, E_{K_{6}^{(2)}}(K_2^{(1)}), I_{6}^{(2)}, E_{K_{1}^{(3)}}(K_0^{(0)}), I_{1}^{(1)} \} \).

2. After decryption, users \( U_1, \ldots, U_5 \) will have \( K_0^{(0)} \) from \( M_{rkey} \). Users \( U_6 \) and \( U_9 \) will have \( K_{14}^{(2)} \) from \( M_{rkey} \) and calculate \( K_2^{(1)} = h^1(K_{14}^{(2)}), K_0^{(0)} = h^2(K_{14}^{(2)}) \). Users \( U_7, U_8, U_{10} \) will have \( K_6^{(2)} \) and \( K_2^{(1)} \) from \( M_{rkey} \), and calculate \( K_0^{(0)} = h^1(K_2^{(1)}) \). The group key is \( GK_s = K_0^{(0)} = h^2(r_1) \).

![Figure 3.3: Updated tree \( T_s' \) when joining new users \( U_9 \) and \( U_{10} \)](image)

In light of Theorem 3.3, part \( (iv) \), and using an argument similar to the proof of Theorem 3.5, we have

**Theorem 3.6** User join provides join secrecy (backward secrecy). That is, for any session \( S_s \) where \( S_1 \leq S_s \leq S_M \), an arbitrary size collusion \( \mathcal{C} = \bigcup \mathcal{J}_s, S_s \leq S_a \leq S_M, \mathcal{E}_a = \text{Join} \), cannot distinguish the group key \( GK_b, S_1 \leq S_b < S_s \), from a random string.
3.4 Key Recovery

**Theorem 3.7** User eviction together with user join provide evict-join secrecy (forward-backward secrecy). That is, for any sessions \( S_{s_1}, S_{s_2} \) where \( S_1 \leq s_1 < s_2 \leq S_M \), an arbitrary size collusion \( C = (\bigcup R_{a_1}) \bigcup (\bigcup J_{a_2}), S_1 \leq s_{a_1} \leq S_{s_1}, E_{a_1} = \text{Evict}, s_{a_2} \leq S_{a_2} \leq S_M, E_{a_2} = \text{Join}, \) cannot distinguish the group key \( GK_b, S_{s_1} \leq s_b < S_{a_2} \), from a random string.

### 3.4 Key Recovery

In this section we propose a solution to increase reliability of re-keying messages by allowing legitimate users to recover their previous group keys in each session. More specifically, in each session \( S_s \), legitimate users can recover \( k \) previous group keys \( GK_{s-k}, \ldots, GK_{s-1} \). A straightforward approach to provide this property is to give \( GK_{s-k}, \ldots, GK_{s-1} \) in session \( S_s \) encrypted with the session key \( GK_s \). It is easy to show that this satisfies the security requirement of forward secrecy, but does not satisfy the security requirements of backward secrecy and forward-backward secrecy.

Our proposed key recovery technique is as follows. A group key \( GK \) is broken into two values \( P \) and \( F \). For each group key, the value \( P \) is part of the re-keying messages in the \( k \) preceding sessions while the value \( F \) is part of the re-keying messages in the \( k \) subsequent sessions. More specifically, for a group key \( GK_s \) in session \( S_s \), \( P_s \) is given in sessions \( S_{s-k}, \ldots, S_{s-1} \) and \( F_s \) is given in sessions \( S_{s+1}, \ldots, S_{s+k} \). Therefore, computing (recovering) the group key \( GK_s \) requires a pair \( P_s \) and \( F_s \), where \( P_s \) and \( F_s \) can be obtained in a session \( S_{s_p} \) and a session \( S_{s_f} \), respectively, for any \( s-k \leq s_p < s_f \leq s+k \). This implies that the group key \( GK_s \) can be recovered in any session \( S_{s+1}, \ldots, S_{s+k} \). In general, the message in session \( S_s \) will include two sets, \( \mathcal{P}_s = \{P_{s+1}, \ldots, P_{s+k}\} \) and \( \mathcal{F}_s = \{F_{s-k}, \ldots, F_{s-1}\} \)\(^6\) which makes it possible to compute (recover) \( k \) preceding session keys \( GK_{s-k}, \ldots, GK_{s-1} \). A snapshot of the sets \( \mathcal{P} \) and \( \mathcal{F} \) over multiple sessions is given in Figure 3.4.

In practice, the value \( P \) can be chosen randomly and the value \( F \) will be computed from \( GK \) and \( P \), that is, \( F = GK \oplus P \) where \( \oplus \) denotes the bitwise Exclusive-OR operation. This is possible since for a key \( GK_s \) in session \( S_s \), \( P_s \) is given in sessions before session \( S_s \) and \( F_s \) is given in sessions after session \( S_s \). Thus, after obtaining \( P_s \) and \( F_s \), it is easy to compute \( GK_s = P_s \oplus F_s \). In general, the set \( \mathcal{P} \) contains random values and the set \( \mathcal{F} \) contains values that are computed. Observe that for a session \( S_s \), the set \( \mathcal{P}_s \) inherits \( k-1 \) elements from the set \( \mathcal{P}_{s-1} \), that is, \( \mathcal{P}_s \cap \mathcal{P}_{s-1} = \{P_{s+1}, \ldots, P_{s+k-1}\} \), and

\(^6\)This is not the case for the first \( k \) sessions \((s = 1, \ldots, k)\).
3.4. Key Recovery

![Diagram](image)

Figure 3.4: The sets \( P \) and \( F \) in sessions \( S_{s-1}, S_s, \) and \( S_{s+1} \)

the element \( P_{s+k} \) will be randomly generated. Also, the set \( F_s \) inherits \( k - 1 \) elements from the set \( F_{s-1} \), that is, \( F_s \cap F_{s-1} = \{ F_{s-k}, \ldots, F_{s-2} \} \), and the element \( F_{s-1} \) is computed from \( F_{s-1} = GK_{s-1} \oplus P_{s-1} \).

We can incorporate the above key recovery method into the re-keying system. The result will be a system with the ability to not only establish the group keys, but also recover previous group keys that are lost, while maintaining all the security properties. To do so, the GC multicasts a key recovery message \( M_{\text{rec}} \) in addition to a re-keying message \( M_{\text{rkey}} \). Suppose an update to the multicast group \( U_s \) is required. The update is by either a user eviction or a user join operation. Without loss of generality, we assume a user eviction operation is invoked in session \( S_s \).

**Group Controller.** The GC does the following.

1. Performs the re-keying algorithm to obtain \( GK_s \) and multicasts \( M_{\text{rkey}} \).

2. Generates the set \( P_s = \{ P_{s+1}, \ldots, P_{s+k} \} \) and computes the set \( F_s = \{ F_{s-k}, \ldots, F_{s-1} \} \) where \( \forall F_j \in F_s, F_j = GK_j \oplus P_j \). Next, the GC multicasts key recovery message \( M_{\text{rec}} = \{ E_{GK_s}(P_s \cup F_s) \} \).

**Users.** Each user \( U \in U_s \setminus R_s \) does the following.

1. Receives \( M_{\text{rkey}} \) and performs the re-keying algorithm to obtain \( GK_s \).

2. Receives \( M_{\text{rec}} \) and decrypts \( D_{GK_s}(M_{\text{rec}}) \) to obtain the sets \( P_s \) and \( F_s \). User \( U \) may recover the previous session keys \( GK_{s-k}, \ldots, GK_{s-1} \) by computing \( GK_j = P_j \oplus F_j \) where \( P_j \in P_g \), for \( s - k \leq j \leq s - 1 \), \( j - k \leq g \leq j - 1 \), and \( F_j \in F_g \), for
3.4. Key Recovery

\[ s - k \leq j \leq s - 1, j + 1 \leq g \leq s. \] User \( U \) may retain \( \mathcal{F}_s \) to be used in subsequent sessions.

**Theorem 3.8** Recovering \( k \) previous group keys requires (i) a message length of \( 2k \log_2 q \) bits and (ii) a legitimate user to perform \( k \) XOR operations.

**Theorem 3.9** The key recovery system maintains,

(i) evict secrecy (forward secrecy) in the sense of Theorem 3.5.

(ii) join secrecy (backward secrecy) in the sense of Theorem 3.6.

(iii) evict-join secrecy (forward-backward secrecy) in the sense of Theorem 3.7 provided that \( S_{s_2} \geq S_{s_1+2k} \).

**Proof:** (sketch) The colluders have access to some key recovery messages. We show that using the information, the colluders cannot discover keys of the sessions that none of them belong to, unless they can break the encryption algorithm \( E \).

(i) Observe that the colluding users in \( \mathcal{C} = \bigcup \mathcal{R}_a, S_1 \leq S_a \leq S_s, E_a = \text{Evict}, \) know only the values \( P_2, \ldots, P_{s+k-1} \) and \( F_1, \ldots, F_{s-2} \). Using these values, \( \mathcal{C} \) may recover \( GK_2, \ldots, GK_{s-2} \) but not \( GK_{s-1}, \ldots, GK_{s+k-1} \) since they do not know the values \( F_{s-1}, \ldots, F_{s+k-1} \). It follows that they cannot find group keys \( GK_s, \ldots, GK_M \).

(ii) Observe that the colluding users in \( \mathcal{C} = \bigcup \mathcal{J}_a, S_s \leq S_a \leq S_M, E_a = \text{Join}, \) know only the values \( P_{s+1}, \ldots, P_M \) and \( F_{s-k}, \ldots, F_{M-1} \). Using these values, \( \mathcal{C} \) may recover \( GK_{s+1}, \ldots, GK_{M-1} \) but not \( GK_{s-k}, \ldots, GK_s \) since they do not know the values \( P_{s-k}, \ldots, P_s \). It follows that they cannot find group keys \( GK_1, \ldots, GK_{s-1} \).

(iii) Let \( S_{s_2} = S_{s_1+2k} \). Users in \( \bigcup \mathcal{R}_a, S_1 \leq S_a \leq S_{s_1}, E_a = \text{Evict}, \) know only the values \( P_2, \ldots, P_{s_1+k-1} \) and \( F_1, \ldots, F_{s_1-2} \). Users in \( \bigcup \mathcal{J}_b, S_{s_1+2k} \leq S_b \leq S_M, E_b = \text{Join}, \) know only the values \( P_{s_1+2k+1}, \ldots, P_M \) and \( F_{s_1+k}, \ldots, F_{M-1} \). Using these values, \( \mathcal{C} \) cannot recover \( GK_{s_1-1}, \ldots, GK_{s_1+2k} \) which means they cannot find group keys \( GK_{s_1}, \ldots, GK_{s_2-1} \). Observe that if \( S_{s_2} < S_{s_1+2k} \), \( \mathcal{C} \) may find some group keys of \( GK_{s_1}, \ldots, GK_{s_2-1} \). \( \square \)

Since efficiency and recovery capability depend on \( k \), in practice we must choose \( k \) by considering the network rate of packet loss. \( k \) can be dynamically chosen to suit network conditions.
3.5 Further Discussion and Conclusion

So far, we only allowed one operation in a session: either user eviction or user join. To increase efficiency we may consider user eviction and user join in the same session. Suppose in session $S_s$, the GC needs to evict a subset of users $R_s$ and join a subset of new users $J_s$. To obtain the new session key $GK_s$, the GC first removes leaves of the evicted users and the redundant internal nodes, then adds the new internal nodes and the new leaves, and rearranges the levels of the affected nodes and keys to form $T'_s$. The GC then performs the re-keying operation with $T = T'_s$ and $K(T) = K(R_s) \cup K(J_s)$ (note that $K(R_s) \cap K(J_s) \neq \emptyset$). Users do the necessary adjustment to their nodes and keys, and perform re-keying operations to update (obtain) their keys including the session key $GK_s$.

In some cases, the session key that is used to encrypt group communication must be updated without changing membership of the group. To refresh the encryption key, the GC might generate a new root key, encrypt it with its children keys and multicast the encrypted version to group members. Another way to produce a fresh encryption key for the same group is to employ a pseudo-random function $f_k$ [35]. An encryption key is determined from $f_k$ on a unique number with $k$ being the session key of the group. Note that $f_k$ will give a new encryption key for a new unique number, in which case the GC only needs to multicast a new unique number to establish a fresh encryption key.

We have considered the re-keying problem for groups in multicast environments with unreliable communication. We have proposed a re-keying scheme that can be used for multiple user eviction and join. We have proven security of the scheme and shown that, relative to the trivial method, it has better efficiency in terms of the number of rounds, communication bandwidth and user computation. The scheme also requires less computation compared to other known methods. We have also considered the problem of reliability of re-keying when the packets can be lost, and proposed a key recovery technique that allows the keys for $k$ previous sessions to be calculated, hence providing robustness against loss of packets. We have evaluated security and efficiency of the resulting method.
Chapter 4

A Collaborative Re-keying Scheme

4.1 Introduction

The secure multicast re-keying schemes in the previous chapter and in [17, 18, 58, 85, 87] assume one trusted party (the group controller) generates and distributes users’ keys in the initialisation phase, and also manages key updates. In this chapter we study the decentralised setting of re-keying scheme where collaborations of group users can take over group controller’s responsibilities. We do not require a group controller (GC) for key generation and distribution in the system setup, or for user eviction and addition operations. This model allows the system to function without requiring a GC and has the advantages of decentralised systems as described in Chapter 1.

4.1.1 A Summary of Our Contribution

We construct a secure and efficient collaborative re-keying scheme based on a logical key hierarchy (LKH) and a key agreement protocol whereby two or more parties jointly initialise the system, execute group operations and establish new session keys with communication over only multicast channels. We may consider the proposed scheme as an extension of the two-party Diffie-Hellman (DH) key exchange protocol [26] where the extension involves multiple parties in dynamic environments. The basic idea is to successively run the basic two-party DH protocol from leaf to root direction in order to update keys of a logical tree including the session key. We note that unlike other collaborative schemes such as [16, 40, 80, 81], efficient only for small groups, our proposed collaborative scheme is scalable to large groups and has low computation, communication and storage costs for re-keying. In fact, it has at most $\log_2 n$ rounds of communication, at most $\log_2 n$ modular exponentiations and $\log_2 n + 1$ keys for a user (only one of those keys is kept secret). The scheme is stateful requiring secret keys to be updated in each session. The scheme supports arbitrary sets of evicted users.
and new users, and provides forward, backward and forward-backward secrecy with arbitrary collusions.

We show the efficiency comparison of the proposed scheme and well-known collaborative schemes [16, 81] in Table 4.1. The user storage and communication costs are measured in terms of the number of keys, and the computation cost is measured in terms of the number of exponentiations. We note that scheme [81] requires a single user to store \( n + 1 \) keys and other users to store one key. User eviction of the scheme requires a single authorised user to compute \( n \) exponentiations and other authorised users to compute one exponentiation. Moreover, user admission of the scheme requires new users to compute \( n \) exponentiations and existing users to compute one exponentiation. The proposed scheme and schemes [16, 81] allow eviction of up to \( n \) users and admission of an arbitrary number of users.

### Table 4.1: Efficiency comparison of the proposed scheme and several relevant schemes

<table>
<thead>
<tr>
<th></th>
<th>Proposed Scheme</th>
<th>BD [16]</th>
<th>STW [81]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Eviction</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Storage</td>
<td>( \log_2 n + 1 )</td>
<td>1</td>
<td>1 or ( n + 1 )</td>
</tr>
<tr>
<td><strong>Single User</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>( \leq \log_2 n )</td>
<td>( 2n )</td>
<td>( n )</td>
</tr>
<tr>
<td>Computation</td>
<td>( \leq \log_2 n )</td>
<td>( n + 2 )</td>
<td>1 or ( n )</td>
</tr>
<tr>
<td>Rounds</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Multiple (( \beta )) Users</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>( \leq n - 1 )</td>
<td>( 2n )</td>
<td>( n )</td>
</tr>
<tr>
<td>Computation</td>
<td>( \leq \log_2 n )</td>
<td>( n + 2 )</td>
<td>1 or ( n )</td>
</tr>
<tr>
<td>Rounds</td>
<td>( \leq \log_2 n )</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>User Join</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Storage</td>
<td>( \log_2 n + 1 )</td>
<td>1</td>
<td>1 or ( n + 1 )</td>
</tr>
<tr>
<td><strong>Single User</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>( \leq \log_2 n )</td>
<td>( 2n )</td>
<td>( 2n - 1 )</td>
</tr>
<tr>
<td>Computation</td>
<td>( \leq \log_2 n )</td>
<td>( n + 2 )</td>
<td>1 or ( n - 1 )</td>
</tr>
<tr>
<td>Rounds</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Multiple (( \alpha )) Users</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>( \leq n - 1 )</td>
<td>( 2n )</td>
<td>( n(\alpha + 1) - \frac{1}{2}(\alpha^2 - \alpha + 2) )</td>
</tr>
<tr>
<td>Computation</td>
<td>( \leq \log_2 n )</td>
<td>( n + 2 )</td>
<td>1 or ( n )</td>
</tr>
<tr>
<td>Rounds</td>
<td>( \leq \log_2 n )</td>
<td>2</td>
<td>( \alpha + 1 )</td>
</tr>
</tbody>
</table>

\( n \) is the total number of users after user eviction or user join.

We note that there are some similarity in the scheme descriptions of this chapter and the previous chapter, nevertheless, we deliberately separate those schemes into two chapters for the sake of clarity.
4.1.2 Protocols

The join and evict protocols in the collaborative scheme are described in Tables 4.2 and 4.3, respectively.

Table 4.2: Join protocol

If $E_s = \text{Join}$

| Input:  | $U_s$, $J_s \subseteq \mathcal{N} \setminus U_s$, $\mathcal{P}_s \subseteq U_s \cup J_s$ |
| Process: | for some rounds:
| - some $U_z \in \mathcal{P}_s$ multicasts a re-keying message $M_{rkey}$
| - all $U_i \in U_s \cup J_s$, $U_i \neq U_z$
| - all $U_i \in J_s$ do computation on $M_{rkey}$ and secrets of former rounds
| - some $U_i \in U_s$ do computation on $M_{rkey}$ and $K(U_i)$ |
| Output: | - all $U_i \in J_s$ obtain secret information $K(U_i)$
| - all $U_i \in U_s \cup J_s$ share a group key $GK_s$
| - $U_{s+1} = U_s \cup J_s$ |

The join operation comprises several rounds wherein new users in $J_s$ gradually obtain their secret key sets, and also a group key is gradually formed for the enlarged group. In every round, each sponsor $U_z$ in a subset of $\mathcal{P}_s$ multicasts a re-keying message $M_{rkey}$ containing information for the new users $U_i \in J_s$ to ascertain some pieces of their secret key sets $K(U_i)$. Each re-keying message in a round also contains some information to establish the group key $GK_s$, requiring some users $U_i \in U_s$ to perform computation on the messages and $K(U_i)$. After fulfilling all rounds, the new users have their complete secret key sets and users in $U_s \cup J_s$ share a common key.

Table 4.3: Evict protocol

| Input:  | $U_s$, $R_s \subseteq U_s$, $\mathcal{P}_s \subseteq U_s \setminus R_s$ |
| Process: | for some rounds
| - some $U_z \in \mathcal{P}_s$ multicasts a re-keying message $M_{rkey}$
| - all $U_i \in U \setminus R_s$, $U_i \neq U_z$
| - some $U_i \in U \setminus R_s$ do computation on $M_{rkey}$ and $K(U_i)$ |
| Output: | - all $U_i \in U \setminus R_s$ share a group key $GK_s$
| - all $U_i \in R_s$ have unusable secret information $K(U_i)$
| - $U_{s+1} = U_s \setminus R_s$ |

The evict operation is divided into a number of rounds as well. In each round, each evictor $U_z$ in a subset of $\mathcal{P}_s$ multicasts a re-keying message $M_{rkey}$ and authorised users
4.2. A Collaborative LKH Construction

$U_i \in \mathcal{U} \setminus \mathcal{R}_s$ perform computation on the messages and their secret key sets $\mathcal{K}(U_i)$ to establish the group key $GK_s$. The authorised users also disable the secret key sets of the evicted users.

**Organisation of this chapter.** We propose a collaborative re-keying scheme in Section 4.2, and give security and efficiency assessments. The application of the scheme to group operations is described in Section 4.3. Section 4.4 contains a summary of this chapter.

The main results of this chapter appeared in the *Proceedings of The Fifth Australasian Conference on Information Security and Privacy – ACISP 2000* [50].

### 4.2 A Collaborative LKH Construction

The notions of two-party key agreement protocol and extended logical key hierarchy are central to the construction. They are defined as follows [59].

**Definition 4.1** A two-party key agreement protocol involves two users establishing a common secret key $K$. They use public functions $f()$ and $h()$, and private keys $K_i$ for $U_i$, $i = 1, 2$. User $U_1$ sends $Y_1 = f(K_1)$ to user $U_2$, and user $U_2$ sends $Y_2 = f(K_2)$ in return, then they can compute $K'_1 = h(K_1, Y_2)$ and $K'_2 = h(K_2, Y_1)$, respectively. If $f()$ and $h()$ are chosen such that $K'_1 = K'_2$, then the two users will share a common key. For a key $K$, $Y = f(K)$ is called the shadow key of $K$.

For simplicity, we employ the Diffie-Hellman key exchange protocol [26] to determine the shared common key. Let $p$ and $q$ be large primes that satisfy $q \mid p - 1$, and $g$ be an element in $\mathbb{Z}_p^*$ whose order is $q$. Let $K_1, K_2 \in \mathbb{Z}_q^*$ be the keys belonging to users $U_1$ and $U_2$, respectively, and let $Y_1 = g^{K_1} \mod p, Y_2 = g^{K_2} \mod p$ be the shadow keys. The shared key is $K = K'_1 = (Y_2)^{K_1} = (Y_1)^{K_2} = K'_2 = (g^{K_1 K_2} \mod p) \mod q$.

The extended logical key hierarchy is an extension of the logical key hierarchy defined in Chapter 3. Although many of the components are the same, we describe them all here for completeness.

**Definition 4.2** An extended logical key hierarchy (x-LKH) is a tree structure where each node corresponds to a key and each leaf corresponds to a user. A user knows not only the keys of nodes along the path from the user’s leaf to the root (as in LKH, Definition 3.2) but also the shadow keys of nodes that are siblings of nodes along the path from the user’s leaf to the root.
4.2. A Collaborative LKH Construction

We require a full binary tree (a binary tree in which each node has exactly zero or two children)\(^1\) in the construction. Let \(U = \{U_1, \ldots, U_n\}\) be the set of users and \(T\) be the full binary tree with \(n\) leaves. Nodes of the tree are divided into internal nodes and leaves. Each node is given a label \(I_{w}^{l}\) and a key \(K_{w}^{l}\), called the node key, where \(l\) is the level of the node and \(w\) is a unique number identifying the node. Node keys are divided into internal keys and leaf keys. Let \(\hat{I}_{w}^{l}\) denote the sibling of \(I_{w}^{l}\), and \(Y_{w}^{l}\) denote the shadow key of \(K_{w}^{l}\). Node labels and shadow keys are publicly known while node keys are kept secret. A user \(U\) is attached to a leaf \(I_{w}^{t_U}\) where \(t_U\) is the level of the leaf corresponding to user \(U\).

Let \(A(I_{w}^{l}) = \{I_{w}^{0}, \ldots, I_{w}^{l-2}, I_{w}^{l-1}\}\) be the set of ancestors of \(I_{w}^{l}\). Let the set \(\mathcal{I}(U) = A(I_{w}^{t_U}) \cup \{I_{w}^{t_U}\}\) contain the leaf \(I_{w}^{t_U}\) and all ancestors of that leaf. Let \(\bar{\mathcal{I}}(U) = \{\hat{I}_{w}^{l} : I_{w}^{l} \in \mathcal{I}(U)\}\) be the set of siblings of nodes in \(\mathcal{I}(U)\).\(^2\) User \(U\) knows the node keys \(\mathcal{K}(U) = \{K_{w}^{l} : I_{w}^{l} \in \mathcal{I}(U)\}\) corresponding to the nodes in \(\mathcal{I}(U)\), and the shadow keys \(\mathcal{Y}(U) = \{Y_{w}^{l} : I_{w}^{l} \in \bar{\mathcal{I}}(U)\}\) corresponding to the nodes in \(\bar{\mathcal{I}}(U)\). Only the \(\mathcal{K}(U)\) are kept secret by the user. Note that all users have the root node \(I_{w}^{0}\) with corresponding root key \(K_{w}^{0}\). Figure 4.1 gives an example of a tree structure for 8 users, where, for example, user \(U_1\) knows \(\mathcal{K}(U_1) = \{K_{0}^{0}, K_{1}^{(1)}, K_{3}^{(2)}, K_{7}^{(3)}\}\) and \(\mathcal{Y}(U_1) = \{Y_{2}^{(1)}, Y_{4}^{(2)}, Y_{8}^{(3)}\}\).

\[\text{Figure 4.1: An example of a tree structure for 8 users}\]

Characteristics of node keys

A leaf key is randomly generated by the user corresponding to the leaf, and is known only by that user. An internal key, which is known to more than a user, is computed as follows. An internal node in a full binary tree has two child nodes. For an internal node \(I_{w}^{l}\), let \(\mathcal{D}(I_{w}^{l}) = \{I_{w1}^{l+1}, I_{w2}^{l+1}\}\) denote the set of its two child nodes. Computation of

\(^1\)Although the tree may be unbalanced, for the sake of clarity we assume a balanced tree in our efficiency analysis.

\(^2\)The root node \(I_{w}^{0}\) does not have a sibling so we may consider \(\bar{\mathcal{I}}(U) \neq \hat{I}_{w}^{0}\).
4.2. A Collaborative LKH Construction

the internal key $K_{w}^{(l)}$ requires the knowledge of the key of one of the two child nodes and the shadow key of the other child node. More specifically,

$$
K_{w}^{(l)} = (Y_{w_{1}}^{(l+1)}K_{w_{2}}^{(l+1)})
= (Y_{w_{2}}^{(l+1)}K_{w_{1}}^{(l+1)})
= (g^{K_{w_{1}}^{(l+1)}K_{w_{2}}^{(l+1)}} \mod p) \mod q .
$$

The above procedure implies that internal keys are computed in the direction of the leaf to the root. Then we have the following lemma.

Lemma 4.1 Knowing a leaf key together with shadow keys of siblings of nodes along the path from the leaf to the root, it is possible to calculate internal keys of nodes along the path from the leaf to the root.

Therefore instead of securely storing all node keys $K(U)$, it is enough for the user $U$ to securely keep the leaf key $K_{t_{U}}^{(l_{U})}$ and store the shadow keys $Y(U)$ in unprotected memory. All internal keys $K(U) \setminus \{K_{t_{U}}^{(l_{U})}\}$ can be calculated from these information. As an example, user $U_1$ in Figure 4.1 keeps $K_{7}^{(3)} \in K(U_1)$ secure and stores $Y(U_1) = \{Y_{2}^{(1)}, Y_{4}^{(2)}, Y_{8}^{(3)}\}$. This allows him to compute internal keys $K_{0}^{(0)}, K_{1}^{(1)}, K_{3}^{(2)} \in K(U_1)$. Another advantage of storing $Y(U)$ is in reducing the size of the multicast message as the shadow keys might be reused in the key update operation (see next section). Nevertheless, note that in practice a shadow key is longer (an element of $\mathbb{Z}_{p}^{*}$) than a node key (an element of $\mathbb{Z}_{q}$), and so larger storage is required to store $Y(U)$.

In general the key $K_{w}^{(l)}$ of a node $I_{w}^{(l)}$ is computed in the following form.

$$
K_{w}^{(l)} = \begin{cases} 
K_{w}^{(l)} & \text{if } I_{w}^{(l)} \text{ is a leaf} \\
g \prod_{I_{d}^{(b)} \in \mathcal{I}(I^{(l)}), K_{w}^{(b)}} & \text{otherwise}
\end{cases}
$$

Let $\mathcal{U}(I_{w}^{(l)}) = \{U : I_{w}^{(l)} \in \mathcal{I}(U)\}$ be the set of users $U$ having $I_{w}^{(l)}$ in $\mathcal{I}(U)$ (leaves having ancestor $I_{w}^{(l)}$). Observe that $K_{w}^{(l)}$ is a function of the leaf keys $K_{t_{U}}^{(l_{U})}$ of all users $U \in \mathcal{U}(I_{w}^{(l)})$. For example, $K_{1}^{(1)}$ in Figure 4.1 is computed as

$$
K_{1}^{(1)} = \left( g^{(g^{(K_{7}^{(3)}K_{8}^{(3)})} \cdot g^{(K_{9}^{(3)}K_{10}^{(3)})}} \mod p \right) \mod q .
$$

We call a node key that is consistent with equation (4.2) an LKH consistent node key.
4.2.1 Updating Internal Keys

We give a method for users to interactively update the root key $K_w^{(0)}$ by updating a subset of internal keys (including the root key) in a logical key hierarchy. All membership event protocols, which we propose in the next section, use the method as their fundamental operation.

Suppose there is a set $\mathcal{I}(T)$ consisting of some internal nodes in the tree $T$ that have the property that for an internal node $I_w^{(l)} \in \mathcal{I}(T)$, all ancestors of the node are also in the set, that is, $\mathcal{A}(I_w^{(l)}) \subset \mathcal{I}(T)$. Observe that $\mathcal{I}(T)$ always includes the root node $I_w^{(0)}$.

Let $\mathcal{K}(T) = \{K_w^{(l)} : I_w^{(l)} \in \mathcal{I}(T)\}$ be the key set of the node set $\mathcal{I}(T)$. Suppose all node keys in the tree $T$, excluding those in $\mathcal{K}(T)$, are LKH consistent keys. We require the key set $\mathcal{K}(T)$ to be updated to a new key set $\mathcal{K}'(T) = \{K_w^{(l)} : I_w^{(l)} \in \mathcal{I}(T)\}$ of LKH consistent keys.

In the following, we show how users in $\mathcal{U}(\mathcal{I}_w^{(l)})$ update the internal key $K_w^{(l)} \in \mathcal{K}(T)$ to $K_w^{(l')} \in \mathcal{K}'(T)$, for all $I_w^{(l)} \in \mathcal{I}(T)$. Observe that updating $\mathcal{K}(T)$ implies updating the root key $K_w^{(0)} \in \mathcal{K}(T)$. We assume each user $U \in \mathcal{U}$ knows the node keys $\{K_w^{(l)} : I_w^{(l)} \in \mathcal{I}(U) \setminus \mathcal{I}(T)\} \subset \mathcal{K}(U)$ and the shadow keys $\{Y_w^{(l)} : I_w^{(l)} \in \mathcal{I}(U) \setminus \mathcal{I}(T)\} \subset \mathcal{Y}(U)$.

The user $U$ needs to update node keys $\{K_w^{(l)} : I_w^{(l)} \in \mathcal{I}(U) \cap \mathcal{I}(T)\} \subset \mathcal{K}(U)$ and shadow keys $\{Y_w^{(l)} : I_w^{(l)} \in \mathcal{I}(U) \cap \mathcal{I}(T)\} \subset \mathcal{Y}(U)$ accordingly.

**Protocol to Update $\mathcal{K}(T)$**

Recall that an internal key is computed from keys of its two child nodes, and $\mathcal{D}(I_w^{(l)})$ denotes the child node set of $I_w^{(l)}$. Observe that for all nodes $I_w^{(l)} \in \mathcal{I}(T)$, $\mathcal{D}(I_w^{(l)})$ satisfies one of the following three cases.

- **Case I**: $\mathcal{D}(I_w^{(l)}) \cap \mathcal{I}(T) = \emptyset$,
- **Case II**: $|\mathcal{D}(I_w^{(l)}) \cap \mathcal{I}(T)| = 1$, and
- **Case III**: $|\mathcal{D}(I_w^{(l)}) \cap \mathcal{I}(T)| = 2$, i.e., $\mathcal{D}(I_w^{(l)}) \subseteq \mathcal{I}(T)$.

Let $t = \max\{l : I_w^{(l)} \in \mathcal{I}(T)\}$ be the deepest level of nodes in $\mathcal{I}(T)$. The key update process starts from internal nodes at level $t$ and goes towards the internal node at level 0 (root node). The protocol to compute $K_w^{(l+1)}$ for an internal node $I_w^{(l)} \in \mathcal{I}(T)$ is as follows. Note that the child nodes in $\mathcal{D}(I_w^{(l)})$ are $I_{w_1}^{(l+1)}$ and $I_{w_2}^{(l+1)}$, and the user set $\mathcal{U}(I_w^{(l)})$ is the union of two distinct user subsets $\mathcal{U}(I_{w_1}^{(l+1)})$ and $\mathcal{U}(I_{w_2}^{(l+1)})$.

**Protocol for case I** The new internal key is computed from $K_{w_1}^{(l+1)}$ and $K_{w_2}^{(l+1)}$ by users $U \in \mathcal{U}(I_w^{(l)})$ as $K_w^{(l+1)} = (Y_{w_2}^{(l+1)})^{K_{w_1}^{(l+1)}} = (Y_{w_1}^{(l+1)})^{K_{w_2}^{(l+1)}} = (g^{K_{w_1}^{(l+1)}}K_{w_2}^{(l+1)} \mod p) \mod q$. 

4.2. A Collaborative LKH Construction

Protocol for case II Without loss of generality, let the child node $I_{w_1}^{(l+1)} \in \mathcal{D}(I_{w_1}^{(l)}) \cap \mathcal{I}(T)$ and the other child node $I_{w_2}^{(l+1)} \notin \mathcal{I}(T)$. Note that there exists a new key $K_{w_1}^{(l+1)}$ for $I_{w_1}^{(l+1)}$ which is known to users in $\mathcal{U}(I_{w_1}^{(l+1)})$. This is because $I_{w_1}^{(l+1)} \in \mathcal{I}(T)$ and the key update process is from level $h$ to level 0. The new internal key $K_w^{(l)}$ is computed from $K_{w_1}^{(l+1)} \in \mathcal{K}'(T)$ and $K_{w_2}^{(l+1)}$.

1. **Transmission**: A privileged user $U_z \in \mathcal{U}(I_{w_1}^{(l+1)})$ computes the new shadow key $Y_{w_1}^{(l+1)} = g^{K_w^{(l+1)}} \mod p$, and multicasts the re-keying message $\mathcal{M}_{rkey} = \{Y_{w_1}^{(l+1)} \parallel I_{w_1}^{(l+1)}\}$ for users in $\mathcal{U}(I_{w_2}^{(l+1)})$.

2. **Calculation**: Users $U \in \mathcal{U}(I_{w_1}^{(l+1)})$ compute the new internal key $K_w^{(l)}$ as follows.

   (a) Users $U \in \mathcal{U}(I_{w_1}^{(l+1)}) : K_w^{(l)} = (Y_{w_1}^{(l+1)} K_{w_1}^{(l+1)}) = (g^{K_{w_1}^{(l+1)}} K_{w_1}^{(l+1)} \mod p) \mod q$.

   (b) Users $U \in \mathcal{U}(I_{w_2}^{(l+1)}) : K_w^{(l)} = (Y_{w_1}^{(l+1)} K_{w_2}^{(l+1)}) = (g^{K_{w_1}^{(l+1)}} K_{w_2}^{(l+1)} \mod p) \mod q$.

Protocol for case III Both child nodes $I_{w_1}^{(l+1)}, I_{w_2}^{(l+1)} \in \mathcal{I}(T)$. Note that there exist new keys $K_{w_1}^{(l+1)}, K_{w_2}^{(l+1)} \in \mathcal{K}'(T)$ which are known to users in $\mathcal{U}(I_{w_1}^{(l+1)})$ and $\mathcal{U}(I_{w_2}^{(l+1)})$, respectively. The new internal key $K_w^{(l)}$ is computed from both new keys. It requires a privileged user $U_{z_1} \in \mathcal{U}(I_{w_1}^{(l+1)})$ and another privileged user $U_{z_2} \in \mathcal{U}(I_{w_2}^{(l+1)})$ to exchange re-keying messages. That is, each of them independently (and simultaneously) transmits a re-keying message $\mathcal{M}_{rkey}$ like the protocol for case II. Users $U_i \in \mathcal{U}(I_{w}^{(l)})$ compute the new internal key as $K_w^{(l)} = (g^{K_{w_1}^{(l+1)}} K_{w_2}^{(l+1)} \mod p) \mod q$.

A summary of the key update procedures is given in Tables 4.4 and 4.5.

<table>
<thead>
<tr>
<th>Table 4.4: Procedures for updating internal keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{K}'(T) = \emptyset$</td>
</tr>
<tr>
<td>for $l = h$ to 0 {</td>
</tr>
<tr>
<td>for all $I_w^{(l)} \in \mathcal{I}(T)$ {</td>
</tr>
<tr>
<td>if $\mathcal{D}(I_w^{(l)}) \cap \mathcal{I}(T) = \emptyset$</td>
</tr>
<tr>
<td>$K_w^{(l)} = \text{Protocol for case I}$</td>
</tr>
<tr>
<td>if $</td>
</tr>
<tr>
<td>$K_w^{(l)} = \text{Protocol for case II}$</td>
</tr>
<tr>
<td>if $</td>
</tr>
<tr>
<td>$K_w^{(l)} = \text{Protocol for case III}$</td>
</tr>
<tr>
<td>$\mathcal{K}'(T) = \mathcal{K}'(T) \cup {K_w^{(l)}}$</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
4.2. A Collaborative LKH Construction

<table>
<thead>
<tr>
<th>Users $U \in \mathcal{U}(T, \mathcal{K}(U), \mathcal{Y}(U))$</th>
<th>Users $U \in \mathcal{U}(I_{w_2}^{(l+1)})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_r^{(l)}<em>{w_1} = ((Y</em>{w_1}^{(l+1)}) K_{w_1}^{(l+1)} \mod p) \mod q$</td>
<td>$K_r^{(l)}<em>{w_1} = ((Y</em>{w_2}^{(l+1)}) K_{w_2}^{(l+1)} \mod p) \mod q$</td>
</tr>
<tr>
<td>$U_z : Y_{w_1}^{(l+1)} = g^{K_r^{(l+1)}_{w_1}} \mod p$</td>
<td>$U_z : Y_{w_2}^{(l+1)} = g^{K_r^{(l+1)}_{w_2}} \mod p$</td>
</tr>
<tr>
<td>$K_r^{(l)}<em>{w_1} = ((Y</em>{w_1}^{(l+1)}) K_{w_1}^{(l+1)} \mod p) \mod q$</td>
<td>$K_r^{(l)}<em>{w_2} = ((Y</em>{w_2}^{(l+1)}) K_{w_2}^{(l+1)} \mod p) \mod q$</td>
</tr>
<tr>
<td>Case I</td>
<td>Case II*</td>
</tr>
</tbody>
</table>

*Assuming $I_{w_1}^{(l+1)} \in \mathcal{I}(T)$.

The key update gives not only the new key set $\mathcal{K}'(T)$ but also the new shadow key set $\mathcal{Y}'(T) = \{Y_r^{(l+1)} : I_{w_1}^{(l+1)} \in \mathcal{I}(T)\}$. Observe that users $U \in \mathcal{U}$ need to update their node keys $K_r^{(l)}_{w_1}$ to $K_r^{(l+1)}_{w_1}$, for all $I_{w_1}^{(l+1)} \in \mathcal{I}(U) \cap \mathcal{I}(T)$. This can be done by updating their shadow keys $Y_{w_1}^{(l+1)}$ to $Y_{w_1}^{(l+1)}$, for all $I_{w_1}^{(l+1)} \in \mathcal{I}(U) \cap \mathcal{I}(T)$. This gives us the following corollary.

**Corollary 4.1** Knowing the leaf key together with the shadow keys of nodes in $\mathcal{I}(U) \setminus \mathcal{I}(T)$ and the new shadow keys of nodes in $\mathcal{I}(U) \cap \mathcal{I}(T)$, it is possible for user $U$ to calculate the internal keys of nodes in $\mathcal{I}(U) \setminus \mathcal{I}(T)$ and the new internal keys of nodes in $\mathcal{I}(U) \cap \mathcal{I}(T)$.

It is easy to see that the number of re-keying messages (new shadow keys) is equal to the size $|\mathcal{I}(T)|$ in the key update. Furthermore, it is necessary to know the smallest set of privileged users $\mathcal{P}$ required to multicast the re-keying messages. The properties of the set $\mathcal{P}$ are as follows:

- The cardinality of $\mathcal{P}$ is equal to the cardinality of a node set $\mathcal{I}(\mathcal{P})$, $\mathcal{I}(\mathcal{P}) \subseteq \mathcal{I}(T)$.

---

3It is unnecessary to compute and multicast the shadow key of root node $I_{w_1}^{(0)}$, so we may consider $\mathcal{Y}'(T) \not\equiv Y_r^{(0)}$. 

that has the following property: for a node $I^{(l)}_w \in \mathcal{I}(\mathcal{P})$, the node is not an ancestor of any other nodes in $\mathcal{I}(T)$, that is, $\mathcal{A}(I^{(l)}_w) \cap \mathcal{I}(T) = \emptyset$.

- The set $\mathcal{P}$ contains one element from each $\mathcal{U}(I^{(l)}_w)$, for all $I^{(l)}_w \in \mathcal{I}(\mathcal{P})$. That is, $\mathcal{P} = \{ U_{z_1}, U_{z_2}, \ldots, U_{z_{|\mathcal{P}|}} \}$ such that $U_{z_1} \in \mathcal{U}(I^{(l)}_{w_1}), U_{z_2} \in \mathcal{U}(I^{(l)}_{w_2}), \ldots, U_{z_{|\mathcal{P}|}} \in \mathcal{U}(I^{(l)}_{w_{|\mathcal{P}|}})$ and $I^{(l)}_w, I^{(l)}_{w_1}, \ldots, I^{(l)}_{w_{|\mathcal{P}|}} \in \mathcal{I}(\mathcal{P})$.

It is obvious that for each $I^{(l)}_w \in \mathcal{I}(\mathcal{P})$, the new shadow key $Y^{r(l)}_w$ can be multicasted by a privileged user $U \in \mathcal{P}$. Moreover observe that for $I^{(l)}_w \in \mathcal{I}(T) \setminus \mathcal{I}(\mathcal{P})$, the new shadow key $Y^{r(l)}_w$ can be multicasted by at least one privileged user $U \in \mathcal{P}$. This is because a node $I^{(l)}_w \in \mathcal{I}(T) \setminus \mathcal{I}(\mathcal{P})$ is an ancestor of at least one node $I^{(l)}_{w_2} \in \mathcal{I}(\mathcal{P})$ and so $\mathcal{U}(I^{(l)}_{w_2}) \subseteq \mathcal{U}(I^{(l)}_w)$. The privileged user that multicasts $Y^{r(l)}_{w_2}$ can also multicast $Y^{r(l)}_w$.

The above observation gives the following corollary.

**Corollary 4.2** Updating $\mathcal{K}(T)$ involves a set $\mathcal{P}$ of privileged users computing and multicasting re-keying messages (new shadow keys $Y^{r(l)}(T)$). The set $\mathcal{P}$ contains one user from each $\mathcal{U}(I^{(l)}_w)$, for all $I^{(l)}_w \in \mathcal{I}(T)$ such that $\mathcal{A}(I^{(l)}_w) \cap \mathcal{I}(T) = \emptyset$.

In practice, the privileged set $\mathcal{P}$ consists of users having more resources, such as high computation power and bandwidth. If in the middle of a key update process, say when updating the key of node $I^{(l)}_w$, the privileged user $U_z \in \mathcal{U}(I^{(l)}_w)$ cannot perform his role further because of network failure, then he can pass his role to another user $U \in \mathcal{U}(I^{(l)}_w)$ for a successful key update. This flexibility increases the reliability of re-keying.

**Theorem 4.1** Assuming $T$ is a full and balanced binary tree, updating $\mathcal{K}(T)$ requires

(i) a communication cost of at most $(n - 1)(\log_2 p + \log_2 q)$ bits\(^4\), (ii) transmissions of at most $\log_2 n$ rounds, and (iii) a user computation cost of at most $\log_2 n$ modular exponentiations.

**Proof:** (i) The highest communication cost is when $\mathcal{I}(T)$ consisting of all internal nodes of $T$. In this case, there are $|\mathcal{I}(T)| = n - 1$ new shadow keys requiring $(n - 1)(\log_2 p + \log_2 q)$ bits of multicast to update $\mathcal{K}(T)$.

(ii) When $\mathcal{I}(T)$ consists of all internal nodes, there are $h = \log_2 n$ levels in $\mathcal{I}(T)$ and updating each internal key requires message exchanges (protocol for case III). While internal keys at the same level can be updated simultaneously, internal keys at different

\(^4\)We assume that a node label is $\log_2 q$ bits long.
levels need to be updated in a sequence (from higher level to lower level). So updating involves \( \log_2 n \) sequences of transmissions (rounds).

(iii) The highest computation cost is for users in \( \mathcal{U}(I_{w}^{(l)}) \) where \( I_{w}^{(l)} \in \mathcal{I}(T) \) is the parent of leaves. In this case, a user in \( \mathcal{U}(I_{w}^{(l)}) \) needs to update all his internal keys, requiring \( \log_2 n \) modular exponentiations (note that updating a single internal key requires a single modular exponentiation).

It is worth pointing out that when \( \mathcal{I}(T) = \{I_{w}^{(0)}, I_{w}^{(1)}, \ldots, I_{w}^{(l-1)}, I_{w}^{(l)}\} \), only a single privileged user is needed to multicast the new shadow keys \( \mathcal{Y}'(T) = \{Y_{w}^{(1)}, \ldots, Y_{w}^{(l-1)}, Y_{w}^{(l)}\} \) in only a single round. This is possible because all keys in \( \mathcal{K}(T) \) are updated using protocol for case II and so there is no re-keying message exchange. The privileged user may pre-compute all new internal keys to generate all new shadow keys, then he concatenates the new shadow keys into a single re-keying message and multicast it.

**Theorem 4.2** The key update protocol satisfies the following properties, assuming all re-keying messages (shadow keys) are accessible.

(i) All users calculate the same new root key.

(ii) A user \( U \) can only find the new keys for \( \mathcal{K}(U) \cap \mathcal{K}(T) \) and not any other new key.

(iii) A passive adversary who knows the keys in \( \mathcal{K}(T) \) cannot discover any new key in \( \mathcal{K}'(T) \).

(iv) A passive adversary who knows new keys in \( \mathcal{K}'(T) \) cannot discover any key in \( \mathcal{K}(T) \).

**Proof:** (sketch) (i) Tables 4.4 and 4.5 show that for all \( I_{w}^{(l)} \in \mathcal{I}(T) \), all users \( U \in \mathcal{U}(I_{w}^{(l)}) \) will calculate the same new internal key for the node \( K_{w}^{(l)} \). Observe that the root node \( I_{w}^{(0)} \) is always in \( \mathcal{I}(T) \) and all users in \( \mathcal{U} \) are indeed those in \( \mathcal{U}(I_{w}^{(0)}) \). This guarantees that all users in \( \mathcal{U} \) will calculate the same new root key \( K_{w}^{(0)} \).

(ii) The key update protocol explicitly shows that a user \( U \) is always able to determine new keys for \( \mathcal{K}(U) \cap \mathcal{K}(T) \). Equation (4.1) demonstrates that computation of the node key of an internal node requires the knowledge of at least a node key of one of its two child nodes. This implies that knowing a key in \( \mathcal{K}(U) \) only leads to computation of another node key which is also in \( \mathcal{K}(U) \). So, the user \( U \) only has enough information to compute those new keys for \( \mathcal{K}(T) \) that are also new keys for \( \mathcal{K}(U) \cap \mathcal{K}(T) \).
(iii) Equation (4.2) exhibits that computation of the node key of an internal node requires the knowledge of at least a node key of its descendants. Observe that for all new keys in $\mathcal{K}'(T)$, the node keys of their descendants are never in $\mathcal{K}(T)$. So the adversary cannot find $\mathcal{K}'(T)$ from $\mathcal{K}(T)$. Next, we consider the attack where the adversary tries to finds new keys in $\mathcal{K}'(T)$ from re-keying messages (shadow keys).

In the light of equation (4.2), an internal key is in the form

$$K_w^{(l)} = g^{K_1 K_2} \mod p,$$

where

$$K_i = \begin{cases} 
1. \text{ a random value (leaf key)} \\
2. \text{ an internal key}
\end{cases}$$

for $i = 1, 2$. All shadow keys are publicly known, including $g^{K_1} \mod p$ and $g^{K_2} \mod p$. Without loss of generality, suppose $K_w^{(l)}$ is a new key in $\mathcal{K}'(T)$ and without knowing both $K_1$ and $K_2$, the adversary wants to find $K_w^{(l)}$. He may derive $K_w^{(l)}$ from $g^{K_1} \mod p$ and $g^{K_2} \mod p$, however, the difficulty to do so is equivalent to Computational Diffie-Hellman (CDH) problem which is believed to be hard.

The adversary may also discover $K_w^{(l)}$ by finding $K_i$ from $g^{K_i} \mod p$, for $i \in \{1, 2\}$. If $K_i$ is an internal key, the difficulty of finding $K_w^{(l)}$ is the same as CDH problem shown above. If $K_i$ is a random value (leaf key), the difficulty is equal to Discrete Logarithm (DL) problem which is an intractable problem. The CDH and DL problems are described in Chapter 1.

(iv) Similar to (iii).

4.3 Group Operations

The first operation in a group is system setup. This is followed by user evictions and user joins for the rest of the system lifetime.

4.3.1 System Initialisation

The group controller (GC) does the following to initialise the system.

1. Chooses two large primes $p$ and $q$, such that $q \mid p - 1$, and a generator $g$ of a multiplicative subgroup of $\mathbb{Z}_p^*$ whose order is $q$. 

2. Constructs a full binary tree $T_0$ with $n_0 = |\mathcal{U}_0|$ leaves, gives a unique label $I^{(l)}_w$ to every node, and attaches every user $U \in \mathcal{U}_0$ to a leaf $I^{(lv)}_w$. The information in steps 1 and 2 is made public.

3. Randomly chooses every leaf key $K^{(lv)}_w$ from $\mathbb{Z}_q^*$ and computes every internal key $K^{(l)}_w$, using equation (4.1), from leaf to root. These keys are secret.

4. Sends the leaf key $K^{(lv)}_w$ over a secure unicast channel and the shadow keys $\mathcal{Y}(U)$ over a multicast channel to every user $U \in \mathcal{U}_0$. Each user $U$ keeps $K^{(lv)}_w$ as a secret key and stores $\mathcal{Y}(U)$, which will allow him to compute other secret keys $\mathcal{K}(U) \setminus \{K^{(lv)}_w\}$, referring to Lemma 4.1.

The system can be initialised without the GC’s presence. Instead, all users $U \in \mathcal{U}_0$ can collaboratively perform the task as the following shows.

1. They agree on $p, q, g$, and $T_0$ (structure, labels and their positions in the tree), and publish the information.

2. Each user $U$ randomly chooses a key $K^{(lv)}_w$ from $\mathbb{Z}_q^*$ for his corresponding leaf $I^{(lv)}_w$, and keeps the leaf key in $\mathcal{K}(U)$. The user also multicasts a shadow key $Y^{(lv)}_w$ of his corresponding leaf $I^{(lv)}_w$, where other users $U' \in \mathcal{U}(I^{(tv)}_w) \subset \mathcal{U}_0$ will store the shadow key in $\mathcal{Y}(U')$.

3. To generate internal keys, they invoke the key update protocol in Section 4.2.1 with (i) $T = T_0$ and (ii) $\mathcal{I}(T)$ consisting of all internal nodes. Observe that $\mathcal{K}(T) = \emptyset$ and the set does not contain LKH consistent keys.

The protocol outputs $\mathcal{K}'(T)$ and $\mathcal{Y}'(T)$. Note that $\mathcal{K}'(T)$ gives LKH consistent internal keys for $T_0$, and in the light of Theorem 4.2, part (ii), every user $U \in \mathcal{U}_0$ will obtain his secret keys $\mathcal{K}(U) \setminus \{K^{(tv)}_w\} = \{K^{(l)}_w : I^{(l)}_w \in \mathcal{I}(U) \setminus \{I^{(tv)}_w\}\} \subset \mathcal{K}'(T)$ and no other key. Furthermore, by Corollary 4.1, the user only needs to store the shadow keys $\mathcal{Y}(U)$.

**Theorem 4.3** The collaborative re-keying scheme allows group members to initialise the system. Assuming a full and balanced binary tree, a user has to store $\log_2 n \log_2 p + \log_2 q$ bits, but only $\log_2 q$ bits of the storage have to be kept secure.

Shadow keys are effectively public, so it is possible to publish the shadow keys in a public bulletin board. A user stores only $\log_2 q$ bits of secret information.
4.3.2 User Eviction

Suppose in session $S$, users in $R$ are evicted from $U$ and a group key $GK$ is established for authorised users $U \setminus R$. User eviction requires all users $U \in U \setminus R$ to update their keys $K(U) \cap K(U')$, for all users $U' \in R$ (and so update the root key shared by users in $U$), such that the evicted users do not know the updated keys.

The user eviction protocol involves two consecutive operations: updating the tree and updating the keys.

Updating the tree  Suppose $T$ is the logical key hierarchy for $U$. Users in $U \setminus R$ update $T$ with respect to the following procedures.

1. Users $U \setminus R$ prune leaves corresponding to all users in $R$, prune internal nodes having no child, and prune internal nodes having only one child by replacing the parent with the child. Observe that the “replacing” nodes and their descendants (if any) will be shifted to lower levels.

2. Users $U \setminus R$ adjust the levels $l$ of shifted nodes while retaining their unique numbers $w$.

3. For every pruned node $I^{(l)}(w)$; (i) users $U \in U(I^{(l)}(w)) \subset U \setminus R$ delete the node from $I(U)$ and its corresponding node key from $K(U)$; and (ii) users $U \in U(I^{(l)}(w)) \subset U \setminus R$ delete the node from $\widehat{I}(U)$ and its corresponding shadow key from $Y(U)$.

4. For every replacing node $I^{(l)}(w)$, users $U \in U(I^{(l)}(w)) \subset U \setminus R$ add the node to $I(U)$.

5. For every shifted node $I^{(l)}(w)$; (i) users $U \in U(I^{(l)}(w)) \subset U \setminus R$ adjust the corresponding level in $I(U)$ and its corresponding node key’s level in $K(U)$; and (ii) users $U \in U(I^{(l)}(w)) \subset U \setminus R$ adjust the corresponding level in $\widehat{I}(U)$ and its corresponding shadow key’s level in $Y(U)$.

This guarantees that every node in the updated tree $T'$ has exactly one sibling (a full binary tree) and only users (leaves) in $U \setminus R$ remain in $T'$. Let the set $\mathcal{K}(R) = \bigcup(K(U) \cap K(U')) : U \in U \setminus R, U' \in R$ consist of internal keys that belong to evicted users (including the root key $K^{(0)}(w)$) with respect to $T'$. Observe that keys in $\mathcal{K}(R)$ are not LKH consistent keys. Let $\mathcal{I}(R) = \{I^{(l)}(w) : K^{(l)}(w) \in \mathcal{K}(R)\}$.

Updating the keys  The following steps are taken to update $\mathcal{K}(R)$.
4.3. Group Operations

1. For every replacing node \( I_w^{(l)} \), a user \( U \in \mathcal{U}(I_w^{(l)}) \subseteq \mathcal{U}_s \setminus \mathcal{R}_s \) multicasts the shadow key \( Y_w^{(l)} \), which will be stored by other users \( U' \in \mathcal{U}(I_w^{(l)}) \subseteq \mathcal{U}_s \setminus \mathcal{R}_s \) in \( \mathcal{Y}(U') \).

2. Users \( \mathcal{U}_s \setminus \mathcal{R}_s \) invoke the key update protocol in Section 4.2.1 with (i) \( T = T'_s \) and (ii) \( \mathcal{T}(T) = \mathcal{T}(\mathcal{R}_s) \).

The protocol outputs \( \mathcal{K}'(T) = \mathcal{K}'(\mathcal{R}_s) \), consisting of updated keys for \( \mathcal{K}(\mathcal{R}_s) \) including the updated root key \( K_w^{(0)} \in \mathcal{K}'(\mathcal{R}_s) \). Theorem 4.2, parts (i) and (ii), guarantees that all users \( U \in \mathcal{U}_s \setminus \mathcal{R}_s \) obtain new keys for \( \mathcal{K}(U) \cap \mathcal{K}(\mathcal{R}_s) \) and share the new root key \( K_w^{(0)} \) as the group key \( GK_s \), and no other new keys.

**Theorem 4.4** The user eviction protocol allows a set of evictors \( \mathcal{P}_s \subseteq \mathcal{U}_s \setminus \mathcal{R}_s \) to collaboratively remove an arbitrary number of users \( \mathcal{R}_s \) from a group \( \mathcal{U}_s \). The evictor set size and the operation cost follow Corollary 4.2 and Theorem 4.1, respectively.

**Theorem 4.5** The collaborative scheme provides evict secrecy (forward secrecy). That is, for any session \( S_s \) where \( S_1 \leq S_s \leq S_M \), an arbitrary size collusion \( \mathcal{C} = \bigcup \mathcal{R}_a, S_1 \leq S_a \leq S_s, E_a = \text{Evict} \), cannot obtain the group key \( GK_b, S_s \leq S_b \leq S_M \).

**Proof:** (sketch) For simplicity, we assume \( \mathcal{C} = \mathcal{R}_s \) for a single session \( S_s \). We need to show that the collusion \( \mathcal{C} \) cannot find \( \mathcal{K}'(\mathcal{R}_s) \). It is always the case that after the updating the tree operation, the colluders only know the keys in \( \mathcal{K}(\mathcal{R}_s) \) and no other node keys of the updated tree \( T'_s \). The rest of the proof is similar to that of Theorem 4.2, part (iii).

We can easily extend the security proof above to multiple sessions. Observe that even if \( \mathcal{R}_1 \cup \mathcal{R}_2 \cup \cdots \cup \mathcal{R}_{s-1} \) collude with \( \mathcal{R}_s \), they only know \( \mathcal{K}(\mathcal{R}_s) \) and so cannot find \( \mathcal{K}'(\mathcal{R}_s) \).

In the next session \( S_{s+1} \), the user group will be \( \mathcal{U}_{s+1} = \mathcal{U}_s \setminus \mathcal{R}_s \) with logical tree hierarchy \( T_{s+1} = T'_s \). Since the evicted users cannot find \( \mathcal{K}'(\mathcal{R}_s) \), their secret keys are outdated and unusable for operation in the next sessions.

**Example 4.1** Suppose \( \mathcal{R}_s = \{U_3, U_6, U_8\} \) are to be evicted from \( \mathcal{U}_s \) in Figure 4.1.

1. Users \( \mathcal{U}_s \setminus \mathcal{R}_s = \{U_1, U_2, U_4, U_5, U_7\} \) update the tree structure as shown in Figure 4.2 where nodes \( I_4^{(2)}, I_5^{(2)}, I_6^{(2)}, I_9^{(3)}, I_{12}^{(3)} \), and \( I_{14}^{(3)} \) have been pruned. Nodes in dashed line are the shifted nodes and they have been rearranged to new levels. The replacing nodes are \( I_{10}^{(3)}, I_{11}^{(3)}, I_{13}^{(3)} \). Keys of grey coloured nodes \( \mathcal{K}(\mathcal{R}_s) = \{K_0^{(0)}, K_1^{(1)}, K_2^{(1)}\} \) need to be updated.
4.3. Group Operations

2. User $U_4$ multicasts $Y_{10}^{(3)}$ to users $U_1$ and $U_2$. User $U_5$ multicasts $Y_{11}^{(3)}$ to user $U_7$. User $U_7$ multicasts $Y_{13}^{(3)}$ to users $U_5$. Moreover, users in $\mathcal{U}_s \setminus \mathcal{R}_s$ invoke the key update protocol in Section 4.2.1 with (i) $T = T'_s$ and (ii) $\mathcal{I}(T) = \mathcal{I}(\mathcal{R}_s) = \{j_0^{(0)}, j_1^{(1)}, j_2^{(1)}\}$. The output is $\mathcal{K}'(T) = \mathcal{K}'(\mathcal{R}_s) = \{K_r^{(0)}, K_r^{(1)}, K_r^{(1)}\}$ and the group key (the new root key $K_r^{(0)}$) is computed as

$$GK_s = (g^{(a \times b)} \mod p) \mod q,$$

where

$$a = g^{(g^{(\kappa_2^{(3)} \kappa_3^{(3)})} \times K_{10}^{(2)})} \mod p,$$

$$b = g^{(K_{12}^{(2)} \kappa_3^{(2)})} \mod p.$$

![Figure 4.2: Updated tree $T'_s$ for group $\mathcal{U}_s \setminus \mathcal{R}_s$](image)

4.3.3 User Join

Suppose in session $S_s$, new users in $\mathcal{J}_s \subseteq \mathcal{N} \setminus \mathcal{U}_s$ join $\mathcal{U}_s$. Each new user $U \in \mathcal{J}_s$ corresponds to a node set $\mathcal{I}(U)$, and will be given a node key set $\mathcal{K}(U)$ and a shadow key set $\mathcal{Y}(U)$. Furthermore, a group key $GK_s$ will be established for authorised users $\mathcal{U}_s \cup \mathcal{J}_s$. This requires all users $U \in \mathcal{U}_s$ to update their keys of nodes $\mathcal{I}(U) \cap \mathcal{I}(U')$, for all users $U' \in \mathcal{J}_s$ (and so updating the root key shared by users in $\mathcal{U}_s$), such that the new users only know the updated keys.

The user join protocol consists of two consecutive steps: updating the tree and updating the keys.

**Updating the tree** Suppose $T_s$ is the logical key hierarchy for $\mathcal{U}_s$. Users in $\mathcal{U}_s$ update $T_s$ with respect to the following procedures.

1. Users $\mathcal{U}_s$ add $|\mathcal{J}_s|$ new leaves. A new internal node must be created to attach a new leaf. The new internal node replaces an existing node and the existing node
becomes another child node of the new internal node. Observe that the displaced node and its descendants (if any) will be shifted to higher levels.

2. Users $U_s$ adjust the levels $l$ of shifted nodes while retaining their unique numbers $w$.

3. For every new leaf $I_w^{(t,v)}$, users $U \in \mathcal{U}(\hat{I}_w^{(l)}) \subset U_s$ add it to $\hat{I}(U)$.

4. For every new internal node $I_w^{(l)}$: (i) users $U \in \mathcal{U}(\hat{I}_w^{(l)}) \subset U_s$ add the node to $\hat{I}(U)$; and (ii) users $U \in \mathcal{U}(\hat{I}_w^{(l)}) \subset U_s$ add the node to $\hat{I}(U)$.

5. For every replaced node $I_w^{(l)}$, users $U \in \mathcal{U}(\hat{I}_w^{(l)}) \subset U_s$ delete the node from $\hat{I}(U)$ and its corresponding shadow key from $\mathcal{Y}(U)$.

6. For every shifted node $I_w^{(l)}$: (i) users $U \in \mathcal{U}(\hat{I}_w^{(l)}) \subset U_s$ adjust the node level in $\hat{I}(U)$ and its corresponding node key’s level in $\mathcal{K}(U)$; and (ii) users $U \in \mathcal{U}(\hat{I}_w^{(l)}) \subset U_s$ adjust the node level in $\hat{I}(U)$ and its corresponding shadow key’s level in $\mathcal{Y}(U)$.

7. Users $U_s$ correspond each new user $U' \in \mathcal{J}_s$ to a new leaf $I_w^{(t,v)}$ and ancestors of the new leaf $\mathcal{A}(I_w^{(t,v)})$, i.e., $\hat{I}(U')$.

This ensures that the updated tree $T'_s$ is a full binary tree consisting of all users (leaves) in $U_s \cup \mathcal{J}_s$. Let the set $\mathcal{I}(\mathcal{J}_s) = \bigcup (\mathcal{I}(U) \cap \mathcal{I}(U')) : U \in U_s, U' \in \mathcal{J}_s$ consist of nodes in $T'_s$ that correspond to new users (including the root node $I_w^{(0)}$). Observe that keys in $\mathcal{K}(\mathcal{J}_s) = \{K_w^{(l)} : I_w^{(l)} \in \mathcal{I}(\mathcal{J}_s)\}$ are not LKH consistent keys.

**Updating the keys** The following steps update $\mathcal{K}(\mathcal{J}_s)$.

1. Each new user $U' \in \mathcal{J}_s$ randomly chooses his leaf key $K_w^{(t,v)}$ from $Z_q^*$, and multicasts the shadow key $Y_w^{(l)}$ that will be stored by other users $U \in \mathcal{U}(I_w^{(l)}) \subset U_s \cup \mathcal{J}_s$.

2. For each new user $U' \in \mathcal{J}_s$ with new leaf $I_w^{(t,v)}$, an existing user $U \in \mathcal{U}(\hat{I}_w^{(t,v)})$ multicasts shadow keys $Y_w^{(l)}$, for all $I_w^{(l)} \in \hat{I}(U') \setminus \mathcal{I}(\mathcal{J}_s)$, which will be stored in $\mathcal{Y}(U')$ by the new user.

3. Users $U_s \cup \mathcal{J}_s$ invoke the key update protocol in Section 4.2.1 with (i) $T = T'_s$ and (ii) $\mathcal{I}(T) = \mathcal{I}(\mathcal{J}_s)$.

The protocol outputs $\mathcal{K}'(T) = \mathcal{K}'(\mathcal{J}_s)$, consisting of updated keys for $\mathcal{K}(\mathcal{J}_s)$ including the updated root key $K_w^{(0)} \in \mathcal{K}'(\mathcal{J}_s)$. Theorem 4.2, parts (i) and (ii), guarantees that all users $U \in U_s \cup \mathcal{J}_s$ obtain new keys for $\mathcal{I}(U) \cap \mathcal{I}(\mathcal{J}_s)$ and share the new root key $K_w^{(0)}$ as the group key $GK_s$, and no other new keys.
Theorem 4.6 The user join protocol allows a set of sponsors $P_s \subseteq U_s \cup \mathcal{J}_s$ to collaboratively admit an arbitrary number of new users $\mathcal{J}_s$ to a group $U_s$. The sponsor set size and the operation cost follow Corollary 4.2 and Theorem 4.1, respectively.

Theorem 4.7 The collaborative scheme provides join secrecy (backward secrecy). That is, for any session $S_s$ where $S_1 \leq S_s \leq S_M$, an arbitrary size collusion $C = \cup \mathcal{J}_a, S_s \leq S_a \leq S_M, E_a = \text{Join}$, cannot obtain the group key $G_{K_b, S_1} \leq S_b < S_s$.

Proof: (sketch) For clarity, we assume $C = \mathcal{J}_a$ for a single session $S_s$. We need to prove that the collusion $C$ cannot obtain $\mathcal{I}(\mathcal{J}_a)$. It is always the case that after the updating the tree operation, $C$ only knows $\mathcal{I}(\mathcal{J}_a)$ and not any node keys of the updated tree $T'_a$. The rest of the proof is similar to that of Theorem 4.2, part (iv).

For multiple session security, observe that although $\mathcal{J}_a, \mathcal{J}_a+1, \ldots, \mathcal{J}_M$ collude, they only know $\mathcal{I}(\mathcal{J}_a)$ and so cannot find $\mathcal{K}(\mathcal{J}_a)$.

Theorem 4.8 The collaborative scheme provides evict-join secrecy (forward-backward secrecy). That is, for any sessions $S_{s_1}, S_{s_2}$ where $S_1 \leq S_{s_1} < S_{s_2} \leq S_M$, an arbitrary size collusion $C = (\cup \mathcal{R}_{a_1}) \cup (\cup \mathcal{J}_{a_2}), S_1 \leq S_{a_1} \leq S_{s_1}, E_{a_1} = \text{Evict}_{s_2} S_{s_2} \leq S_{a_2} \leq S_M, E_{a_2} = \text{Join}_{s_2}$, cannot obtain the group key $G_{K_b, S_{s_1}} \leq S_b < S_{s_2}$.

Proof: (sketch) It can be easily derived from the proof for Theorem 4.5 and Theorem 4.7.

In the next session $S_{s+1}$, the user group will be $U_{s+1} = U_s \cup \mathcal{J}_s$ with logical tree hierarchy $T_{s+1} = T'_s$. The new users obtain appropriate secret keys and so they can participate in subsequent sessions.

Example 4.2 Suppose $\mathcal{J}_s = \{U_9, U_{10}\}$ join $U_s$ in Figure 4.1.

1. Users in $U_s$ update the tree structure as shown in Figure 4.3 where two new leaves $I_{17}^{(3)}$ and $I_{18}^{(4)}$, and two new internal nodes $I_{15}^{(2)}$ and $I_{16}^{(3)}$ (nodes in bold line) have been added. Nodes in dashed line are the shifted nodes and they have been rearranged to new levels. Keys of grey coloured nodes $\mathcal{K}(\mathcal{J}_s) = \{K_0^{(0)}, K_1^{(1)}, K_2^{(1)}, K_4^{(2)}, K_{15}^{(2)}, K_{16}^{(3)}\}$ need to be updated.

2. New users $U_9$ and $U_{10}$ randomly generate leaf keys $K_{17}^{(3)}$ and $K_{18}^{(4)}$, respectively, and individually multicast $Y_{17}^{(3)}$ and $Y_{18}^{(4)}$. Users $U_7$ and $U_8$ will store $Y_{17}^{(3)}$ and user $U_4$ will store $Y_{18}^{(4)}$. 


3. Either user $U_7$ or $U_8$ multicasts shadow keys $Y_6^{(3)}, Y_5^{(2)}$ to new user $U_9$, and user $U_4$ multicasts shadow keys $Y_{10}^{(4)}, Y_9^{(3)}, Y_3^{(2)}$ to new user $U_{10}$.

4. Users in $U_s \cup J_s$ invoke the key update protocol in Section 4.2.1 with (i) $T = T'_s$, (ii) $\mathcal{I}(T) = \mathcal{I}(J_s) = \{I_0^{(0)}, I_1^{(1)}, I_2^{(2)}, I_4^{(2)}, I_{15}^{(3)}, I_{16}^{(3)}\}$. The output is $\mathcal{K}'(T) = \mathcal{K}'(J_s) = \{K_0^{(0)}, K_1^{(1)}, K_2^{(2)}, K_4^{(2)}, K_{15}^{(3)}, K_{16}^{(3)}\}$ and the group key (the new root key $K_0^{(0)}$) is computed as

$$GK_s = (g^{(a \times b)} \mod p) \mod q,$$

where

$$a = g \left( g^{(K_{10}^{(4)} K_{18}^{(4)} \times K_9^{(3)}) \times g^{(K_{7}^{(3)} K_{8}^{(3)})}} \right) \mod p,$$

$$b = g \left( g^{(K_{13}^{(4)} K_{14}^{(4)} \times K_{17}^{(3)}) \times g^{(K_{17}^{(3)} K_{12}^{(3)})}} \right) \mod p.$$

![Figure 4.3: Updated tree $T'_s$ for group $U_s \cup J_s$](image)

### 4.4 Conclusion

We have considered the re-keying problem in collaborative environments. Our construction combines the two-party key agreement protocols with a logical key hierarchy to allow a session key to be collaboratively computed by multiple parties. We have proven that the proposed re-keying scheme is secure and efficient. We have also demonstrated how to use the re-keying scheme to initialise the system without the existence of a group controller, and to provide secure user eviction and user join operations.
Chapter 5

A Dynamic and Stateless Revocation Scheme

5.1 Introduction

Group management in traditional group key distribution systems is centralised. There is a fixed group controller (group manager), GC, and an initial set of users, $U_0$. The GC initialises the system by generating and securely delivering individual keys to the users. After initialisation, the GC establishes authorised groups by broadcasting a message allowing users of the target subgroup to compute a common group key. Most literature on group key distribution schemes focuses on efficient methods of forming authorised groups under such a system [3, 31, 37, 39, 55, 62, 64, 84]. One common approach is treating unauthorised users in a session as those who must be temporarily revoked from $U_0$. So a group key distribution scheme is also called revocation scheme.

A dynamic group key distribution system allows the group controller to be dynamic and so any group member can assume the role of the group manager in the transmission phase of an event protocol. We consider the scenario that after the initialisation phase, any group member can replace the GC as the group initiator, hence allowing a decentralised model. Allowing a group member to form a subgroup has numerous applications. For example in dynamic teleconferences, where users want to transmit data to a subgroup of users, a fixed group controller model results in a very inefficient solution. This is because either all communications from users must go through the group controller and then broadcasted to the designated group, or numerous group keys need to be established. These solutions have drawbacks such as single point of failure, and high communication overhead for the group controller and also communication delay. Dynamic revocation schemes alleviate these problems and result in more efficient solutions.

Another simple solution to the problem of dynamic group initiator is to employ the
scheme of a single group controller as a building block and associate a single group controller scheme to each user such that the user is the group controller and the rest of the group are the receivers. The obvious drawback of this solution is that the key storage of each user is prohibitively large, that is $n - 1$ times the storage of a user plus the storage of a group controller in a single group controller scheme. This is linear in $n$, and for large groups is unacceptable.

In some cases secret keys of users are generated and stored in computing devices at manufacturing time and the keys are unchangeable. Accordingly, it is preferable to assume users cannot change their secret keys after system initialisation, i.e., their secret keys are stateless.

### 5.1.1 A Summary of Our Contribution

We propose a stateless revocation scheme that can be used by any member of a group to form an arbitrary subgroup of his choice. This is achieved by a single broadcast message from the user to the group and allows the group key to be computed by the users of the target subgroup. The proposed scheme can be easily modified to a centralised setting where the GC is the only group initiator in the system. We assume that the group is fixed (no new users are added) and users require to form subgroups at different times.

The construction uses the logical key hierarchy in conjunction with the Shamir secret sharing scheme [77] and the Diffie-Hellman key agreement [26]. The main advantage of the proposed system over other dynamic and stateless revocation systems [2, 75] is scalability for large groups and higher levels of collusion resilience. Our scheme provides $n$-resilience while the schemes in [2, 75] provide $t$-resilience such that when $t$ is large, say $t \approx n$, they become very inefficient. The bandwidth and computation of the system in [2] for any $t', t' \leq t$, revoked users is of order $t$. Our scheme has efficient storage while keeping bandwidth and computation costs of order $\log_d n$ for a tree degree $d$. We also give a variant of the scheme that reduces the number of system keys and the amount of published information.

Table 5.1 compares the efficiency of our scheme and its variant, and the schemes in [2, 75]. The user storage and communication costs are measured in terms of the number of keys. Our scheme, its variant and the scheme in [2] are public-key based and so the computation cost of these schemes is measured in terms of the number of exponentiations. The scheme in [75] is symmetric-key based and so the computation cost of this scheme is measured in terms of the number of decryptions. Our scheme can
be used to revoke up to \( n \) users \((\beta \leq n)\), while its variant and the scheme in [2] limit the number of revoked users to \( t \) \((\beta \leq t)\), and the scheme in [75] requires the number of revoked users to be small \((\beta \approx 0)\).

Table 5.1: Efficiency comparison of the proposed schemes and several relevant schemes

<table>
<thead>
<tr>
<th></th>
<th>Proposed Schemes</th>
<th>AMM [2]</th>
<th>SW [75]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Storage</strong></td>
<td>( \log d n + 1 )</td>
<td>( t(\log d n - 1) + 1 )</td>
<td>1</td>
</tr>
<tr>
<td><strong>Public Keys</strong></td>
<td>( d(t(\log d n - 1) + 1) )</td>
<td>( t )</td>
<td>( O(\log n) )</td>
</tr>
<tr>
<td><strong>Collusion Resilience</strong></td>
<td>( n )</td>
<td>( t )</td>
<td>( n + t )</td>
</tr>
<tr>
<td><strong>Single User</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>( (d - 1)\log d n )</td>
<td>( (d - 1)(t(\log d n - 1) + 1) )</td>
<td>( t + 1 )</td>
</tr>
<tr>
<td>Computation</td>
<td>( (d - 1)\log d n )</td>
<td>( (d - 1)(t(\log d n - 1) + 1) )</td>
<td>( t + 1 )</td>
</tr>
<tr>
<td><strong>Multiple (( \beta )) Users</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Communication</td>
<td>( \leq (1 - \frac{\beta}{d})n )</td>
<td>( \leq (d - 1)(t(\log d n - 1) + 1) )</td>
<td>( t + 1 )</td>
</tr>
<tr>
<td>Computation</td>
<td>( \leq (1 - \frac{\beta}{d})n )</td>
<td>( \leq (d - 1)(t(\log d n - 1) + 1) )</td>
<td>( t + 1 )</td>
</tr>
</tbody>
</table>

We prove security of our scheme and give a method of providing authentication to the system. We point out that methods such as the Subset Difference (SD) [62] and the Layered Subset Difference (LSD) [37] can also be employed to provide efficient dynamic and stateless revocation schemes. Finally we discuss permanent revocation within our scheme.

5.1.2 Protocol

Table 5.2: Subgroup protocol

<table>
<thead>
<tr>
<th>If ( E_s = \text{Subgroup} )</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>( U_s, G_s \subseteq U_s, P_s = {U_z}, U_z \in G_s )</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>( - U_z \xrightarrow{M} \text{ all } U_i \in G_s, U_i \neq U_z )</td>
</tr>
<tr>
<td></td>
<td>( - \text{ all } U_i \in G_s \text{ perform computation on } M \text{ and } K(U_i) )</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>( - \text{ all } U_i \in G_s \text{ share a group key } GK_s )</td>
</tr>
<tr>
<td></td>
<td>( - U_{s+1} = U_s )</td>
</tr>
</tbody>
</table>

There is only a group initiator who broadcasts a subgroup message \( M \) that allows each user \( U_i \in G_s \) to calculate the group key \( GK_s \) using his secret keys in \( K(U_i) \) while preventing users outside \( G_s \) from doing so.
5.2. The Dynamic Scheme

Organisation of this chapter. In Section 5.2 we describe a dynamic revocation scheme with stateless storage, and show how to transform the scheme into a centralised setting. We assess security and efficiency of the system, and show how to provide authentication for this system. In Section 5.3 we propose a new method of system initialisation with improved efficiency, but do not change the revocation operation. In Section 5.4 we present additional discussions and summarise this chapter.

The main parts of this chapter appeared in the Proceedings of The Fourth International Conference on Information Security and Cryptology – ICISC 2001 [51].

5.2 The Dynamic Scheme

The notions of logical key hierarchy, Diffie-Hellman key exchange protocol, and threshold secret sharing are central to our construction. The definitions of logical key hierarchy (LKH) and Diffie-Hellman key exchange protocol can be found in Chapters 3 and 4, respectively. Shamir threshold secret sharing [77] is described in Chapter 1. Below we add another property of Shamir threshold secret sharing.

A new share $K_j$ (a value of $F(x)$ at $x = j$) can be generated by any $t+1$ users as $K_j = \sum_{i \in \Phi} K_i \times \psi(\Phi, i, j) \mod q$, where $\psi(\Phi, i, j) = \prod_{c \in \Phi, c \neq i} \frac{1 - \epsilon}{1 - c} \mod q$.

Note that $j = 0$ gives the secret $K_0$.

5.2.1 System Initialisation

The group controller (GC) is in charge of the system setup and does the following.

1. Generates two large primes $p, q$, where $q \mid p - 1$, and a generator $g$ of a multiplicative subgroup of $\mathbb{Z}_p^*$ with order $q$. The GC then publishes $p, q$ and $g$.

2. Builds an LKH tree of degree $d$ with $n_0$ leaves.1 Every node in the tree is either a leaf or a parent with $d$ child nodes. Let $\mathcal{I}$ denote the set of all nodes in the tree. Each node is labelled by a unique number $w$ such that $w \neq 0$.

3. Logically associates each user $U \in \mathcal{U}_0$ with a leaf of the tree. Knowledge of the tree structure together with node labels and users’ association with leaves are public. Figure 5.1 is an example of a tree structure. The nodes are labelled by

1Although the tree may not be full and balanced, we assume a full and balanced tree in our efficiency analysis.
5.2. The Dynamic Scheme

$w$, for $1 \leq w \leq |\mathcal{I}|$, starting from root node to leaf nodes, and from left to right directions.

Figure 5.1: A tree structure of degree 2 for 8 users

4. Generates a set of secret keys, $\mathcal{K}(\mathcal{I}) = \{K_w : w \in \mathcal{I}, K_w \in \mathbb{Z}_q^\ast\}$, and a set $\mathcal{Y}(\mathcal{I}) = \{Y_w : w \in \mathcal{I}, Y_w = g^{K_w} \mod p\}$ of public keys. For all $w \in \mathcal{I}$, node $w$ is associated with a pair of secret key $K_w$ and public key $Y_w$.

5. Publishes all the public keys and securely sends to each user $U \in \mathcal{U}_0$ the set of secret keys, $\mathcal{K}(U) \subset \mathcal{K}(\mathcal{I})$, from $U$’s leaf to the root. User $U$ keeps these keys as his secret information. For example, the logical key hierarchy of Figure 5.1 is shown in Figure 5.2 so the set of secret keys for $U_1$, for example, is $\mathcal{K}(U_1) = \{K_1, K_2, K_4, K_8\}$.

Figure 5.2: A logical key hierarchy for Figure 5.1

The system setup above requires the GC to generate $\frac{d_n - 1}{d - 1}$ secret keys and to publish $\frac{d_n - 1}{d - 1}$ public keys. A user has to store secret keys from a leaf to the root (height of tree), which is $h + 1$, where $h = \log_d n$ keys.

**Theorem 5.1** In the above scheme, the storage sizes for a group controller and a user are $\frac{d_n - 1}{d - 1}\log_2 q$ bits and $(\log_d n + 1)\log_2 q$ bits, respectively. There are $\frac{d_n - 1}{d - 1}\log_2 p$ bits of public keys.
5.2. User Revocation

Suppose in session $S_s$, a group member $U_z$ in $U_s$ wants to form a subgroup $G_s \subseteq U_s$. This can be achieved by revoking the users $U_s \setminus G_s$ and forming a group key for the subgroup $G_s$. The privileged set for this operation is $P_s = \{U_z\}$ where $U_z \in G_s$. Let $I(U)$ be the set of nodes from user $U$’s leaf to the root.

1. **Transmission:** A group initiator $U_z \in P_s$ does the following.
   (a) Randomly chooses an element $r \in \mathbb{Z}_q^*$ and calculates $\tilde{g} = g^r \mod p$.
   (b) Uses the algorithm in Section 5.2.3 to find a set $T$ of nodes that satisfy the following conditions:
      (i) $\forall U \in G_s, T \cap I(U) \neq \emptyset$;
      (ii) $\forall U \in U_s \setminus G_s, T \cap I(U) = \emptyset$.
      (The algorithm guarantees that $\forall U \in G_s, |T \cap I(U)| = 1$, which means each user in $G_s$ has exactly one node in $T$.) Let $|T| = m$.
   (c) Chooses a set $\Theta$ of $m - 1$ distinct elements from $\mathbb{Z}_q^*$ such that $T \cap \Theta = \emptyset$.
      Then for all $j \in \Theta$, calculates
      $$Y_j^* = \prod_{w \in T} (Y_w)^{\psi(T,w,j)} \mod p$$
      and $\tilde{Y}_j = (Y_j)^r \mod p$. Note that $Y_w, \forall w \in T$ are in $\mathcal{Y}(I)$ while $Y_j, \forall j \in \Theta$ are new public keys.
   (d) Broadcasts subgroup message
      $$\mathcal{M} = \{\tilde{g}, \tilde{Y}_j \mid j : j \in \Theta\}.$$  

2. **Calculation:** Each user $U \in G_s$ uses a secret key $K_v = K(U)$, where $v \in T \cap I(U)$, and the received message $\mathcal{M}$ to calculate the group key as follows.
   $$GK_s = (\tilde{g}^{K_v})^{\psi(T \cup \{v\}, v, 0)} \times \prod_{j \in \Theta} (\tilde{Y}_j)^{\psi(T \cup \{v\}, j, 0)} \mod p.$$  

**Theorem 5.2** The user revocation protocol allows any group member to form a subgroup $G_s$ from a group $U_s$.

**Proof:** We need to show that all users in $G_s$ are able to compute the group key $GK_s$ based on their secret keys and the broadcast data.
Without loss of generality, assume \( T = \{1, 2, \cdots, m\} \). Let the secret keys and public keys associated with \( T \) be \( \{K_w : w \in T\} \) and \( \{Y_w = g^{K_w} \mod p : w \in T\} \), respectively. Notice that there implicitly exists a unique polynomial \( F(x) \) of degree at most \( m - 1 \) such that \( g^{F(w)} = g^{K_w} \), for all \( w \in T \). Using the public keys of \( T \), that is, \( \{g^{K_w} \mod p : w \in T\} \), one can calculate \( \prod_{w \in T} (Y_w)^{\psi(T,w,j)} \mod P \).

So, each user in \( G_s \) computes the group key as

\[
G_{K_s} = (g^{K_v})^{\psi(\Theta \cup \{v\}, v, 0)} \times \prod_{j \in \Theta} (Y_j)^{\psi(\Theta \cup \{v\}, j, 0)} = (g^{r \times F(v)})^{\psi(\Theta \cup \{v\}, v, 0)} \times \prod_{j \in \Theta} (g^{r \times F(j)})^{\psi(\Theta \cup \{v\}, j, 0)} = \prod_{j \in (\Theta \cup \{v\})} (g^{r \times F(j)})^{\psi(\Theta \cup \{v\}, j, 0)} = g^{r \times \sum_{j \in (\Theta \cup \{v\})} F(j) \times \psi(\Theta \cup \{v\}, j, 0)} = g^{r F(0)} \mod p .
\]

**Theorem 5.3** Assuming that the Computational Diffie-Hellman (CDH) problem is hard, an arbitrary collusion \( C \subseteq U_s \setminus G_s \) cannot find the group key \( G_{K_s} \) for any session \( S_s \) where \( S_1 \leq S_s \leq S_M \) and \( E_s = \text{Subgroup} \). Thus the user revocation protocol provides subgroup secrecy.

**Proof:** First we note that \( C \) cannot gain any information regarding \( F(x) \) from their secret keys \( K(C) = \bigcup_{U \in C} K(U) \) because \( T \cap I(U) = \emptyset, \forall U \in C \). So they cannot obtain the value \( F(0) \) to compute the group key \( g^{r F(0)} \).

Further we show that \( C \) is not able to find the group key \( G_{K_s} \) from broadcast messages. The CDH (Computational Diffie-Hellman) problem, which is believed to be hard [26], is the basis of our revocation protocol. The CDH problem is described in Chapter 1.

Our proof uses a “reduction argument”. That is, if there exists an oracle (probabilistic polynomial-time) \( G \) that can compute \( G_{K_s} \) using all the information known to \( U_s \setminus G_s \), then the same oracle can be used to solve the CDH problem.
It is sufficient to show that if there exists a probabilistic polynomial-time algorithm \( G \) that on inputs \( g^e, g^{F(j)}, \forall j \in \Theta \) and \( g^{F(w)}, \forall w \in T \), outputs \( g^{F(0)} \) with a non-negligible probability, and \( F(x) \) is a polynomial of degree at most \( m - 1 \), then \( G \) can be used to solve the CDH problem. That is, given \( g^{X_1} \) and \( g^{X_2} \), where \( X_1 \) and \( X_2 \) are two randomly chosen elements of \( \mathbb{Z}_q^* \), \( G \) can be used to find \( g^{X_1 X_2} \). Let \( \Theta = \{ j_1, \ldots, j_{m-1} \} \). We randomly choose \( m - 1 \) elements \( L_1, \ldots, L_{m-1} \in \mathbb{Z}_q^* \) and construct a unique polynomial \( H(x) \) of degree at most \( m - 1 \) such that \( H(j_e) = L_e, \forall e, 1 \leq e \leq m - 1 \), and \( g^{H(0)} = g^{X_2} \). This can be used to calculate \( g^{H(j_e)} = g^{L_e}, 1 \leq e \leq m - 1 \), and also \( g^{H(\alpha)} \) for all \( \alpha \in \mathbb{Z}_q \), and hence \( g^{H(w)}, \forall w \in T \). Furthermore, since we know \( L_e \) we can compute

\[
(g^{X_1})^{L_e} = (g^{X_1})^{H(j_e)}, \quad j_e \in \Theta .
\]

Now if \( G \) is given the inputs \( g^{X_1}, (g^{X_1})^{L_e}, e = 1, \ldots, m - 1; \) and \( g^{H(w)}, w \in T, \) it will output \( g^{X_1 H(0)} = g^{X_1 X_2} \). This means \( G \) can solve the CDH problem which contradicts the hardness assumption of the CDH problem. \( \square \)

The above scheme can be used for multiple sessions with a different \( r \) for each session. We can extend the above proof to multiple sessions as given below.

**Proof:** For simplicity, we assume that the scheme is run twice (two sessions) for the same \( T \) and we show that an adversary who can collude with the users in \( U_s \setminus G_s \), after seeing all the broadcasts (and even the group key for the first session), is not able to compute the group key of the second session. We may further assume that \( \Theta \) is the same for two sessions and so the polynomial \( F(x) \) will be the same. The only different values in the two runs are the random values \( r_1 \) and \( r_2 \), respectively. We will again employ the “reduction arguments” for the proof. Assume that \( G \) is a probabilistic polynomial-time algorithm that on inputs \( g^{r_1}, g^{r_2}, g^{r_1 F(j)}, g^{r_2 F(j)}, \forall j \in \Theta, g^{F(w)}, \forall w \in T \) and \( g^{r_1 F(0)} \), outputs \( g^{r_2 F(0)} \) with a non-negligible probability. We show that we can use \( G \) to solve the CDH problem. Let \( g^{X_1}, g^{X_2} \) be two elements with \( X_1, X_2 \) randomly chosen from \( \mathbb{Z}_q^* \). As before, let \( \Theta = \{ j_1, \ldots, j_{m-1} \} \) and choose \( m - 1 \) random elements \( L_1, \ldots, L_{m-1} \in \mathbb{Z}_q^* \). There exists a unique polynomial \( H(x) \) of degree at most \( m - 1 \) such that \( H(j_e) = L_e, \forall e, 1 \leq e \leq m - 1 \), and \( g^{H(0)} = g^{X_2} \). We also randomly choose \( r_1 \) and compute \( g^{r_1 H(0)} \). We can feed \( G \) with the following data: (1) \( g^{r_1}, g^{r_1 H(j_e)}, \forall j_e \in \Theta, g^{H(w)}, \forall w \in T \) and \( g^{r_1 H(0)} \) (i.e., all the information obtained by the adversary from the first session); and (2) \( g^{X_1}, (g^{X_1})^{L_e}, e = 1, \ldots, m - 1 \) (the information from the second session). By the assumption of \( G \), it outputs \( g^{X_1 H(0)} = g^{X_1 X_2} \), which shows that \( G \) can solve the CDH problem and we obtain a contradiction, and therefore the desired result
5.2. The Dynamic Scheme

5.2.3 An Algorithm for Finding $T$

We want to find a set of nodes $T$ where all users in $G_s$, but not any users in $U_s \setminus G_s$, have at least a node in the set. Let the root of the tree be at level zero and leaves be at level $h$ (similar to Figure 5.1). The algorithm for finding $T$ is as follows.

Recall that $I$ denotes the set of all nodes in the tree and let $I(l) = \{w \in I : l \leq h\}$ for $l = 0$ to $h$. That is, $I(l)$ is the set of nodes at level $l$. Also, recall that $I(U)$ denotes the set of nodes of the revoked users and let $I(U)_{(l)}$ denote the set of nodes at level $l$ of $I(U)$. That is, $I(U)_{(l)} = \bigcup_{U \in \mathcal{G}_s} I(U)_{(l)}$. The algorithm is shown in Table 5.3.

Table 5.3: An algorithm to find $T$

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td>$T = \emptyset$, $U_{left} = G_s$</td>
</tr>
<tr>
<td>(2)</td>
<td>for $l = 0$ to $h$ {</td>
</tr>
<tr>
<td>(3)</td>
<td>$I_{temp1} = I^{(l)} \setminus I(U)_{(l)}$</td>
</tr>
<tr>
<td>(4)</td>
<td>$U_{temp} = {U : U \in U_{left}, w_U^{(l)} \in I_{temp1}}$</td>
</tr>
<tr>
<td>(5)</td>
<td>$I_{temp2} = {w_U^{(l)} : w_U^{(l)} \in I(U), U \in U_{temp}}$</td>
</tr>
<tr>
<td>(6)</td>
<td>$T = T \cup I_{temp2}$</td>
</tr>
<tr>
<td>(7)</td>
<td>$U_{left} = U_{left} \setminus U_{temp}$</td>
</tr>
<tr>
<td>(8)</td>
<td>if $U_{left} = \emptyset$</td>
</tr>
<tr>
<td>(9)</td>
<td>stop</td>
</tr>
<tr>
<td>(10)</td>
<td>}</td>
</tr>
</tbody>
</table>

Intuitively, the algorithm works as follows. It starts from the root and visits all nodes in each level, before moving to the next level up. A node is put in $T$ if the following conditions are satisfied.

(i) the node does not belong to a revoked user, and

(ii) no other node on its path to the root is in $T$.

Let $U_{left}$ be the set of users in $G_s$ who do not have any node in $T$.

1. Initialise $T = \emptyset$ and $U_{left} = G_s$.

---

2For an unbalanced tree, level $h$ is located at the lowest leaves of the tree where $h + 1$ is height of the unbalanced tree.
2. Repeat 2.1 – 2.5 for each level from \( l = 0 \) to \( l = h \).

   2.1 put all nodes in level \( l \), except those belonging to the revoked users, in \( I_{\text{temp1}} \).

   2.2 look for users in \( U_{\text{left}} \) who have at least one node in \( I_{\text{temp1}} \). These users are kept in \( U_{\text{temp}} \). It is possible that nodes in \( I_{\text{temp1}} \) do not belong to any user in \( U_{\text{temp}} \).

   2.3 select nodes in \( I_{\text{temp1}} \) that belong to at least one user in \( U_{\text{temp}} \). These nodes are stored in \( I_{\text{temp2}} \).

   2.4 add \( I_{\text{temp2}} \) to \( T \) and subtract \( U_{\text{temp}} \) from \( U_{\text{left}} \).

   2.5 check if \( U_{\text{left}} \) is empty, that is if all users in \( G_s \) have at least a node in \( T \).

      If this is the case, then the algorithm stops; otherwise it goes to the next level.

Theorem 5.4 The output set \( T \) of the above algorithm satisfies the following properties.

(i) \( \forall U \in G_s, T \cap I(U) \neq \emptyset \), and

(ii) \( \forall U \in U_s \setminus G_s, T \cap I(U) = \emptyset \).

The algorithm guarantees that \( \forall U \in G_s, |T \cap I(U)| = 1 \) and \( T \) is a minimal set.

Proof (sketch): Step (3) excludes all nodes belonging to the revoked users and fulfills property (ii). Note that a node at a lower level belongs to more users and a node at the highest level (a leaf node, \( l = h \)) belongs to a single user. In step (2), the algorithm runs from the lowest level (\( l = 0 \)) to the highest level (\( l = h \)), and constructs \( T \) from the most common nodes to the least common nodes. Together with steps (4), (5) and (7), the algorithm ensures that once the node at level \( l \) of the user \( U, w_U^{(l)} \), is in \( T \), nodes on the higher levels belonging to the same user will not be in \( T \). This guarantees that \( \forall U \in G_s, |T \cap I(U)| = 1 \) and results in \( T \) being minimal. The algorithm may terminate at level \( l, l < h \), if property (i) is satisfied. Otherwise, it will proceed to level \( l = h \) to guarantee this property.

Theorem 5.5 The above scheme allows revocation of \( |U_s \setminus G_s| \) users, \( 1 \leq |U_s \setminus G_s| \leq |U_s| - 1 \), from a group \( U_s \). Revocation of one user requires transmission of \( ((d - 1) \log_d n_s - 1)(\log_2 p + \log_2 q) + \log_2 p \) bits and revocation of multiple users requires transmission of at most \( ((1 - \frac{1}{d})n_s - 1)(\log_2 p + \log_2 q) + \log_2 p \) bits.

Proof: Recall that \( |U_s| = n_s \). Revoking one user always gives \( |\Theta| = (d - 1) \log_d n_s \) nodes and so \( |\Theta| = (d - 1) \log_d n_s - 1 \) elements. The required bandwidth is \( ((d -
The Dynamic Scheme

5.2. The Dynamic Scheme

1) \(\log_d n_s - 1) \log_2 p + \log_2 q + \log_2 p\) bits. For multiple user revocation, the worst case is when \(|\mathcal{U}_s \setminus \mathcal{G}_s| = \frac{n_s}{d}\) users and the leaves associated with the revoked users have different parents. In this case, \(|\mathcal{T}| = n_s - \frac{n_s}{d} = (1 - \frac{1}{d})n_s\) nodes and so \(|\Theta| = (1 - \frac{1}{d})n_s - 1\) elements. The required bandwidth is \(((1 - \frac{1}{d})n_s - 1)(\log_2 p + \log_2 q) + \log_2 p\) bits.

Corollary 5.1 Establishing a group key \(GK_s\) requires a user in \(\mathcal{G}_s\) to compute \((d - 1) \log_d n_s\) and at most \((1 - \frac{1}{d})n_s\) modular exponentiations for revocations of \(|\mathcal{U}_s \setminus \mathcal{G}_s| = 1\) and \(|\mathcal{U}_s \setminus \mathcal{G}_s| > 1\) users, respectively.

We highlight the existence of natural subgroups in LKH due to the fact that a parent node is an ancestor of a subset of leaves. Precisely, a parent node \(w\) at level \(l\) is the ancestor of a subgroup \(\{U : I(U) \ni w\}\) of \(d^{h-l}\) users (leaves) assuming a full and balanced \(d\)-ary tree with the highest level \(h\). The users can use the node key \(K_w\) as the group key without extra transmission and computation. The number of such subgroups is equal to the number of parent nodes in the tree, which is \(\frac{n_s - 1}{d-1}\).

Example 5.1 Consider Figure 5.1 and suppose user \(U_1\) wants to form a subgroup \(\mathcal{G}_s = \{U_1, U_2, U_5, U_6, U_7, U_8\}\) by revoking \(\mathcal{U}_s \setminus \mathcal{G}_s = \{U_3, U_4\}\). The group initiator \(U_1\) does the following.

1. Generates \(r\) and calculates \(\bar{g} = g^r \mod p\).

2. Sets \(I(\mathcal{U}_s \setminus \mathcal{G}_s) = \{1, 2, 5, 10, 11\}\) and executes the algorithm in Section 5.2.3. The result is \(\mathcal{T} = \{3, 4\}\) as shown in Figure 5.3 where dashed nodes are nodes belonging to \(\mathcal{U}_s \setminus \mathcal{G}_s\) and bold nodes are nodes of the minimal set \(\mathcal{T}\).

3. Suppose \(U_1\) chooses \(\Theta = \{16\}\). Then \(U_1\) uses \(Y_3\) and \(Y_4\) to calculate \(Y_{16}, \bar{Y}_{16} = (Y_{16})^r \mod p\) and broadcasts subgroup message \(\mathcal{M} = \{\bar{g}, \bar{Y}_{16} \| 16\}\).

4. Users \(U_1\) and \(U_2\) use \(K_4\), and users \(U_5, \ldots, U_8\) use \(K_3\) to calculate \(GK_s\).

5. Users \(U_3\) and \(U_4\) are not able to calculate \(GK_s\) since they do not have \(K_3\) or \(K_4\).

5.2.4 Authentication

In group applications such as private teleconferencing, authenticity of data and the identity of the sender is very important. Being inspired by [2] and [66], we describe below a technique that can be used to prevent modification or forging of broadcast data, and to identify the sender.

Suppose a sender \(U_i\) wants to broadcast data \(msg\), then he needs to do the following.
5.2. The Dynamic Scheme

1. Use the secret key, $K_w \in K(U_i)$, corresponding to his leaf node as his secret identity $id$, that is $id = K_w$.

2. Generate a random number $r$ over $\mathbb{Z}_q^*$ and compute $\bar{g} = g^r \mod p$.

3. Calculate $\text{hash} = f_{\text{owh}}(msg \parallel i \parallel \bar{g}) \mod q$, where $f_{\text{owh}}$ is a publicly known one way hash function, and calculate a signature $\text{sign} = (-\text{hash} \times r) + id \mod q$.

4. Broadcast the signed data $\mathcal{M} = \{msg \parallel i \parallel \bar{g} \parallel \text{sign}\}$.

Receivers verify as follows.

1. Compute $\text{hash}' = f_{\text{owh}}(msg \parallel i \parallel \bar{g}) \mod q$ from $\mathcal{M}$ and assign $id' = Y_w$, where $Y_w$ is the public key of $K_w$ (the leaf key of the sender $U_i$).

2. Check $id' \overset{?}{=} g^{\text{sign}} \times (\bar{g})^{\text{hash}'} \mod p$. If they are equal, then receivers accept the integrity of the data and authenticity of the sender. Otherwise, either data or the sender, or both, are tampered with.

The above authentication system can be used by a group initiator to authenticate broadcast data $\bar{g}$ and $\bar{Y}_j : j \in \Theta$ sent during the revocation stage, as described below.

The group initiator $U_z$ computes $msg = \{\bar{g} \parallel \bar{Y}_j \parallel j : j \in \Theta\}$ (we combine all data into one message), calculates $\text{hash} = f_{\text{owh}}(msg \parallel z)$ and $\text{sign} = (-\text{hash} \times r) + id \mod \mathbb{Z}_q$ ($id$ is the secret key corresponding to $U_z$’s leaf node), and broadcasts signed subgroup message $\mathcal{M} = \{msg \parallel z \parallel \text{sign}\}$. To verify, users in subgroup $G_s$ will calculate $id'$ and $\text{hash}'$, and check $id' \overset{?}{=} g^{\text{sign}} \times (\bar{g})^{\text{hash}'} \mod p$.

The authentication technique is based on DL (Discrete Logarithm) problem. Refer to [66] for further security proof.
5.3 A Variant of Key Generation and Allocation

5.2.5 A Centralised Protocol

There is a group controller (GC) who initially sets up the group and at a later stage revokes the memberships as required by the subgroup to be formed. The dynamic scheme in Section 5.2 can be easily converted to the centralised model. The two systems have similar performances except, in this model, public keys are not required. The user revocation works as follows.

1. Transmission: The GC does the following.
   
   (a) Same as step 1a of the protocol in Section 5.2.2.
   
   (b) Finds $T$ and uses Lagrange interpolation to generate a polynomial $F(x)$ of degree $m - 1$ such that $F(w) = K_w$, for all $w \in T$,
   
   $$F(x) = \sum_{w \in T} K_w \times \Psi(T, w) \mod q .$$
   
   (c) Chooses $\Theta$ and calculates $K_j = F(j), \tilde{Y}_j = (\tilde{y})^{K_j} \mod p$, for all $j \in \Theta$.
   
   (d) Similar to step 1d of the protocol in Section 5.2.2.

2. Calculation: Similar to step 2 of the protocol in Section 5.2.2.

The correctness and security of the centralised protocol is the same as the dynamic one.

Alternatively, the GC can choose a random value for the group key $GK_s$, encrypt $GK_s$ with the secret key $K_w, \forall w \in T$, and broadcast the result. Each user $U \in G_s$ decrypts the encrypted broadcast using his secret key $K_v$, where $v \in T \cap I(U)$, to find $GK_s$. An arbitrary collusion of revoked users $U_s \setminus G_s$ cannot discover $GK_s$ as they do not have any secret key $K_w$ used to encrypt the group key (remember the properties of the set $T$).

5.3 A Variant of Key Generation and Allocation

In the dynamic scheme described in the previous section, the total number of keys in the system is $|I|$ (the number of nodes in the tree). In this section we propose a variant of the scheme that reduces the number of system keys while maintaining security. It has the advantage of reducing the storage required by the GC and the amount of published information.
The reduction in the number of system keys is at the cost of reducing the collusion resilience of the system. That is, the modified system provides collusion resilience for up to \( t \) colluders where \( t \) is a pre-determined threshold parameter. This may be a disadvantage for some applications where we cannot bound the collusion size beforehand, while in other situations it may be a reasonable assumption.

The basic idea is as follows. For a \( d \)-ary tree of height \( h + 1 \), we choose \( dh \) keys for the system and allocate a key to each node in such a way that, \( d \) distinct keys are assigned to the nodes in the \( l \)th level, \( 1 \leq l \leq h \), such that the \( d \) children of the same parent have distinct keys. Each user is associated with a leaf but not all leaves are assigned to users. The keys of a user are his leaf key together with all the node keys along the path to the root. The leaves corresponding to the users are chosen in such a way that for any set of \( t \) users, \( \{U_{i_1}, \ldots, U_{i_t}\} \), and a user \( U_{\nu} \notin \{U_{i_1}, \ldots, U_{i_t}\} \), there exists at least one key which belongs to user \( U_{\nu} \), but does not belong to users in \( \{U_{i_1}, \ldots, U_{i_t}\} \). In other words, \( d \) and \( h \) must be chosen such that \( n < d^h \) (note that there are \( d^h \) leaves of the tree and \( n \) is the number of users). Figure 5.4 illustrates the leaf assignment for 9 users in a 3-ary tree of height 4. The system requires only 9 keys.

![Figure 5.4: A tree structure with \( d = 3 \) and \( h = 3 \) for 9 users](image)

In the following we give a construction for this approach using polynomials over finite fields.

### 5.3.1 System Initialisation

The group controller (GC) does the following. Note that \( n_0 = |U_0| \).

1. Generates \( p, q \) and \( g \), similar to the system initialisation in Section 5.2.1, and publishes them.

2. Selects \( t \), the required level of collusion resilience, chooses a prime \( d \) and computes \( u = \lceil \log_d n_0 \rceil \). Next, the GC chooses the tree depth \( h \) such that \( h > t(u - 1) \) and \( h \leq d \).
3. Forms a set of polynomials $F_d[x]_u = \{F(x) \in F_d[x] : \text{deg}(F(x)) \leq u - 1\}$ and associates a polynomial $F_i(x) \in F_d[x]_u$ to a user $U_i$, for $1 \leq i \leq n_0$. Note that $|F_d[x]_u| = d^n \geq n_0$.

4. Chooses a set of $h$ distinct elements of $\mathbb{Z}_d$, $L = \{\alpha^{(1)}, \cdots, \alpha^{(h)}\}$, each associated with one level of the tree. To each user $U_i, 1 \leq i \leq n_0$, the GC assigns an identity vector $V_i = (F_i(\alpha^{(1)}), \cdots, F_i(\alpha^{(h)})) = (v_i^{(1)}, \cdots, v_i^{(h)})$ over $\mathbb{Z}_d$.

5. Generates a set of secret keys $K(I) = \{K_{\alpha}^{(l)} : K_{\alpha}^{(l)} \in \mathbb{Z}_q^*, 0 \leq a \leq d - 1, 1 \leq l \leq h\}$ and a set of public keys $Y(I) = \{Y_{\alpha}^{(l)} = g^{K_{\alpha}^{(l)}} \mod p : 0 \leq a \leq d - 1, 1 \leq l \leq h\}$.

6. All public keys are published. For $1 \leq i \leq n_0$, the GC secretly sends a set of secret keys $K(U_i) = \{K_{v_i}^{(l)} : 1 \leq l \leq h\} \subset K(I)$ to user $U_i$.

Observe that the underlying structure above is a full and balanced tree of degree $d$ with $d^h$ leaves. Since $n_0 < d^h$, only some of the leaves are associated with the users. There are only $dh$ secret keys. The system initialisation requires the GC to generate

$$dh = d(t(u - 1) + 1)$$

$$\approx d(t(\log_d n - 1) + 1)$$

secret keys, and to publish $dh \approx d(t(\log_d n - 1) + 1)$ public keys. A user has to store $h \approx t(\log_d n - 1) + 1$ secret keys.

### 5.3.2 User Revocation

The revocation protocol is the same as that described in Section 5.2.2. We need to show that with the above key allocation, $T$ as defined in Section 5.2.2 is not an empty set. This is true because each user has a subset of $h$ keys that corresponds to a polynomial of degree at most $u - 1$. It follows that the number of common keys of any two users is at most $u - 1$. This is because if two users, $U_i$ and $U_{i'}$, have a common key at level $l$, it means that $F_i(\alpha^{(l)}) = F_{i'}(\alpha^{(l)})$. The condition $h > t(u - 1)$ yields that a set $T$ satisfying the required conditions can be found. Since the cardinality of $T$ determines the transmission overhead, we would like the size of $T$ to be as small as possible. To find $T$, we can use the same algorithm as in Section 5.2.3. However, the resulting $T$ is not necessarily minimal.

**Theorem 5.6** The variant scheme has the following properties.
5.3. A Variant of Key Generation and Allocation

(i) Assuming that the Computational Diffie-Hellman (CDH) problem is hard, a collusion \( C \subseteq U_s \setminus G_s, |C| \leq t \) cannot find the group key \( GK_s \) for any session \( S_s \) where \( S_1 \leq S_s \leq S_M \) and \( E_s = \text{Subgroup} \).

(ii) The transmission overhead for \( |U_s \setminus G_s|, 1 \leq |U_s \setminus G_s| \leq t \), user revocation from a group \( U_s \) is at most \( ((d-1)(t(\log_d n - 1) + 1) - 1)(\log_2 p + \log_2 q) + \log_2 p \) bits. A user in \( G_s \) needs to compute at most \( (d-1)(t(\log_d n - 1) + 1) \) modular exponentiations.

(iii) The storage of a group controller is \( d(t(\log_d n - 1) + 1) \) bits, the storage of a user is \( t(\log_d n - 1) + 1 \) bits, and there are \( d(t(\log_d n - 1) + 1) \) bits of public keys.

Proof: (i) Similar to the proof for Theorem 5.3, but in this case \( C \) is bounded by \( t \).

(ii) The maximum value of \(|T|\) corresponds to one user revocation, in which case

\[
|T| = dh - h = (d-1)h
\]

nodes and so \(|\Theta| \approx (d-1)(t(\log_d n - 1) + 1) - 1\) elements. This worst case requires \(((d-1)(t(\log_d n - 1) + 1) - 1)(\log_2 p + \log_2 q) + \log_2 p \) bits of bandwidth and user computation of \( (d-1)(t(\log_d n - 1) + 1) \) modular exponentiations.

Revocation of multiple users (at most \( t \) users) has smaller \(|T|\) and so the required bandwidth and user computation are less than the worst case.

(iii) It is straightforward from the system initialisation.

Example 5.2 Let \( n_0 = 9, t = 2 \) and \( d = 3 \). It follows that \( u = 2, h > 2 \), i.e., \( h = 3 \) and \( \mathcal{F}_3[x]_2 = \{ F(x) \in \mathcal{F}_3[x] : \deg(F(x)) \leq 1 \} \). Polynomials for users are the following.

\[
\begin{align*}
F_1(x) &= 0 \mod 3 \quad F_2(x) = x \mod 3 \quad F_3(x) = 2x \mod 3 \\
F_4(x) &= 1 \mod 3 \quad F_5(x) = 1 + x \mod 3 \quad F_6(x) = 1 + 2x \mod 3 \\
F_7(x) &= 2 \mod 3 \quad F_8(x) = 2 + x \mod 3 \quad F_9(x) = 2 + 2x \mod 3
\end{align*}
\]

Let \( L = \{2, 0, 1\} \), then vectors for the users are the following.

\[
\begin{align*}
V_1 &= (0, 0, 0) \quad V_2 = (2, 0, 1) \quad V_3 = (1, 0, 2) \\
V_4 &= (1, 1, 1) \quad V_5 = (0, 1, 2) \quad V_6 = (2, 1, 0) \\
V_7 &= (2, 2, 2) \quad V_8 = (1, 2, 0) \quad V_9 = (0, 2, 1)
\end{align*}
\]
Moreover, sets of secret keys for the users are the following.

\[ \mathcal{K}(U_1) = \{ K_0^{(1)}, K_0^{(2)}, K_0^{(3)} \} \]
\[ \mathcal{K}(U_2) = \{ K_2^{(1)}, K_0^{(2)}, K_1^{(3)} \} \]
\[ \mathcal{K}(U_3) = \{ K_1^{(1)}, K_0^{(2)}, K_2^{(3)} \} \]
\[ \mathcal{K}(U_4) = \{ K_1^{(1)}, K_1^{(2)}, K_1^{(3)} \} \]
\[ \mathcal{K}(U_5) = \{ K_0^{(1)}, K_1^{(2)}, K_2^{(3)} \} \]
\[ \mathcal{K}(U_6) = \{ K_2^{(1)}, K_1^{(2)}, K_0^{(3)} \} \]
\[ \mathcal{K}(U_7) = \{ K_2^{(1)}, K_2^{(2)}, K_2^{(3)} \} \]
\[ \mathcal{K}(U_8) = \{ K_1^{(1)}, K_2^{(2)}, K_0^{(3)} \} \]
\[ \mathcal{K}(U_9) = \{ K_0^{(1)}, K_2^{(2)}, K_1^{(3)} \} \]

Figure 5.4 is a tree structure for the example above. Note that each secret key \( K_a^{(l)} \)
(or public key \( Y_a^{(l)} \)) is labelled by a unique pair \( a \) and \( l \). In the example, the unique
pair \( a \) and \( l \) is mapped to a unique number \( w = a + (l - 1)d + 1 \) in the tree.

**Authentication**

We may use the technique in Section 5.2.4 with a slight change to provide the variant
scheme with sender authentication and message authentication. The authentication
capabilities require the sender to hold a secret that is unknown by others. The technique
in Section 5.2.4 assumes that the sender’s leaf key \( K_w \) is unique to him, so it can be
used for his secret identity \( id \). This assumption is not guaranteed in the variant scheme
since a leaf key is known by more than one user. Instead, the sender \( U_i \) in the variant
scheme can use \( \sum_{w \in \mathcal{I}(U_i)} K_w \mod q \), which is unique to him, for his secret identity \( id \).
In this case, the receivers compute \( id' = \prod_{w \in \mathcal{I}(U_i)} Y_w \mod p \) from public keys during verification.

### 5.4 Further Discussion and Conclusion

In this section we point out some interesting aspects of the proposed dynamic and
stateless revocation scheme, and summarise this chapter.

**Other Variants.** The Subset Difference (SD) method in [62] and the Layered Subset
Difference (LSD) method in [37] are based on binary tree structures. It has been shown
in their work that using SD or LSD method to generate and allocate system’s secret
keys gives efficient user storage. Also, for a given set of authorised users, the method
provides an algorithm to find a small set of secret keys that are known only to the
authorised users (this means efficient communication bandwidth, see Chapter 1).

We may apply one of the methods and its algorithm to the proposed dynamic
construction. That is, the GC uses the method for secret key generation and allocation.
The GC also generates public keys corresponding to the secret keys in the initialisation
phase. To form an authorised subgroup of his choice, a group member uses the set
cover algorithm of the method to discover the *indices of secret keys* that are known only to subgroup users (note that the set cover algorithm, tree structure and indices are public). By referring to the discovered indices, the group initiator uses the public versions of the secret keys to generate a subgroup message and further broadcasts the message whereby users in the subgroup compute the common key. The following describes the operation in more detail.

Let $\mathcal{I}$ be the set of indices of secret keys in the system. Let $\mathcal{K}(\mathcal{I}) = \{K_w : K_w \in \mathbb{Z}_q^*, w \in \mathcal{I}\}$ and $\mathcal{K}(U) \subset \mathcal{K}(\mathcal{I})$ be the secret key set in the system and the secret key set of each user $U \in \mathcal{U}_s$, respectively. These keys are generated and allocated using either SD or LSD method by the GC. Moreover, the GC generates public key set $\mathcal{Y}(\mathcal{I}) = \{Y_w = g^{K_w} \mod p, w \in \mathcal{I}\}$.

Suppose a group member $U_z$ would like to form a subgroup $G_s$ by revoking users in $\mathcal{U}_s \setminus G_s$. The group initiator $U_z$ uses the set cover algorithm provided by the method to discover a set of indices $T$ satisfying: (i) $\forall U \in G_s, T \cap \{w : K_w \in \mathcal{K}(U)\} \neq \emptyset$ and (ii) $\forall U \in \mathcal{U}_s \setminus G_s, T \cap \{w : K_w \in \mathcal{K}(U)\} = \emptyset$. The algorithm guarantees the two properties of $T$ since it always outputs the indices of secret keys that are known only to subgroup users. After obtaining $T$, $U_z$ follows steps 1a, 1c and 1d of the protocol in Section 5.2.2 to generate and broadcast the subgroup message. Users $U \in G_s$ use a secret key $K_v \in \mathcal{K}(U)$, where $v \in T$, to compute the common key as in step 2 of the protocol in Section 5.2.2.

**Permanent Revocation.** The proposed user revocation is on a temporary basis where a user is effectively revoked for a session only. It might be necessary to permanently revoke a user, for example when his keys are compromised. In this case it is necessary to update the system keys. Permanent revocation (eviction) needs the assistance of the GC. We show below a method, inspired by [2], to evict users. Suppose users in $\mathcal{R}_s = G_s \setminus \mathcal{U}_s$ must be evicted. That is, the system keys must be updated such that the key information known to the evicted group has to be changed. In the following, we show how to update the keys.

1. The GC uses the revocation protocol in Section 5.2.2 to temporarily revoke the users in $\mathcal{R}_s$ and obtain a group key $GK_s$.

2. The GC and all users in $\mathcal{U}_s$ update their secret keys as $K'_w = K_w \times GK_s \mod q$.

3. The GC replaces public value $g$ with $g' = g^{\frac{1}{GK_s}} \mod p$.

Using this method, public keys in the system remain unchanged as $Y_w = (g')^{K'_w} \mod p$. 
It is possible to generate the system keys using a pseudo-random function $f_K$ [35], in which case the GC only needs to hold a single secret key $K$ which is the index to a pseudo-random function family. The system’s secret keys are obtained as $K_w = f_K(w) \mod q$. In this case the GC computes $K'_w = f_K(w) \times GK_s \mod q$. For every permanent revocation, the GC only needs to update the group key without the need to change $f_K(w)$.

**Chapter Summary.** We have considered the problem of establishing a common key among subgroups of the original group, with dynamic and fixed group initiator. This is important in many multi-party applications including teleconferencing and Pay-TV systems. We have approached the problem as a revocation problem and have proposed a construction with dynamic group initiator and with proven security. The proposed scheme is efficient in practice and resilient against arbitrary collusions. The dynamic setting of the scheme is made possible by the existence of public keys. Finding efficient solutions without the need for public keys is an interesting research direction. We have also shown that the scheme can be easily modified to a scheme with fixed group initiator, without compromising security. A method of adding authentication to the scheme has also been suggested.

In addition, we have proposed an extension to the basic scheme to reduce the number of system keys (and public keys accordingly), and showed that further extensions using SD and LSD methods are possible to achieve superior performance. We also have demonstrated how users can be permanently removed from the group.
Chapter 6

Group Key Distribution Schemes with Decentralised User Join – An Algebraic Approach

6.1 Introduction

In general a group key distribution scheme requires two cryptographic operations: user join and user revocation (either temporary, permanent, or both). Most published papers have been concerned with only user revocation; many researchers have overlooked the join problem, simply assuming the group is static such as [31, 55, 84] and Chapter 5, or: when a new user joins the group, a new common key is generated and sent to the new user using a secure channel. To provide backward security, the new common key is sent to the old group users encrypted with the old common key which makes it decryptable by all the users [17, 18, 75]. Such a simplistic solution has two obvious drawbacks: (i) the new joined user has only a single session membership because he only has the group key of the session and is not eligible for future group operations; (ii) it requires the group controller (GC) to be on-line for the operation, and when the GC is unavailable no join operation can be performed. Furthermore, it is usually assumed that admission requires a new user to go through an initialisation process similar to other users of $U_0$ to obtain unique key information from the GC [2, 62, 64].

Allowing group users to perform user join and revocation provides system availability and flexibility which are important in many applications, such as ad-hoc networks. The schemes in [2, 75] and Chapter 5 allow a group user to take the role of the GC for the revocation operation, but not for the join operation. It is desirable to construct group key distribution schemes that have this decentralised setting for both user revocation and join. That is, after system initialisation each group member can form a subgroup by sending a single broadcast message. Moreover, collaborations of several group members may admit new members to the system. This is achieved by securely
sending data to a new user such that the new user can compute key information. Systems with the dynamic setting for user revocation and the collaborative setting for user join have higher reliability since they function without a group controller.

Allowing join without the assistance of the GC is essential for flexible and reliable operation of groups in applications such as a rescue operation, where new volunteers or recruits must join the group and, because of a broken communication link, it is not possible to contact the GC. In these situations the group must be able to function independently: that is, group members must be able to admit new users to the group. Another example is where new recruits may urgently require to join a group when the director (GC) is absent. Having a system where several managers can cooperate to grant membership is then helpful. However, this mechanism must be designed so that it cannot be abused by corrupt group members. One solution requires several specified trusted managers to cooperate to grant membership.

Allowing group members to admit new members is also an important requirement for content distribution systems. For example, suppose a content distributor wishes to create a network of branches to distribute content, some through a central branch and some only through local branches (for example local news). In systems with a centralised group controller, the controller generates all key sets and gives subsets to each branch who will give them to their local subscribers. This solution requires the key sets to be designed for all users in advance. A more efficient and flexible solution is to give some autonomy to local branches and allow them to generate some key sets for their local subscribers.

### 6.1.1 A Summary of Our Contribution

Our focus in this chapter is decentralised user join. We introduce a novel scenario for dynamic group key distribution schemes which allows any group member to (i) form a subgroup of existing users \( U \), and (ii) sponsor new users to groups initiated by him. A sponsored member can participate in groups that are initiated by the sponsor but remains outside \( U \). A new user will join the universe of users \( U \) if he receives enough sponsorships. There are specified subsets of group members, called access sets, able to admit new users to the group \( U \). Cooperation of users in an access set is required to exact admission of a new user; arbitrary subsets of users cannot do so. This provides a mechanism for self sufficiency, flexibility in admitting new users and security in the sense of ensuring that the new members have the approval of specified subsets of users. Therefore after the initialisation phase, the group members can fully control the group
and there is no need for the GC. A group key is only computable by members of an authorised group. A collusion of users outside the authorised group will have no information about the key, even if all messages broadcast through the lifetime of the system are available.

We propose a construction that caters for the above scenario using symmetric polynomials over finite fields, and cumulative arrays. We use threshold cryptography [77] to replace the GC and distribute the trust to group members. We describe two dynamic group key distribution schemes with safe and flexible join operations. The first scheme provides sponsorship and full join with a threshold admission structure, where any subset of users of a specified size $(t+1)$ is an access set. We give a second scheme with an arbitrary admission structure, where the collection of access sets also includes some arbitrary access sets of up to some size $(t)$. We call such access sets $t$-arbitrary access sets. We also give a variant of the second scheme with better efficiency. Both schemes allow subgroup establishment of size at least $|U| - t$ users and are secure against a collusion of at most $t$ corrupt users, assuming any access set is not a subset of the collusion. The efficiency of the schemes does not depend on the number of users $n$ but is, rather, only a function of $t$. It is reasonable to assume such a collection of subgroups, since in practice the number of users requesting secure communication can be bounded accurately a priori. We firstly assume that key storage is stateless in the subgroup operation but later remove this assumption and allow key storage to be stateful in the join operation to maintain security. We present user eviction (permanent removal) methods and demonstrate traceability of colluders in the proposed schemes. We also show our proposed schemes satisfy the flexibility and security requirements, and we evaluate their efficiency.

6.1.2 Protocols

The subgroup protocol of this scheme is identical to that in Table 5.2 of Chapter 5. The join and evict protocols are as follows.

Table 6.1 shows the protocol for a join event. For a sponsorship operation, a single sponsor $U_z \in P_s$ transmits a join message $\mathcal{M}$ to each new user $U_i \in J_s$ over a secure unicast channel (we assume such a channel exists between the sponsor and each new user). A sponsorship operation requires a new user $U_i$ to keep only the message as his secret information $K(U_i)$. Full join operation is a multiple sponsorship requiring sponsors $U_z$ in an access set $P_s$ to individually transmit join messages $\mathcal{M}$ to each new user $U_i \in J_s$. A new user $U_i$ uses these messages to determine his secret information
Table 6.1: Join protocol

<table>
<thead>
<tr>
<th>If $E_s = \text{Join}$*</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> $U_s, \mathcal{J}_s \subseteq N \setminus U_s$, $\mathcal{P}_s \subseteq U_s$</td>
</tr>
<tr>
<td><strong>Process:</strong></td>
</tr>
<tr>
<td>- all $U_z \in \mathcal{P}_s$ broadcast $\mathcal{M}$</td>
</tr>
<tr>
<td>- all $U_i \in \mathcal{J}_s$ perform computation on $\mathcal{M}$</td>
</tr>
<tr>
<td>- follow subgroup protocol with $G_s = U_s \cup \mathcal{J}_s$</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
</tr>
<tr>
<td>- all $U_i \in \mathcal{J}_s$ obtain secret information $K(U_i)$</td>
</tr>
<tr>
<td>- all $U_i \in U_s \cup \mathcal{J}_s$ share a group key $GK_s$</td>
</tr>
<tr>
<td>- $U_{s+1} = U_s \cup \mathcal{J}_s$</td>
</tr>
</tbody>
</table>

*For sponsorship, $\mathcal{P}_s = \{U_z\}, U_z \in U_s$

$K(U_i)$.

Table 6.2: Evict protocol

<table>
<thead>
<tr>
<th>If $E_s = \text{Evict}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> $U_s, \mathcal{R}_s \subseteq U_s$, $\mathcal{P}_s = {U_z}, U_z \in U_s \setminus \mathcal{R}_s$</td>
</tr>
<tr>
<td><strong>Process:</strong></td>
</tr>
<tr>
<td>- $U_z \overset{\mathcal{M}}{\rightarrow} U_i \in U_s \setminus \mathcal{R}_s$</td>
</tr>
<tr>
<td>- all $U_i \in U_s \setminus \mathcal{R}_s$ perform computation on $\mathcal{M}$ and $K(U_i)$</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
</tr>
<tr>
<td>- all $U_i \in U_s \setminus \mathcal{R}_s$ share a group key $GK_s$</td>
</tr>
<tr>
<td>- all $U_i \in \mathcal{R}_s$ have unusable secret information $K(U_i)$</td>
</tr>
<tr>
<td>- $U_{s+1} = U_s \setminus \mathcal{R}_s$</td>
</tr>
</tbody>
</table>

Table 6.2 exhibits the evict protocol. To evict users in $\mathcal{R}_s$, a single evictor $U_z \in \mathcal{P}_s$ broadcasts a single eviction message $\mathcal{M}$ and each authorised user $U_i \in U_s \setminus \mathcal{R}_s$ performs computation on the message and his secret information $K(U_i)$, whereby the authorised users share a common key $GK_s$ and the secret information of evicted users is disabled.

Organisation of this chapter. Section 6.2 gives a basic construction of a dynamic group key distribution scheme that provides sponsorship and full join with threshold access sets using symmetric polynomials. User eviction methods and traceability are also discussed in that section. In Section 6.3, we extend the basic dynamic group key distribution scheme using cumulative arrays to allow $t$-arbitrary access sets in the system. We furthermore show how to add new $t$-arbitrary access sets during sessions. Section 6.4 gives a variant of the extended scheme with better efficiency. Additional comments and conclusions are given in Section 6.5.
6.2. A Decentralised GKDS

This construction is inspired by the Diffie-Hellman key exchange protocol and threshold secret sharing. The descriptions of both cryptographic tools can be found in Chapter 4 and Chapter 1, respectively. In this scheme (i) any user can establish a group key for a subgroup, (ii) any user can sponsor a new user to a group initiated by him, and (iii) any \( t + 1 \) or more users is an access set being able to grant full membership to a new user.

6.2.1 System Initialisation

The group controller (GC) initialises the system as follows.

1. Generates two large primes \( p, q \) such that \( q \mid (p - 1) \), and chooses a generator \( g \) of a multiplicative subgroup of \( \mathbb{Z}_p^* \) with order \( q \). \( p, q \) and \( g \) are made public.

2. Chooses a value for the system parameter \( t \) and constructs a random symmetric polynomial,

\[
F(x, y) = \sum_{l=0}^{t} \sum_{k=0}^{t} a_{l,k} x^l y^k \mod q,
\]

where the \( a_{l,k} \in \mathbb{Z}_q(0 \leq l \leq t, 0 \leq k \leq t) \) are randomly chosen, and \( a_{l,k} = a_{k,l} \) for all \( l, k \). The polynomial \( F(x, y) \) is kept secret.

3. Calculates a polynomial \( F_i(x) = F(x, i) \) for each user \( U_i \in U_0 \), and gives the polynomial to the user over a secure unicast channel. Note that \( F_i(x) = \sum_{l=0}^{t} A_{i,l} x^l \mod q \), where \( A_{i,l} = \sum_{k=0}^{l} a_{i,l} x^k \mod q \). User \( U_i \) keeps the polynomial \( F_i(x) \) as his secret information, \( \mathcal{K}(U_i) = \{ F_i(x) \} \).

The secret information of the GC is the symmetric polynomial \( F(x, y) \) of degree \( t \) in \( x \) and \( y \), and that of each user is the polynomial \( F_i(x) \) of degree \( t \) in \( x \).

**Theorem 6.1 ([9])** The scheme requires a group controller to store \( \frac{(t+1)(t+2)}{2} \log_2 q \) bits and a user to store \( (t + 1) \log_2 q \) bits of secret information.
6.2. A Decentralised GKDS

Note that a user $U_i$ has a secret polynomial $F_i(x)$ and any other user $U_j$ knows a point $F_i(i')$ of $F_i(x)$ because $F_i(i') = F_j(i)$ (the symmetric property of $F(x,y)$) and $U_j$ has the polynomial $F_j(x)$. The polynomial $F_i(x)$ and users’ points give a secret sharing system with threshold $t + 1$, secret $F_i(0)$ and shares $F_i(i')$. In general each user corresponds to a unique secret sharing system and it gives an instance of Anzai et al. [2] system (see Chapter 1).

All $a_{i,k}$ in the polynomial $F(x,y)$ are randomly chosen. To reduce the GC’s storage, we may use a pseudo-random number generator. Let $H_e()$ be a secure pseudo-random generator where $e$ is the key. The GC produces $a_{i,k} = H_e(\min (l,k), \max (l,k)) \mod q$ and so needs to only store the key $e$ which is $\log_2 q$ bits long.

6.2.2 Subgroup Event

We give a subgroup protocol where a user in $\mathcal{U}_s$ can use to form a subgroup $\mathcal{G}_s$ and establish a group key $GK_s$ for the subgroup. The privileged set for this operation consists of a group initiator $U_z \in \mathcal{G}_s$, so that $\mathcal{P}_s = \{U_z\}$. The protocol can be used to form a subgroup with $|\mathcal{G}_s| \geq |\mathcal{U}_s| - t$. Let $\mathcal{U}_s = \{i : U_i \in \mathcal{U}_s\}$ and $\mathcal{G}_s = \{i : U_i \in \mathcal{G}_s\}$ be the sets of identities.

1. Transmission: A group initiator $U_z \in \mathcal{G}_s$ does the following.
   
   (a) Randomly generates an element $r \in \mathbb{Z}_q^*$ and calculates $\bar{g} = g^r \mod q$.

   (b) Chooses a set $\Theta \subseteq \mathbb{Z}_q^*$ such that $\Theta \cap \mathcal{U}_s = \emptyset$, $a \neq b$, for all $a, b \in \Theta$, and $|\Theta| = t + |\mathcal{G}_s| - |\mathcal{U}_s|$. Let $\Delta = \Theta \cup (\mathcal{U}_s \setminus \mathcal{G}_s)$.

   (c) Uses his secret polynomial $F_z(x) \in \mathcal{K}(U_z)$ to calculate $\bar{g}_j = \bar{g}^{F_z(j)} \mod p$, for all $j \in \Delta$, and broadcasts the subgroup message
   
   $\mathcal{M} = \{\bar{g}, z, \bar{g}_j \mid j : j \in \Delta\}$.

2. Calculation: Each user $U_i \in \mathcal{G}_s$ uses his secret polynomial $F_i(x) \in \mathcal{K}(U_i)$ and the subgroup message $\mathcal{M}$ to compute the group key as

   $$GK_s = (\bar{g}^{F_z(z)})^{\psi(\Delta \cup \{i\}, i)} \times \prod_{j \in \Delta} (\bar{g}_j)^{\psi(\Delta \cup \{i\}, j)} \mod p.$$

A summary of the subgroup protocol is shown in Table 6.3.

**Theorem 6.2** The subgroup protocol allows any group member to form a subgroup $\mathcal{G}_s$ with at least $|\mathcal{U}_s| - t$ users from a group $\mathcal{U}_s$. It requires a broadcast message of length...
Table 6.3: The subgroup protocol

<table>
<thead>
<tr>
<th>A group initiator $U_z \in \mathcal{G}_s$</th>
<th>Users $U_i \in \mathcal{G}_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle p, q, g, t, \tilde{U}_s, \tilde{G}_s, F_z(x) \rangle$</td>
<td>$\langle p, q, F_i(x) \rangle$</td>
</tr>
</tbody>
</table>

$r \in \mathbb{Z}_q^*$, $\Theta \subseteq \mathbb{Z}_q^*$
$\Theta \cap \tilde{U}_s = \emptyset$
$|\Theta| = t + |\tilde{G}_s| - |\tilde{U}_s|$
$\Delta = \Theta \cup (\tilde{U}_s \setminus \tilde{G}_s)$
$\bar{g} = g^r \mod p$
$\bar{g}_j = \bar{g}^{F_z(j)} \mod p, \forall j \in \Delta$

$\mathcal{M} = \{\bar{g}, \bar{g}_j | j \in \Delta\}$

$\bar{v} = (\bar{g}^{F_i(z)})^{\psi(\Delta \cup \{i\}, i)} \mod p$
$w = \prod_{j \in \Delta} (\bar{g}_j)^{\psi(\Delta \cup \{i\}, j)} \mod p$
$GK_s = v \times w \mod p$

$(t + 1)(\log_2 p + \log_2 q)$ bits, and requires a subgroup user to compute $t + 1$ modular exponentiations.

**Proof:** Recall that in the system, the group initiator $U_z$ has a secret polynomial $F_z(x)$ and any other user $U_i$ knows a point $F_z(i)$ of $F_z(x)$, because $F_z(i) = F_i(z)$ (the symmetric property of $F(x, y)$) and $U_i$ has the polynomial $F_i(x)$. We may consider $F_z(x)$ as a $(t + 1)$-secret sharing system with secret $F_z(0)$ and each user $U_i$ holding a share $F_z(i)$.

To form a subgroup $\mathcal{G}_s$, $|\mathcal{G}_s| \geq |\tilde{U}_s| - t$, the group initiator $U_z$ broadcasts the message $\mathcal{M}$ consisting of

(i) the $|\tilde{U}_s| - |\mathcal{G}_s|$ shares (at the exponent) belonging to users in $\tilde{U}_s \setminus \mathcal{G}_s$. That is, $\bar{g}_j$ for all $j \in \tilde{U}_s \setminus \mathcal{G}_s$, and

(ii) the $t + |\mathcal{G}_s| - |\tilde{U}_s|$ auxiliary shares (at the exponent). That is, $\bar{g}_j$ for all $j \in \Theta$.

Thus, the broadcast message $\mathcal{M}$ contains $|\Delta| = t$ shares of $F_z(x)$ (at the exponent).

Every user $U_i \in \mathcal{G}_s$ can compute the group key from $\mathcal{M}$ and his share $F_i(z)$ (=$F_z(i)$). Note that $\bar{g}^{F_z(i)}$ ($=\bar{g}^{F_z(i)}$) is not included in $\mathcal{M}$ so user $U_i$ can reach the threshold $t + 1$ of $F_z(x)$ (at the exponent). The group key is computed by employing the Lagrange interpolation formula (at the exponent level) as follows.

$$GK_s = (\bar{g}^{F_i(z)})^{\psi(\Delta \cup \{i\}, i)} \times \prod_{j \in \Delta} (\bar{g}_j)^{\psi(\Delta \cup \{i\}, j)}$$
6.2. A Decentralised GKDS

\[ (g^{F_z(i)})^{\psi(\cup \cup \{i\},i)} \times \prod_{j \in \Delta} (g^{F_z(j)})^{\psi(\cup \cup \{i\},j)} \]

\[ = \prod_{j \in \Delta \cup \{i\}} (g^{F_z(j)})^{\psi(\cup \cup \{i\},j)} \]

\[ = g^{\sum_{j \in \Delta \cup \{i\}} F_z(j) \times \psi(\cup \cup \{i\},j)} \]

\[ = g^{F_z(0)} \]

\[ = g^{rF_z(0)} \mod p . \]

Since the threshold is \( t + 1 \), only \( t \) shares (at the exponent) can be released in \( M \). The \( t \) shares could be shares belonging to users in \( U_s \setminus G_s \), auxiliary shares, or both. This implies that the protocol can revoke at most \( |U_s \setminus G_s| = t \) users and so the subgroup \( G_s \) has at least \( |U_s| - t \) users at a time.

It is straightforward to show that \((t + 1)(\log_2 p + \log_2 q)\) bits are broadcasted in the protocol and that each authorised user has to compute \( t + 1 \) modular exponentiations.

\[ \square \]

**Theorem 6.3** The subgroup protocol provides Subgroup Secrecy – For any session \( S_s \) where \( S_1 \leq S_s \leq S_M \) and \( E_s = \text{Subgroup} \), a collusion \( C \subseteq U_s \setminus G_s, |C| \leq t \) cannot find the group key \( G K_s \).

**Proof:** We note that \( C \) (at most \( t \) non-subgroup users) cannot obtain \( F_z(0) \) from their secret keys \( K(C) = \bigcup_{U_i \in C} K(U_i) = \{ F_i(x) : U_i \in C \} \) and so cannot calculate the group key \( G K_s = g^{rF_z(0)} \) (\( g \) is publicly known). This is because \( C \) has at most \( t \) shares, \( F_z(i) (= F_i(z)) : U_i \in C \), while calculating \( F_z(0) \) requires \( t + 1 \) shares. Moreover, \( C \) cannot obtain \( G K_s \) from the broadcast message \( M \) (only containing \( t \) shares at the exponent) and computation on \( g^{F_z(i)} : U_i \in C \) will not give any additional information because they are already released in \( M \).

Now we consider the case where \( C \) tries to find the group key \( G K_s \) from broadcast messages. The proof uses the Decisional Diffie-Hellman (DDH) assumption that can be found in Chapter 1.

We use a “reduction argument”: if there exists an algorithm (probabilistic polynomial-time) \( V \), that uses the secret keys \( K(C) \), the broadcast message \( M \) and all public information, to distinguish \( G K_s = g^{rF_z(0)} \) from a random value, then \( V \) contradicts the DDH assumption.

Without loss of generality, we assume that the group initiator is \( U_z \) who performs polynomially many subgroup operations. We assume \( C = \{ U_d \} \ (t = 1) \). Let \( V \) be the algorithm that on input of values \( F_z(d) \), polynomially many tuples \( (g^r, g^{rF_z(d)}, g^{rF_z(0)}) \).
generated with randomly chosen \( r_j \)'s, and a pair \( g^r, g^{rF_z(d)} \), distinguishes between \( g^{rF_z(0)} \) and a random value.

Let \( V' \) be the algorithm that uses \( V \) to break the DDH assumption. \( V' \) is given input \( g^a, g^b \) and \( C \), and has to decide whether \( C \) is \( g^{ab} \) or a random value. \( V' \) generates inputs to \( V \). Let \( F_z(0) = b \) and \( r = a \). \( V' \) generates a randomly chosen set \( \mathcal{C}' = \{ U_i \} \). \( V' \) also generates values \( F_z(d') \), random \( r_j \)'s and tuples \( \langle g^{r_j}, g^{r_jF_z(d')}, g^{r_jb} \rangle \), and gives them to \( V \). Then \( V' \) gives the values \( g^a, g^{aF_z(d)} \) to \( V \), takes the output of \( V \) and outputs the same value. In this way \( V' \) can distinguish between \( g^{ab} \) and \( C \), contradicting the DDH assumption.

Observe that to be efficient, the threshold \( t \) has to be relatively small and consequently the above protocol can be used to form relatively large subgroups. To form a small subgroup, say \( |\mathcal{G}_s| \approx 0 \), we may use the following simple protocol to establish the group key.

1. **Transmission**: A group initiator \( U_z \in \mathcal{G}_s \) does the following.
   
   (a) Randomly chooses a group key \( GK_s \in \mathbb{Z}_p^{*} \) and encrypts it with keys \( K(z,i) = F_z(i) \) for all \( U_i \in \mathcal{G}_s \).
   
   (b) Broadcasts to all users a subgroup message
   
   \[ \mathcal{M} = \left\{ z, E_{K_{(z,i)}}(GK_s) \mid i : U_i \in \mathcal{G}_s \right\}, \]
   
   where \( E_{K(z,i)}() \) is a symmetric cipher with key \( K(z,i) \).

2. **Calculation**: Each user \( U_i \in \mathcal{G}_s \) computes \( K_{(z,i)} = F_i(z) \) (from the symmetric property of \( F(x,y) \)) and decrypts \( E_{K_{(z,i)}}(GK_s) \in \mathcal{M} \) to obtain the group key \( GK_s \).

It is obvious that only users in \( \mathcal{G}_s \) can obtain the group key \( GK_s \). The broadcast size is \( |\mathcal{G}_s| \log_2 p + (|\mathcal{G}_s| + 1) \log_2 q \) bits and an authorised user performs a single decryption operation. In general the protocol can be used to form subgroups of any size.

### 6.2.3 Join Event

We divide user join into two subclasses: **sponsorship (partial join)** and **full join**. The importance of these operations is that the admission of new users to groups is controlled by the user(s) instead of by the GC\(^1\) and so the scheme is fully decentralised.

\(^1\)To our knowledge, in all the previous GKDS the join operation, if possible at all, is GC-dependent requiring an on-line GC.
6.2. A Decentralised GKDS

Sponsorship (Partial Join): The new user is sponsored by an existing group member and receives secret information from him through a secure unicast channel. The new user can only calculate the session keys for groups formed by his sponsor and is effectively a passive user whose membership is fully controlled by his sponsor.

Full Join: The new user is sponsored by multiple group members and receives secret information from them. If he receives enough sponsorship, he becomes a group member with the same rights as other group users (i.e., for example he can perform subgroup and user join operations).

Our proposed join protocol adds a new step prior to the transmission and calculation steps in order to satisfy the backward secrecy requirement of the join operation. In the new step, the system’s secret symmetric polynomial \( F(x, y) \) and users’ secret polynomials are updated or refreshed. Each update or refresh uses a group key known to all existing users but not to new users, and the system’s polynomial and users’ polynomials are self-refreshing without further communication after the group key is established. We note that updating a secret and shares of the secret has been considered in proactive secret sharing schemes [38].

Sponsorship (Partial Join)

The sponsorship operation allows a new user to be in a group formed by his sponsor. The privileged set of this operation is \( \mathcal{P}_s = \{ U_z \} \), where the sponsor \( U_z \) can be any user in \( \mathcal{U}_s \).

Suppose a sponsor \( U_z \in \mathcal{U}_s \) wants to form a group that includes a set of new users \( \mathcal{J}_s \subseteq \mathcal{N} \setminus \mathcal{U}_s \). \( U_z \) needs to sponsor the new users into \( \mathcal{U}_s \) as follows. Let \( \tilde{\mathcal{U}}_s = \{ i : U_i \in \mathcal{U}_s \} \) be the set of identities.

1. Self-refreshing: Users \( U_i \in \mathcal{U}_s \) do the following.
   
   (a) The sponsor \( U_z \) performs a refresh event to establish a group key \( GK_s \) for the group \( \mathcal{U}_s \) by invoking the subgroup protocol in Section 6.2.2 with \( \mathcal{G}_s = \mathcal{U}_s \).
   
   (b) The users \( U_i \in \mathcal{U}_s \) update their secret information \( K(U_i) = \{ F'_i(x) \} \) where \( F'_i(x) = GK_s + F_i(x) \mod q \).

2. Transmission and Computation: For each new user \( U_i \in \mathcal{J}_s \), the sponsor \( U_z \) assigns a unique identity \( i \in \mathbb{Z}_q \) such that \( i \notin \tilde{\mathcal{U}}_s \) and \( i \neq 0 \), then uses his
secret polynomial $F'_z(x) \in \mathcal{K}(U_z)$ to compute $f'_{(z,i)}(x) = F'_z(i)$ (a share of $F'_z(x)$) and securely sends it to the new user $U_i$. The new user $U_i$ keeps $f'_{(z,i)}$ as his secret information $\mathcal{K}(U_i) = \{f'_{(z,i)}\}$.

After giving secret information to all new users, the sponsor $U_z$ can establish session keys for groups that include the new users in subsequent sessions. In this case, the sponsor will become a group initiator who executes the subgroup protocol in Section 6.2.2 to establish the group keys. The new users can only calculate the group keys using their secret information and cannot perform other operations. Observe that an execution of refresh event (subgroup operation with $G_s = U_s \cup J_s$) following the sponsorship protocol will give a group key for users in $U_s \cup J_s$.

Note that only the sponsor $U_z \in U_s$ can perform operation on groups that include the new users in $J_s$ (since $U_z$ is the manager of those groups). Other users in $U_s \setminus \{U_z\}$ cannot do so since the new users do not have shares of secret information (polynomials) belonging to users in $U_s \setminus \{U_z\}$. To do so, they have to sponsor the new users into their own groups (we assume the sponsors assign the same identity for a new user). At most $t$ sponsors are allowed to sponsor the same new users to maintain the passiveness property of the new users (this can be done by, for example, publishing the status of existing groups).

In practical applications, sponsorship provides a solution for a head office (group controller) to share some admission capability to several trusted branches (users). The trusted branches can create their own groups while the head office will still be able to monitor the whole system.

**Theorem 6.4** The sponsorship (partial join) protocol with a self-refreshing step allows a group member to sponsor a set $J_s$ of arbitrary number of new users to a group $U_s$. Sponsoring a new user requires a transmission of $\log_2 q$ bits over a secure unicast channel. An arbitrary collusion of at most $t$ new users sponsored by one or more group members cannot learn all group keys that were established prior to their admission.

**Proof:** It is straightforward from the above description. The security proof follows from that of the full join protocol in Theorem 6.5 (to be given later). \)

**Full Join**

To have the same capability as existing users, new users must be sponsored by at least $t + 1$ group members. Let $P_s \subseteq U_s, |P_s| \geq t + 1$, be the privileged set of sponsors
and \( \bar{P}_s = \{ z : U_z \in P_s \} \) be the set of identities. Suppose the sponsors agree on the admission of a set of new users \( J_s \) to the group \( U_s \).

1. **Self-refreshing:** Similar to step 1 of sponsorship (partial join) protocol. A sponsor \( U_z \in P_s \) performs a refresh event to create a group key \( GK_s \), and users \( U_i \in U_s \) use the common key to update their secret information \( K(U_i) = \{ F'_i(x) \} \).

2. **Transmission:** Sponsors \( U_z \in P_s \) do the following.

   (a) Agree on a unique identity \( i \in \mathbb{Z}_q^* \) for each new user \( U_i \in J_s \) such that \( i \notin \bar{U}_s \).

   (b) Use their secret polynomials \( F'_z(x) \in K(U_z) \) to compute \( f'_{(z,i)} = F'_z(i) \) and independently distribute join messages

   \[ M = \{ f'_{(z,i)} \parallel z \} \]

   over secure unicast channels to each new user \( U_i \in J_s \).

3. **Calculation:** Each new user \( U_i \in J_s \) receives the join messages \( M \) from all sponsors and computes his secret information using Lagrange interpolation as follows.

   \[ K(U_i) = \left\{ \sum_{z \in \bar{P}_s} f'_{(z,i)} \Psi(\bar{P}_s, z) \mod q \right\}. \]

A summary of the full join protocol is shown in Table 6.4. After all new users obtain their secret information, any user in \( U_s \cup J_s \) performs a refresh event (subgroup operation with \( G_s = U_s \cup J_s \)) to establish a group key for \( U_s \cup J_s \).

<table>
<thead>
<tr>
<th>Sponsors ( U_z \in P_s )</th>
<th>New users ( U_i \in J_s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle F'_z(x) \rangle )</td>
<td>( \langle q \rangle )</td>
</tr>
<tr>
<td>( f'_{(z,i)} = F'_z(i) )</td>
<td>( M = { f'_{(z,i)} \parallel z } )</td>
</tr>
<tr>
<td>( F'<em>i(x) = \sum</em>{z \in \bar{P}<em>s} f'</em>{(z,i)} \Psi(\bar{P}_s, z) \mod q )</td>
<td></td>
</tr>
</tbody>
</table>

Note that a full join protocol is indeed a collection of multiple sponsorship (partial join) protocols performed by different sponsors. It follows that the security of full join protocol subsumes the security of sponsorship (partial join) protocol.
Theorem 6.5 The full join protocol with a self-refreshing step satisfies Join Secrecy –
For any session \( S_s \) where \( S_1 \leq S_s \leq S_M \), a collusion \( C \subseteq \bigcup S_s : S_s \leq S_a \leq S_M, E_a = \) Join and \(|C| \leq t \) cannot find the group keys \( GK_b : S_1 \leq S_b < S_s \).

Proof: Although the system secret \( F(x, y) \) is updated in every join operation, we may assume secret information of \( C, K(C) = \{ F'_i(x) : U_i \in C \} \) is determined from an updated system secret \( F'(x, y) = GK_s + F(x, y) \mod q \), as when \( C \) tries to find group keys of previous sessions. Referring to the self-refreshing step, the group key \( GK_s \) is established by a sponsor \( U_z \in P_s \) before joining the new users (the collusion \( C \)). Since \( C \) do not know \( GK_s \), they cannot find group keys of sessions before \( S_s \). So the security is guaranteed by the difficulty of finding \( GK_s \). The rest of the proof is similar to that of Theorem 6.3. The proof also implies that \( C \) cannot use the join messages \( M \) to break join secrecy.

\[ \square \]

Theorem 6.6 Using the full join protocol, any \( t + 1 \) or more group members may fully admit a set \( J_s \), of arbitrary size, of new users to a group \( U_s \), while any \( t \) or less group members cannot do so. It requires transmission of \( 2(t+1) \log_2 q \) bits over secure unicast channels to admit a new user.

Proof: After the self-refreshing step, the system secret \( F(x, y) \) has been updated to \( F'(x, y) = GK_s + F(x, y) \mod q \). To have a full membership in the group \( U_s \), a new user \( U_i \in J_s \) must have secret information \( F'(x, y)|_{y=i} \). Since \( U_i \) receives at least \( t + 1 \) pieces of information from the sponsors, that is \( f'_z = F'_i(z) = F'_i(i) \), for all \( z \in P_s \) (in general \( |P_s| \geq t + 1 \)), he can interpolate to obtain \( K(U_i) = \{ F'_i(x) = F'(x, i) \} \). Clearly, less than \( t + 1 \) pieces of information are not enough to compute \( F'(x, i) \). The sponsors need to transmit messages of a total of \( 2(t+1) \log_2 q \) bits to give the \( t + 1 \) pieces of information.

\[ \square \]

Discussion

The backward secrecy of the user join protocol relies on a Self-refreshing step where secret information in the system is updated before admitting new users. As an alternative, we may apply the following method to ensure backward secrecy, with the extra cost of additional encryption and decryption operations in each execution of the subgroup protocol in Section 6.2.2.

The idea is to encrypt the subgroup message with a key shared by all group users. That is, prior to the execution of the subgroup protocol in session \( S_s \), the group initiator
(or any user in $\mathcal{U}_s$) performs a refresh operation to establish a key $K$ for the group $\mathcal{U}_s$ (users in $\mathcal{U}_s$ do not update their secret information). When executing the subgroup protocol, the group initiator is required to encrypt the subgroup message using the key $K$ and broadcast the encrypted version, and the authorised users are required to decrypt the encrypted broadcast so that they can compute the session key. Note that we may use the same key $K$ to encrypt subgroup messages of consecutive sessions (i.e., as long as the group memberships are unchanged).

Using this alternative method, a sponsor $U_z$ only needs to securely send shares $F_z(i)$ to new users. The backward secrecy of user join relies on the encryption of subgroup messages in previous sessions. The new users, even $t$ of them colluding, do not have the encrypting keys and so they cannot access the content of the encrypted broadcasts (assuming the encryption algorithm is secure). It follows that the collusion does not have adequate information to compute the previous session keys. The argument in Theorem 6.3 implies the security proof of this method.

6.2.4 Evict Event

The aim of this operation is to establish a common key for group users, excluding some evicted users, and prevent the evicted users from being parts of any authorised groups in future sessions. It requires that the evicted users (even collusion of them) cannot obtain any future group keys. Moreover, in decentralised systems the evicted users cannot participate in any future group operations. A naive approach to satisfying the requirements is to generate a new system secret (i.e., a new random symmetric polynomial) in a session of eviction, and exclude all currently evicted users from the transmission of shares (i.e., new polynomials) of the new secret. This approach is clearly secure but very inefficient since it essentially re-initialises the system. In the following we discuss two approaches of adding user eviction to the scheme that utilise the subgroup protocol in Section 6.2.2. Suppose in session $S_s$, a set of users $\mathcal{R}_s$ are evicted from $\mathcal{U}_s$ so the set $\mathcal{U}_s \setminus \mathcal{R}_s$ consists of authorised users for the session.

Trivial Approach

An evictor performs the subgroup protocol with $\mathcal{G}_s = \mathcal{U}_s \setminus \mathcal{R}_s$ that gives a common key $GK_s$ for the authorised group $\mathcal{G}_s$. For an effective eviction, executions of subgroup protocol in subsequent sessions must consistently also exclude $\mathcal{R}_s$ from authorised groups, that is, $\mathcal{R}_s \subseteq \mathcal{U}_a \setminus \mathcal{G}_a$ (i.e., $\mathcal{R}_s \cap \mathcal{G}_a = \emptyset$), $E_a \in \{\text{Subgroup, Evict, Refresh}\}$ for $S_a = S_s, \cdots, S_M$ so that the evicted users cannot find future group keys. Consequently,
the set of excluded users is monotonically increasing and, to maintain security, it must not exceed \( t \) users in a session. Referring to the argument in Theorem 6.3 gives the following theorem.

**Theorem 6.7** Using the trivial approach for user eviction satisfies evict secrecy – For any session \( S_s \) where \( S_1 \leq S_s \leq S_M \), a collusion \( C \subseteq \bigcup \mathcal{R}_a, S_1 \leq S_a \leq S_s, E_a = \text{Evict}, \) and \( |C| \leq t \), cannot find the group keys \( \text{GK}_b, S_s \leq b \leq S_M \).

Note that the size of the excluded user set is bounded above by \( t \) where \( t \) is the degree of polynomial used in the system. Small values of \( t \) restrict the number of excluded users, while large \( t \) values give inefficient communication cost. In the following we give a method of having scalable user eviction with reasonable communication cost by sacrificing efficiency of user storage. The idea is to employ multiple independent polynomials with monotonically increasing degrees (for example, \( t, 2t, 3t, \cdots \)). By default, the group initiator uses a polynomial with the lowest, but sufficiently large degree, to exclude users in the execution of subgroup protocol. When the size of the excluded user set exceeds the degree of the currently used polynomial, a group initiator replaces it with another polynomial with higher degree in the execution of the subgroup protocol. With this method, the communication cost depends on the number of excluded users, and increases when there are more users to be excluded. To provide users with the required polynomials, during system initialisation, the GC generates random symmetric polynomials with monotonically increasing degrees and securely delivers a share of every polynomial to each user. This requires large user storage. Note that as those polynomials are determined during the initialisation phase, their degrees should be chosen by taking into account the worst scenario.

This approach has stateless user storage and, using the above method, it is possible to evict arbitrarily large numbers of users. An evicted user is prevented from computing future group keys, but he may participate in future group operations, for example, contributing to the admission of new users.

We may separate the broadcast message of a subgroup protocol into two parts. The first part related to the evicted users of the preceding sessions and the second part related to the excluded users of the current session. The first part is usually much longer than the second part. To have efficient execution time in a session, the first part can be computed and broadcasted prior to the session (for example, during system idle time) since the evicted users were already known. Users of the session might do some pre-computation towards the group key using the first part, however, only authorised
users may complete the group key computation when the second part is broadcasted in the session.

**Self-refreshing Approach**

An evictor performs the subgroup protocol with $G_s = U_s \setminus R_s$ that gives a common key $GK_s$, and the authorised users $U_i \in G_s$ update their secret information $K(U_i) = \{F_i'(x) = GK_s + F_i(x) \mod q\}$ (and so update the system secret $F'(x, y) = GK_s + F(x, y) \mod q$). Successive group operations must use the updated secret information in their protocol executions, and successful executions of the subgroup protocol will not exclude $R_s$ from authorised groups. Without knowing $GK_s$, the evicted users $R_s$ cannot update their secret information to match the new system secret. Their outdated secret information is unusable both for computing future group keys and for contributing to further group operations, and so they are effectively evicted. The security proof of above user eviction follows the argument in Theorem 6.3. Nevertheless, this approach does not completely satisfy evict secrecy: it is secure only if the colluders belong to the same session. A collusion of evicted users from different sessions will break the system as the following shows.

The user group in session $S_{s+1}$ is $U_{s+1} = U_s \setminus R_s$ and they know the group key $GK_s$. Without loss of generality, assume another set of users $R_{s+1}$ is evicted from $U_{s+1}$. An evictor applies the subgroup protocol with $G_{s+1} = U_s \setminus R_{s+1}$ to obtain a group key $GK_{s+1}$. The group key is obviously unknown to $R_{s+1}$ and also $R_s$ since users in $R_s$ do not have $GK_s$ to match the updated system secret. However, if some users from $R_{s+1}$ and some users from $R_s$ collude, they will succeed in obtaining $GK_{s+1}$.

For example, a user $U_i$ in $R_{s+1}$ who knows the group key $GK_s$ can reveal it to a user $U_i'$ in $R_s$. Then user $U_i'$ will know his updated secret information $K(U_i') = \{F_i'(x) = GK_s + F_i'(x) \mod q\}$, which can be used to obtain $GK_{s+1}$ and subsequent group keys.

This approach has stateful user storage and allows at most $t$ users to be evicted in a session. Assuming a single session collusion, a colluder is prevented from both computing subsequent group keys and performing any role in future group operations.

**Discussion**

It is worth pointing out that in practice, user eviction can be addressed by applying certain policies. For example, group members who voluntarily return their smart-cards at the end of a subscription period will be returned their deposited moneys. A smart-card contains the secret information of a user and a user will be effectively evicted by
6.2. A Decentralised GKDS

returning his smart-card as he no longer has the secret information to participate in group operations (assuming the smart-card is secure). This approach would reduce the number of users that need to be evicted using the trivial approach or the self-refreshing approach.

6.2.5 Traceability

The aim of traitor tracing is to trace the colluders whose keys are used to build an illegal decryption device (pirate device). Boneh and Franklin in [15] constructed a public key tracing scheme based on the Decisional Diffie-Hellman assumption. Their scheme guarantees deterministic tracing if the extracted keys are in canonical form and black-box confirmation test is possible. Naor and Pinkas in [64] proposed similar tracing algorithms for their revocation scheme. We use the same approach as [64]. The main difference from [64] is that in our case traitors’ secret keys are polynomials. The pirate device can calculate the group keys from broadcast messages. The group controller (GC) will act as the tracer.

Let the key information of the user $U_i$ in canonical form be written as a vector $V_i = (1, i, i^2, \ldots, i^t) \pmod{q}$, and a polynomial $F_i(x)$ which is the inner product of $V_i$ and $(B_0, B_1, B_2, \ldots, B_t)$ where $B_k = \sum_{l=0}^{t} a_{l,k} x^l \pmod{q}$, $k = 0, \ldots, t$, are the coefficients of the symmetric polynomial $F(x,y) = \sum_{k=0}^{t} B_k y^k \pmod{q}$.

Pirate Key

Consider a set $C = \{U_{i_1}, U_{i_2}, \ldots, U_{i_w}\}$, $w \leq t$, of colluders who construct a pirate device. In the following we show that if colluders create a linear combination of the vectors $V_{pr} = \alpha_1 V_{i_1} + \alpha_2 V_{i_2} + \cdots + \alpha_w V_{i_w} \pmod{q}$ and a linear combination of the polynomials $F_{pr}(x) = \alpha_1 F_{i_1}(x) + \alpha_2 F_{i_2}(x) + \cdots + \alpha_w F_{i_w}(x) \pmod{q}$, and supply the pair $V_{pr}$ and $F_{pr}(x)$ to a pirate device\footnote{We assume that this is the only way for colluders to construct the pirate device.}, the device can calculate the group key from a broadcast message $\mathcal{M}$.

Deriving group key  We begin by describing the subgroup message $\mathcal{M}$ broadcasted by a privileged user $U_z$ that holds the secret polynomial $F_z(x)$. Let $\bar{C} = \{ i : U_i \in C \}$ be the set of colluders’ identities. Without loss of generality, suppose the message is, $\mathcal{M} = \{ \bar{y}, z, \bar{y}^{F_z(j)} \pmod{p} \mid j \in \Delta \}$
where $\Delta \subseteq \mathbb{Z}_q^*$, $|\Delta| = t$ and $\bar{\mathcal{C}} \not\subseteq \Delta$. Let $\Delta = \{j_1, j_2, \ldots, j_t\}$. The pirate device can construct vectors $V_j = (1, j, j^2, \ldots, j^t) \pmod{q}$ for all $j \in \Delta$ and form the matrix equation

$$
\begin{pmatrix}
V_{j_1} \\
V_{j_2} \\
\vdots \\
V_{j_t} \\
V_{pr}
\end{pmatrix}
\begin{pmatrix}
z_0 \\
z_1 \\
\vdots \\
z_{t-1} \\
z_t
\end{pmatrix}
= \begin{pmatrix}
F_z(j_1) \\
F_z(j_2) \\
\vdots \\
F_z(j_t) \\
F_{pr}(z)
\end{pmatrix} \pmod{q},
$$

(6.1)

where $z_0, z_1, \ldots, z_{t-1}, z_t$ are coefficients of the polynomial $F_z(x)$ and $F_z(j_1), F_z(j_2), \ldots, F_z(j_t)$ are $t$ points of the polynomial. These values are unknown to the pirate device. We may represent the coefficients of the polynomial $F_z(x)$ as a vector $Z = (z_0, z_1, \ldots, z_{t-1}, z_t)$. We first show that $F_{pr}(z) = V_{pr} \cdot Z \pmod{q}$, where "\cdot" denotes inner product operation. From the symmetric property of the polynomial $F(x, y)$, we know that $F_i(z) = V_i \cdot Z \pmod{q}$ for all $i \in \bar{\mathcal{C}}$. Then we have

$$
F_{pr}(z) = \sum_{i \in \bar{\mathcal{C}}} \alpha_i F_i(z)
= \sum_{i \in \bar{\mathcal{C}}} \alpha_i (V_i \cdot Z)
= \left( \sum_{i \in \bar{\mathcal{C}}} \alpha_i V_i \right) \cdot Z
= V_{pr} \cdot Z \pmod{q}.
$$

Let $A, B$ and $C$ denote the three matrices in equation (6.1), from left to right. We have $AB = C$. To deduce the group key $\tilde{g}^{F_z(0)} (= \tilde{g}^{z_0})$, the pirate device calculates the inverse of the matrix $A (= A^{-1})$ and modifies the matrix equation to $B = A^{-1}C$. Observe that multiplying the first row of the matrix $A^{-1}$ by matrix $C$ will give $z_0$, that is, $z_0 = d_1F_z(j_1) + d_2F_z(j_2) + \cdots + d_tF_z(j_t) + d_{t+1}F_{pr}(z) \pmod{q}$ where $d_1, d_2, \ldots, d_t, d_{t+1}$ are elements of the first row of the matrix $A^{-1}$. Then the group key is found as

$$
\tilde{g}^{z_0} = (\tilde{g}^{F_z(j_1)})^{d_1} \times (\tilde{g}^{F_z(j_2)})^{d_2} \times \cdots \times (\tilde{g}^{F_z(j_t)})^{d_t} \times (\tilde{g}^{F_{pr}(z)})^{d_{t+1}} \pmod{p}.
$$

Notice that the values needed to calculate the group key $\tilde{g}^{z_0}$ are known by the pirate device using the broadcast message $\mathcal{M}$ and the pair $V_{pr}$ and $F_{pr}(x)$.

**Tracing Algorithm**

Traitor tracing may be of one of the following two types.
1) **Black box tracing** If the keys in the pirate decoder cannot be extracted, the GC examines the output of the pirate box on chosen inputs and uses this information to trace traitors. The tracing algorithm follows the methods proposed in [64] and uses a black-box confirmation test. The test can determine if a candidate set of users have contributed to the construction of a given pirate box. The black-box confirmation test for a candidate set $T$ is as follows. Let $|T| \leq t$ and $\tilde{T} = \{ i : U_i \in T \}$.

1. If $|T| < t$, generate a random set $W$, $|W| = t$, of users such that $T \subseteq W$. If $|T| = t$ then $W = T$.

2. Construct a symmetric polynomial $P(x, y)$ such that $P(x, i) = F(x, i)$ for all $U_i \in W$ but $P(x, i) \neq F(x, i)$ for all $U_i \notin W$.

3. Generate a message $M = \{ g, v, \tilde{g}^{P(j, v)} \mod p \mid j \in D \}$ for a random element $v \in \mathbb{Z}_q^*$ and a random set $D \subseteq \mathbb{Z}_q^*$ such that $|D| = t$ and $\tilde{T} \cap D = \emptyset$, and feed $M$ to the pirate device.

4. If the output of the pirate device is $\tilde{g}^{P(0, v)}$, the GC knows $T \cap C = C$. Otherwise, the GC may conclude $T \cap C = \emptyset$. Note that if $T \cap C \neq \emptyset$ and $T \cap C \neq C$, the algorithm cannot guarantee anything.

The tracer examines $t$-subsets of users and uses the confirmation test to determine if a candidate set contains all the colluders. Once such a $t$-set is found, then a tracing algorithm similar to [64] can be used to identify a specific traitor.

2) **Tracing when the pirate key is accessible** We assume that if a pirate decoder can decrypt with negligible error then the pirate key inside the decoder is a linear combination of secret keys of colluders.

Given the pirate key in canonical form and assuming that at most $\frac{t}{2}$ colluders have contributed to the construction of the pirate device, the tracing problem is the same as the decoding problem in a (Generalised) Reed-Solomon code that has a polynomial time algorithm [56]. We have the following result.

**Theorem 6.8** Given a pirate key which is a linear combination of at most $\frac{t}{2}$ keys of group members, it is possible to trace colluders whose keys appear with non-zero coefficients in the linear combination.

For more details on the algorithm and its correctness, refer to [64].
6.3  The Extended Scheme

The full join protocol in Section 6.2.3 (we will refer to full join as join in the rest of this chapter) has the property that any $t + 1$ or more group members can admit new users. An increase in the value of $t$ makes join events less accessible, as approval of many users is required. In some cases it is desirable that less than $t + 1$ users be able to join new users. That is, users in some specified subsets might need higher privileges. We extend the group key distribution scheme in Section 6.2 to provide this property.

We need additional cryptographic tools that will be employed in the construction. The notion of cumulative scheme and the concept of $(v,v)$-threshold scheme are described in Section 1.4.

In the extended scheme a privileged set of sponsors can be any subset of at least $t + 1$ users, or an access set of a minimal access structure. That is, $\mathcal{P}_s \in \Gamma^- \cup \{ \mathcal{A} : \mathcal{A} \subseteq \mathcal{U}_s, |\mathcal{A}| \geq t + 1 \}$, where $\Gamma^-$ is a collection of $t$-arbitrary access sets $\mathcal{A} \subseteq \mathcal{U}_0, |\mathcal{A}| \leq t$, defined by the GC. The extended scheme inherits all properties of the basic scheme and also has the property of not only any $t + 1$ or more users, but also all users of a $t$-arbitrary access set, being able to grant full membership to a new user.

6.3.1 System Initialisation

The group controller (GC) performs the following steps.

1. Performs steps 1, 2, and 3 of the system initialisation in Section 6.2.1.

2. Specifies a minimal access structure $\Gamma^-$ over $\mathcal{U}_0$ and let $\mathcal{U}^- = \bigcup_{\mathcal{A} \in \Gamma^-} \mathcal{A}$. Note that $\mathcal{U}^- \subseteq \mathcal{U}_0$.

3. Constructs and publishes a $w \times v$ cumulative array $C(\Gamma^-) = [c_{ij}]$ where $w$ and $v$ are the cardinalities of the sets $\mathcal{U}^-$ and $\mathcal{F}$, respectively. Let each user $U_i \in \mathcal{U}^-$ correspond to a set $\beta_i$ consisting of all columns indexed by $j$ where $c_{ij} = 1$, i.e., $\beta_i = \{ j : c_{ij} = 1 \}$.

4. Randomly chooses $v - 1$ symmetric polynomials of degree $t$ in $x$ and $y$ over $\mathbb{Z}_q$, $Y_1(x, y), Y_2(x, y), \cdots, Y_{v-1}(x, y)$, and calculates

$$Y_v(x, y) = F(x, y) - \sum_{j=1}^{v-1} Y_j(x, y) \mod q.$$  \hspace{1cm} (6.2)

All polynomials in this step are kept secret.
5. Associates column \( j \) of the cumulative array with the symmetric polynomial \( Y_j(x, y) \), for \( 1 \leq j \leq v \). Observe that the set \( F \) of the cumulative array \( C(\Gamma^-) \) is \( F = \{ Y_1(x, y), Y_2(x, y), \ldots, Y_v(x, y) \} \).

6. Gives elements of \( F \) to each user \( U_i \in \mathcal{U}^- \) if and only if \( c_{ij} = 1 \) through a secure unicast channel. Thus, the set \( \alpha_i = \{ Y_j(x, y) : j \in \beta_i \} \).

The secret information of the GC is \( F(x, y) \) and \( F \). The secret information of each user \( U_i \in \mathcal{U}^- \) is \( K(U_i) = \{ F_i(x), \alpha_i \} \) and that of each user \( U_i \in \mathcal{U}_0 \setminus \mathcal{U}^- \) is \( K(U_i) = \{ F_i(x) \} \). Observe that the set \( \alpha_i \) consists of at most \( v \) symmetric polynomials of degree \( t \).

**Theorem 6.9** In the extended scheme, (i) storage of a group controller is \((v+1)(t+1)(t+2)/2\) \( \log_2 q \) bits (ii) storage of a user not in \( \mathcal{U}^- \) is \((t+1)\log_2 q \) bits and (iii) storage of a user in \( \mathcal{U}^- \) is at most \((1 + v(t+2)/(2))t + 1\) \( \log_2 q \) bits of secret information.

In assessing security, we assume that the collusion \( C, |C| \leq t \), does not contain a \( t \)-arbitrary access set, \( C \not\supseteq A, \forall A \in \Gamma^- \). We show that \( C \) only knows secret information \( K(C) = \bigcup_{U_i \in C} K(U_i) = \{ F_i(x), \alpha_i : U_i \in C \} \) and cannot gain additional information about \( F(x, y) \) from \( K(C) \).

**Theorem 6.10** Any collusion \( C, |C| \leq t \), can obtain the session secret \( F(0, 0) \) for session \( S_s \) if and only if some \( t \)-arbitrary access set \( A \) of \( \Gamma^- \) is contained in \( C \).

**Proof:** If there is a \( t \)-arbitrary access set \( A \) in \( C \) then the only unknown in equation (6.2) is \( F(x, y) \), which can be thus obtained.

Without loss of generality, we take the most powerful \( C \) not containing a \( t \)-arbitrary access set to be of size \( t \), and to hold all \( Y_j(x, y), 1 \leq j \leq v - 1 \). Using the symmetry property in \( F(x, y) \) and their key information, the colluders can calculate \( t \) points, \( F(i, k), U_i \in C, F(x, k), \forall U_k \not\in C \). Equation (6.2) gives them one equation for \( F(k', k), U_{k'} \not\in C \) but with another unknown \( Y_{v}(k', k) \) also. The colluders cannot solve this, so cannot find \( F(0, 0) \) letting \( U_{k'} = U_k = U_0 \not\in N \).

### 6.3.2 Subgroup Event

Subgroup protocol for the extended scheme is similar to the subgroup protocol in Section 6.2.2. The difference is in this case, the formed subgroup \( \mathcal{G}_s \) also has to satisfy \( A \not\subseteq \mathcal{U}_s \setminus \mathcal{G}_s \), for all \( A \in \Gamma^- \) in order to maintain subgroup secrecy.
In the extended scheme a user $U_i$ only uses $F_i(x)$ to process the subgroup protocol. Thus, Theorem 6.10 and the results in Section 6.2.2 guarantee security and efficiency of the protocol.

### 6.3.3 Join Event

Join protocol for the extended scheme has the property that sponsorship of at least $t+1$ users can join a new user. This follows from the join protocol in Section 6.2.3. In this section, we show in particular how sponsors in a privileged set ($t$-arbitrary access set) $\mathcal{P}_s \in \Gamma^-$ grant membership to new users.

The admission of new users in $\mathcal{J}_s \subseteq \mathcal{N} \setminus U_s$ to the group $U_s$ is as follows, recalling the identity set $\hat{U}_s = \{i : U_i \in U_s\}$. Note that $\mathcal{P}_s \subseteq U^- \subseteq U_s$.

1. **Self-refreshing:** Same as step 1 of the join protocol in Section 6.2.3. The result is the group key $GK_s$, shared by users $U_i \in U_s$, and the updated secret information $F'_i(x)$ of the users. In addition, users $U_i \in U^-$ need to update secret information

   \[
   \alpha'_i = \{Y'_j(x,y) = u + Y_j(x,y) \mod q : j \in \beta_i\},
   \]

   where $u = \frac{GK_s}{v} \mod q$ (assuming $q > v$). After this step, the secret information of users $U_i \in U^-$ is $\mathcal{K}(U_i) = \{F'_i(x), \alpha'_i\}$ and that of users $U_i \in U_s \setminus U^-$ is $\mathcal{K}(U_i) = \{F'_i(x)\}$.

2. **Transmission:** Sponsors $U_z \in \mathcal{P}_s$ do the following.

   (a) Same as step 2a of the join protocol in Section 6.2.3.

   (b) Use their secret information $\alpha'_z \in \mathcal{K}(U_z)$ to compute $Y'_{j,i}(x) = Y'_j(x,i)$, for all $j \in \beta_z$ (note that $Y'_j(x,y) \in \alpha'_z$) and individually send join messages

   \[
   \mathcal{M} = \{Y'_{j,i}(x) \parallel j : j \in \beta_z\}
   \]

   to each new user $U_i \in \mathcal{J}_s$ over secure unicast channels. It is possible $\beta_{z_1} \cap \beta_{z_2} \neq \emptyset$, for some $U_{z_1}, U_{z_2} \in \mathcal{P}_s, U_{z_1} \neq U_{z_2}$. It is then enough for one sponsor to send $Y'_{j,i}(x)$, by convention the sponsor with the lowest $z$.

3. **Calculation:** Using the join messages $\mathcal{M}$ sent by all sponsors, each new user $U_i \in \mathcal{J}_s$ computes his secret information $\mathcal{K}(U_i)$ as follows. Let $\beta_P = \bigcup_{U_z \in \mathcal{P}_s} \beta_z$.

   \[
   \mathcal{K}(U_i) = \left\{ \sum_{j \in \beta_P} Y'_{j,i}(x) \mod q \right\}.
   \]
6.3. The Extended Scheme

Table 6.5: The join protocol of extended scheme

<table>
<thead>
<tr>
<th>Sponsors $U_z \in P_s$</th>
<th>New users $U_i \in J_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y'<em>{j,i} = Y'</em>{j}(x, i), \forall j \in \beta_z$</td>
<td>$Y_0(x) = Y_0(x, i)$, $i \in J_s$</td>
</tr>
</tbody>
</table>

A summary of the extended scheme’s join protocol is shown in Table 6.5.

Theorem 6.11 In the extended scheme, users in a $t$-arbitrary access set may perform join operations for a set $J_s$ of any number of new users, while any $t$ or less users that do not contain all users in a $t$-arbitrary access set cannot do so. It requires transmission of $v(t + 2) \log_2 q$ bits over secure unicast channels to join a new user.

Proof: After the self-refreshing step, the system secret $F(x, y) = F'(x, y) = G \Gamma + F(x, y) \mod q$. To be in $U_s$, a new user $U_i \in J_s$ has to have secret information $F'(x, y) |_{y=i}$. By the property of cumulative array $C(\Gamma)$, $\beta_p = \{1, \ldots, v\}$ is guaranteed. Thus $U_i$ obtains $Y'_{j,i}(x)$, for $1 \leq j \leq v$, from the messages. He can then compute $K(U_i)$ as follows, (referring to equation (6.2))

$$\sum_{j \in \beta_p} Y'_{j,i}(x) \mod q = \sum_{j \in \beta_p} Y'_j(x, i) \mod q$$

$$= \sum_{j \in \beta_p} \left( \frac{G \Gamma}{v} + Y_j(x, i) \right) \mod q$$

$$= G \Gamma + \sum_{j \in \beta_p} Y_j(x, i) \mod q$$

$$= G \Gamma + F(x, i) \mod q$$

Clearly $t$ users that do not contain all users in a $t$-arbitrary access set cannot admit a new user $U_i$ since the new user will not receive sufficient information to compute $F'(x, i)$.

A new user needs $v$ polynomials of degree $t$ to compute $F'(x, y)$ requiring a total communication cost of $v(t + 2) \log_2 q$ bits. \qed
Following a join event, session $S_{s+1}$ has a user group $U_{s+1} = U_s \cup J_s$. Since new users in $J_s$ only obtain secret information $F_i(x)$, none of the new users are in $U^-$. In other words, $J_s \subseteq U_{s+1} \setminus U^-$. Using a similar argument in Theorem 6.5 we have the following.

**Theorem 6.12** The extended scheme satisfies Join Secrecy – For any session $S_s$ where $S_1 \leq S_s \leq S_M$, a collusion $C \subseteq \bigcup J_a$, $S_i \leq S_a \leq S_M$, $E_a = \text{Join}$, and $|C| \leq t$ cannot find the group keys $GK_{b}, S_1 \leq S_b < S_s$.

### 6.3.4 Adding New Access Sets

The access sets in $\Gamma^-$ are subsets of $U_0$ and defined during the system initialisation. In practice it will be desirable to have flexible and dynamic $\Gamma^-$, in the sense that new $t$-arbitrary access sets can be appended to $\Gamma^-$ during system lifetime.

Let $\tilde{\Gamma}^-$ be a collection of new access sets with cardinalities at most $t$ defined over new users in $J_s$ and existing users in $U_s$. After adding $\tilde{\Gamma}^-$ to $\Gamma^-$, the new collection will be $\Gamma^- \cup \tilde{\Gamma}^-$ and it is assumed to be minimal. Let $\tilde{U}^- = \bigcup_{A \in \tilde{\Gamma}^-} A$ and recall $U^- = \bigcup_{A \in \Gamma^-} A$. Note that $U^- \subseteq U_0 \subseteq U_s$. We need to give secret information $\alpha_i$ to users $U_i \in \tilde{U}^- \setminus U^-$. More specifically, the users in $(\tilde{U}^- \setminus U^-) \cap J_s$ are new users for the group $U_s$ so they also need secret information $F_i(x)$ as well as $\alpha_i$. The users in $(\tilde{U}^- \setminus U^-) \setminus J_s$ are existing users in $U_s$, so only require secret information $\alpha_i$.

We describe two methods for users in a $t$-arbitrary access set $A \in \Gamma^-$ to perform the extension without the GC's assistance, while maintaining security of the system.

**Pre-Defined New Access Sets**

Assume the new $t$-arbitrary access sets are known at the design time of the system. The idea is to reserve some ghost users $\hat{U}$ and ghost $t$-arbitrary access sets $\hat{\Gamma}^-$ that will be used during system lifetime. During the system initialisation (see Section 6.3.1) the GC will do the following.

1. In step 2, also specify the set $\hat{U}$ and the access structure $\hat{\Gamma}^-$, where the access sets in $\hat{\Gamma}^-$ are defined on $U_0$ and $\hat{U}$. Without loss of generality, we assume that $\hat{U} = (\bigcup_{A \in \hat{\Gamma}^-} A) \setminus U^-$. We furthermore assume that the access structure $\Gamma^- \cup \hat{\Gamma}^-$ is minimal and identifiers of the ghost users in $\hat{U}$ are unique in the system.

2. In step 3, construct and publish the cumulative array $C(\Gamma^- \cup \hat{\Gamma}^-)$ instead of $C(\Gamma^-)$. 


In this approach, the new \( t \)-arbitrary access sets are constrained to \( \tilde{\Gamma}^- \subseteq \hat{\Gamma}^- \), new users are \((\tilde{U}^- \setminus \mathcal{U}^-) \cap \mathcal{J}_s \subseteq \tilde{U})\) and existing users are \((\tilde{U}^- \setminus \mathcal{U}^-) \setminus \mathcal{J}_s \subseteq \mathcal{U}_0\). The privileged set for this operation is an access set \( \mathcal{P}_s \in \Gamma^- \).

1. For \( \mathcal{J}_s \neq \emptyset \), sponsors \( U_z \in \mathcal{P}_s \) invoke the join protocol in Section 6.3.3 to give secret information \( F'_i(x) \) to the new users \( U_i \in \mathcal{J}_s \). Note that identifiers of the new users in \((\tilde{U}^- \setminus \mathcal{U}^-) \cap \mathcal{J}_s \) are chosen from the identifiers of the ghost users in \( \tilde{U} \). All secret information is updated after this step.

2. **Transmission:** To give secret information \( \alpha'_i \) to each user \( U_i \in \tilde{U}^- \setminus \mathcal{U}^- \), each sponsor \( U_z \in \mathcal{P}_s \) constructs the message
   \[
   \mathcal{M} = \{ Y'_j(x,y) \mid j : j \in \beta_z \cap \beta_i \},
   \]
   encrypts \( \mathcal{M} \) with the key \( F'_z(i) \) using a symmetric key cipher and sends the encrypted message to user \( U_i \). Note that if more than one sponsor holds \( Y'_j(x,y) \), it is enough for one of them (for example, the sponsor with the lowest \( z \)) to send a copy of \( Y'_j(x,y) \).

3. **Calculation:** Each user \( U_i \in \tilde{U}^- \setminus \mathcal{U}^- \) decrypts the encrypted messages using keys \( F'_i(z) \), for all \( U_z \in \mathcal{P}_s \) (note that the symmetric property of \( F'(x,y) \) gives \( F'_i(i) = F'_i(z) \)), and keeps the messages as the secret information. Recalling \( \beta_p = \bigcup_{U_z \in \mathcal{P}_s} \beta_z \),
   \[
   \alpha'_i = \{ Y'_j(x,y) : j \in \beta_p \cap \beta_i \}. \]
   Since \( \beta_p = \{1, \ldots, v\} \), we can assure that each user \( U_i \in \tilde{U}^- \setminus \mathcal{U}^- \) obtains correct \( \alpha'_i = \{ Y'_j(x,y) : j \in \beta_i \} \).

**Post-Defined New Access Sets**

This method allows any new \( t \)-arbitrary access sets to be added during the system lifetime and gives a flexible extension for the access structure. We use the idea of redistribution schemes [25] to deliver secret information \( \alpha_i \). With this method, \( \alpha_i \) is delivered to users \( U_i \in \tilde{U}^- \cup \mathcal{U}^- \) (not only users in \( \tilde{U}^- \setminus \mathcal{U}^- \)). Adding \( \hat{\Gamma}^- \) to \( \Gamma^- \) is performed by an access set \( \mathcal{P}_s \in \Gamma^- \).

1. Same as step 1 of the pre-defined new access sets approach.

2. Sponsors \( U_z \in \mathcal{P}_s \) construct and publish a new \( \tilde{w} \times \tilde{v} \) cumulative array \( C(\Gamma^- \cup \hat{\Gamma}^-) = [c_{ie}] \) where \( \tilde{w} \) and \( \tilde{v} \) are the cardinalities of the sets \( \mathcal{U}^- \cup \tilde{U}^- \) and \( \hat{\mathcal{F}} \),
respectively. Each user $U_i \in \mathcal{U}^- \cup \hat{\mathcal{U}}^-$ corresponds to a set $\hat{\mathcal{B}}_i$ consisting of all columns indexed by $e$ where $c_{ie} = 1$, i.e., $\hat{\mathcal{B}}_i = \{e : c_{ie} = 1\}$.

3. Each sponsor $U_z \in \mathcal{P}_s$ randomly generates $\hat{v} - 1$ symmetric polynomials of degree $t$ in $x$ and $y$ over $\mathbb{Z}_q$, $Y_1^{(j)}(x,y), Y_2^{(j)}(x,y), \ldots, Y_{\hat{v}-1}^{(j)}(x,y)$, and calculates

$$Y_e^{(j)}(x,y) = Y_j'(x,y) - \sum_{c=1}^{\hat{v}-1} Y_c^{(j)}(x,y) \mod q,$$

for all $j \in \beta_z$. Note that $Y_j'(x,y) \in \alpha_z'$. If $\beta_{z_1} \cap \beta_{z_2} \neq \emptyset$, for any $U_{z_1}, U_{z_2} \in \mathcal{P}_s$, $U_{z_1} \neq U_{z_2}$, there will be more than one version of $Y_e^{(j)}(x,y)$ for all $j \in \beta_{z_1} \cap \beta_{z_2}$. It is required that there is only one version of $Y_e^{(j)}(x,y)$. Again, this can be achieved by a policy where the sponsor with the lowest $z$ computes it.

4. Recall that $\beta_P = \bigcup_{U_z \in \mathcal{P}_s} \beta_z = \{1, \ldots, v\}$. Observe that the column $e$ of the new cumulative array relates to the polynomial

$$W_e(x,y) = \sum_{j \in \beta_P} Y_e^{(j)}(x,y) \mod q,$$

for $e = 1, 2, \ldots, \hat{v}$. Thus, the set $\hat{\mathcal{F}}$ of the new cumulative array $C(\Gamma^- \cup \hat{\Gamma}^-)$ is $\hat{\mathcal{F}} = \{W_1(x,y), W_2(x,y), \ldots, W_{\hat{v}}(x,y)\}$. Each user $U_i \in \mathcal{U}^- \cup \hat{\mathcal{U}}^-$ needs to obtain secret information $\hat{\alpha}_i = \{W_e(x,y) : e \in \hat{\mathcal{B}}_i\}$.

5. **Transmission:** To give secret information $\hat{\alpha}_i$ to each user $U_i \in \hat{\mathcal{U}}^- \cup \hat{\mathcal{U}}^-$, each sponsor $U_z \in \mathcal{P}_s$ constructs the message

$$\mathcal{M} = \{Y_e^{(j)}(x,y) \parallel j \parallel e : j \in \beta_z, e \in \hat{\mathcal{B}}_i\},$$

encrypts $\mathcal{M}$ with key $F_z'(i)$ using a symmetric key cipher, and sends the encrypted message to user $U_i$. The message is constructed with respect to the policy in step 3.

6. **Calculation:** Each user $U_i \in \hat{\mathcal{U}}^- \cup \hat{\mathcal{U}}^-$ decrypts the encrypted messages using keys $F_i'(z)$, for all $U_z \in \mathcal{P}_s$, and formulates the secret information

$$\hat{\alpha}_i = \left\{\sum_{j \in \beta_P} Y_e^{(j)}(x,y) \mod q : e \in \hat{\mathcal{B}}_i\right\}.$$

It is clear that each user $U_i \in \hat{\mathcal{U}}^- \cup \hat{\mathcal{U}}^-$ obtains correct $\hat{\alpha}_i = \{W_e(x,y) : e \in \hat{\mathcal{B}}_i\}$. 
6.4 A Variant of the Extended Scheme

We propose a variant of the extended scheme with less user storage and lower transmission costs in user join. The costs are significantly reduced for large $h$, say $h \approx t$, where $h = \min\{|\mathcal{A}| : \mathcal{A} \in \Gamma^\sim\}$.

6.4.1 System Initialisation

In the join protocol in Section 6.2.3, collaborations of $t + 1$ users admit new users using their $F_i(x)$. If the collaboration has less than $t + 1$ users, no join can be performed. In this variant we give extra information to members of the access structure, to reduce the number of collaborators they need for user admission. Let $\bar{U}_0 = \{i : U_i \in U_0\}$. The group controller (GC) distributes $t + 1 - h$ polynomials $F(x, d), d \not\in \bar{U}_0$, among members of $\Gamma^\sim$, such that each $t$-arbitrary access set can construct the $F(x, d)$. In detail,

1. Same as steps 1, 2, and 3 of the system initialisation in Section 6.3.1.

2. Chooses and publishes a set $\mathcal{D}$ of $t + 1 - h$ distinct elements from $\mathbb{Z}_q^*$ such that $\mathcal{D} \cap \bar{U}_0 = \emptyset$.

3. For each $d \in \mathcal{D}$, randomly chooses $v - 1$ polynomials of degree $t$ over $\mathbb{Z}_q$, $Y_1^{(d)}(x), Y_2^{(d)}(x), \ldots, Y_{v-1}^{(d)}(x)$, and calculates

$$Y_v^{(d)}(x) = F(x, d) - \sum_{j=1}^{v-1} Y_j^{(d)}(x) \mod q. \quad (6.3)$$

Let $\mathcal{Y}_j = \{Y_j^{(d)}(x) : d \in \mathcal{D}\}$, for all $1 \leq j \leq v$. All polynomials here are secret.

4. Associates column $j$ of the cumulative array with the set $\mathcal{Y}_j$, for $1 \leq j \leq v$.

Observe that the set $\mathcal{F}$ of the cumulative array $C(\Gamma^\sim)$ is $\mathcal{F} = \{\mathcal{Y}_1, \ldots, \mathcal{Y}_v\}$.

5. Gives elements of $\mathcal{F}$ to each user $U_i \in \mathcal{U}^\sim$ if and only if $c_{ij} = 1$ through a secure unicast channel. Thus, the set $\alpha_i = \{\mathcal{Y}_j : j \in \beta_i\}$.

Observe that the set $\alpha_i$ consists of at most $v(t + 1 - h)$ polynomials of degree $t$.

**Theorem 6.13** In the variant scheme, (i) storage of a group controller is $(v(t + 1 - h)) + \frac{(t+2)}{2}(t + 1) \log_2 q$ bits (ii) storage of a user not in $\mathcal{U}^\sim$ is $(t + 1) \log_2 q$ bits and (iii) storage of a user in $\mathcal{U}^\sim$ is at most $(1 + v(t + 1 - h))(t + 1) \log_2 q$ bits of secret information. The security property in Theorem 6.10 and its proof apply to the variant scheme also, using equation (6.3) rather than (6.2).
6.4.2 Subgroup Event

The subgroup protocol here differs from the subgroup protocol in Section 6.3.2 only in the chosen set $\Theta$, which in this case also has to satisfy $\Theta \cap \mathcal{D} = \emptyset$.

6.4.3 Join Event

While at least $t + 1$ users can admit a new user following the join protocol in Section 6.2.3, we show here how sponsors in a privileged set ($t$-arbitrary access set) $\mathcal{P}_s \in \Gamma^-$ admit a set $\mathcal{J}_s \subseteq \mathcal{N} \setminus \mathcal{U}_s$ of new users. Recall that the identity sets $\tilde{\mathcal{U}}_s = \{i : U_i \in \mathcal{U}_s\}$ and $\bar{\mathcal{P}}_s = \{z : U_z \in \mathcal{P}_s\}$, and $\mathcal{P}_s \subseteq \mathcal{U}^- \subseteq \mathcal{U}_s$.

1. **Self-refreshing:** Same as step 1 of the join protocol in Section 6.3.3. In this case, users $U_i \in \mathcal{U}^-$ update their secret information $\alpha'_i$ as follows.

   \[
   \alpha'_i = \{Y'_j : j \in \beta_i\}, \text{ where } Y'_j = \{Y'^{(d)}(x) = u + Y^{(d)}(x) \mod q : d \in \mathcal{D}\}
   \]

   and $u = \frac{GK_v}{v} \mod q$ (assuming $q > v$). It follows that users $U_i \in \mathcal{U}^-$ has $\mathcal{K}(U_i) = \{F'_i(x), \alpha'_i\}$ and users $U_i \in \mathcal{U}_s \setminus \mathcal{U}^-$ has $\mathcal{K}(U_i) = \{F'_i(x)\}$ as their secret information.

2. **Transmission:** Sponsors $U_z \in \mathcal{P}_s$ do the following.

   (a) Same as step 2a of the join protocol in Section 6.3.3.

   (b) Use their secret information $\mathcal{K}(U_z) = \{F'_z(x), \alpha'_z\}$ to do the following for each new user $U_i \in \mathcal{J}_s$.

   i. Compute $f'_{(z,i)} = F'_z(i)$.

   ii. Choose a set $\mathcal{D}_s$ of $t + 1 - |\mathcal{P}_s|$ elements from the set $\mathcal{D}$, $\mathcal{D}_s \subseteq \mathcal{D}$ (note that $|\mathcal{P}_s| \geq h$ so $|\mathcal{D}_s| \leq |\mathcal{D}|$), and for all $j \in \beta_z$, compute $y'_{(j,i)}^{(d)} = Y'^{(d)}(i)$, for all $d \in \mathcal{D}_s$. Note that $Y'^{(d)}(x) \in \mathcal{Y}'_j \in \alpha'_z$. All sponsors $U_z \in \mathcal{P}_s$ choose the same $\mathcal{D}_s$.

   iii. Individually send join messages

   \[
   \mathcal{M} = \{f'_{(z,i)} \parallel z\} \cup \{y'_{(j,i)}^{(d)} \parallel j \parallel d : j \in \beta_z, d \in \mathcal{D}_s\}
   \]

   to $U_i$ over a secure unicast channel. When $\beta_{z_1} \cap \beta_{z_2} \neq \emptyset$, $U_{z_1}, U_{z_2} \in \mathcal{P}_s, U_{z_1} \neq U_{z_2}$, only one sponsor sends $y'_{(j,i)}^{(d)}$. 

3. Calculation: Each new user $U_i \in \mathcal{J}_s$ computes his secret information $\mathcal{K}(U_i)$ using the join messages given by all sponsors as follows, for $\beta_P = \bigcup_{U_s \in \mathcal{P}_s} \beta_z$.

$$\mathcal{K}(U_i) = \left\{ \sum_{z \in \mathcal{P}_s} f'_{(z,i)} \Psi(\tilde{\mathcal{P}}_s \cup \mathcal{D}_s, z) + \sum_{d \in \mathcal{D}_s} \sum_{j \in \beta_P} y'_{(j,i)}^{(d)} \Psi(\tilde{\mathcal{P}}_s \cup \mathcal{D}_s, d) \mod q \right\}.$$

A summary of the variant scheme’s join protocol is shown in Table 6.6.

Table 6.6: The join protocol of variant scheme

<table>
<thead>
<tr>
<th>Sponsors $U_s \in \mathcal{P}_s$</th>
<th>New users $U_i \in \mathcal{J}_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(F'_s(x), \alpha'_s, \mathcal{D})$</td>
<td>$(q)$</td>
</tr>
</tbody>
</table>
| $f'_{(z,i)} = F'_z(i)$  
$\mathcal{D}_s \subseteq \mathcal{D}$  
$|\mathcal{D}_s| = t + 1 - |\mathcal{P}_s|$  
$y'_{(j,i)}^{(d)} = Y'_j^{(d)}(i), \forall j \in \beta_z, \forall d \in \mathcal{D}_s$  
$a_1 = \{f'_{(z,i)} \parallel z\}$  
$a_2 = \{y'_{(j,i)}^{(d)} \parallel j \parallel d : j \in \beta_z, d \in \mathcal{D}_s\}$  
$\mathcal{M} = \{a_1 \cup a_2\}$ | $b_1 = \sum_{z \in \tilde{\mathcal{P}}_s} f'_{(z,i)} \Psi(\tilde{\mathcal{P}}_s \cup \mathcal{D}_s, z) \mod q$  
$b_2 = \sum_{d \in \mathcal{D}_s} \sum_{j \in \beta_P} y'_{(j,i)}^{(d)} \Psi(\tilde{\mathcal{P}}_s \cup \mathcal{D}_s, d) \mod q$  
$F'_i(x) = b_1 + b_2 \mod q$ |

**Theorem 6.14** In the variant scheme, users in a $t$-arbitrary access set may perform join events for a set $\mathcal{J}_s$ of any number of new users, while any $t$ or less users that do not contain all users in a $t$-arbitrary access set cannot do so. Joining a new user requires transmission of at most $(2h + 3w(t + 1 - h)) \log_2 q$ bits over secure unicast channels. Also, the argument in Theorem 6.5 guarantees join secrecy of the variant scheme.

**Proof:** Observe that the system secret $F(x, y)$ has been updated to $F'(x, y) = GK_s + F(x, y) \mod q$ following the self-refreshing step. To be in $\mathcal{U}_s$, a new user $U_i \in \mathcal{J}_s$ has to obtain secret information which is the evaluation of the system secret $F'(x, y)$ at $y = i$. From the messages, $U_i$ gets $|\tilde{\mathcal{P}}_s|$ pieces of information, that is, $f'_{(z,i)} = F'_z(i) = F'(i, z)$ for all $z \in \tilde{\mathcal{P}}_s$ (in general $|\tilde{\mathcal{P}}_s| \leq t$). From the property of cumulative array $C(\Gamma^-)$, $\beta_P = \{1, \cdots, v\}$ holds and so $U_i$ obtains $y'_{(j,i)}^{(d)}$, for $1 \leq j \leq v$, for all $d \in \mathcal{D}_s$. From those, $U_i$ may form $|\mathcal{D}_s| = t + 1 - |\mathcal{P}_s|$ pieces of information, that is, $F'(i, d)$, for all
6.5. Further Discussion and Conclusions

\(d \in \mathcal{D}_s\), as follows, referring to equation (6.3).

\[
\sum_{j \in \beta_{\mathcal{P}}'} y'_{j, i}^{(d)} \mod q = \sum_{j \in \beta_{\mathcal{P}}'} Y_j^{(d)}(i) \mod q \\
= \sum_{j \in \beta_{\mathcal{P}}'} \left(\frac{G\mathcal{K}_s}{v} + Y_j^{(d)}(i)\right) \mod q \\
= G\mathcal{K}_s + \sum_{j \in \beta_{\mathcal{P}}'} Y_j^{(d)}(i) \mod q \\
= G\mathcal{K}_s + F(i, d) \mod q \\
= F'(i, d).
\]

Interpolation of all \(|\mathcal{P}_s| + |\mathcal{D}_s| = |\mathcal{P}_s| + t + 1 - |\mathcal{P}_s| = t + 1\) pieces of information (\(|\mathcal{P}_s| = |\mathcal{P}_s|\)) gives \(\mathcal{K}(U_i) = \{F'(x, i) = F'_i(x)\}\). It is obvious that \(t\) users that do not contain all users in a \(t\)-arbitrary access set give insufficient information for a new user \(U_i\) to compute \(F'(x, i)\).

Observe that the sponsors transmit a total of \((2|\mathcal{P}_s| + 3v(t + 1 - |\mathcal{P}_s|)) \log_2 q\) bits and the worst case of transmission is when \(|\mathcal{P}_s| = h\). See Theorem 6.5 for the security proof.

\[\Box\]

Discussion

We can add new \(t\)-arbitrary access sets into \(\Gamma^-\) of the extended scheme by applying the two methods described in Section 6.3.4. Note that (i) using the method of pre-defined new access sets, we require the value \(h = \min\{\mathcal{A} : \mathcal{A} \in \Gamma^- \cup \Gamma^-\}\), and (ii) using the method of post-defined new access sets, we require \(|\mathcal{A}| \geq h\), for all \(\mathcal{A} \in \Gamma^-\), where \(h\) is the minimum cardinality of access sets in \(\Gamma^-\). This is because if \(|\mathcal{A}| < h\), the sponsors in \(\mathcal{P}_s = \mathcal{A}\) cannot give enough information for a new user to compute his secret information.

6.5 Further Discussion and Conclusions

In many cases, there is a need to change a group key without revoking or adding users. An obvious example is the self-refreshing step described earlier. It is also possible that a group user accidentally loses the group key and wants to update the group key without changing group memberships. To change the group key, the user can follow the subgroup protocol with the set \(\mathcal{G}_s = \mathcal{U}_s\) (since no one is being revoked), or use the sponsorship protocol with the set \(\mathcal{J}_s = \emptyset\) (since no one is being sponsored). However,
the latter needs more computation for the group users to update their secret information.

We have introduced a novel scenario for dynamic group key distribution schemes whereby a sponsorship operation can be performed by any user, and a full user join operation can be performed by users in an access set of an access structure. We have given applications of such a decentralised system and presented two dynamic group key distribution schemes for that system. One has an access structure consisting of any combination of \( t + 1 \) or more users, and the other extends the first to have some defined access sets of size at most \( t \). We have also given a variant of the extended scheme with better efficiency. All schemes allow subgroup establishment of size at least \( |U| - t \) users. Our constructions use symmetric polynomials and cumulative arrays, and they are secure against a collusion of at most \( t \) users assuming any \( t \)-arbitrary access set is not a subset of the collusion. We have shown that the schemes are consistent and secure, and evaluated their efficiency. Moreover, we have shown that traitor tracing is possible and further discussed several approaches for user eviction.

We note that sponsorship and full join are also useful in other applications such as hierarchical-groups where users in different levels have different rights. Constructing group key distribution schemes that allow an arbitrary structure of group users to perform both user revocation and user join is an interesting problem for further research.
A Group Key Distribution Scheme with Decentralised User Join – A KDP Approach

7.1 Introduction

We address the problems of enabling any group user to securely establish a session key among users of a designated subgroup, and allowing several group users to securely enroll a new user to the group. We present a construction that caters for the scenario using key distribution patterns (KDP). The basic idea is to design the structure of keys in such a way that each user holds a different subset of keys, such that each pair of users share at least a key that is not known to any other users – a similar method has been used in the single group controller scheme [48].

7.1.1 A Summary of Our Contribution

We give a dynamic group key distribution scheme containing two protocols to form subgroups, called the OR protocol and the AND protocol, whereby a user chooses a session key and broadcasts encrypted versions of the key which can only be decrypted by members of the target subgroup (we assume a secure encryption tool is used). Both protocols are efficient: they require storage of $O(\log n)$ keys for both the group controller (GC) and the user, and both the OR protocol for large subgroups (i.e., small number of revoked users) and the AND protocol for small subgroups (i.e., large number of revoked users) achieve the communication cost of $O(\log n)$ keys in a single broadcast. Therefore, the key storages of the GC and the user, and the communication costs of the scheme can simultaneously achieve $O(\log n)$ keys. Furthermore, because of the identical key structure in both protocols and the property that they are most efficient for different ranges of subgroup sizes, they can be seen as complementary to each other. This provides high flexibility in practice. We also show how to improve the communication efficiency of the OR protocol by using an erasure code. Our system is
inspired by [48], where an erasure code is used to improve communication efficiency in secure broadcast systems.

We also consider user join in this scheme. We show that decentralised user join is possible and give an algorithm that allows a new user to obtain his private key information from existing members of the group. Our user admission method can also be applied to the KDP based revocation scheme proposed in [61]. The proposed scheme provides security against collusion of up to \( t \) users and has stateless key storage in subgroup and join operations.

The schemes proposed in this chapter and Chapter 6 use similar settings. In Table 7.1 we compare the efficiency of the subgroup protocols and the join protocols proposed in the two chapters. We use the key length as the unit to measure the user storage and communication costs. The computation cost of user join is measured in terms of the number of decryptions. The computation cost of user revocation is measured in terms of the number of decryptions for the scheme in this chapter, and in terms of the number of exponentiations for the scheme in Chapter 6. This is because the scheme in this chapter is symmetric-key based while the scheme in Chapter 6 is public-key based. We recall that the OR and AND protocols in this chapter can be used to revoke small number of users (\( \beta \approx 0 \)) and large number of users (\( \beta \approx n \)), respectively, and both protocols bound the number of new users to \( n (\alpha + \gamma \leq n) \), where \( \gamma \) is the number of existing users). The scheme in Chapter 6 can be used to revoke any subset of users of size at most \( t \) (\( \beta \leq t \)), and the system supports an arbitrary number of new users.

The subgroup and join protocols of the proposed scheme are identical to those in Table 5.2 (Chapter 5) and Table 6.1 (Chapter 6), respectively.

Organisation of this chapter. Section 7.2 gives a basic construction for the OR protocol and the AND protocol using key distribution patterns. It also shows that the communication cost of the OR scheme can be improved through the application of erasure codes. It also describes a slight extension of the scheme to support user admission, and demonstrates how a new joined user obtains his personal key set from some existing users. Finally Section 7.3 concludes the chapter.

### 7.2 A KDP Construction

The notion of key distribution pattern is central to the construction. The definition of key distribution pattern is given in Chapter 1.

#### 7.2.1 System Initialisation

The group controller (GC) does the following.

1. Designs a \((\mathcal{X}, \mathcal{B}) = (m, N, t)\)-KDP with \(N = |\mathcal{U}_0|\) and chooses a large number \(q\). The \((m, N, t)\)-KDP is public.

2. Randomly generates a set of \(m\) keys \(\mathcal{K}(\mathcal{X}) = \{K_1, K_2, \ldots, K_m\} \subseteq \mathbb{Z}_q\), \(\mathcal{K}(\mathcal{X}) \neq 0\), corresponding to the set \(\mathcal{X} = \{x_1, x_2, \ldots, x_m\}\). All keys in \(\mathcal{K}(\mathcal{X})\) are kept secret.

3. Securely sends a subset \(\mathcal{K}(U_i) = \{K_j : x_j \in \mathcal{B}_i\} \subseteq \mathcal{K}(\mathcal{X})\) to each user \(U_i \in \mathcal{U}_0\) over a secure unicast channel. All keys in \(\mathcal{K}(U_i)\) are secret information for user \(U_i\).

\(^1\)The number \(q\) should be large enough such that the symmetric encryption algorithm \(E()\) used in the next section is secure against brute force attack.
It is obvious that the number of keys for the GC is $|\mathcal{K}(\mathcal{X})| = m$ and for each user is $|\mathcal{K}(U_i)| < m$. Therefore, the scheme requires a group controller to store $m \log_2 q$ bits and a user to store less than $m \log_2 q$ bits of secret information.

### 7.2.2 Subgroup Event

We present two subgroup protocols, called the OR protocol and the AND protocol, using key distribution patterns. (AND and OR protocols are also used in [55], however, our application is different as we consider decentralised model of group communication.) Both protocols allow any user in a group $U_s$ to form a subgroup $\mathcal{G}_s \subseteq U_s$ and establish a group key $GK_s$ for the subgroup. That is, the privileged set contains a group initiator $U_z$, $P_s = \{U_z\}$, where $U_z$ can be any user in $\mathcal{G}_s$. Without loss of generality, we assume that $N = |U_s|$.

**OR Protocol**

This protocol can be used to form a subgroup with $|\mathcal{G}_s| \geq |U_s| - t$.

1. **Transmission:** A group initiator $U_z \in \mathcal{G}_s$ does the following.
   
   (a) Randomly chooses a group key $GK_s \in \mathbb{Z}_q$, $GK_s \neq 0$, and encrypts it with all his keys except those keys incident to $\mathcal{K}(U_i)$, for all $U_i \in U_s \setminus \mathcal{G}_s$. That is, encrypts $GK_s$ with keys $K_j$, for all $x_j \in B_z, x_j \notin \bigcup_{U_i \in U_s \setminus \mathcal{G}_s} B_i$.
   
   (b) Broadcasts subgroup message

   $$\mathcal{M} = \{E_{K_j}(GK_s) \parallel x_j \in B_z \setminus \bigcup_{U_i \in U_s \setminus \mathcal{G}_s} B_i\}$$

   to all the users. $E_{K_j}(\cdot)$ is a symmetric key encryption algorithm with key $K_j$.

2. **Calculation:** Each user $U_i \in \mathcal{G}_s$ uses one of his keys $K_j$, where $x_j \in B_i \cap (B_z \setminus \bigcup_{U_i \in U_s \setminus \mathcal{G}_s} B_i)$, to decrypt $E_{K_j}(GK_s)$ using the corresponding decryption algorithm $D_{K_j}(\cdot)$ and obtain the group key $GK_s$.

A summary of the OR protocol is shown in Table 7.2.

**Theorem 7.1** The OR protocol provides Subgroup Secrecy – For any session $S_s$ where $S_1 \leq S_s \leq S_M$ and $E_s = \text{Subgroup}$, a collusion $\mathcal{C} \subseteq U_s \setminus \mathcal{G}_s, |\mathcal{C}| \leq t$ cannot find the group key $GK_s$.  

Table 7.2: The OR protocol

<table>
<thead>
<tr>
<th>$A$ group initiator $U_z \in \mathcal{G}_s$</th>
<th>$\langle \mathcal{X}, \mathcal{B} \rangle, K(U_z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_{K_s} \in \mathbb{Z}<em>q, G</em>{K_s} \neq 0$</td>
<td>$\phi = B_z \setminus (\bigcup_{i \in U_s \setminus G_s} B_i)$</td>
</tr>
<tr>
<td>$M = {E_{K_j}(G_{K_s}) \mid x_j \in \phi}$</td>
<td>$\theta = B_\nu \cap (B_z \setminus (\bigcup_{i \in U_s \setminus G_s} B_i))$</td>
</tr>
<tr>
<td></td>
<td>choose a key $K_j : x_j \in \theta$</td>
</tr>
<tr>
<td></td>
<td>$G_{K_s} = D_{K_j}(E_{K_j}(G_{K_s}))$</td>
</tr>
</tbody>
</table>

**Proof:** The group key $G_{K_s}$ is encrypted using keys $K_j$, for all $x_j \in B_z \setminus (\bigcup_{i \in U_s \setminus G_s} B_i)$. It is clear that users in $U_s \setminus G_s$, even if they collude, do not possess the keys used to encrypt $G_{K_s}$ in the broadcast message $M$, and so they are unable to obtain the group key.

**Theorem 7.2** The OR protocol allows any group member to form a subgroup with at least $N - t$ users from a group of $N$ users. It requires a broadcast message of less than $2m \log_2 q$ bits and requires an authorised user to compute a decryption operation.

**Proof:** From the definition of KDP, every user $U_\nu \in \mathcal{G}_s$ has at least one key $K_j$ where $x_j \in B_z \setminus (\bigcup_{i \in U_s \setminus G_s} B_i)$ so he can decrypt $E_{K_j}(G_{K_s})$ to obtain the group key. The above property holds if $|U_s \setminus G_s| \leq t$ requiring $|G_s| \geq |U_s| - t$ users.

Since a group initiator $U_z$ has less than $m$ keys, the broadcast message $M$ contains less than $m$ encrypted group keys. Assuming $x_j$ is $\log_2 q$ bits long and the output of the symmetric encryption algorithm is also $\log_2 q$ bits long, the size of $M$ is less than $2m \log_2 q$ bits. A user in $\mathcal{G}_s$ only needs to decrypt an encrypted group key.

**Example 7.1** Let $\mathcal{X} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ and $\mathcal{B} = \{B_1, B_2, B_3, B_4, B_5, B_6, B_7, B_8, B_9, B_{10}, B_{11}, B_{12}\}$ defined as follows.

<table>
<thead>
<tr>
<th>$B_1$ = {4, 5, 6, 7, 8, 9}</th>
<th>$B_2$ = {2, 3, 5, 6, 8, 9}</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_3$ = {2, 3, 4, 6, 7, 8}</td>
<td>$B_4$ = {2, 3, 4, 5, 7, 9}</td>
</tr>
<tr>
<td>$B_5$ = {1, 2, 3, 7, 8, 9}</td>
<td>$B_6$ = {1, 3, 4, 6, 7, 9}</td>
</tr>
<tr>
<td>$B_7$ = {1, 3, 4, 5, 8, 9}</td>
<td>$B_8$ = {1, 3, 5, 6, 7, 8}</td>
</tr>
<tr>
<td>$B_9$ = {1, 2, 3, 4, 5, 6}</td>
<td>$B_{10}$ = {1, 2, 4, 5, 7, 8}</td>
</tr>
<tr>
<td>$B_{11}$ = {1, 2, 5, 6, 7, 9}</td>
<td>$B_{12}$ = {1, 2, 4, 6, 8, 9}</td>
</tr>
</tbody>
</table>

Then $(\mathcal{X}, \mathcal{B})$ is a $(9, 12, 1)$-KDP.
7.2. A KDP Construction

Assume that after initialisation, user (group initiator) $U_1$ wants to establish a common key with the rest of users except $U_3$. Since $B_1 \setminus B_3 = \{5, 9\}$, using the OR protocol, user $U_1$ encrypts the key $GK_s$ using $K_5$ and $K_9$, and broadcasts subgroup message $\mathcal{M} = \{E_{K_5}(GK_s), E_{K_9}(GK_s)\}$ to the group. It is easy to see that every user except $U_3$ can decrypt at least one encrypted $GK_s$ in the subgroup message using $K_5$ or $K_9$. On the other hand, user $U_3$ does not know $K_5$ and $K_9$, and so he can not decrypt the encrypted group key.

**AND Protocol**

Observe that the OR protocol can only be efficiently used to form large subgroups. We present a variant, called the AND protocol, which can be used to efficiently form small subgroups, that is $|G_s| \approx 0$.

1. **Transmission**: A group initiator $U_z \in G_s$ does the following.

   (a) Computes a key with each user $U_i \in G_s$ as follows.
   
   $K_{(z,i)} = \bigoplus K_j : x_j \in B_z \cap B_i$,
   
   where $\bigoplus$ denotes the Exclusive-OR operation (assuming that all the keys are strings of the same length).

   (b) Randomly chooses a group key $GK_s \in \mathbb{Z}_q$, $GK_s \neq 0$, and encrypts it with key $K_{(z,i)}$, for all $U_i \in G_s$.

   (c) Broadcasts subgroup message
   
   $\mathcal{M} = \{E_{K_{(z,i)}}(GK_s) \parallel i : U_i \in G_s\}$,
   
   where $E_{K_{(z,i)}}()$ is a symmetric key encryption algorithm with key $K_{(z,i)}$, to all the users.

2. **Calculation**: Each user $U_i \in G_s$ computes $K_{(z,i)}$ and decrypts $E_{K_{(z,i)}}(GK_s)$, using the corresponding decryption algorithm $D_{K_{(z,i)}}()$, to obtain the group key $GK_s$.

A summary of the AND protocol is shown in Table 7.3.

**Theorem 7.3** The AND protocol provides Subgroup Secrecy – For any session $S_s$ where $S_1 \leq S_s \leq S_M$ and $E_s = \text{Subgroup}$, a collusion $C \subseteq U_s \setminus G_s, |C| \leq t$ cannot find the group key $GK_s$. 
Proof: The group key $GK_s$ is encrypted using keys $K_{(z,i)}$, for all $U_i \in \mathcal{G}_s$. Let $C = \{U_{k_1}, U_{k_2}, \ldots, U_{k_8}\}$. $C$ succeed in obtaining the group key only if they can calculate $K_{(z,i)}$ for some $U_i \in \mathcal{G}_s$. This is not possible because indeed for each $U_i \in \mathcal{G}_s$, there exists at least an $x \in B_z \cap B_i$ but $x \notin B_{k_1} \cup B_{k_2} \cup \cdots \cup B_{k_8}$ by referring to the definition of KDP. It follows that $K_{(z,i)}$, for all $U_i \in \mathcal{G}_s$, is unknown to $C$, and so they cannot decrypt any encrypted group key in the broadcast message $M$. 

\[ \text{Theorem 7.4} \quad \text{The AND protocol allows any group member to form a subgroup with} \ \ d \ \ \text{users,} \quad 1 \leq d \leq N - 1, \quad \text{from a group of} \ N \ \ \text{users. It requires a broadcast message of} \ 2d \log_2 q \ \text{bits and requires an authorised user to perform a decryption operation.} \]

\[ \text{Proof:} \quad \text{It is straightforward to show that each user} \ U_i \ \text{in subgroup} \ \mathcal{G}_s \ \text{can compute the key} \ K_{(z,i)} \ \text{to retrieve the group key} \ GK_s. \ \text{To prove that} \ 1 \leq |\mathcal{G}_s| \leq |\mathcal{U}_s| \ (\mathcal{G}_s \ \text{can be any subset of} \ \mathcal{U}_s), \ \text{we need to show that indeed} \ B_i \cap B_j \neq B_i \cap B_{k_1} \ \text{for any} \ i, j, k_1 \in \{1, 2, \ldots, N\}. \ \text{From the definition of KDP, there exists at least an} \ x \in B_i \cap B_j \ \text{but} \ x \notin B_{k_1} \cup B_{k_2} \cup \cdots \cup B_{k_8} \ \text{for any} \ i, j, k_1, k_2, \ldots, k_8 \in \{1, 2, \ldots, N\}. \ \text{It follows that} \ x \notin B_{k_1} \cup B_{k_2} \cup \cdots \cup B_{k_8} \ \text{and so} \ B_i \cap B_j \neq B_i \cap B_{k_1}. \]

The number of encrypted group keys in the broadcast message $M$ depends on the subgroup size $d$. Assuming the identifier $i$ is $\log_2 q$ bits long and the output of symmetric encryption algorithm is also $\log_2 q$ bits long, the size of $M$ is $2d \log_2 q$ bits. A user in $\mathcal{G}_s$ only decrypts an encrypted group key. 

\[ \text{Example 7.2} \ \text{Let} \ (\mathcal{X}, \mathcal{B}) \ \text{be the} \ (9,12,1)-KDP \ \text{defined in example 7.1 for the OR protocol. Suppose, after initialisation, user} \ (\text{group initiator}) \ U_1 \ \text{wants to form a subgroup with} \ U_{11} \ \text{and} \ U_{12} \ \text{only. Since} \ B_{11} \cap B_{12} = \{5,6,7,9\} \ \text{and} \ B_1 \cap B_{12} = \{4,6,8,9\}, \ \text{using the AND protocol, user} \ U_1 \ \text{computes keys} \ K_{(1,11)} = K_5 \oplus K_6 \oplus K_7 \oplus K_9 \ \text{and} \ K_{(1,12)} = K_4 \oplus K_6 \oplus K_8 \oplus K_9, \ \text{encrypts the group key} \ GK_s \ \text{using the computed keys,} \]
and broadcasts the subgroup message $\mathcal{M} = \{ E_{K_{(1,11)}}(G_{K_s}), E_{K_{(1,12)}}(G_{K_s}) \}$. Observe that only user $U_{11}$ can compute $K_{(1,11)}$ and only user $U_{12}$ can compute $K_{(1,12)}$, and they will be able to decrypt one encrypted $G_{K_s}$ in the subgroup message. However, no non-subgroup user has enough components to compute $K_{(1,11)}$ or $K_{(1,12)}$, and so cannot decrypt the encrypted group key.

Efficiency of the System

Efficiency of the system is directly related to the parameters of the underlying key distribution pattern. For given $t$ and $N$, we expect $m$ to be as small as possible. Equivalently, for given $m$ and $t$, we expect $N$ to be as large as possible.

Constructing KDP with maximal $N$ for a given $m$ has been extensively studied in literature [29, 36, 61, 67, 72, 83]. Mitchell and Piper [61], and Gong and Wheeler [36] gave explicit constructions for $(m, N, t)$-KDP with $m = \mathcal{O}(N)$, which is more efficient than the trivial construction that gives each pair of users an individual key requiring $m = \mathcal{O}(N^2)$. Moreover, Dyer et al. [29] showed the existence of $(m, N, t)$-KDP with $m = \mathcal{O}(\log N)$. The above observation leads to the following corollary.

**Corollary 7.1** Let $\mathcal{U}_s$ be the user group in session $S_s$ and $\mathcal{G}_s \subseteq \mathcal{U}_s$ be the subgroup to be formed. We assume $|\mathcal{U}_s| = N$.

1. For $|\mathcal{G}_s| \geq |\mathcal{U}_s| - t$, there exists an OR protocol such that (i) the storage of a group controller is $\mathcal{O}(\log N \log_2 q)$ bits, (ii) the storage of each user is less than $\mathcal{O}(\log N \log_2 q)$ bits, and (iii) the broadcast (subgroup) message consists of $\mathcal{O}(\log N \log_2 q)$ bits.

2. For $1 \leq |\mathcal{G}_s| \leq |\mathcal{U}_s| - 1$, there exists an AND protocol such that (i) the storage of a group controller is $\mathcal{O}(\log N \log_2 q)$ bits, (ii) the storage of each user is less than $\mathcal{O}(\log N \log_2 q)$ bits, and (iii) the broadcast (subgroup) message consists of $|\mathcal{G}_s| \log_2 q$ bits.

### 7.2.3 Improving Communication Efficiency

We show how to improve communication efficiency (broadcast size) of the basic OR protocol using erasure codes. The basic idea is to first encode the group key with an erasure code, then apply the basic OR protocol (slightly modified) to the codeword. Kumar et al. [48] proposed a group key establishment method using the centralised model (a fixed trusted authority is required to perform the membership event) that
employs erasure codes and cover-free families (CFF). Our method is inspired by their work but for a decentralised model and using KDP. See Chapter 1 for a brief description of erasure codes.

In order to apply the erasure codes to improve the communication cost, we extend the definition of KDP to $\alpha$-KDP.

**Definition 7.1** Let $\mathcal{X} = \{x_1, \cdots, x_m\}$ be a set and $\mathcal{B} = \{\mathcal{B}_1, \cdots, \mathcal{B}_N\}$ be a family of subsets of $\mathcal{X}$. The pair $(\mathcal{X}, \mathcal{B})$ is called an $(m, N, t)$-$\alpha$-key distribution pattern ($(m, N, t)$-$\alpha$-KDP) if

$$|(\mathcal{B}_i \cap \mathcal{B}_j) \setminus (\bigcup_{e=1}^{t} \mathcal{B}_{ke})| \geq \alpha$$

for any $(t + 2)$-subset $\{i, j, k_1, \cdots, k_t\}$ of $\{1, 2, \cdots, N\}$.

For $\alpha = 1$, an $(m, N, t)$-$\alpha$-KDP is the same as an $(m, N, t)$-KDP. It is easy to see that $\alpha$-KDP with $\alpha = 2$ can be constructed by concatenating multiple KDPs. Also, an $(m, N, t)$-$\alpha$-KDP is an $(m, N, t - 1)$-$\alpha'$-KDP with $\alpha' \geq \alpha + 1$. We note that most techniques for constructing KDPs can be generalised to $\alpha$-KDP in a straightforward manner. It would be interesting to look for more sophisticated solutions.

Now the basic OR protocol can be modified as follows. Assume that keys of the users are elements of a finite field $\mathbb{F}_q$. Let $(\mathcal{X}, \mathcal{B})$ be an $(m, N, t)$-$\alpha$-KDP and let the keys for users be allocated as in the system initialisation. Our aim is to modify the broadcast (subgroup) message in the basic OR protocol using erasure codes. For simplicity, we assume that $|\mathcal{B}_i| = \ell$ ($|\mathcal{K}(U_i)| = \ell$), for all $\mathcal{B}_i \in \mathcal{B}$, and the group key $GK_s$ is an element in $\mathbb{F}_q^\ell$.

1. **Transmission:** A group initiator $U_z \in G_s$ does the following.

   (a) Divides the group key $GK_s$ into $w$ pieces $GK_s = (GK_{s1}, GK_{s2}, \cdots, GK_{sw})$, then encodes $GK_s$ using an $[\ell, w, \alpha]_q$ erasure code $(C, D)$ to obtain a code-word $C(GK_s) = (c_1, c_2, \cdots, c_{\ell})$.

   (b) Uses all his keys, except those keys incident to $\mathcal{K}(U_i)$, for all $U_i \in U_s \setminus G_s$, to encrypt the corresponding components of $C(GK_s)$. That is, encrypts components $c_j \in C(GK_s)$ with keys $K_j$, for all $x_j \in \mathcal{B}_z, x_j \notin \bigcup_{U_i \in U_s \setminus G_s} \mathcal{B}_i$.

   (c) Broadcasts the subgroup message

   $$\mathcal{M} = \{E_{K_j}(c_j) : x_j \in \mathcal{B}_z \setminus (\bigcup_{U_i \in U_s \setminus G_s} \mathcal{B}_i)\}.$$
7.2. A KDP Construction

2. Calculation: Since each user in $G_s$ has at least $\alpha$ keys that are incident with the group initiator $U_z$, the user can decrypt $\alpha$ messages of $E_{K_j}(c_j)$ to obtain $\alpha$ components of $C(GK_s)$. He then applies the erasure code to obtain the group key $GK_s$. All users that are not in $G_s$, as in the basic OR scheme, cannot find the group key.

In an $[n,k,m]_q$ erasure code, the length of the codeword $C(v)$ is $n \log_2 q$ bits, whereas the length of the source message $v$ is $k \log_2 q$ bits, and hence the ratio which indicates the extra bandwidth is $\frac{n}{k}$. We note that the basic OR scheme uses an $[n;1;1]_q$ erasure code in the construction. In general, we expect $k$ to be as large as possible to minimize the extra bandwidth. We also note that to use this construction with $k > 1$, the parameter $\alpha$ in the $(m,N,t)$-KDP must satisfy $\alpha \geq k$ which requires larger $m$ for the same values of $N$ and $t$, and so needs more keys. A detailed analysis for this tradeoff is an interesting problem for further research.

7.2.4 Join Event

Joining new users can be achieved by designing the system such that there are some pre-designed keys to be used by the new users. Let $(\mathcal{X}, \mathcal{B})$ be an $(m, N, t)$-KDP and $U_s = \{U_{i_1}, \ldots, U_{i_{ns}}\}$ be the set of users in session $S_s$. Each user $U_{i_e} \in U_s$ obtains a subset of keys $K(U_{i_e}) \subseteq K(\mathcal{X})$ corresponding to the block $\mathcal{B}_{i_e} \in \mathcal{B}$. To allow for adding new users, it is essential that the $(\mathcal{X}, \mathcal{B}) = (m, N, t)$-KDP satisfies $N > n_s$. This is always true. Since the keys of the new users should come from users in $U_s$, using the properties of KDP, we may assume without loss of generality, that $\mathcal{X} = \bigcup_{e=1}^{n_s} \mathcal{B}_{i_e}$.

Suppose a set of new users $J_s \subseteq N \setminus U_s$ join the group $U_s$. Assuming $N \geq |U_s| + |J_s|$, each new user $U_{i'} \in J_s$ chooses a block $\mathcal{B}_{i'} \in \mathcal{B}$, where $i' \notin \{i_1, \ldots, i_{ns}\}$ (note that the KDP is public), and finds the privileged set of his sponsors $\mathcal{P}_{s(i')} = \{U_{z_1}, \ldots, U_{z_r}\} \subseteq U_s$ such that

$$\mathcal{B}_{i'} = \bigcup_{e=1}^{r} \mathcal{B}_{z_e} \ . \quad (7.1)$$

The new user $U_{i'}$ gets his keys from the sponsors as follows.

1. Transmission: Each sponsor $U_{z_e} \in \mathcal{P}_{s(i')} \cap \mathcal{P}_{s(i')} \cap \mathcal{P}_{s(i')}$ independently sends a join message $M_{s(i')}$, consisting of keys indexed by $\mathcal{B}_{z_e} \cap \mathcal{B}_{i'}$, over a secure unicast channel, that is

$$M_{(z_e)} = \{K_j : x_j \in \mathcal{B}_{z_e} \cap \mathcal{B}_{i'}\} \ .$$
2. **Calculation:** The new user \( U_{i'} \) obtains his key set from the join messages,

\[
\mathcal{K}(U_{i'}) = \bigcup \mathcal{M}_{U_{z_i}} : U_{z_i} \in \mathcal{P}_{s(i')}
\]

\[
= \{ K_j : x_j \in \mathcal{B}_{i'} \}.
\]

A summary of the join protocol is shown in Table 7.4. After all new users obtain their key set, a user in \( \mathcal{U}_s \cup \mathcal{J}_s \) performs the OR protocol in Section 7.2.2 with \( \mathcal{G}_s = \mathcal{U}_s \) to establish a group key for \( \mathcal{U}_s \cup \mathcal{J}_s \).

| Sponsors \( U_{z_i} \in \mathcal{P}_{s(i')} \) \((X, B), \mathcal{K}(U_{z_i})\) | New users \( U_{i'} \in \mathcal{J}_s \) \((X, B)\)
|-------------------------------|------------------|
| request \( \mathcal{M}_{U_{z_i}} = \{ K_j : x_j \in \mathcal{B}_{z_i} \cap \mathcal{B}_{i'} \} \) | find \( \mathcal{P}_{s(i')} \subseteq \mathcal{U}_s \)
| \( \mathcal{M}_{U_{z_i}} \) | \( \mathcal{K}(U_{i'}) = \bigcup_{U_{z_i} \in \mathcal{P}_{s(i')}} \mathcal{M}_{U_{z_i}} \)

Suppose \( \mathcal{U}_N \) is the user set where every user \( U_i \) in the set corresponds to a block \( B_i \in \mathcal{B} \) of an \( (X, B) = (m, N, t) \)-KDP. For a session \( S_s \), where \( \mathcal{U}_s \subset \mathcal{U}_N \), the users in \( \mathcal{U}_N \setminus \mathcal{U}_s \) might be viewed as ghost users. It is worth pointing out that when running an OR or AND protocol in a session \( S_s \), the group initiator has to treat the ghost users as unauthorised users who are not in the subgroup \( \mathcal{G}_s \) (i.e., the non subgroup users are \( \mathcal{U}_N \setminus \mathcal{G}_s \), instead of \( \mathcal{U}_s \setminus \mathcal{G}_s \)). This is to preserve backward secrecy of the system. Note that this requirement will affect the number of actual users that can be revoked in an OR protocol, as the number of users outside the subgroup is limited to \( t \).

**Theorem 7.5** *The join protocol provides Join Secrecy* – For any session \( S_s \) where \( S_1 \leq S_s \leq S_M \), a collusion \( C \subseteq \bigcup \mathcal{J}_a \), \( S_s \leq S_a \leq S_M \), \( E_a = \text{Join} \), and \( |C| \leq t \) cannot find the group keys \( GK_{b,s} : S_1 \leq S_b < S_s \).

**Proof:** The new users in \( C \) are ghost users in sessions \( S_b \), for \( S_1 \leq S_b < S_s \), and they are never in subgroup \( \mathcal{G}_b \), for \( S_1 \leq S_b < S_s \). \( C \) cannot obtain the common keys of the sessions. Observe that we may consider join secrecy as multiple instances of subgroup secrecy, and so its proof is essentially identical to that of Theorem 7.1 or Theorem 7.3. \( \square \)
Theorem 7.6  The join protocol allows subsets of group members to admit a set $J_s$ of new users to a group $U_s$ when $N \geq |U_s| + |J_s|$. It requires transmissions of less than $m \log_2 q$ bits over secure unicast channels to admit a new user.

Proof: The condition $N \geq |U_s| + |J_s|$ guarantees that each new user $U_\nu \in J_s$ corresponds to a block $B_\nu \in B$. With the assumption $X = \bigcup_{i \in U_s} B_i$, there clearly exists a subset of group members in $U_s$ who can cover the $B_\nu$ of the new user and give the key set $K(U_\nu)$ to him. The key set $K(U_\nu)$ consists of less than $m$ keys, so the required bandwidth to transmit $K(U_\nu)$ is less than $m \log_2 q$ bits.

Discussion on Privileged Sets

The user join protocol above raises the following questions.

1. What is the minimum value of $r$ that satisfies equation (7.1)?

2. How can we find a set $P_s$ of users in $U_s$ that satisfies equation (7.1)?

We show that the first question in general is equivalent to Set Cover problem and is NP-complete. For the second question, we use the simple, greedy algorithm in [86] to find the privileged set with a good approximation.

Set Cover Problem  It is not difficult to see that our first question to find the minimal $r$ is equivalent to a set cover problem. (See Chapter 1 for a brief description of set cover problem.) Indeed, let $(X, B)$ be a KDP and $B' = \{B_{i_1}, \cdots, B_{i_{mn}}\} \subseteq B$ with $X = \bigcup_{B_i \in B'} B_i$. For each $B_\nu \in B$ but $B_\nu \notin B'$ we define $B_\nu^* = \{B_\nu \cap B : B \in B'\}$. Then, it is straightforward to see that our first question is exactly finding the minimum-sized of $B_\nu^*$ that covers $B_\nu$ (the set cover problem for the set system $(B_\nu, B_\nu^*)$). Since it is a well-known fact [22] that the set cover problem is NP-complete, it follows that our problem of finding the minimal number of privileged users in the user join protocol is NP-complete as well.

An Approximation Algorithm  The basic idea of the greedy algorithm is to cover the block $B_\nu$ corresponding to the new user $U_\nu$ by choosing the block that covers the largest number of “uncovered” elements at each stage. The algorithm to find the set cover is as follows.
Input: \((X, B), B' = \{ B_{i_1}, \cdots, B_{i_x} \} \subset B \) and \( \bigcup_{i=1}^{n_x} B_{i_e} = X \), \( B_{i'} \in B, B_{i'} \notin B' \)

\[ Y = B_{i'} \]

\[ C = \emptyset \]

while \((Y \neq \emptyset)\) {
    choose \(B_i\) in \(B'\) that covers the most elements of \(B_{i'}\)
    include \(B_i\) to \(C\)
    \(Y = Y \setminus B_i\)
}

return \(C\);

The set \(P_s\) consists of users \(U_i\) corresponding to blocks \(B_i \in C\). The algorithm has an approximation bound of \(\ln \ell\), where \(\ell = \max\{|B_i|: B_i \in B\}\) as the following shows.\(^2\)

Note the following inequality.

**Lemma 7.1 ([22])** For all \(d > 0\),

\[
(1 - \frac{1}{d})^d \leq \frac{1}{e},
\]

where \(e\) is the base of the natural logarithm.

**Theorem 7.7 ([86])** The above greedy algorithm has the optimum ratio bound of at most \(\ln \ell\).

**Proof:** Let \(D\) be the optimum set cover and let \(d\) denote the size of \(D\). Recall that \(C\) is the output of the greedy algorithm and let \(c\) denote the size of \(C\) minus 1. The following shows that \(\frac{c}{d} \leq \ln \ell\).

Without loss of generality, assume that \(|B_{i'}| = \ell\) where \(B_{i'}\) is the block chosen by the new join user. Initially there are \(\ell_0 = \ell\) elements needed to be covered. Note that there exists an optimum set cover of size \(d\) and by the pigeonhole principle, there must exist at least one block that covers at least \(\frac{\ell_0}{d}\) elements. This is because if every block covers less than \(\frac{\ell_0}{d}\) elements, then no collection of \(d\) blocks could cover all \(\ell_0\) elements.

At each stage, the greedy algorithm chooses the largest block which means it chooses a block that covers at least these many elements at the first stage. Therefore, there are at most \(\ell_1 = \ell_0 - \frac{\ell_0}{d} = \ell_0(1 - \frac{1}{d})\) elements left to be covered after the first stage.

Using the same argument at the subsequent stage where these \(\ell_1\) elements could be covered by an optimum set cover of size \(d\), the greedy algorithm will choose a block

\(^2\)Note that [22] gives a stronger result achieving the approximation factor of \(\ln \beta\), where \(\beta = \max\{|B_i \cap B_j|: B_i, B_j \in B\}\). However, their algorithm is much more complicated.
that covers at least \( \ell_1 \) elements, leaving at most \( \ell_2 = \ell_1(1 - \frac{1}{d}) = \ell_0(1 - \frac{1}{d})^2 \) elements for the next stage.

After using the same argument for \( c \) stages, each stage succeeds in covering at least a fraction of \((1 - \frac{1}{d})\) of the remaining elements, the number of elements that remain uncovered after \( c \) blocks have been chosen by the greedy algorithm is at most \( \ell_c = \ell_0(1 - \frac{1}{d})^c \).

Considering the largest value of \( c \) where the greedy algorithm has covered all but the last block of the greedy cover, there will be some elements left to be covered. The particular interest is the largest value of \( c \) such that

\[
1 \leq \ell_0 \left(1 - \frac{1}{d}\right)^c.
\]

It can be rewritten as

\[
1 \leq \ell_0 \left[\left(1 - \frac{1}{d}\right)^d\right]^{c/d}.
\]

By lemma 7.1 it will be

\[
1 \leq \ell_0 \left(\frac{1}{e}\right)^{c/d}.
\]

Multiplying by \( e^{c/d} \) and taking natural logs, the result is

\[
\frac{c}{d} \leq \ln \ell_0.
\]

The fact that \( \ell_0 = \ell \) completes the proof.

\[\blacksquare\]

### 7.3 Conclusion

We have considered the problem of forming subgroups by dynamic group members, and proposed two protocols based on key distribution patterns. One has the best performance when the subgroup size is close to the total number of group members, and the other when the subgroup size is very small. We have proven security and assessed efficiency of both protocols, and improved communication efficiency of the first protocol using erasure codes. The construction of efficient schemes when the subgroup size is close to half of the full group is an interesting problem for further research. Moreover, we have slightly extended the proposed scheme to have user admission capability and also given the algorithm for a new joined user to discover his sponsors and obtain a secret key set from them.
Chapter 8

Conclusion

A group key distribution system is designed to protect communication within an authorised group by providing a group key for the authorised users. An authorised group is a result of a membership event: subgroup, join, evict, or refresh. A group key distribution system must guarantee that unauthorised users cannot discover the group key shared by the authorised members even if they collude and know some portion of the secret information, including some other group keys, and eavesdrop all transmissions in the system.

A group key distribution scheme consists of an algorithm for initialising the system, and an algorithm for establishing a new group key for each possible membership event in the system. Group key distribution schemes are varied by (i) collusion size and the underlying cryptographic problem, (ii) storage, communication and computation overheads and (iii) authorised group size. Designing group key distribution schemes that have superior performance has been a major research area [4, 5, 7, 8, 17, 18, 23, 48, 52, 55, 71, 76, 81, 84, 85, 87]. Another aspect of group key distribution scheme is system model. Group key distribution schemes with new properties have also obtained attention to accommodate for diverse group applications [2, 24, 42, 47, 49, 50, 51, 53, 75, 80, 81].

8.1 Fulfilment of Aims and Objectives

The aims and objectives of the thesis were outlined in the first chapter. The following is the highlights of our major contributions.

Chapter 2 A general model for group key distribution schemes was given. The model accommodates various settings and concerns in group communications.
Chapter 3: A secure and efficient re-keying scheme for eviction and join of arbitrary number of users was proposed. It outperforms the trivial method of repeating the eviction or join procedure of a single user: a method that most previous schemes such as [17, 18, 85, 87] adopted to evict or join multiple users. When evicting or joining a single user, it also outperforms some previous schemes including [85, 87] with respect to the communication and computation costs, while performing equally to other efficient re-keying schemes such as [17, 18, 58]. The proposed scheme was proven to maintain forward secrecy, backward secrecy and forward-backward secrecy with an arbitrary number of colluders. A key recovery method complement to the proposed scheme in providing reliable re-keying was also proposed.

Chapter 4: A secure and efficient re-keying scheme for collaborative setting was proposed. The proposed scheme is scalable for large groups and has better re-keying performance than previous collaborative schemes including [16, 40, 80, 81] with respect to the communication and computation overheads, while retaining the number of rounds feasible. Security arguments based on DL and CDH problems were also given showing that the proposed scheme provides forward secrecy, backward secrecy and forward-backward secrecy with an arbitrary number of colluders.

Chapter 5: A notion of dynamic setting, including its applications, was introduced and a stateless revocation scheme for the setting was proposed. It gives flexibility in forming subgroups where any group member can be the group initiator for the subgroup of his choice. The proposed scheme is superior to [2, 75] in terms of communication cost and collusion size, and a collusion of an arbitrary number of adversaries was proven not to compromise the subgroup secrecy assuming the CDH problem is hard. The techniques in the proposed scheme can be easily applied to other stateless revocation schemes such as [37, 62] to have the dynamic property.

Chapter 6: A novel setting for dynamic group key distribution system, including its applications, was presented and two schemes for the setting were proposed. In the setting, (i) a subgroup operation and a sponsorship operation can be performed by any group member, and (ii) a full join operation requires collaborations from members of an access set of an access structure. The first proposed scheme considered an access structure consisting of all $\alpha$-subsets of users, $\alpha \geq t + 1$, while the second proposed scheme extended the first one to have some $\beta$-subsets of users, $\beta \leq t$, as the access
sets. An improvement on the second scheme was also suggested. The security of both proposed schemes is based on DDH assumption, and they were proven to be secure against a collusion of at most $t$ adversaries that does not contain an access set.

Chapter 7 A dynamic group key distribution scheme that allows several group members to cooperatively admit a new user to the group was proposed. It has efficient subgroup protocol where both storage and communication costs are equivalent to $O(\log n)$ keys, where $n$ is the group size.

8.2 Further Work

In addition to the open problems encountered in the previous chapters, further improvement to security and efficiency of existing group key distribution schemes is an attractive research challenge. This issue has gained attention, particularly in the context of broadcast encryption systems and stateless revocation systems. Further exploring other settings and their applications in group communication is also a compelling problem. Subsequently, the problem of constructing schemes for the new settings arises which needs careful attention.

The proposed schemes in the thesis used (i) the key refreshing method and (ii) the method of consistent revocation of ghost users to preserve backward secrecy of user join event. User eviction also used similar methods where evicted users are consistently revoked instead of ghost users to preserve forward secrecy. The question remains whether there are other (efficient) methods of user join and user eviction that meet the security requirement, and it is worthy of resolving.

The thesis considered the security requirement of subgroup secrecy, forward secrecy, backward secrecy or forward-backward secrecy in the proposed group key distribution schemes. It will be interesting to define new security classes that are applicable to group communication, and to construct efficient schemes that satisfy the new security requirements.
Appendix A

On the Security of Anzai et al. Scheme (Asiacrypt ’99)

A.1 Introduction

We show that the group key distribution scheme proposed by Anzai et al. in [2] does not provide backward security. We then suggest a way to repair the scheme, using the idea of proactive secret sharing [38]. See Chapter 1 for the brief review of the scheme.

A.2 Security of Anzai et al. Scheme

It is easy to see that in the user revocation operation, an authorised user can use the broadcast \((t \text{ shares at the exponent form})\), together with his own share to compute the group key through the variant of Lagrange interpolation (at exponent). Moreover, the scheme can be applied to multiple revocation operations as long as the total number of revoked users is not greater than \(t\). It is shown that the security of revocation operation in the scheme is based on Decisional Diffie-Hellman (DDH) problem [2, 64].

We will show that the join operation in the scheme does not maintain system secrecy. That is, the new joined users are able to discover the group keys for previous sessions (and so learn past communication in the system) through the following simple attack. Suppose in session \(S_T\) a revocation operation has revoked \(\{U_{i_1}, \ldots, U_{i_m}\}\), \(m \leq t\), from the group \(\{U_1, \ldots, U_n\}\) resulting in the subgroup \(\{U_1, \ldots, U_n\} \setminus \{U_{i_1}, \ldots, U_{i_m}\}\) that share a group key \(K_T\). Also assume that in the next session, \(S_{T+1}\), a new user \(U_d\) joins the group \(\{U_1, \ldots, U_n\}\). Since \(U_d\) has access to the broadcast message \((G, G_j \parallel j : j \in \mathcal{V})\) of the revocation operation (broadcast messages are publicly known and may be stored by users), using his secret information (share) \(s_d\), he can calculate the group key \(K_T\) like any authorised user in \(\{U_1, \ldots, U_n\} \setminus \{U_{i_1}, \ldots, U_{i_m}\}\). This is because \(U_d\) is not treated as a revoked user in session \(S_T\) and his secret information \((G_d = y_d^r \bmod p)\) is
A.2. Security of Anzai et al. Scheme

not included in the broadcast message. Thus, he has enough information to calculate the group key $K_T$.

A.2.1 How to Repair Anzai et al. Scheme

In the scheme, users receive different points of a common polynomial, i.e., different shares in a secret sharing scheme. New users also obtain shares from the same secret sharing scheme and this leads to the security violation in user join. New users of a session can be viewed as ghost users in the previous sessions and so can recover the previous group keys if they can access and record the broadcast messages of those sessions. To prevent this security breach, a different secret should be used for each user join operation.

We modify user join in the scheme to become stateful by updating the secret and shares of the underlying secret sharing scheme in each session of user join. (Updating the secret and shares has been considered in proactive secret sharing schemes [38].) The modified user join does not require any extra key storage for users, and no communication over secure channels is required to update the secret and shares. Each update or refresh uses a common key known to all existing users but not to the new users, and the secret and shares are self-refreshing after the common key is established.

Assume that each user $U_i$ in the group $\{U_1, \cdots, U_n\}$ has a share $F(i)$ of a polynomial of degree at most $t$, $F(x)$. Without loss of generality, let $U_{n+1}$ and $U_{n+2}$ be the new users that join $\{U_1, \cdots, U_n\}$ and the new group will be $\{U_1, \cdots, U_{n+2}\}$. The modified user join in the scheme is as follows.

1. A group controller, or a user in the group $\{U_1, \cdots, U_n\}$, establishes a common key $K$ for the group using user revocation protocol without revoking anyone ($K = g^{rF(0)} \mod p$).

2. The group controller updates the secret polynomial $F(x)$ to $F'(x) = K + F(x) \mod q$. Each user $U_i$ in $\{U_1, \cdots, U_n\}$ updates his secret information $s'_i = K + s_i \mod q$ (i.e. $F'(i)$) as his new secret information.

3. The group controller distributes shares $s'_{n+1} = F'(n+1)$ and $s'_{n+2} = F'(n+2)$ to new users $U_{n+1}$ and $U_{n+2}$, respectively, through secure channels, and publishes the public keys $y'_{n+1} = g^{s'_{n+1}} \mod p$ and $y'_{n+2} = g^{s'_{n+2}} \mod p$. Users $U_{n+1}$ and $U_{n+2}$ keeps $s'_{n+1}$ and $s'_{n+2}$, respectively, as their secret information.

4. The group controller updates public keys $y'_i = g^K \times y_i \mod p$, for all $i \in \{1, \cdots, n\}$. 
The modified user join requires extra communication and computation: (i) a broadcast message of \((t + 1)\log_2 p + t\log_2 q\) bits and (ii) each group member to compute \(t + 1\) modular exponentiations in order to add any number of new users.

**Theorem A.1** A collusion of up to \(t\) new joined users in the modified join protocol cannot discover the group keys for previous sessions.

**Proof:** The common key \(K\) is established to update the secret and shares before joining the new users (the collusion), and the colluders obtain shares (secret information) of the updated secret polynomial \(F'(x)\). Since the colluders do not know \(K\), they cannot convert their secret information to find the group keys established prior to their admission. So the security is guaranteed by the difficulty of finding the common key \(K\), which is equal to the Decisional Diffie-Hellman (DDH) problem. \(\square\)


