Key Management Protocols for Wireless Networks

Mukesh Singhal, Rendong Bai, Yun Lin,
Yongwei Wang, Mengkun Yang, Qingyu Zhang
Lab for Advanced Networking
Department of Computer Science
University of Kentucky
Lexington, KY 40506

Abstract

Key management is a critical issue for both wired and wireless secure communications. In this paper, we investigate a variety of key management protocols used mainly in wireless networks environments with different features. In particular, we review those for two-party communication, wired group communication, and wireless group communication. We analyze these protocols for security vulnerabilities against attacks. We also analyze the pros and cons of each protocol and give performance comparisons among related approaches.

Keywords: Key management, wireless networks, group communication, security.

1. Introduction

Many emerging network applications, such as video conferencing, pay-per-view media distribution, real-time information services, collaborative work, are based on group communications. They require efficient delivery of data from one or more sender(s) to a group of receivers. Many of these applications distribute sensitive information and hence require provisions for secure data transmission and membership management.

The current multicast service in the Internet successfully provides an efficient, best-effort data delivery to large groups [Dee88]. However, it does not provide data confidentiality. Furthermore, it provides little control over who can participate in a group. Any user can join or leave a multicast group and any user can send data to a multicast group. An attacker can intercept and modify packets easily because copies of plaintext traverse many more links than those of a unicast communication. Thus, a secure group communication service is necessary to ensure that only authorized members can get access to group data and only authorized members can send information to the group. Cryptographic mechanisms are used to satisfy such requirements. Specifically, messages are encrypted by senders using a group key that is only known to group members. The overall security of the group communication totally depends on the secrecy and strength of the group key. Therefore, group key management is a critical issue in secure group communication.

Generally, group key establishment protocols can be classified into key agreement and key distribution. Key agreement is also called contributory key establishment. It means that all

participants take equal part in the key generation and guarantee for their part that resulting key is fresh. Key distribution means that the key is generated by one party and distributed to all other participants. Key distribution is more suitable to large groups.

The group key management system must be able to cope with the demands of various applications. Besides confidentiality, integrity, and authenticity requirements, the following features are desirable in a group key management system for dynamic groups.

- Scalability. The size of a multicast group may vary from a few to tens of thousands. The rate of join/leave requests and the expected lifetime of a member may vary largely in different applications. The group key management system should not make arbitrary assumptions about group size. Membership changes should only affect a small subset of members so that the system can support large dynamic groups.

- Forward and backward secrecy. Forward secrecy means that a departed member can't access future data after it has left the group. To keep forward secrecy, the system must change the group key after a current member leaves, or is evicted from the system. Backward secrecy means that a joining member can't access past group data. To keep backward secrecy, the system must change the group key after a new member joins. The process of establishing new group key securely upon membership changes is referred to as *group rekeying*.

- Reliability. Since IP Multicast only provides best-effort data delivery, rekeying messages may be lost or delayed. If a receiver didn't get the rekeying information, it can't decrypt messages encrypted with the new group key. More seriously, if the sender missed new group key, data may be exposed to departed members because the sender still uses old key to encrypt group data. So the system must provide reliable transport of rekeying messages, or provide a recovery mechanism for a member to get missed rekeying messages in a timely manner.

- Resistance to attacks. Since we assume the network infrastructure is insecure, unauthorized members may eavesdrop on the group communication. A subset of departed members may collude to try to discover new group keys. A subset of current members may collude to try to discover the keying material of other valid members to impersonate the victims. Thus, a key management system should be resistant to attacks from both inside and outside the group.

- Protocol Independence. To be applicable as widely as possible, the system should not make any assumption about underlying multicast protocols and routing algorithms.

Recently, mobile computing has become very popular due to dramatic decrease in weight and cost of mobile devices, and the increase of bandwidth and new wireless services. Introducing mobile users into group communications greatly improves the flexibility, availability of group applications. However, power, storage, and bandwidth limitations restrict the ability of mobile users to participate in the group as fixed hosts. The broadcast nature of wireless medium makes attacks easier, and host mobility increases the complexity of membership dynamics. The group key management system must consider the following issues in mobile wireless environments:

- Simplicity. The system only requires little additional software/hardware in mobile hosts.

- Efficiency. To make the optimal use of limited processing and communication resources, the system must be both communicationally and computationally efficient.

- Robustness. The system should be able to work gracefully in case of communication disruptions and minimize the effect as far as possible.

- The role of base stations. Except in ad hoc networks, all mobile hosts rely on base stations to relay their communication. "What kind of role should the base station play?" and "To what degree we can trust them?" are two important questions.

In case of ad hoc networks which have no pre-existing infrastructure, secure group communication is more complex. The lack of infrastructure means the lack of central entities, fixed routers, name servers, certificate authorities, and so on. Thus standard key management techniques in wired networks can not be applied straightforwardly. The network topology of ad hoc networks may change rapidly, so any protocol bound closely with network topology is not suitable. These are all concerns we should take into account when going a further step into ad hoc environments.

A variety of protocols for secure group communication have been proposed in literature in recent years. We summarize some of the representative protocols in this survey. We briefly introduce these protocols and examine their pros and cons. The rest of this paper is organized as follows. In Section 2, we describe the necessary background on mobile networks and cryptography. In Section 3, we introduce some typical key management protocols for two-party communications. In Section 4, we discuss several key management protocols for group communications. In Section 5, we describe key exchange protocols in wireless networks and we draw some conclusions in Section 6.

2. Background

2.1 The Specific Nature of Mobile Environments

Security, authentication and access control are vital features that must be present in any communications network. These features are more important in case of wireless mobile communications than in wired communications because of the widely shared nature of the wireless medium. In fact, the mobile wireless environment has some specific characteristics which influence the feasibility and efficiency of the security protocols:

- The unique characteristics of the wireless medium. A wireless link is likely to be limited in bandwidth. Also the error rates on a wireless link is much higher than that of a wired link.
- Different types of communication paths involved, one of which is radio link, particularly vulnerable to attack.
- Location privacy. Any leakage of specific signaling information on the network can lead to an eavesdropper to approximately “locate” the position of a subscriber and thus hindering the subscriber’s privacy.

– Computational limitation. Compared with typical communications devices, the mobile station is limited in computational power. In fact, mobile and base stations have different levels of computational power.

2.2 Mobile Security Requirements

The following presents the requirements in radio link security [BM98]:

– Mutual authentication of the mobile and base stations.
– Confidentiality and integrity of the information exchanged between the mobile and base station.
– Confidentiality of the identity of the mobile station.
– Acceptable cost of the computation involved to the mobile station.

Compared with radio link security, the end-to-end security between mobile users and their communications partners typically includes confidentiality and integrity of user data, non-repudiation. The detailed requirements will vary depending on the applications.

2.3 Cryptography

Traditional cryptography is based on the sender and receiver of a message knowing and using the same secret key: the sender uses the secret key to encrypt the message, and the receiver uses the same secret key to decrypt the message. This method is known as *secret-key* or *symmetric cryptography*. The main problem with this method is having the sender and the receiver agree on the secret key that an eavesdropper will not be able to determine.

The second class of cryptography methods is *public-key cryptography* or *asymmetric cryptography*. In a public-key cryptosystem, every entity has a pair of keys: a public key and a private key. Data encrypted with the public key can be decrypted only with the corresponding private key. The public key is published, so that anyone can encrypt messages with it. But the private key is kept secret, thus only the key owner can decrypt the messages correctly. That's the essence of public-key cryptography introduced by Diffie and Hellman in 1976.

Key management consists of a set of techniques and procedures supporting the establishment and maintenance of keying relationships between authorized parties. A keying relationship is the state wherein communicating entities share common data (keying material) to facilitate cryptographic techniques. This data may include public or secret keys, initialization values, and additional non-secret parameters [MOV96].

The vast majority of key agreement protocols are based on Diffie-Hellman key exchange protocol. Diffie-Hellman key exchange protocol is a typical contributory key exchange protocol in which the session key is established from the contribution components provided by all the entities in the communication group.

The following table demonstrates how the Diffie-Hellman key exchange protocol works. The goal is for Alice and Bob (two entities in communication) to agree upon a shared secret that an eavesdropper will not be able to determine. This shared secret is used by Alice and Bob to independently generate keys for symmetric encryption that will be used to encrypt the data

stream between them. The "key" aspect of this approach is that neither the shared secret nor the encryption key ever travel over the network.

Alice and Bob agree on two numbers "**p**" and "**g**"	"**g**" is called the base or generator
Alice picks a secret number "**a**"	Alice's secret number = **a**
Bob picks a secret number "**b**"	Bob's secret number = **b**
Alice computes her public number : $\mathbf{x} = \mathbf{g}^{\mathbf{a}} \bmod \mathbf{p}$	Alice's public number = **x**
Bob computes his public number: $\mathbf{y} = \mathbf{g}^{\mathbf{b}} \bmod \mathbf{p}$	Bob's public number = **y**
Alice and Bob exchange their public numbers	Alice knows **p, g, a, x, y** Bob knows **p, g, b, x, y**
Alice computes $\mathbf{k_a} = \mathbf{y}^{\mathbf{a}} \bmod \mathbf{p}$	$\mathbf{k_a} = (\mathbf{g}^{\mathbf{b}} \bmod \mathbf{p})^{\mathbf{a}} \bmod \mathbf{p}$ $= (\mathbf{g}^{\mathbf{b}})^{\mathbf{a}} \bmod \mathbf{p}$ $= \mathbf{g}^{\mathbf{ba}} \bmod \mathbf{p}$
Bob computes $\mathbf{k_b} = \mathbf{x}^{\mathbf{b}} \bmod \mathbf{p}$	$\mathbf{k_b} = (\mathbf{g}^{\mathbf{a}} \bmod \mathbf{p})^{\mathbf{b}} \bmod \mathbf{p}$ $= (\mathbf{g}^{\mathbf{a}})^{\mathbf{b}} \bmod \mathbf{p}$ $= \mathbf{g}^{\mathbf{ab}} \bmod \mathbf{p}$
Alice and Bob then agree on the session key $\mathbf{k_{ab}} = \mathbf{k_a} = \mathbf{k_b}$	$\mathbf{g}^{\mathbf{ba}} \bmod \mathbf{p} = \mathbf{g}^{\mathbf{ab}} \bmod \mathbf{p}$ i.e., $\mathbf{k_a} = \mathbf{k_b}$

Table 1: Diffie-Hellman key exchange protocol.

There are requirements on the numbers picked (e.g., minimum size, ranges, etc.), which is known as "Diffie-Hellman parameters". The value of **p** should be larger than 2 and **g** should be an integer that is smaller than **p**. Besides, **a** and **b** should less than **p-1**.

2.4 Terminology and Notations

The following notations will be used throughout the paper unless noted otherwise:

M : mobile entity
B : base station
PK_x^+ : x's public key
PK_x^- : x's private key
$Cert(x)$: x's certificate
N_x : random challenge generated by x
$\{x\}_K$: x encrypted with key K
k_{ab} : session key between a and b
s : shared secret
r_x : random value chosen by x
T_X : time stamp issued by X
N : number of protocol parties (group members)
M_i : i-th group member; $i \in \{1,..., N\}$
h : height of a tree
$\langle l, v \rangle$: v-th node at level l in a tree
T_i : M_i's view of the key tree

$\hat{T}_i$: M_i 's modified tree after membership operation

p, q : prime integers

α : exponentiation base

3. Key Establishment in Two-Party Communications

In this section several protocols for establishing a secret key between two nodes are discussed. These protocols provide link-level security for mobile communication. That is security is provided in a machine-to-machine manner, compared with end-to-end security. These protocols also form the basis of establishing a secret key among a group of nodes.

3.1 Beller-Chang-Yacobi Protocols

Beller et al. [BCY91][BCY92][BCY93][BM98] proposed a series of protocols which combine asymmetric and symmetric cryptographic algorithms in order to meet the specific requirements imposed by the asymmetry of the computational power between a mobile host and a base station. The public key cryptosystem underlying BCY protocols uses Modulo Square Root (MSR) technique in which encryption and decryption are implemented by calculating a modulo square and a modulo square root, respectively. Computation power involved is acceptable for a mobile station.

BCY protocol family includes three variants: Basic MSR protocol, Improved MSR (IMSR) protocol, and MSR+DH protocol. Each has different feature related to security.

The Basic MSR protocol works as follows:

1. B→M: B, PK_B^+
2. M→B: $\{k_{BM}\}_{PK_B^+}$
3. M→B: $\{M, Cert(M)\}_{k_{BM}}$

Firstly, the base station *B* sends its public key PK_B^+ to the mobile station *M*. *M* then uses PK_B^+ to encrypt the session key k_{BM} and sends it to *B*. Only the base station can decrypt the session key using its private key. At the same time, the mobile also sends its identity and certificate encrypted by k_{BM} for authentication to the base station. Messages in steps 1 and 2 are based on public key cryptography while message in step 3 uses secret key cryptography.

In fact, this protocol doesn't allow authentication of the base station at the mobile host. This means, anyone can masquerade as the base station and initiate a session with a mobile host by sending his own public key and id in step 1.

To address this problem, the IMSR protocol adds $Cert(B)$, the certificate of the base station, in the first message. Besides this feature, the IMSR protocol is the same as the Basic MSR protocol.

The IMSR protocol works as follows:

1. B→M: B, $PK_B^+, Cert(B)$
2. M→B: $\{k_{BM}\}_{PK_B^+}$
3. M→B: $\{M, Cert(M)\}_{k_{BM}}$

Upon receiving message 1, the mobile decrypts $Cert(B)$ using the public key of the Certificate Authority (CA), and then it verifies the identity of the base station. The owner of a certificate should keep its certificate secret from other mobile users and eavesdroppers to prevent others from masquerading.

Note that the IMSR protocol is not immune to replay attack. If a malicious attacker copies messages $\{k_{BM}\}_{PK_B^+}$ and $\{M, Cert(M)\}_{k_{BM}}$, replays them later after the session between M and B ends, the base station can not determine if these messages indeed came from M. Even if the attacker doesn't know k_{BM} and can't do anything further, at least the session initiated by the attacker will be incorrectly charged to M. Situation might be worse if the old session key k_{BM} is obtained by the attacker, since using it to decrypt message $\{M, Cert(M)\}_{k_{BM}}$ the attacker can acquire $Cert(M)$, which is all it needs to masquerade as M.

MSR + DH protocol works as follows:

1. $B \rightarrow M$: B, $PK_B^+, Cert(B)$
2. $M \rightarrow B$: $\{k\}_{PK_B^+}, \{M, PK_M^+, Cert(M)\}_k$

This protocol adopts the idea of Diffie-Hellman protocol. The session key is not transmitted directly. PK_B^+ and PK_M^+ are exchanged to establish a shared secret s between B and M using Diffie-Hellman technique (Please refer to Section 2.3 for the details). The session key is calculated as encryption of k with s.

With improved security, the protocol requires more computation since both parties need to calculate a full modular exponentiation to establish the session key.

3.2 Beller-Yacobi Protocol

The Beller-Yacobi protocol [BM98][BY93] adds a challenge to address the replay problem in the previous protocols. The reason why this algorithm is preferred is that the digital signature is used on the challenge at the mobile and signature can largely be executed before choosing the message to be signed, i.e., the mobile can do most of the work when idle. The protocol runs in the following steps:

1. $B \rightarrow M$: B, $PK_B^+, Cert(B)$
2. $M \rightarrow B$: $\{k_{BM}\}_{PK_B^+}$
3. $B \rightarrow M$: $\{N_B\}_{k_{BM}}$
4. $M \rightarrow B$: $\{M, PK_M^+, Cert(M), \{N_B\}_{PK_M^-}\}_{k_{BM}}$

The first two messages are same as in the IMSR protocol. Upon receiving the encrypted message from M in step 2, B decrypts the message and gets the session key k_{BM}. Then B sends to M a challenge N_B (a large random number) encrypted using k_{BM}. The mobile then signs N_B using its private key and return it to B together with its id, public key PK_M^+, and certificate, all encrypted with k_{BM}. Finally, B decrypts this message using k_{BM} and decrypts N_B using PK_M^+ and checks if the N_B is the one expected.

This protocol is resistant to replay attacks. If an attacker replays a message $\{k_{BM}\}_{PK_B^+}$, it can not get the correct N_B since it doesn't know k_{BM}. Even if it knows an old session key and

can get the correct N_B, it still can not generate correct $\{N_B\}_{PK_M^-}$ since it doesn't know the private key of *M*. In both of the cases, the attacker can not respond with the correct message in step 4.

However, if an attacker is a registered user of *B*, and at the same time it can talk with *M* as a base station, it can succeed in spoofing as *M* as follows:
A is an attacker.

1. $B \rightarrow A$: $B,\ PK_B^+, Cert(B)$
2. $A \rightarrow B$: $\{k_{BA}\}_{PK_B^+}$
3. $B \rightarrow A$: $\{N_B\}_{k_{BA}}$
4. $A \rightarrow M$: $A,\ PK_A^+, Cert(A)$
5. $M \rightarrow A$: $\{k_{AM}\}_{PK_A^+}$
6. $A \rightarrow M$: $\{N_B\}_{k_{AM}}$
7. $M \rightarrow A$: $\{M,\ PK_M^+, Cert(M), \{N_B\}_{PK_M^-}\}_{k_{AM}}$
8. $A \rightarrow B$: $\{M,\ PK_M^+, Cert(M), \{N_B\}_{PK_M^-}\}_{k_{BA}}$

A starts a session with *B* and gets the random naunce N_B for the session. Then *A* turns to initiate a talk with *M* pretending as a base station. In this way, A gets the critical message $\{M,\ PK_M^+, Cert(M), \{N_B\}_{PK_M^-}\}$ from *M* and forwards it to *B* after encrypting it using the session key k_{BA}. B can not detect the threatening standing between *M* and it.

An improved BY protocol fights this attack by signing not only N_B but together with the session key and the ids of the mobile itself and the base. It works as follows:

1. $B \rightarrow M$: $B,\ PK_B^+, Cert(B),\ N_B$
2. $M \rightarrow B$: $\{k_{BM}\}_{PK_B^+},\ \{M, PK_M^+, Cert(M)\}_{k_{BM}}, \{hash(B, M, N_B, k_{BM})\}_{PK_M^-}$
3. $B \rightarrow M$: $\{N_B\}_{k_{BM}}$

Thus, the attacker *A* standing between *M* and *B* like above can't succeed by just forwarding the message containing *M*'s signature which is acquired in another session with M, even if A chooses the same session key k_{BA} with B as the one with M k_{AM}.

1. $B \rightarrow A$: $B,\ PK_B^+, Cert(B), N_B$
2. $A \rightarrow M$: $A,\ PK_A^+, Cert(A), N_B$
3. $M \rightarrow A$: $\{k_{AM}\}_{PK_A^+},\ \{M, PK_M^+, Cert(M)\}_{k_{AM}}, \{hash(A, M, N_B, k_{AM})\}_{PK_M^-}$

Now the message that B is expecting from A is

$$\{k_{BA}\}_{PK_B^+},\ \{M, PK_M^+, Cert(M)\}_{k_{BA}}, \{hash(B, M, N_B, k_{BA})\}_{PK_M^-}$$

The first two parts can be generated by *A* from message 3. However *A* can't generate the last component correctly. So *A* can't masquerade as *M* talking with *B*.

3.3 Aziz and Diffie's Protocol

The Aziz and Diffie protocol [AD94] [BM98] uses both public-key and secret-key cryptography techniques. The public-key cryptography provides the means for session key setup and authentication. Secret-key cryptography is used to provide privacy for bulk data transmission. The protocol works as follows:

alg-list: a list of flags representing secret-key algorithms provided by the mobile;
sel-alg: the flag representing the particular algorithm selected by the base station;
X_B, X_M : are contribution components for the session key provided by *B* and *M*, respectively.

1. $M \rightarrow B : Cert(M), N_M, a\lg\_list$
2. $B \rightarrow M : Cert(B), \{X_B\}_{PK_M^+}, sel\_a\lg, \{hash(\{X_B\}_{PK_M^+}, sel\_a\lg, N_M, a\lg\_list)\}_{PK_B^-}$
3. $M \rightarrow B$: $\{X_M\}_{PK_B^+}, \{hash(\{X_M\}_{PK_B^+}, \{X_B\}_{PK_M^+})\}_{PK_M^-}$

First, M sends to B its certificate, a challenge and a list of algorithms. The certificate binds the id of *M* with *M*'s public key. Using corresponding CA's public key, *B* can decrypt $Cert(M)$ and get the public key of *M*. N_M is to avoid replaying attacks. *B* responds with its certificate and session key contribution component encrypted by PK_M^+, and the preferred algorithm. To avoid the man-in-the-middle attack, a digest of vulnerable items is calculated and appended to the message. Similarly, *M* responds to *B* with its contribution component for the session key. With the knowledge of both contribution components, both sides can calculate the session key.

In the phase of session key establishment, the mobile has to perform two computationally expensive operations based on public key cryptography: one decryption to get X_B in step 2, and one encryption to do the digital signature in step 3.

In [M95], Meadows showed a possible attack on this protocol as follows.
A is an attacker who is also a registered user of *B*.

1. $M \rightarrow B : Cert(M), N_M, a\lg\_list$
2. $A \rightarrow B : Cert(A), N_M, a\lg\_list$
3. $B \rightarrow A : Cert(B), \{X_{BA}\}_{PK_A^+}, sel\_a\lg, \{hash(\{X_{BA}\}_{PK_A^+}, sel\_a\lg, N_M, a\lg\_list)\}_{PK_B^-}$
4. $A \rightarrow M : Cert(B), \{X_{BA}\}_{PK_M^+}, sel\_a\lg, \{hash(\{X_{BA}\}_{PK_M^+}, sel\_a\lg, N_M, a\lg\_list)\}_{PK_B^-}$
5. $B \rightarrow M : Cert(B), \{X_{BM}\}_{PK_M^+}, sel\_a\lg, \{hash(\{X_{BM}\}_{PK_M^+}, sel\_a\lg, N_M, a\lg\_list)\}_{PK_B^-}$
   (intercepted and discarded by A)
6. $M \rightarrow B$: $\{X_M\}_{PK_B^+}, \{hash(\{X_M\}_{PK_B^+}, \{X_B\}_{PK_M^+})\}_{PK_M^-}$

A sits between *M* and *B*. Just after *M* starts a session with *B*, *A* initiates another talk with *B* by replaying *M*'s challenge N_M. Then A forwards to *M* the *B*'s contribution component for the session key between *A* and *B* (steps 3,4) while discarding the *B*'s contribution component for the session key between *M* and *B*. Thus, the mobile calculates the session key with X_{BA} and X_M while the base station calculates that with X_{BM} and X_M. This means the mobile and the base station agree on the session key with different values and they can't do the following encryption and decryption correctly.

3.4 Park's Protocol

Park's protocol [BP98][P97] is a modified version of an earlier protocol by Yacobi and Shmuely [YS89] and it works as follows:

1. $B \rightarrow M$: $g^{PK_B^- + r_B}$ *(In* the *Yacobi-Shmuely protocol, "* $PK_B^- + r_B$ *" is sent.)*
2. $M \rightarrow B$: $PK_M^- + r_M$

The public keys of two sides are $PK_B^+ = g^{-PK_B^-}$ and $PK_M^+ = g^{-PK_M^-}$ respectively. The session key is $K_{BM} = g^{r_B r_M}$, which is calculated by M as $K_{BM} = (g^{PK_B^- + r_B} PK_B^+)^{r_M}$ and by B as $K_{BM} = (g^{PK_M^- + r_M} PK_M^+)^{r_B}$

Compared with the Yacobi-Shmuely protocol in which both the mobile host and the base station have to do two exponentiation operations, this protocol require the mobile M to perform only one exponentiation operation, which makes it more suitable for key establishment in wireless environments.

In [MM98], an attack is shown upon the Park's protocol. Suppose an attacker A knows an old session key $K_{BM} = g^{r_B r_M}$ and stores the exchanged information $g^{PK_B^- + r_B}$, $PK_M^- + r_M$, which was to established K_{BM}. A can masquerade as B successfully as follows.

K'_{BM} is the new session key.

1. $A \rightarrow M$: $g^{PK_B^- + r_B}$
2. $M \rightarrow A$: $PK_M^- + r_M'$
3. A: $(PK_M^- + r_M') - (PK_M^- + r_M) = r_M' - r_M$;

   $K'_{BM} = (g^{PK_B^- + r_B} PK_B^+)^{r_M' - r_M} K_{BM} = (g^{PK_B^- + r_B} g^{-PK_B^-})^{r_M' - r_M} g^{r_B r_M} = g^{r_B r_M'}$

In the same way the Yacobi-Shmuely protocol can be attacked in both directions due to its symmetry of the message exchange.

In fact, anyone A can run the protocol like B together with M by constructing the first message as $(PK_B^+)^{-1} g^{r_A} = (g^{-PK_B^-})^{-1} g^{r_A} = g^{PK_B^- + r_A}$. The steps followed as normal, i.e., A and M should agree on a session key $K_{AM} = g^{r_A r_M}$ finally. M takes the communication partner as B, however, it is NOT B actually. In another word, there is no authentication of B at M.

3.5 ASPeCT Protocol

ASPeCT [BP98][HP98] is the abbreviation of Advanced Security for Personal Communication Technologies. This protocol is used within the third generation mobile communications system, also known in Europe as UMTS (*Universal Mobile Telecommunications System*) for secure billing between a mobile user and a value-added service provider (VASP). The protocol is in fact divided into two separate component protocols: Authentication and Initialization Protocol, Payment Protocol. The former does authentication in various degrees between the user and the VASP, establishes the session key and initializes the subsequent payment protocol. The latter is responsible for making payments for a value-added service. Due to the subject of this paper, we just discuss the Authentication and Initialization Protocol, which is based on asymmetric cryptography.

The protocol uses three functions $h1$, $h2$ and $h3$ that are implemented using one-way hash functions. A trusted third party (TTP) is involved to work as a certificate authority. k is a temporarily used key computable by TTP. $PK_B^+ = g^b$ is the public key contained in *Cert(B)*. *ch_data is* the charging information. *pay_data* is the data needed to initialize the Payment Protocol. T is a time stamp.

The following are the steps of the protocol:

1. $M \rightarrow B$: g^{r_M}, $\{id_M\}_k$, TTP
2. $B \rightarrow M$: $\{r_B, h_2(K_{MB}, r_B, B), ch\_data, T, Cert(B)\}_k$

3. $M \rightarrow B$: $\{\{h_3(g^{r_M}, g^b, r_B, B, ch\_data, T, pay\_data)\}_{PK_M^-}, pay\_data\}_{K_{MB}}$

Between steps 1 and 2, the process is omitted that *B* consults the TTP for the public key of *M* (TTP also includes the value of *k* and a time stamp *T* in the respond message). In step 2, *B* sends the charging information to *M*. In step 3, *M* signs its payment and sends to *B*. The hashing operation avoids message compromise during transmission without detection. The time stamp added in the message aims at preventing replay attacks. Signature prevents repudiation. The session key is calculated by *M* as $K_{MB} = h_1(r_B, ((PK_B^+)^{rM})) = h_1(r_B, ((g^b)^{rM}))$ and by *B* as $K_{MB} = h_1(r_B, (g^{rM})^b)$.

3.6 Accelerating Key Establishment Protocols for Mobile Communication

Public-key cryptosystems have more advantages than secret-key cryptosystems, but they are not fully utilized because of their computational overhead and the low computing power of a mobile station. Lee et al. [LHYC98] proposed techniques for accelerating some of the above key establishment protocols between a mobile station and a base station. The basic idea is to enable a mobile station to borrow computing power from a base station without revealing its secret information. The proposed techniques use SASC (server-aided secret computation) protocol, which can be used to significantly accelerate RSA signature generation. The objective of SASC protocols is to enable the client to efficiently compute $m^s \bmod n$ with the aid of the server, where *m* is a message, *s* is private key, and *n* is the product of two large primes.

Lee et al. proposed techniques for several key establishment protocols, and we will describe the acceleration of improved BY scheme. A mobile station need to execute two public key operations and a private key operation in this key establishment protocol (section 3.2). The operation that requires extensive computation is the signature generation of the mobile station using its private key. The approach to overcome this problem is to make use of the precomputable property of ElGamal algorithm. Their insight is as follows: When the mobile station generates the signature $\{h(B, M, N_B, x)\}_{PK_M^{-1}}$, the signature can be precomputed and stored in advance as it is independent of the message *h(B, M, N_B, x)* to be signed.

The precomputation ($g^r \bmod p$) uses Beguin and Quisquater's server-aided DSS (Digital Signature Standard) scheme [BQ95], which is a splitting-based technique. The session random *r* (the secret of mobile station) is decomposed into several pieces and sent to the base station. The latter computes exponentiation for every piece and returns them to the mobile station, and the mobile station combines these results to get the signature. The precomputation works as follows:

1. The mobile station randomly chooses x_i*s* and b_i*s* which satisfy $r = \sum_{i=0}^{m-1} x_i b_i$, where $0 \le x_i b_i \le h$, and then, it sends b_i*s* to the base station.

2. The base station computes $g^{b_i} \bmod p$, for $0 \le i \le m-1$, and then, it returns them to the mobile station.

3. The mobile station computes $g^r \equiv \prod_{i=0}^{m-1} (g^{b_i})^{x_i} \bmod p$.

The analysis shows that this approach can present outstanding accelerating performance. However, it still remains an open question whether it is possible to accelerate significantly RSA signatures using an insecure server with the possibility of active attacks: that is, when the server returns false values to get some part of secret. Moreover, the rapid advances in computing power of hardware have been resulting in drastic improvements in large-number arithmetic

computations. So the bottleneck probably will shift from computation to other issues such as communication delay.

4.Key Management Protocols for Group Communication

4.1. Tree-Based Key Management

With the rapid development of deploying secure group communication services over the Internet, scalability is becoming a critical issue, especially when the group size is very large. In this section, we will focus our discussion on improving the scalability of key distribution and management, for purpose of accommodating frequent membership changes in large groups.

In tree-based key management, keys are organized into a tree hierarchy, based on different construction strategies. The basic consideration for employing this kind of hierarchy is to reduce the rekeying cost by localizing the effects of member joins or leaves, and therefore, provide higher scalability for secure communications in large dynamically changing groups. Two categories of keys are included in this kind of methods: (1) the *group session key* for encrypting messages exchanged among group members, and (2) the *auxiliary keys* used for securely distributing and updating the group session key in an efficient way.

4.1.1.Model

A communication group with N members $M_1, M_2, \cdots, M_N$, has a trusted server, called *group controller* C (or in short *controller*), who is responsible for managing group memberships, as well as the services related with key distribution and update, such as maintaining the key hierarchy, generating new keys, and initiating rekeying process. At any point, group member(s) can *join* or *leave* (either de-registration or removal by the controller) the group at will, and there's always a mechanism for C to detect these membership changes and initiate the key distribution accordingly. For example, in order to join, a member sends a join request to the *controller* C, which in turn verifies the client's credentials and securely sends group session key and necessary auxiliary keys to the new member. As for de-registration or removal, the controller distributes new generated keys, to prevent old members from compromising future communications.

Simple Key Distribution Center (SKDC) [HMR97] is one of the simplest solutions for group key management, in which *controller* C shares an individual secret key $k_{C,i}$ with each group member M_i. Secret group session key is encrypted by $k_{C,i}$ and distributed sequentially to M_i. When a new member M_{N+1} joins, the cost is not too high, since the new group key k_G' can be encrypted by old group key k_G and multicast to $M_1, M_2, \cdots, M_N$; M_{N+1} gets k_G' from a unicast rekeying message encrypted by $k_{C,N+1}$. However, when a member leaves, we cannot use old group key k_G to encrypt the new key k_G', since the removed member also knows k_G. Instead, k_G' has to be encrypted by each remaining member's individual key $k_{C,i}$ and unicast separately. Apparently this approach does not scale up with the group size, since it requires N encryptions and N rekeying messages.

We can see that communicating the new group session key in a scalable and secure way, especially when members leave, is definitely a non-trivial task. More recent research literatures

explored the scalability problem in group key distributions, based on different novel models or hierarchies. Iolus is the one of them, which addresses scalability problem by dividing a large group into multiple subgroups and employing a hierarchy of group security agents. We will discuss it in a subsequent section.

Another approach that we will discuss in this section divides the whole communication group into several subgroups, based on different strategies. Every subgroup is recursively decomposed into smaller subgroups. Each subgroup has a secret key shared by all its members to provide secure communications among them. The key corresponding to the whole group is the *group session key* K that we are interested in. The keys shared within other subgroups are called *auxiliary keys*, since their ultimate goal is only to help encrypt and distribute the group session key efficiently. The hierarchy of these subgroups naturally leads to a tree rooted at the *group session key*, with keys as internal tree nodes and group members as leaves. By employing key trees, it enables combining more than one member's rekeying messages into only one encrypted message and multicasting it, and therefore, substantially reduces the overhead on the *controller* as well as on the network traffic, compared with SKDC. We call this kind of approaches as *tree-based key management and distribution*. In the rest of this section, we describe several tree-based approaches ([WGL97], [CE99], [GS98]), with especial focus on their corresponding cost on encryption, messaging, and storage.

4.1.2 Group Key Management Using Key Graphs (KG)

Wong *et al.* [WGL97] discussed the scalability problem in key management for group communications, generalized the solutions based on secure subgroups by introducing key graphs, and formalized the notation of secure subgroups. Based on how to group the rekeying messages after a join/leave happens, three different rekeying strategies, i.e., *user-oriented, key-oriented,* and *group-oriented*, are proposed. The analysis and comparisons on their different effects on complexity are also presented.

4.1.2.1 Key Graph Notations

The notion of a *secure group* is formalized as a triple (U, K, R), where

- U is a finite and nonempty set of group members,
- K is a finite and nonempty set of keys, and
- $R \subset U \times K$ is a binary member-key relation.

Group controller knows the member set U and the key set K, and is responsible for maintaining the member-key relation R. Two functions are associated with each secure group (U, K, R):

$keyset(M_i) = \{k | (M_i, k) \in R\}$, which is the set of all keys held by member M_i, and $keyset(\phi) = \phi$;

$userset(k) = \{M_i | (M_i, k) \in R\}$, which represents the set of all members holding key k.

Figure 1 presents a key graph, where

$keyset(M_2) = \{k_{12}, k_{234,} k_{1234}\}$

$keyset(M_3) = \{k_{234,} k_{1234}\}$

$userset(k_1) = \{M_1\}$

$userset(k_{12}) = \{M_1, M_2\}$.

Here $k_{i\cdots j}$ is the key shared by members $\{M_i, \cdots, M_j\}$

Apparently, each member holds two kinds of keys: group session key and some auxiliary keys. All group members are partitioned into several subgroups recursively, and all members belonging to a same subgroup share the same auxiliary key. A subgroup key is different from other subgroup keys, which ensures that members outside a group have no way to decrypt the communications within it.

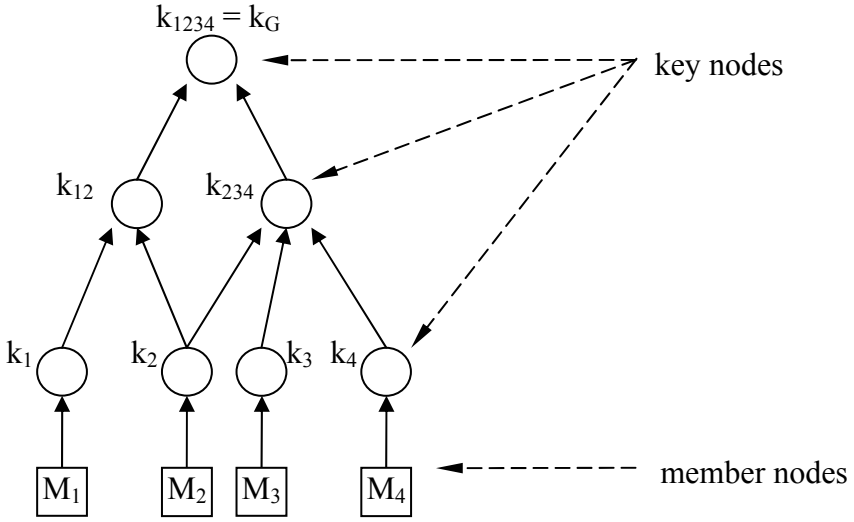


Figure 1: A key graph

The consideration of using auxiliary subgroup keys lies in the observation that when a member M_i leaves the group, only those subgroups that M_i belongs to need to change the corresponding keys, i.e., $keyset(M_i)$. All other subgroups keep their subgroup keys intact and use them to encrypt new keys for future use. Thus by localizing and restricting the effect when a member joins or leaves, we are able to reduce the overhead. In a word, properly constructing subgroups, utilizing auxiliary keys to encrypt rekeying messages, and then multicasting them to subgroups respectively, can result in substantial decrease in encryption cost and overhead on network traffic.

4.1.2.2 Rekeying Process

When a join happens, the *controller* initiates a rekeying process, so that new member(s) can join the secure message transmission in the group, while being unable to decrypt previous communications in that group. When member(s) leave, rekeying process updates the group session key to prevent members who have left the group from compromising the future group communications. Wong *et al.* presented three rekeying strategies, i.e., user-oriented, key-oriented, and group-oriented, which are based on how to construct the rekeying messages and group the encryptions. Next, we explain these rekeying strategies and illustrate them, referring to Figure 2.

User-Oriented Strategy

During the rekeying process, every member $M_i, (i = 1, \cdots, N)$ updates some of its old keys with new keys denoted by the set K_i^{new}, which the member needs for future communication and rekeying process. In the user-oriented approach, the *controller C* puts M_i's new key set K_i^{new} in a single rekeying message, and then encrypts it by an auxiliary key. The auxiliary key used for encryption here is chosen in such a way that it is shared by the largest one ($U_{\max}$) among all those groups satisfying $U = \left\{ M_j \middle| K_j^{new} = K_i^{new} \text{ and } \bigcap_j \left(keyset(M_j) - keyset(x) \right) \neq \phi, j = 1, \cdots, N \right\}$, where $x = \phi$ when member joins, and x is the member that left the group when a leave happens. The purpose for finding the "largest" such group is to minimize the encryption cost, by combining the constructions of rekeying messages encapsulating the same content to the greatest extent. Finally, in order to reduce traffic overhead, rekeying messages are multicast, which can only be decrypted by members holding the proper auxiliary keys.

- An example: When M_9 joins the group, the following actions are executed:

$C \to \{M_1, \cdots, M_6\} : \{k_{1-9}\}_{k_{1-8}}$

$C \to \{M_7, M_8\} : \{k_{1-9}, k_{789}\}_{k_{78}}$

$C \to M_9 : \{k_{1-9}, k_{789}\}_{k_9}$

When M_9 joins, group $\{M_1, \cdots, M_8\}$ changes to $\{M_1, \cdots, M_9\}$, and subgroup $\{M_7, M_8\}$ changes to $\{M_7, M_8, M_9\}$. So M_7 and M_8 need to be granted not only the new group session key $k_G (= k_{1-9})$, but a new subgroup key k_{789} as well. $M_1, \cdots, M_6$ belong to subgroups whose compositions are not affected by M_9's join, and therefore, only need to receive the new session key k_G. All the rekeying messages are generated and multicast by the *controller C*.

- An example: When M_9 leaves the group, the following actions are executed:

$C \to \{M_1, M_2, M_3\} : \{k_{1-8}\}_{k_{123}}$

$C \to \{M_4, M_5, M_6\} : \{k_{1-8}\}_{k_{456}}$

$C \to M_7 : \{k_{1-8}, k_{78}\}_{k_7}$

$C \to M_8 : \{k_{1-8}, k_{78}\}_{k_8}$

When M_9 leaves the group, $M_1, \cdots, M_6$ cannot use the old group session key $k_G (= k_{1-9})$ to encrypt the new session key $k_G' (= k_{1-8})$, because M_9 knows the old key k_G. Instead, k_{123} and k_{456} are used to encrypt the new session key, so that M_9 has no way to decrypt rekeying messages. M_7 and M_8 need to update the keys used in the subgroup previously including M_9, as well as the session key shared in the whole group.

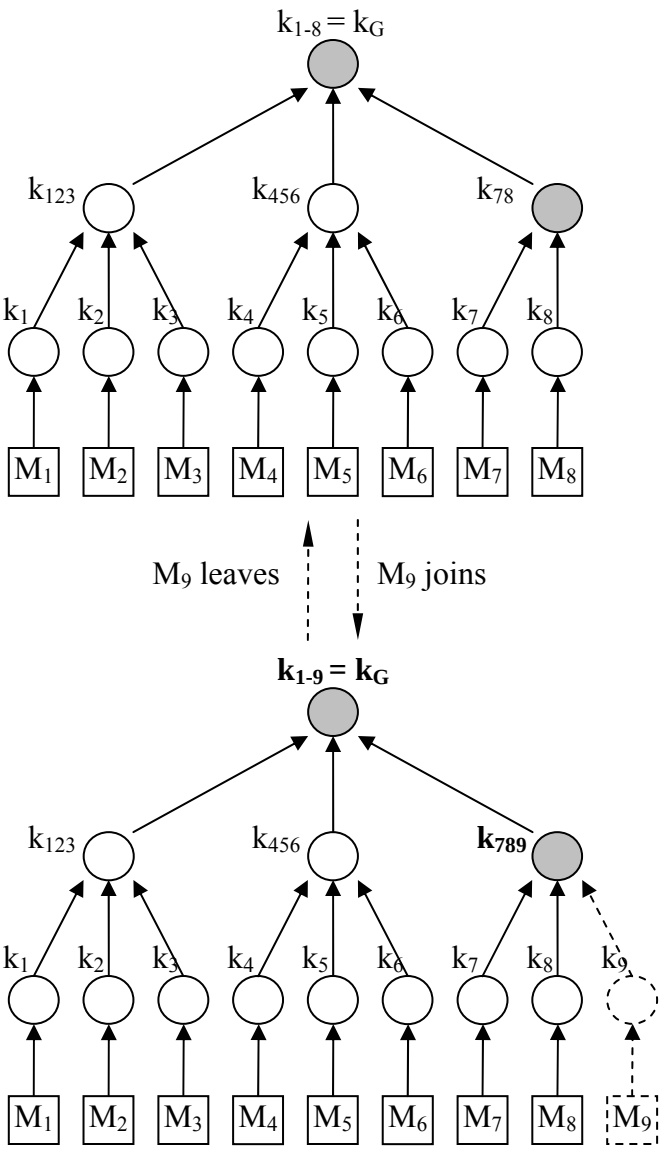


Figure 2: An example

Key-Oriented Strategy

For a group, once a join or leave happens, the set of keys to be updated during the rekeying process are determined accordingly. In the key-oriented approach, each rekeying message only contains a single new key. In order to minimize the encryption cost, the *controller C* chooses an auxiliary key to encrypt a rekeying message such that as many members as possible that need the new key contained in the message hold this auxiliary key.

- An example: When M_9 joins the group, the following actions are executed:

$C \rightarrow \{M_1, \cdots, M_8\} : \{k_{1-9}\}_{k_{1-8}}$

$C \rightarrow \{M_9\} : \{k_{1-9}\}_{k_9}$

$C \rightarrow \{M_7, M_8\}: \{k_{789}\}_{k_{78}}$

$C \rightarrow \{M_9\}: \{k_{789}\}_{k_9}$

New group session key $k_G (= k_{1-9})$ is needed by every group member. The largest subgroup that shares an auxiliary key (k_{1-8}) is $\{M_1, \cdots, M_8\}$, so k_G is encrypted by k_{1-8}. New subgroup key k_{789} is to be held only by $\{M_7, M_8, M_9\}$, and the largest group shares an auxiliary key (k_{78}) is $\{M_7, M_8\}$, so old subgroup key k_{78} is used to encrypt new subgroup key k_{789}. Newly joined member M_9 gets the new session key k_G and the new subgroup key k_{789} from messages encrypted by its individual key k_9.

- An example: When M_9 leaves the group, the following actions are executed:

$C \rightarrow \{M_1, M_2, M_3\}: \{k_{1-8}\}_{k_{123}}$

$C \rightarrow \{M_4, M_5, M_6\}: \{k_{1-8}\}_{k_{456}}$

$C \rightarrow M_7: \{k_{78}\}_{k_7}$

$C \rightarrow M_7: \{k_{1-8}\}_{k_{78}}$

$C \rightarrow M_8: \{k_{78}\}_{k_8}$

$C \rightarrow M_8: \{k_{1-8}\}_{k_{78}}$

When M_9 leaves, we cannot use the old session key k_{1-9} to decrypt any rekeying messages, in order to prevent M_9 from getting knowledge about the new keys. So the new group key k_{1-8} has to be encrypted by subgroup keys k_{123}, k_{456} or k_{78}, where first of all the new subgroup key k_{78} should be delivered to M_7 and M_8, encrypted by k_7 and k_8, respectively.

Group-Oriented Strategy

In group-oriented approach, the *controller C* tries to let a rekeying message to contain as many new keys as possible. New keys are encrypted by appropriate subgroup keys, which are chosen in such a way that the encryption cost is kept as low as possible. When a join happens, the new group is made up of the old group and a new member, so the *controller* will construct a rekeying message for each of these two subgroups. When a leave happens, the new group is just a group excluding the leaving members, so the *controller* will group together all new keys encrypted by appropriate auxiliary keys and multicast in the new group. The main consideration for adopting group-oriented strategy is to take advantage of multicasting to reduce the network overhead.

- An example: When M_9 joins the group, the following actions are executed:

$C \rightarrow \{M_1, \cdots, M_8\}: \{k_{1-9}\}_{k_{1-8}}, \{k_{789}\}_{k_{78}}$

$C \rightarrow \{M_9\}: \{k_{1-9}, k_{789}\}_{k_9}$

When a new member M_9 joins, the whole group is made up of the subgroup $\{M_1, \cdots, M_8\}$ that shares a key k_{1-8}, and the subgroup $\{M_9\}$ that holds the key k_9. The *controller C* constructs and distributes two rekeying messages designated for these two groups, respectively.

- An example: When M_9 leaves the group, the following actions are executed:

$$C \to \{M_1, \cdots, M_8\}: \{k_{1-8}\}_{k_{123}}, \{k_{1-8}\}_{k_{456}} \{k_{1-8}\}_{k_{78}}, \{k_{78}\}_{k_7}, \{k_{78}\}_{k_8}$$

The *controller C* constructs and sends out rekeying messages to all remaining members $\{M_1, \cdots, M_8\}$, using appropriate auxiliary keys to encrypt different new keys, with the goal to minimize the encryption cost by choosing keys shared by as many members as possible. When a member receives the rekeying message, it uses the keys in its *keyset* to extract those new keys that it is supposed to know, while preventing it from knowing other new keys that should not be exposed to it.

4.1.2.3. A Remark

Wong *et al.* consider star, tree and complete graph topology, and presents cost analysis of them, although we only discuss tree topology here. For more details, refer to [WGL97]. Compared with SKDC approach, the number of rekeying messages and the encryption costs are substantially decreased. The numerical results for storage complexity are given in Table 2. The rekeying complexity, when a member joins or leaves using three different rekeying strategies, is listed in Table 3, where *h* is the height of the key tree with degree *d*, and *N* is the group size. Intuitively, each user needs to store *h* keys.

Total number of keys maintained in the whole group	$\frac{d}{d-1}N - \frac{1}{d-1}$
Number of keys held per user	$h = \log_d N + 1$

Table 2: Storage complexity of KG protocol (tree topology)

	User-oriented strategy		Key-oriented strategy		Group-oriented strategy	
	Join	Leave	Join	Leave	Join	Leave
Number of rekeying messages	$h = \log_d N + 1$	$(d-1)(h-1)$	$2(h-1)$	$(d-1)(h-1)$	2	1
Encryption cost	$\frac{h(h+1)}{2} - 1$	$\frac{(d-1)h(h-1)}{2}$	$2(h-1)$	$d(h-1)$	$2(h-1)$	$d(h-1)$

Table 3: Rekeying complexity of KG protocol (tree topology)

As we can see from Table 2 and Table 3, the *controller* needs to maintain $O(N)$ keys, and each user stores $O(\log N)$ keys, and the encryption cost when a member joins is proportional to $O(1) \sim O(\log N)$, and the encryption cost for a leave is $O(\log N) \sim O(\log^2 N)$. Note the undesirable encryption cost $O(\log^2 N)$ is introduced by using user-oriented strategy when a member leaves, but we can easily avoid this relatively higher cost by choosing an alternative strategy, i.e., key-oriented or group-oriented. Hence, compared with previous SKDC approaches, where we have complexity of both encryption and rekeying messaging proportional to *N*, KG method substantially improves the scalability of the key distribution and management for group communications.

However, there remains some space for improvement. For example, Wong *et al.* [WGL97] didn't address how to construct the key tree and how to choose *h* to optimize the overall performance. Another problem is the controller has to maintain $O(N)$ keys, which puts a heavy

burden on both controller's storage as well as computation. Next we describe another key-tree-based method which reduces the number of keys maintained by the *controller* to $O(\log N)$, while resulting in similar rekeying complexity.

4.1.3 Group Key Management using Boolean Function Minimization Techniques (BFMT)

Chang *et al.* [CE99] developed a method for group key management that is based on a interesting and novel idea of defining a user ID (UID), in form of *n*-bit binary string, for each user. The set of keys held by a user is entirely determined by its UID. Since any two users must differ with each other's UID in at least one bit, when one of them leaves the group, other users can always get the new group key from the rekeying messages encrypted by key(s) not held by the leaving one.

Since we only need $n = \log N$ bits to represent a UID, not only the number of keys held by each user is $O(n) = O(\log N)$, but also the number of keys maintained by the *controller C* is reduced to $O(\log N)$. Thus, this new method achieves a substantial improvement over the key graph (KG) approach discussed in the previous section, where the controller has to maintain $O(N)$ keys.

Since multiple members may leave within a short period of time, especially for the applications with high frequency of changes, it is better not to rekey after every member leaves, under the assumption that the harm on the communication secrecy caused by a little bit delay in updating the group session key is acceptable. Rather we can batch these membership changes periodically, encrypt the rekeying messages by keys not held by all those leaving members, in order to reduce the overhead on encryption and distribution of new keys. Chang *et al.* [CE99] explored this cumulative member removal by utilizing minimization techniques in Boolean algebra to decide the auxiliary keys used for encrypting rekeying messages.

4.1.3.1 UID and Key Pair Notations

The whole group maintains *n* auxiliary key pairs, k_i and $\overline{k_i}$, where *i* ranges from 0 to $n-1$. Each key pair corresponds to one bit in UID. Besides the group session key k_G, *n* keys are assigned to every member such that, member M_j holds k_i if *i*th bit of its UID is "1", or $\overline{k_i}$ if *i*th bit of its UID is "0". A UID and corresponding key assignment for a group of members is illustrated in Figure 3.

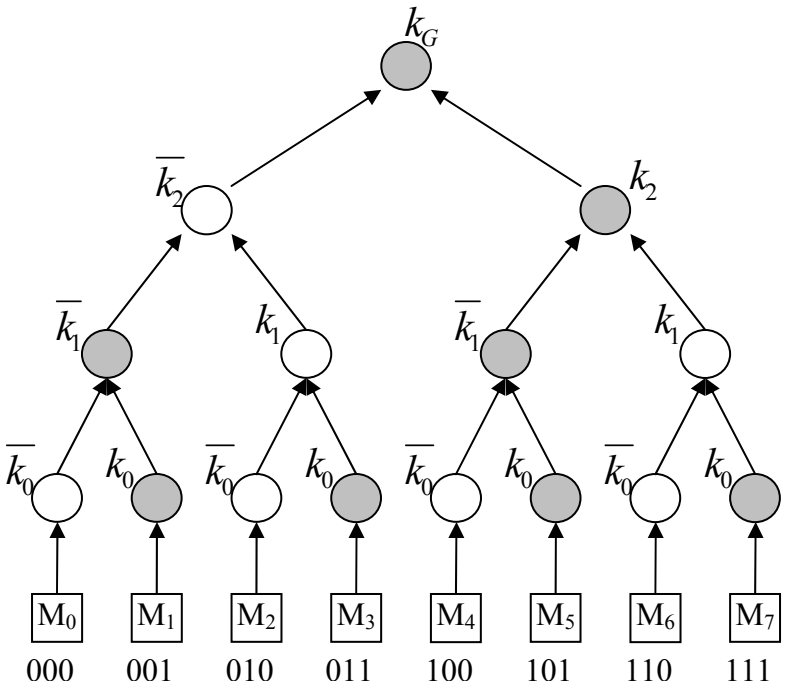


Figure 3: A key tree based on UIDs

This key tree roots at the node with group session key k_G, with auxiliary keys k_i and $\overline{k_i}$ as its internal nodes and group members M_j as its leaves. Each member has a UID as illustrated in the figure. For example, member M_5 has UID 101, which decides that it possesses auxiliary keys k_2, $\overline{k_1}$, and k_0, plus k_G that is shared by all members. All the keys held by a member are denoted by the nodes along its path to the root.

4.1.3.2 Rekeying Process

In general, both k_G and auxiliary keys update whenever a leave happens, so that the leaving member can no longer understand the future communications in the group. In batching approach, group changes are batched together and the rekeying processes are initiated every specified timeout. We call such a rekeying process as a *round*. For the *r*th round, the group session key is denoted as $k_G(r)$, and the auxiliary keys as $k_i(r)$ and $\overline{k_i}(r)$.

Individual Member Removal

In order to update $k_G(r)$, the *controller* computes a new session key $k_G(r+1)$. The new key is encrypted by the keys that are "complementary" to the ones of the departing member. Referring to Figure 3, for example, if M_5 leaves the group, the "complementary" keys are $\overline{keyset(M_5)} = \{\overline{k_2}, k_1, \overline{k_0}\}$. $k_G(r+1)$ can be encrypted using these three keys (i.e., $\{k_G(r+1)\}_{\overline{k_2}}, \{k_G(r+1)\}_{k_1}, \{k_G(r+1)\}_{\overline{k_0}}$) and multicast to all group members. Since M_5 does not know any of these keys, it can not decrypt the multicast rekeying message to get the new session

key. On the other hand, any other member's UID differs in at least one bit with the UID of M_5, therefore, possesses $keyset(M_j)$ such that $\overline{keyset(M_5)} \cap keyset(M_j) \neq \phi$, where $j \neq 5$. This ensures any other member can decrypt at least one data chunk in the rekeying message.

To make sure that the departing member is not able to use its auxiliary keys to decrypt future session key updates, auxiliary keys are also updated using one-way hash function *f*: $k_i(r+1) = f(k_i(r), k_G(r+1))$.

Removal of Multiple Members

Instead of removing the members one by one and sequentially generating and sending rekeying messages individually for every removal, under certain circumstances, it is more desirable to batch removals periodically, in order to minimize the number of rekeying messages as well as encryption cost.

Suppose M_0 and M_4 leave the group. Without batching, a total of 2*3=6 messages will be sent out, since messages encrypted by 3 auxiliary keys respectively are to be generated in each of the 2 rounds. In the cumulative scheme, the minimum set *S* of auxiliary keys, which are not held either by M_0 or M_4, is computed by using minimization technique in Boolean algebra. In our example, $\overline{keyset(M_0)} = \{k_2, k_1, k_0\}$ and $\overline{keyset(M_4)} = \{\overline{k_2}, k_1, k_0\}$, so $S = \{k_1, k_0\} = \overline{keyset(M_0)} \cap \overline{keyset(M_4)}$. Using keys in *S* to encrypt new session key ensures none of the departing members can figure out $k_G(r+1)$, while all other remaining members can always determine it.

We can resort to Boolean functions minimization techniques to compute the set S, when M_0 and M_4 leave. In Table 4 and Figure 4, the corresponding Boolean member function and its Karnaugh map minimization of membership function are illustrated, where input $X_2X_1X_0$ is the UID of a member, output "0" designates for "leaving" the group and "1" for "remaining" in the group, and "X" means UID for M_5 is currently not assigned since M_5 has already been removed from the group. Apparently, the result is $X_1 + X_0$, which corresponds to the set $S = \{k_1, k_0\}$.

4.1.3.3 A Remark

This method has some interesting features.

First, it's easy to understand, since it utilizes the most common computer science idea, i.e., using binary string to represent a member and designing the rekeying process accordingly. It simplifies the rekeying message generation and distribution algorithm. It just needs to compute the complementary key set *S* of the departing members and multicast the new session key encrypted by auxiliary keys in set *S*. Therefore the number of rekeying multicast message is only 1. As for the encryption cost, it is at most $O(n) = O(\log N)$, where *n* is the number of auxiliary key pairs, since it is enough to have *n* bits to represent all *N* users.

Second, it proposes the idea of batching rekeying messages and efficiently solved the minimization of encryption number by borrowing minimization technique from Boolean algebra.

However, it does not present a satisfactory solution to control the high cost, which is proportional to the group size N, for reconstructing the key tree and reassigning the auxiliary

keys, due to the increasement of UID length, resulting from the group expansions. The $O(N)$ complexity may severely limit the scalability of this method.

Input ($X_2X_1X_0$)	Output
000	0
001	1
010	1
011	1
100	0
101	X
110	1
111	1

Table 4: Boolean member function

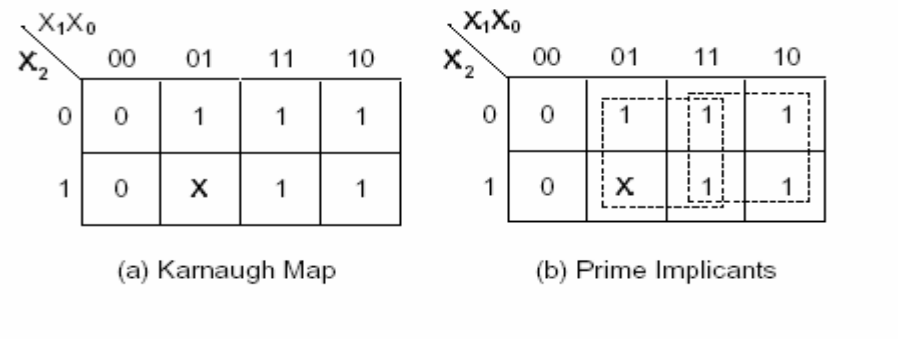


Figure 4: Karnaugh map minimization of membership function

4.1.4 Group Key Management Using One-Way Function Trees (OWFT)

McGrew *et al.* [WS98] proposed an algorithm based on one-way function trees to establish group session key when memberships change. In this method, each node *x* in the binary key tree has both an unblinded key k_x and a blinded key k'_x. The basic observation is that each blinded key is computed from a well-known one-way function $k'_x = g(k_x)$, using the unblinded key on this node as the parameter. Hence even some other node x' has knowledge about the blinded key k'_x, it has no way to figure out the unblinded key k_x because of the "one-way" feature of the function. Details on how to take advantage of this observation to construct a key tree and improve the scalability of key management for large dynamic groups are to be presented later.

The interesting idea of this approach is that the group session key is not delivered directly to each member; rather, members can calculate it "bottom-up" whenever a member joins/leaves, which results in key changes on the path from that member to the root.

In this method, the number of keys maintained in the system is $O(N)$, the number of keys stored at each member is $O(\log N)$, and the multicast rekeying message number is $O(\log N)$.

4.1.4.1 One-Way Function Tree

In this method, the *controller* maintains a binary tree, in which all group members located at leaf nodes, but leaves are not necessarily members. Every node *x* (including leaves) is associated with two keys: an unblinded node key k_x and a blinded node key k'_x, which are computed using a well-known one-way function g and a "mixing" function f : $k'_x = g(k_x)$, and $k_x = f(g(k_{left(x)}), g(k_{right(x)})) = f(k'_{left(x)}, k'_{right(x)})$, where *left(x)* and *right(x)* denote the left and right children of *x*. From this rule, with the unblinded key of a node and the blinded node of its sibling, we can always derive the unblinded key of its father.

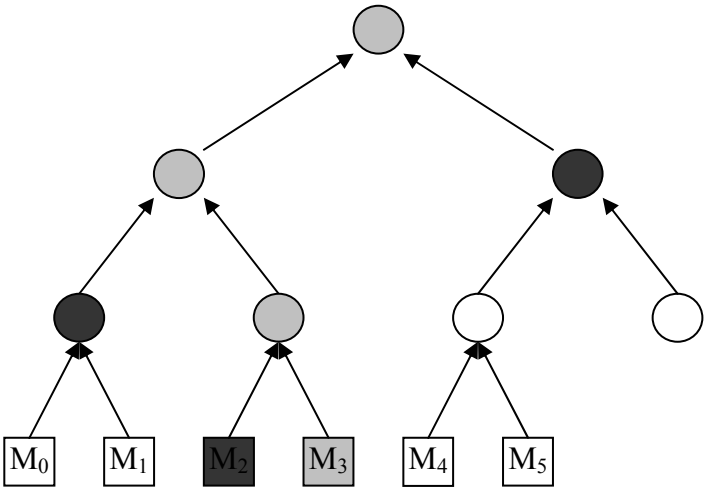


Figure 5: One-Way Function Tree

In the OWFT method, each member knows and only knows all blinded keys of sibling nodes to those nodes along its path to the root. Based on the blinded keys a member knows and the unblinded key of itself, it can compute all the unblinded keys along the path from itself to the root in a bottom-up way. Figure 5 illustrates a one-way function tree, where member M_3 knows all the blinded keys on nodes in black and unblinded key of itself (gray node M_3), therefore the unblinded keys on all other gray nodes on its path to root can be derived and known to M_3. The unblinded key associated with the root which is regarded as the group session key is finally computed independently by every member.

With the "one-way" feature of the function g , even though a node's blinded key is exposed to nodes who are not its descendents (members), there's no way for them to figure out its unblinded key, and therefore, it is used as the secure session key of the subgroup consisting of all its descendents.

4.1.4.2 Rekeying Process

The problem remained is, whenever group member(s) join or leave, it is necessary to update and securely communicate the blinded keys to appropriate member set. Next, rekeying process is outlined.

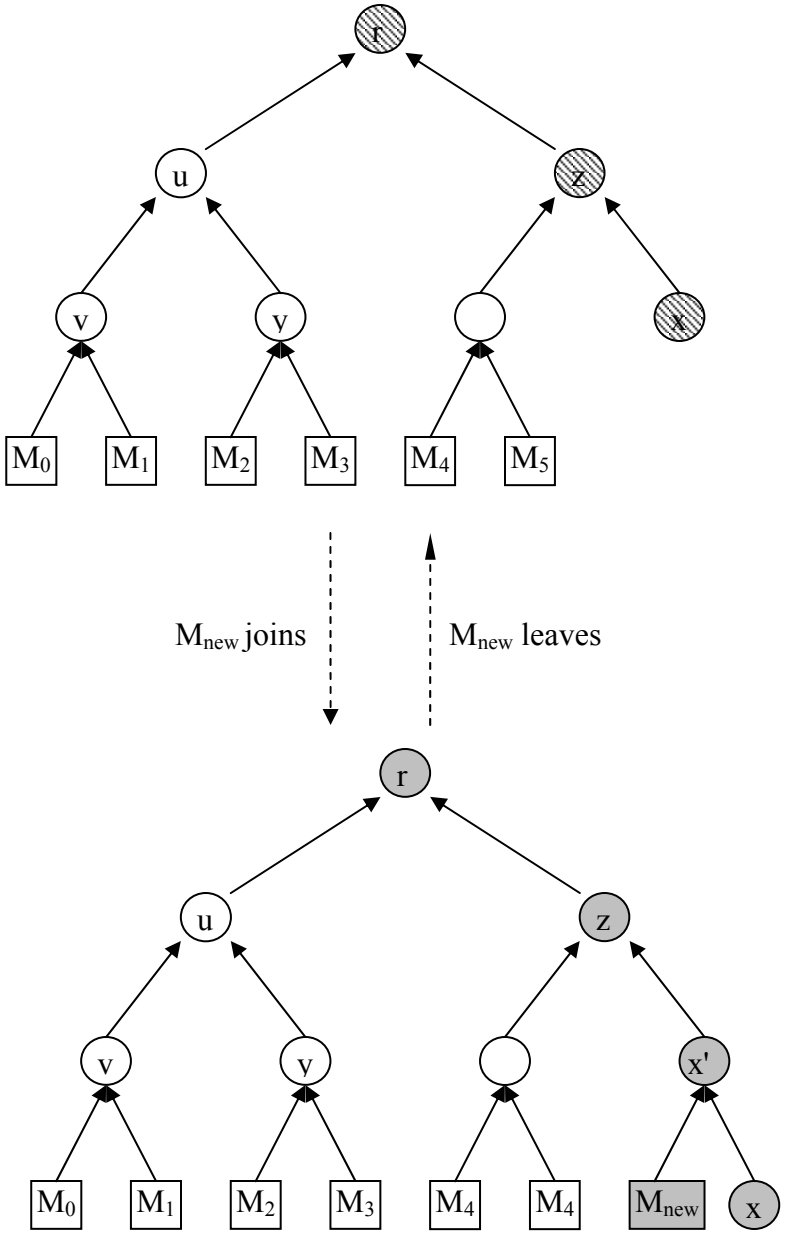


Figure 6: Member Joins/Leaves

Adding a Member

Figure 6 illustrates a situation when a member M_{new} joins. What the *controller* does is to choose a leaf node x, and replace it with a new internal node x' with two children, one of which is x itself and the other is M_{new}. The subgroups affected by this join are descendent sets of the nodes in gray color, and therefore, the unblinded keys on these gray nodes need to be updated so that backward secrecy is ensured.

To compute the new group session key, nodes storing the old versions of the updated blinded keys should be informed their new versions. For example, node *y* should be given the

updated blinded keys k_z' of node z. The set of nodes that needs k_z' is $S_z = \{u, v, y, M_0, M_1, M_2, M_3\}$, which exactly consists of z's sibling u and all descendents of u. This new blind key k_z' is included in a rekeying message, encrypted by the unblinded key k_u of u, and then multicast to set S_z.

Removing a Member

Figure 6 also presents a situation when a member M_{new} leaves. Node x' is removed and replaced by x. All the keys along the path from x to root r are updated. Accordingly, using the same method as when a join happens, the updated blind keys securely delivered to the nodes who stored old versions of them.

4.1.4.3 A Remark

The most novel idea of OWFT is to improve the key management scalability by getting around the direct delivery of group session key by maintaining a one-way function tree and computing the group key independently by each member. The number of rekeying messages and encryption complexity are determined by the number of subgroups that need the updated blinded keys. Since the number of updated blinded keys, each of which corresponds to a subgroup that needs updated keys, is at most *h* (height of tree), the number of the rekeying multicast message, as well as the encryption cost is $O(h) = O(\log N)$. McGrew *et al.* also showed that the number of keys stored in the system is $O(N)$ and the number of keys stored at each member is $O(\log N)$.

However, in OWFT, the controller has to maintain the membership information of $2N-1$ subgroups, since each node in the tree corresponds to a subgroup consisting of itself and all its descendents. The non-trivial maintenance work imposes a heavy load on the controller and may limit the scalability of this algorithm in large group applications.

4.1.5 A Performance Comparison

We now present the performance comparison between KG, BFMF and OWFT.

	KG(Tree Topology)	**BFMT**	**OWFT**
Total number of keys maintained in the system	$O(N)$	$O(\log N)$	$O(N)$
Number of keys stored on each user	$O(\log N)$	$O(\log N)$	$O(\log N)$
Number of multicast rekeying messages	Group-Oriented: $O(1)$	$O(1)$	$O(\log N)$
Encryption cost	$O(\log N) \le Cost \le O(\log^2 N)$	$O(\log N)$	$O(\log N)$

Table 5: A performance comparison

We can see that BFMT has the smallest encryption complexity and rekeying messaging complexity among the three, while reducing the number of keys maintained on the *controller* from $O(N)$ to $O(\log N)$. Also, in KG and OWFT, algorithms are relatively more complex; while in BFMT, no subgroup membership information is maintained, and the algorithm just need

to compute the complementary key set *S* of the departing members and multicast the new session key encrypted by keys in set *S*, which is much easier and straightforward.

4.1.6 A Remark

Basically, key-tree-based approaches improve scalability by reducing encryption cost and the number of rekeying messages, at the cost of larger storage space, due to introducing auxiliary keys, which are shared by members belonging to a same subgroup.

The ultimate goal of adding auxiliary keys and organizing them as a tree architecture, is to ensure that, when new members join or old members leave the group, some rekeying messages can be encrypted aggregately using subgroup keys and multicast to all members in the subgroups, rather than encrypted and unicast to each member separately.

Several novel approaches have been explored and proved to succeed in achieving higher scalability, as mentioned above. However, the controller still remains the single point of failure.

4.2 Tree-based Group Diffie-Hellman protocols

4.2.1 TGDH Protocol

Key-tree-based secure group communication protocols come in two different categories: server-based (or centralized) key distribution protocols and contributory key agreement protocols. The former is suitable for large groups while the latter is suitable for small groups. The protocols mentioned in Section 4.1 belong to the server-based category, because they have a group controller.

TGDH [KPT00] is a contributory group key agreement protocol that unifies two important trends in group key management: 1) key trees to efficiently compute and update group keys and 2) Diffie-Hellman key exchange to achieve provably secure and fully distributed protocols. The result yields a simple, secure and efficient key management solution.

Figure 7 gives an example of TGDH key tree model.

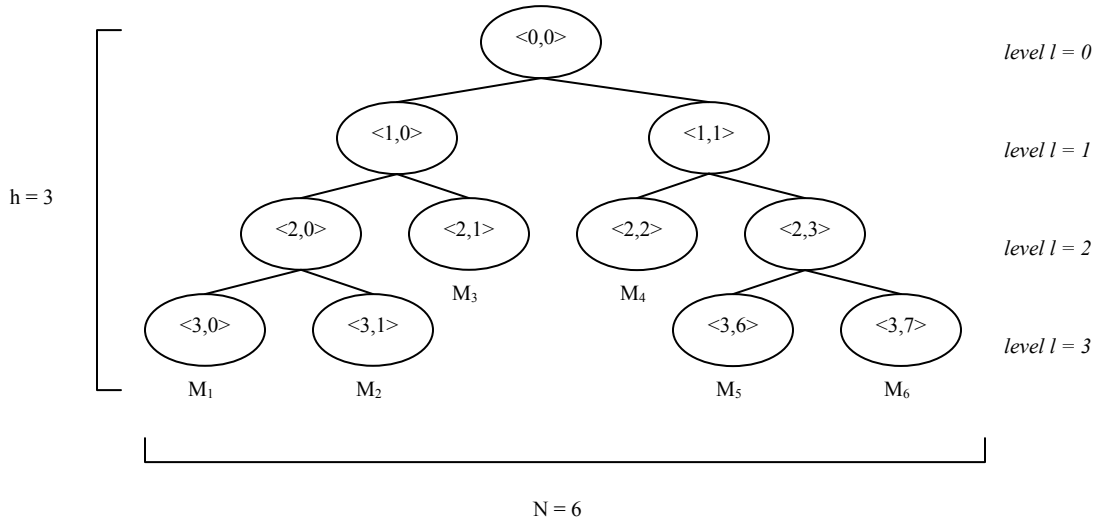


Figure 7: Tree notations for the TGDH protocol

The root is at level 0 and the lowest leaves are at level h. The tree is binary, every node has either two children or it is a leaf node. Every leaf node $<h, v>$ ($0 \le v \le 2^l - 1$, each level l hosts at most 2^l nodes) is associated with a member M_i of the group. Each node $<h, v>$ in the tree has a key $K_{<l, v>}$ and a blinded key $\mathrm{BK}_{<l, v>} = \alpha^{\mathrm{K}_{<l, v>}}$ (α is the exponentiation base). Every member M_i at node $<h, v>$ knows every key along the path from $<h, v>$ to <0, 0>. This path is called the member's ***key-path***, denoted as KEY_i^* for a member M_i. In figure 7, member M_2's key-path is $KEY_2^* = \{<3,1>, <2,0>, <1,0>, <0,0>\}$. M_2 knows all the keys along it's key-path, $\{K_{<3,1>}, K_{<2,0>}, K_{<1,0>}, K_{<0,0>}\}$, and M_2 knows all the blinded keys of the tree, $BK_2^* = \{ BK_{<0,0>}, BK_{<1,0>}, \ldots, BK_{<3,7>}\}$. Actually every member M_i knows all the blinded keys of the tree.

Every key $K_{<l, v>}$ is of the form $\alpha^{\mathrm{K}_{<l+1, 2v>}\mathrm{K}_{<l+1, 2v+1>}}$, where $K_{<l+1, 2v>}$ and $K_{<l+1, 2v>}$ are the keys of left and right child of node $<l, v>$, respectively. So in order to calculate $K_{<l, v>}$, we need to know the key of one child and the blind key of the other child. This is essential for TGDH key calculation.

$K_{<0,0>}$ is the group secret (group key) shared by all members. The group key $K_{<0,0>}$ in Figure 7 is $K_{<0,0>} = \alpha^{\mathrm{K}_{<1,0>}\mathrm{K}_{<1,1>}} = \alpha^{\alpha^{K_{<2,0>}K_{<2,1>}}\alpha^{K_{<2,2>}K_{<2,3>}}} = \alpha^{\alpha^{\alpha^{K_{<3,0>}K_{<3,1>}}K_{<2,1>}}\alpha^{K_{<2,2>}\alpha^{K_{<3,6>}K_{<3,7>}}}}$.

As an example, M_2 can compute $K_{<2,0>}$, $K_{<1,0>}$, $K_{<0,0>}$ using it's own key and the blinded keys $BK_{<3,0>}$, $BK_{<2,1>}$ and $BK_{<1,1>}$. To simplify the protocol description, the term ***co-path*** is introduced, denoted as CO_i^*, which is the set of siblings to each node in the key-path on tree T_i of member M_i. In other words, every member M_i at leaf node $<l, v>$ can derive the group secret $K_{<0,0>}$ from all blinded keys on the CO_i^* and its own key (session random) $K_{<l,v>}$.

In TGDH protocol, a group member might take on a special role, "housekeeping". For example, it can be involved to compute a key and broadcast the blinded keys to the group. This member is called the *sponsor*. The sponsor is not a privileged entity, and it is different for the group controller or group leader in previous protocols. The criteria for selecting a sponsor member vary in different membership events (join, leave, etc.).

TGDH includes protocols in support of the following operations: join, leave, merge, partition and key refresh. Due to space limitation, we will only discuss join and leave protocols.

Join Protocol

A new member M_{n+1} initiates the protocol by broadcasting a join request message that contains its own blinded key $BK_{<0,0>}$. When the current group members receive this message, they generate a new intermediate node and a new member node, and promote the new intermediate node to the parent of its node and the new member node. After updating the tree, only the sponsor can compute the group key, since it is the sibling of the joining node. After computing the group key, the sponsor broadcasts the new tree which contains all blinded keys. All other members update their tree using this message, and compute the new group key.

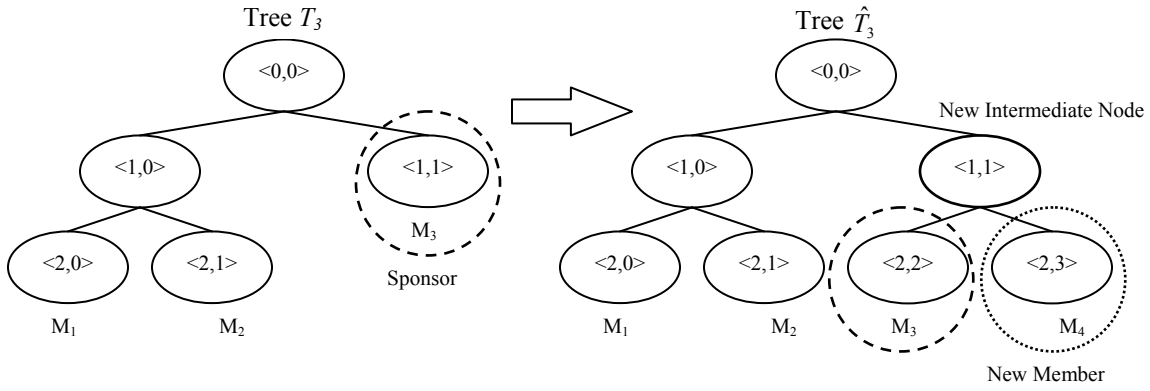


Figure 8: Tree update in a join operation

Figure 8 shows an example of member M4 joining the group. The sponsor M3 performs the following actions:

1. Renames node <1, 1> to <2, 2>,
2. generates a new intermediate node <1, 1> and a new member node <2, 3>,
3. promotes <1, 1> as the parent node of <2, 2> and <2, 3>,
4. computes the new group key K<0,0>,
5. broadcasts the new tree which contains all blinded keys to the group.

Upon receiving the broadcast message, every member can compute the group key.

Leave Protocol

Assume that we have *n* members and a member M_d leaves the group. In this case, the sponsor is the sibling node of M_d. If the sibling is not a leaf node, the sponsor is the right-most leaf node of the subtree which has the sibling node as root of the subtree. In the leave protocol (Figure 9), every member updates its key tree by deleting the node of M_d and its parent node. The sponsor picks a new secret share, computes all keys on its key path up to the root, and broadcasts the new blinded keys of its key path to the group. This information allows all members to recompute the group key.

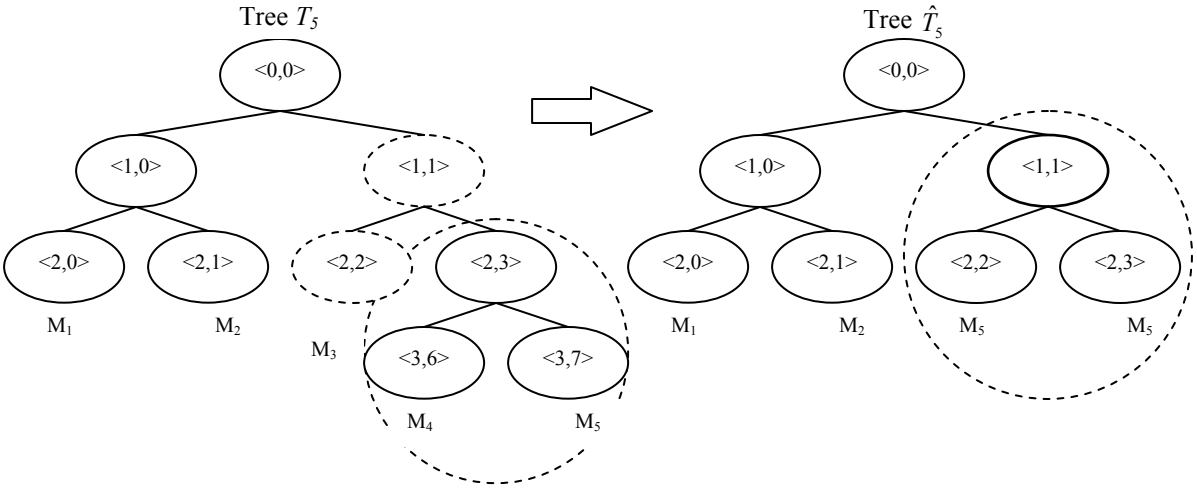


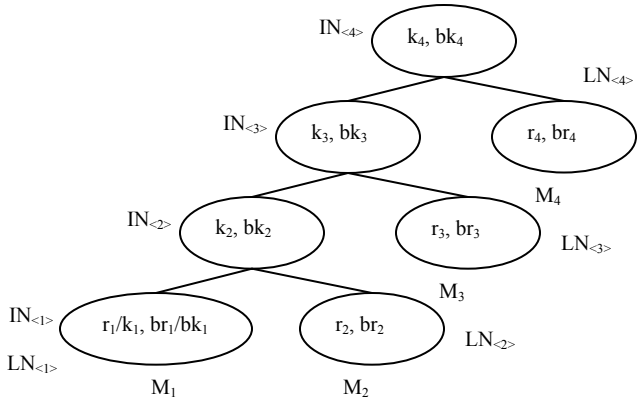
Figure 9: Tree update in a leave operation

Assuming the example of Figure 9, if member M_3 leaves the group, every member deletes node <1, 1> and <2, 2>. After updating the tree, the sponsor M_5 picks a new key $K_{<2,3>}$, recomputes $K_{<1,1>}$, $K_{<0,0>}$, $BK_{<2,3>}$ and $BK_{<1,1>}$, and broadcasts the updated tree with BK_5^*. Upon receiving the broadcast message, all members compute the group key, since the broadcast message contains every blinded key. Note that M_3 cannot compute the group key, because its share is no longer in the group key and M_5 picks a new key share.

4.2.2 STR Protocol

STR [KPT2001] protocol is based on the consideration that the rapid advances in computing have resulted in drastic improvements in large-number arithmetic computations. Thus the bottle neck is shifting from computation to communication. STR protocol tries to allow more liberal use of cryptographic operations while attempting to reduce the communication overhead, which dominates in a WAN environment.

STR is basically an "extreme" version of TGDH, where the key-tree structure is completely imbalanced or stretched out.



r_i :	M_i's secret key
br_i :	M_i's blinded session random, i.e. $\alpha^{r_i} \bmod p$
k_j :	secret key of $IN_{<j>}$, shared among $M_1 \ldots M_j$
$IN_{<l>}$:	Internal tree node at level l
$LN_{<i>}$:	Leaf node associated with member M_i
$T_{<i>}$:	Tree of member M_i
$BT_{<i>}$:	Tree of member M_i including all of its blinded keys

Figure 10: Tree notations for STR protocol

Like TGDH, the STR protocol uses a tree structure that associates the leaves with individual random session contributions of the group members. Every internal (non-leaf) node has an associated secret key and a public blinded key. The secret key is the result of a Diffie-Hellman key agreement between the node's two children (k_1 is an exception and is equivalent to r_1). k_i (i>1) can be computed recursively as follows:

$$k_i = (bk_{i-1})^{r_i} \bmod p = (br_i)^{k_{i-1}} \bmod p = \alpha^{r_i k_{i-1}} \bmod p, \text{ if i>1.}$$

The group key is the key associated with the root node. So the group key in Figure 10 is:

$$k_4 = \alpha^{r_4 \alpha^{r_3 \alpha^{r_2 r_1}}} \bmod p$$

Similar to TGDH protocol, STR also needs a sponsor member, and the selecting of sponsor varies in different membership events.

STR defines protocols for member join, member leave, group merge and group partition. We will only discuss member join and member leave protocols.

Join Protocol

Assume the group has *n* members $\{M_1, \ldots, M_n\}$. When a new member joins the group, a new root node is created for the group tree, with the following two children, the old root on the left and the new member on the right. The new member M_{n+1} broadcasts a join request message which contains its own blinded session secret br_{n+1}. At the same time, the current group's sponsor (M_n) computes a blinded version of the old group key (bk_n) and sends the old tree $BT_{<n>}$ to M_{n+1} with all blinded keys and blinded session randoms.

Then, each M_i increments n = n + 1 and gets the new key tree structure. Now every member can compute the group key:

- All the old members know the old group key, they can use the old group key and the new member's blinded session random.
- The new member uses the blinded old group key and its own session random.

Leave Protocol

Again we assume the group has *n* users $\{M_1, \ldots, M_n\}$, a member M_d (d≤n) leaves the group. In the leave protocol, if d>1, the sponsor M_s is M_{d-1}, otherwise the sponsor is M_2. Upon knowing the leave event from the group communication system, each remaining member updates its key tree by deleting the nodes $LN_{<d>}$ corresponding to M_d and its parent node $IN_{<d>}$. The

nodes above the leaving node are also renumbered. The former sibling $IN_{<d-1>}$ of M_d is promoted to replace (former) M_d's parent. The sponsor then selects a new secret session random, computes all keys and blinded keys up to the root, and broadcasts the $BT_{<s>}$ to the group. This $BT_{<s>}$ allows all members to recompute the new group key.

Table 6 shows a comparison of TGDH protocol and STR protocol. As seen from the table, STR costs less in communication on every membership event.

		Rounds	Messages	Unicast	Broadcast	Exponentiation
TGDH	Join	2	3	0	3	2 log *n*
	Leave	1	1	0	1	Log *n*
STR	Join	1	2	1	1	2
	Leave	1	1	0	1	3*n*/2 + 2

Table 6: Comparison of TGDH and STR

4.3 Distributed Hierarchical Tree Approach

4.3.1 The Iolus Framework

Iolus [Iolus] is a distributed hierarchical tree-based approach, which uses a *secure distribution tree.* The secure distribution tree is composed of a number of smaller secure multicast sub-groups arranged in a hierarchy to create a single *virtual* secure multicast group (see Figure 11).

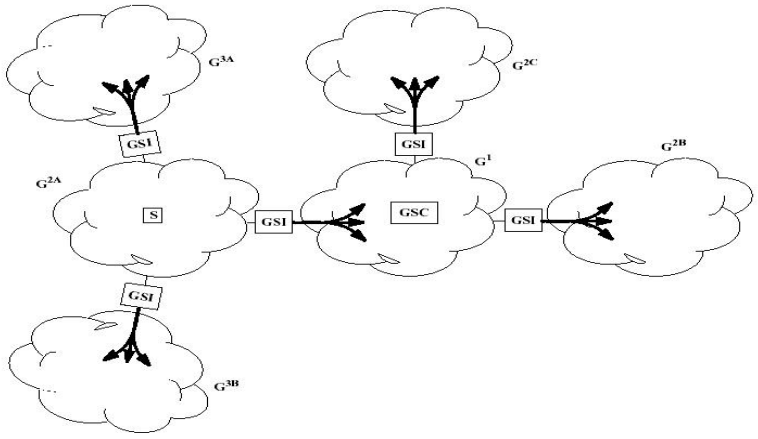


Figure 11. Iolus architecture.

Each subgroup is managed by a subgroup controller called *group security intermediary (GSI)*. GSIs form a hierarchy of subgroups and the top-level subgroup is managed by *group security controller (GSC)*. GSC is ultimately responsible for the security of the entire group. Each GSI joins the subgroup at the next higher level (or the subgroup of GSC) and acts as proxies of the GSC or its parent GSIs.

In Iolus, there is no global group key. Each subgroup maintains its own subgroup key. When a member joins or leaves, it joins or leaves its local subgroup. Therefore, only the subgroup key needs to be changed. The following subsection describes five basic operations of Iolus framework.

4.3.2 Iolus Operations

4.3.2.1 Startup

The startup of the secure communication group requires only that the GSC for the group be started. After that, GSIs and other members apply to join its subgroup.

4.3.2.2 Join

To join a group, a member sends a JOIN request to its designated GSI (or GSC) using a secure unicast channel[1]. Upon receiving the request, the GSI (or GSC) decides whether to approve or deny the request. If the request is approved, the GSI will (1) generate an individual key K_{MBR}, which is shared only with the new member; (2) stores K_{MBR} along with any other relevant information concerning the new member in GSI's private database; (3) sends K_{MBR} to the new member securely.

Then the GSI needs to change current subgroup key K_{SGRP} to a new one K'_{SGRP}. The GSI multicasts a *GRP_KEY_UPDATE* message containing K'_{SGRP} encrypted with K_{SGRP} to its subgroup and sends K'_{SGRP} to the new member via the existing secure unicast channel.

4.3.2.3 Leave

LEAVE operation occurs under two conditions: (1) a member voluntarily leaves the subgroup, or (2) GSI expels a member.

In either case, the GSI needs to change current subgroup key K_{SGRP} to a new one K'_{SGRP} to prevent the leaving member from participating in future communications. To distribute K'_{SGRP} to the subgroup members, the GSI multicasts one message containing n copies of K'_{SGRP} (n is the number of remaining members), each encrypted with a member's individual key K_{MBR}.

4.3.2.4 Key Refresh

With use, K_{SGRP} will "wear out" and need to be changed. A new subgroup key K'_{SGRP} can be distributed by multicasting it to the subgroup encrypted with K_{SGRP}.

4.3.2.5 Data Transmission

Due to the lack of a global group key, sending multicast data is not as simple as multicasting the data to the group encrypted with a group key. Instead, multicast data is relayed by GSIs. More specifically, the sender multicasts the data directly to its local subgroup encrypted with the subgroup key. The parent GSI (if this is not the top-level subgroup) receives multicast data, decrypts it, and re-multicasts it to its parent subgroup encrypted with the subgroup key of its parent subgroup. Similarly, child GSIs get multicast data and remulticast it to their child subgroups.

The advantage of this approach is that there is no global group key. Thus both the frequency and computation/communication overhead of re-keying depends on the size of a subgroup instead of the size of the whole group. However, this approach requires full trust in the

[1] Any of existing unicast security protocol that provides mutual authentication can be used.

GSC and GSIs. It may incur a lot of computational overhead because the GSIs have to re-encrypt all data passing them.

All the schemes mentioned above in Section 4 require that a legitimate receiver is capable of recording the past history of re-key operations and change its keys accordingly. In the case of high packet loss rate, these schemes will not work well. The following two schemes are designed for "stateless" receivers, i.e., they are not constantly on-line and can deduce the new group key from their initial configuration.

4.4 Broadcast Encryption Scheme

The *Broadcast Encryption* [ASW00][FN94] technique allows a center efficiently broadcast information to all users in such a way that only privileged users can decrypt the message. An example scenario is a satellite/cable TV broadcast network. Each user has a special device when he subscribes to pay TV service and can only get the channels he paid for. To solve this problem, key-tree based approaches suggest building a separate key tree for each channel, thus incurring a setup cost of at least *logk* keys per channel for target receivers of size k. The broadcast encryption schemes use a single key structure for all programs and are efficient in two measures, i.e., the number of keys stored at receiver and the number of keys transmitted by the sender.

In order to achieve the efficiency goal and break the theoretical bounds, Abdalla *et al*. [ASW99][FN94] proposed a scheme, which allows a controlled number of users outside the target set (free riders) to occasionally access the multicast data. Abdalla *et al*. introduce *f-redundant* establishment key allocations, which guarantee that the total number of recipients is no more than *f* times the number of intended recipients. A simple multi-level establishment key allocation is a balanced binary tree, built by recursively partitioning the sets of a high level into equally-sized disjoint sets in the next level. The number of keys each receiver holds is only *(1+logn)*. In the environment where membership changes dynamically, the establishment key allocation can be built incrementally. A new partition is created at the beginning of each phase, with virtual "place holder" users. Each new user that joins replaces a virtual user and is assigned the virtual user's keys. The phase ends when all the virtual users have been replaced by real users. Then a new phase starts. A leaving user is marked as non-existing. Once the number of non-existing users in a partition drops below some threshold, the partition is deleted and all the remaining users are rekeyed to a new partition.

Once the establishment key allocation is decided, the next problem is to find a good key cover in which the union of sets contains all the legitimate receivers for each target set. The transmissions needed for re-key operations depend on the number of sets in the cover. Since the *Set Cover* problem is a NP-hard optimization problem, Abdalla *et al.* proposed a greedy approximation algorithm to find a good key cover.

This approach is quite practical for very large groups where some free riders may be tolerated.

4.5 Subset-Difference Based Approach

It is often convenient to think of Broadcast Encryption as a *Revocation Scheme*, which deals with the case where some subsets of the users are excluded from accessing the multicast data. The

Subset-Difference based approach [LNN01][NNL01] is a new revocation scheme that is especially suitable for *stateless* receivers. In such a scenario, a receiver can't record the past history of rekeying operation and update its keys accordingly. Instead, each receiver can deduce the current session key based on the current rekey message and receiver's initial configuration. Stateless receivers are very useful in environments with unreliable communication.

The Subset-Difference based approach allows the group controller to transmit a message to all users such that any non-revoked (remaining) user can decrypt the message correctly, while even a coalition of all revoked members cannot decrypt it. The algorithm consists of three components: 1) Initiation, which assigns every receiver some private information; 2) Broadcast Algorithm at the group controller, which partitions the non-revoked users into disjoint subsets S_{i1}, …, S_{im}, and encrypts the new session key separately by the keys associated with these subsets; 3) Decryption at receiver, which finds out the specific subset this receiver belongs to, deduces the subset key from its private information, and then gets the new session key.

4.5.1 Definitions

Let N be the set of all users (|N|=n), among which *r* users should be revoked. Let R denote the set of revoked users. All users are viewed as leaves in a complete binary tree.

A subset-difference $S_{i,j}$ is defined as set of all leaves in the subtree rooted at V_i but not in V_j, where V_i is an ancestor of V_j. In another words, $S_{i,j}$ consists of the leaves of V_i minus the leaves of V_j. So, a leaf *u* is in subset of the form $S_{i,j}$ iff V_i is one of its ancestors but V_j is not. Figure 12 shows an example of $S_{i,j}$. All black leaves are rooted at V_j, and $S_{i,j}$ consists of only the gray leaves.

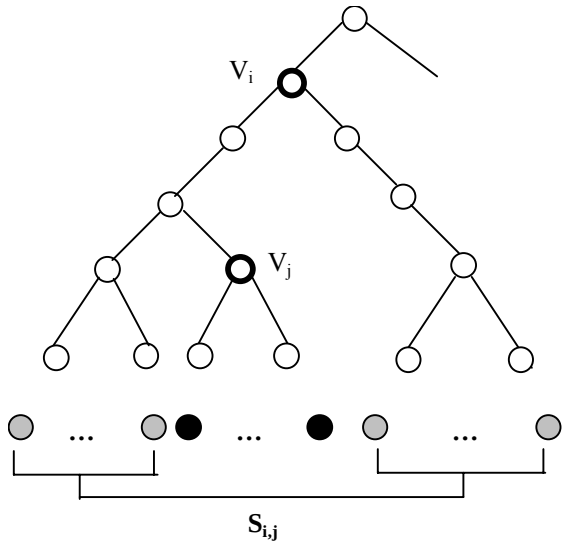


Figure 12. Subset-Difference definition

The *Cover* is a collection of disjoint sets $S_{i1,j1}$, $S_{i2,j2}$ ... ,$S_{im,jm}$ which partitions the non-revoked users N/R. Figure 13 shows an example group with 32 leaves, 12 of which are revoked. The cover consists of 6 subset-differences: $S_{a,b}$, $S_{c,d}$,$S_{e,f}$, $S_{g,h}$, $S_{i,j}$, $S_{k,l}$. The cover size is defined as the number of subsets in the cover. Lotspiech *et al.* give out an efficient algorithm to find a small cover, whose size is only 1.25r on the average.

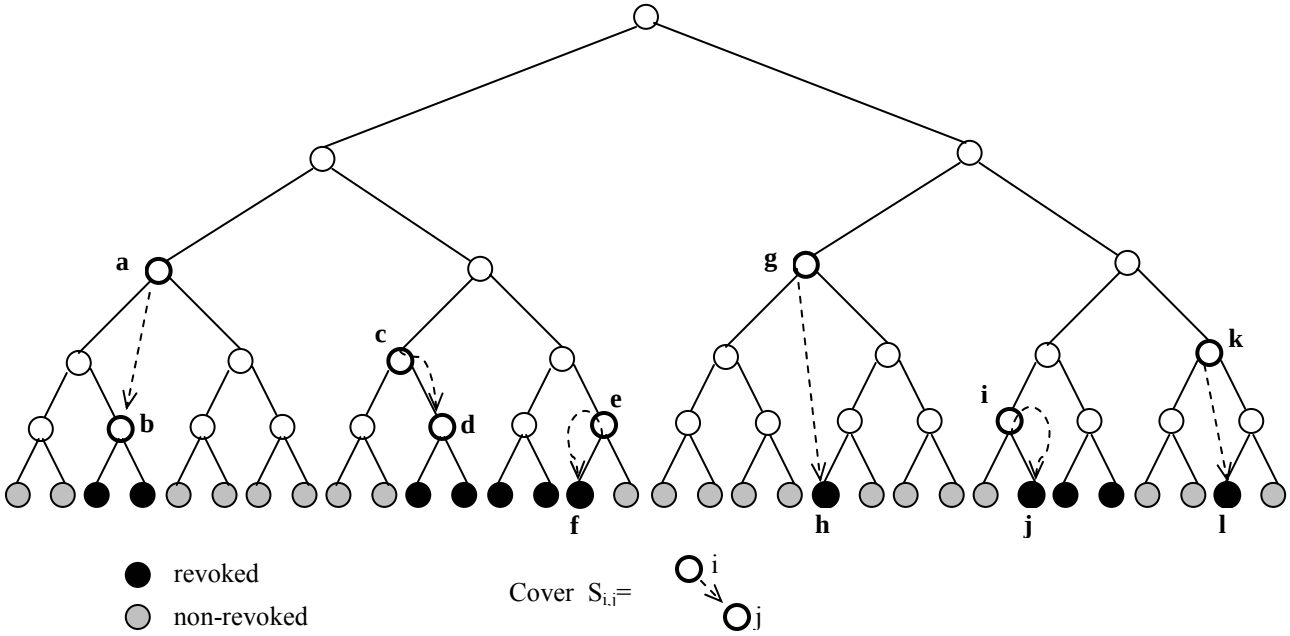


Figure 13: An example of subset cover

4.5.2 Key assignment

Each internal node *i* in the binary tree has a random and independent value LABEL*i*. From this value, we can calculate the keys for all subsets of the form $S_{i,j}$ using pseudo-random functions.

Let G be a pseudo-random sequence generator that triples the input, i.e., whose output length is *three times* the length of input. Let *S* denote the label at one node*, GL(S)* denote the label of left child*, GR(S)* denote the label of right child*,* and *GM(S)* denote the key of this node. Figure 14 shows the idea of G.

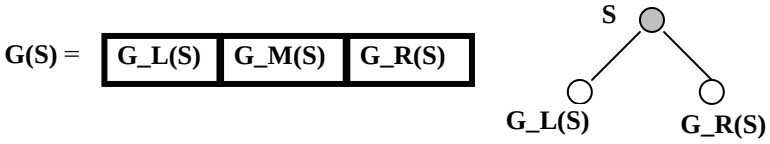


Figure 14: Generator G

For a given subtree T_i rooted at V_i, the labels and keys are assigned in a top-down manner as shown in Figure 15. Let $LABEL_{i,j}$ be the label of node V_j derived in *Ti* from $LABEL_i$, and the key of $S_{i,j}$, denoted by $L_{i,j}$ is $GM(LABEL_{i,j})$. So given $LABEL_i$, computing $L_{i,j}$ requires at most *logn* applications of G.

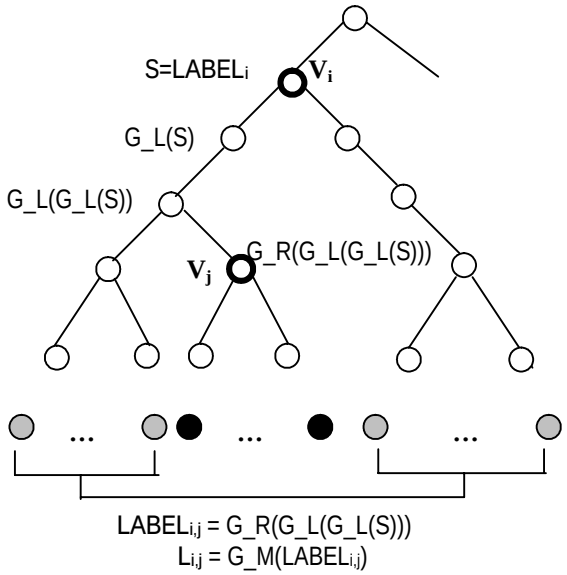


Figure 15 Generation of $LABEL_{i,j}$ and key $L_{i,j}$

Now, let's consider the information each leaf node *u* needs to store in order to derive the key assignment described above. For each subtree T_i such that *u* is a leaf of T_i , the receiver *u* should be able to compute $L_{i,j}$ iff *j* is not an ancestor of *u*. For every V_i, which is ancestor of leave *u*, let the label of V_i be S. *u* receives all labels at nodes that are **hanging off the path** from V_i to *u*. These labels are all derived from S. Figure 16 shows the key assignment process for *u*. In this example, leaf *u* receives the labels of V_{i1}, V_{i2}, V_{i3}, V_{i4}, and V_{i5} that are induced by label of V_i.

Each receiver needs to store $(0.5\log^2 n + 0.5\log n + 1)$ keys. At decryption step, receiver u first finds the subset $S_{i,j}$ it belongs to, computes the corresponding key of $S_{i,j}$, and then decrypts the new session key.

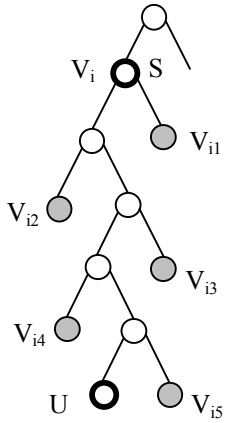


Figure 16. An example of key assignment

This scheme has several advantages. 1) All the receivers are "stateless" and can get new session key from its initial configuration. 2) Rekey message size is small because the remaining receivers are partitioned into 1.25r subsets on average (r is the number of revoked members). 3) Each receiver needs to do one decryption for every rekeying event (plus at most logn

applications of a pseduo-random generator). However, the number of keys stored by each receiver is $0.5\log^2 N$.

5. Key Exchange in Wireless Multicast Network

Few protocols exist to deal with the secure multicast in wireless networks. Among them, most migrate the algorithms from the wired networks, together with a discussion of the wireless environment without concrete algorithm.

5.1 Key distribution protocol of Bruschi et al.

Bruschi and Rosti [BR00] classified base stations into three different trust degrees. In the first degree, the base station is not trusted at all, such system is called non-trusted system. In the second degree, the base station is not fully trusted. It will not be allowed to understand the data traffic but will behave correctly. This system is called semi-trusted system. For the last degree, the base station is fully trust. Such system is called fully-trusted system. For these three scenarios, different key managements with secure multicast service are provided.

As a general method, participants in the secure multicast are classified as data group and control group. Data group (DG) includes all the members interested in receiving data traffic in the group. Control group (CG) includes all base stations involved. Group members also exchange keying material besides data traffic. Three types of keys are used. These are the key used to encrypt/decrypt data traffic (named TEKs), the key used to encrypt/decrypt TEKs (named KEKs) and the key used for encryption in a cell covered by a base station (named CEKs), which is completely managed by the base station and local to each cell.

In the following description of this protocol, we use A => B to denote A broadcast (or multicast) message to B, whereas A→ B denotes A unicast message to B. Besides, A=>B→C or A→B=>C means A sends message to B, then B forwards this message to C.

A non-trusted system

The group manager (GM) and the mobile hosts share the task to manage group dynamics and host mobility. No better solution could be provided.

A semi-trusted system

A protocol with two tier structure is provided. In the first tier, the group manger manages all the support base stations and distributes TEK to the group members when they subscribe to the group. These stations act as agents of the mobile hosts in the data group. Data traffic is symmetrically encrypted using a session key *s_k* . This session key *s_k* is in turn symmetrically encrypted using the TEK *t*. In the second tier, the base station relays the data traffic to the hosts in its cells. Each base station acts as the group manager of a centralized tree VersaKey [WCSWP99] in its cell. It receives the data traffic from GM and encrypts them using local CEK, then broadcast it in the local cell to the mobile hosts M_{s_i}, as the follows:

$$\text{GM => CG: } \{data\_traffic\}_{s\_k}, \{s\_k\}_t$$

$$\forall s_i \in CG,\ s_i \Rightarrow M_{s_i} : \{data\_traffice\}_{s\_k}, \{\{s\_k\}_t\}_{c_{s_i}}$$

After receiving the messages, the mobile hosts first decrypt them using local CEK c_{s_i}, then decrypt again using TEK *t* they received when subscribing the group, and get the *s_k*. Thus they can get data traffic.

Because of the CEKs used for encryption in each cell, host mobility and group dynamics is restricted to the cell level as key update may occur only in the interested cell. Since only those who know both CEK and TEK can know the data traffic, updating CEKs for group dynamics can guarantee the backward and forward traffic secrecy.

The protocol works as follows:

-- Initially, a control group CG and a data group DG are created for every multicast group. A mobile host is added to DG when it subscribes to the service and removed from DG when it terminates the service. Similarly, base station will be added to CG if there is mobile host subscribed to the service in its cell and removed from CG if no mobile hosts subscribed to the DG.

-- When a new mobile host *m* joins the group, it sends subscription request to the GM, forwarded by its base station *s*. This request is digitally singed by *m*'s private key to allow GM to authenticate.

$$m \Rightarrow s \rightarrow GM: \{SUBSCRIBE\}_{PK_m^-}$$

If the request is approved, an *add* operation is executed in the first tier: if the base station is not in the CG, it is added into CG. Meanwhile, GM adds *m* to DG, gives *m* the current TEK *t* used to encrypt the data traffic:

$$GM \rightarrow s \Rightarrow m: \{\{t\}_{PK_m^+}\}_{PK_{GM}^-}$$

Then base station *s* will proceeds with *join* operation int the second tier: if *m* is the only member in the cell, *s* generates CEK *cs*, and a symmetric session key *sm* used for control traffic with *m* only and sends them to *m*:

$$s \Rightarrow m : \{c_s\}_{sm}, \{\{sm\}_{PK_m^+}\}_{PK_s^-}$$

Otherwise, if *s* has been in the CG before *m* joined, then the cell executes a local centralized tree VersaKey scheme. *s* updates its local CEK c_s to $c_s^{'}$ and all the KEKs along the path from leaf *sm* in VersaKey tree to the root. These keys, together with the symmetric session key *sm*, are sent to *m*:

$$s \Rightarrow m : \{sm, k_{n_s-1}^{'}, \ldots, k_2^{'}, k_1^{'}, c_s^{'}\}_{sm}, \{\{sm\}_{PK_m^+}\}_{PK_s^-}$$

Where $k_{n_s-1}^{'}, \ldots, k_2^{'}, k_1^{'}$ are the new keys along the path from the leaf *sm* to the new root $c_s^{'}$. Giving the newcomer a new CEK and a new set of KEKs guarantee it will not receive the data traffic before it join the group.

-- When the mobile host *m* leaves from the group, it sends a *CANCEL* message to GM, forwarded by local base station *s* .

$$M => s \rightarrow GM: \{CANCEL\}_{PK_m^-}$$

GM executes *delete* operation in the first tier to remove *m* from DG. Also GM checks if base station *s* should be deleted from CG. No key change is required in this level.

Meanwhile, *s* executes *leave* operation in the second tier. It changes it local CEK *cs* to prevent *m* from accessing the data traffic after it leaves the cell. *S* executes the following according to the centralized tree VersaKey scheme:

$$s => M_s : \{k'_{n_s-1}\}_{\underline{sm}},\ \{k'_{n_s-2}\}_{k'_{n_s-1}}, \ldots, \{k'_1\}_{k'_2}, \{k'_1\}_{\underline{k_2}}, \{c'_s\}_{k'_1}, \{c'_s\}_{\underline{k_1}}$$

where $\underline{k_i}$ is the sibling of node k_i in the tree, and $k'_{n_s-1}, k'_{n_s-2}, \ldots k'_2, k'_1$ are the new KEKs along the path from the leaf *sm* to the new root c'_s.

-- When a mobile host moves from one cell to its neighbor, a hand-off procedure is performed by the base station. It involves the departed base station *s* and the entered base station *s'* and possibly GM. Because the mobile host is still in the same group, the old base station *s* execute *leave* operation and the new base station *s*' execute *join* operation. When *m* move into a new cell, it sends a message to *s*':

$$m => s' : \{m, mcast\_grp\_id, seq\}_{PK_m^-}$$

where *mcast_grp_id* is the identifier of the multicast group to which *m* belong and *seq* is the sequence number of the last packet received by *m*.

-- The key material should be updated after an interval to guard against cryptanalytic attacks. The update rate depends on the rate that mobile hosts join and leave the group and speed they move from cell to cell. Each base station may perform a pseudo-join operation to force a key update. Also TEKs might be periodically refreshed. The new TEKs can be encrypted in the current TEKs.

In a fully-trusted system

Because the base station is fully trusted, they can have access to both the key encryption key and the traffic encryption keys. Some decryption operations can be performed by them. In this system, each base station decrypts the data traffic received from GM and encrypts it, then broadcasts in its cell.

$$GM => CG : \{data\_traffic\}_t$$

$$\forall s_i \in CG,\ s_i => M_{s_i} : \{data\_traffic\}_{c_{s_i}}$$

There are some weaknesses in this protocol. First, this protocol has high overhead. In case of semi-trusted system, in order to recover the session key, the mobile host need to do two additional decryption operations. Second, it doesn't consider the reliability problem of rekeying. Third, because the traffic between GM and base stations are only encrypted by session key and it doesn't change after member's leaving, a leaving member may understand this traffic.

5.2 Key agreement protocols

In this subsection some key agreement protocols, which means that each group member contributes its share to the group key, are discussed. The first protocol uses one-way function on shares of group members to generate group key. Other protocols are based on the Diffie-Hellman key exchange by extending two parties to multiparty.

5.2.1 A Generic multi-party protocol

Asokan and Ginzboorg [AG00] modified the generic two-party protocol called encrypted key exchange [BM92] and extended it into a multi-party protocol. The protocol is based on a weak shared key. All group members $M_1, \ldots, M_n$ share a weak secret *P*. M_n is the leader and has a key pair (E, D). The protocol works as follows:

(1) $M_n \rightarrow$ all: $M_n, \{E\}_p$

(2) $M_i \rightarrow M_n$: $M_i, \{\{R_i,S_i\}_E\}_P$, i = 1, .. n-1

(3) $M_n \rightarrow M_i$: $\{S_j, j = 1, .. , n\}_{Ri}$, i = 1,.. ,n-1

(4) $M_i \rightarrow M_n$: $M_i, \{S_i, H(S_1,S_2,.. S_n)\}_K$, for some i

In the first step, M_n sends its encryption key *E* to all members. In step two, every member M_i generates a random string R_i, which is used as the symmetric key between itself and M_n, and random share S_i, encrypts them using E and returns them to M_n. In step three, M_n send all the random share S_i encrypted by R_i to every member. Then every member can calculate the session key using one-way function $K = f(S_1, S_2, .., S_n)$. The last step is used for key confirmation.

It is pointed out that this protocol has the following shortcoming. First, to prevent replay attack, *E* cannot be a long term public encryption key. It should be refreshed for each run of the protocol. However, generating a new key pair each time is expensive. Second, some properties (e.g., a product of large prime numbers, as in RSA) of the key *E* may be utilized by the attacker to attempt an dictionary attack on *P*(*E*). Only the unpredictable parts of *E* should be encrypted with *P*.

5.2.2 Burmester and Desmedt's Protocol

Burmester and Desmedt constructed a broadcast system to generate a group key [BD94]. This protocol includes only three steps:

(1) Each M_i selects its random exponent N_i and broadcasts $z_i = \alpha^{N_i}$

(2) Each M_i computes and broadcasts $X_i = (z_{i+1} / z_{i-1})^{N_i} = (\alpha^{N_{i+1}} / \alpha^{N_{i-1}})^{N_i}$. Here, the indices of M_i are taken in a cycle, so M_{n+1} is M_1 and M_n is M_0.

(3) Each M_i compute the key $K = z_{i-1}^{nN_i} \cdot X_i^{n-1} \cdot X_{i+1}^{n-2} \cdots X_{i-2} \bmod p$, where p is a prime number.

The group key will be $K = \alpha^{N_1N_2+N_2N_3+\cdots+N_nN_1}$, which is deducted as follows:

Set $A_{i-1} = (z_{i-1})^{N_i} = \alpha^{N_{i-1}N_i} \pmod p$, $A_i = (z_{i-1})^{N_i} \cdot X_i = \alpha^{N_{i-1}N_i} \cdot (\alpha^{N_{i+1}} / \alpha^{N_{i-1}})^{N_i} = \alpha^{N_iN_{i+1}} \pmod p$, $A_{i+1} = (z_{i-1})^{N_i} \cdot X_i \cdot X_{i+1} = \alpha^{N_{i+1}N_{i+2}} \pmod p$, and so on. Then, $K_i = A_{i-1} \cdot A_i \cdot A_{i+1} \cdots A_{i-2} = \alpha^{N_{i-1}N_i} \cdot \alpha^{N_iN_{i+1}} \cdot \alpha^{N_{i+1}N_{i+2}} \cdots = \alpha^{N_1N_2+N_2N_3+\cdots+N_nN_1}$.

The problem with this protocol is, according to [KPT00], that most of the members need to change their random exponent on every membership change event.

5.2.3 GDH.2 and extensions

Group Diffie-Hellman (GDH) is a class protocols presented by Steiner et al. [STW96][STW00]. GDH.2 and GDH.3 are two of them. These protocols are natural extensions of 2-party Diffie-Hellman key exchange to the n-party case. In 2-party case, each member selects its secret share and sends exponent of this secret share α^{Ni} to its peer. Both members can calculate the key α^{N_1N2} by using its own share. In n-party case, if a member M_i knows exponent of secret share of other members $\alpha^{N_1N2\cdots N_{i-1}N_{i+1}\cdots N_n}$, then using its own share, it can calculate group key as $\alpha^{N_1N2\cdots\cdots N_n}$.

Suppose N_i is the secret exponent of member M_i and α is a generator in the algebraic group. GDH.2 works as follows:

(1) $M_i \rightarrow M_{i+1}$: { $\alpha^{\frac{N_1\ldots N_i}{N_j}}$ | j ∈ [1,i] }, $\alpha^{N_1\ldots N_i}$ i ∈ [1, n-1]

(2) $M_n \rightarrow$ ALL M_i: { $\alpha^{\frac{N_1\ldots N_n}{N_j}}$ | j ∈ [1, n-1] }

Stage (1) consists of n-1 rounds. In every round i, M_i unicasts M_{i+1} a collection of i values. Of these, i-1 items are *intermediate*, which are, $\alpha^{N_2N_3\ldots N_i}$, $\alpha^{N_1N_3\ldots N_i}$ … $\alpha^{N_1N2\ldots N_{i-1}}$, and one is *cardinal*, $\alpha^{N_1\ldots N_i}$. When upflow reaches M_n, M_n can calculate the group key as $\alpha^{N_1N_2\ldots N_n}$. Also, M_n calculates the intermediate values $\alpha^{N_2N_3\ldots N_n}$, $\alpha^{N_1N_3\ldots N_n}$ …, $\alpha^{N_2N_3\ldots N_{n-2}N_n}$. In stage (2), M_n broadcasts these n-1 intermediate values to all group members. When member M_i receives these broadcast intermediate, it can calculate the group key as $\alpha^{N1N_2\ldots N_n}$ by using its own share N_i to the corresponding intermediate. Figure17 shows GDH.2 with 4 members.

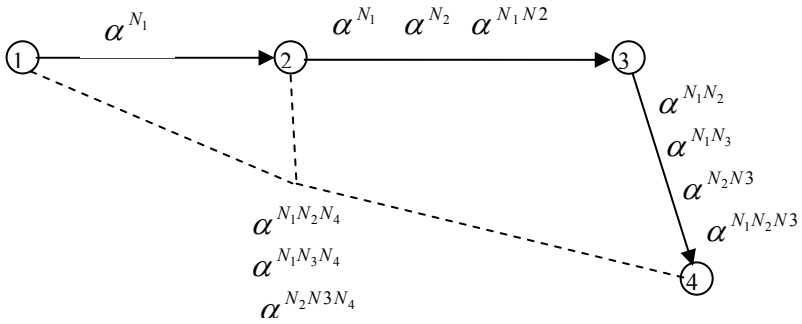


Figure 17: GDH.2 with n=4

In GDH.2, for every M_i, a total of (i+1) exponentiations are required and the computational burden increases with the group size.

To reduce the computational burden on M_i, GDH.3 protocol is provided. In the upflow stage of GDH.3, only one item, *cardinal*, instead of i items in GDH.2, is calculated and sent to the next member. Thus computation overload of i-1 intermediate items is reduced. GDH.3 consists of four stages.

(1) $M_i \rightarrow M_{i+1}$: $\alpha^{\prod N_p | p \in [1,i]}$ $i \in [1, n-2]$

(2) $M_{n-1} \rightarrow$ ALL : $\alpha^{\prod N_p | p \in [1,n-1]}$

(3) $M_i \rightarrow M_n$: $\alpha^{\frac{\prod N_p | p \in [1,n-1]}{N_i}}$

(4) $M_n \rightarrow$ ALL : $\{ \alpha^{\frac{\prod N_p | p \in [1,n]}{N_i}} | i \in [1, n] \}$

First stage collects the exponentiation of every member up to M_{n-1}. After first stage is complete, M_{n-1} obtains $\alpha^{\prod N_p | p \in [1,n-1]}$. In stage 2, M_{n-1} broadcasts $\alpha^{\prod N_p | p \in [1,n-1]}$ to every member. In stage 3, every member factors out (divided by) its own exponent from $\alpha^{\prod N_p | p \in [1,n-1]}$ and sends the result to M_n. In stage 4, M_n raises every message received in stage 3 with its exponent and returns the result back to respective member. Thus every member can now calculate the Key as $\alpha^{\prod N_p | p \in [1,n]}$.

The problem of GDH.3 is that n-1 unicast messages are sent to M_n in stage 3, which may congest M_n.

Asokan and Ginzboorg [AG00] provided a similar extension to GDH.2. In their method, all members share a password *P*. Each member M_i generates a secret share S_i. The protocol works as follows:

(1) $M_i \rightarrow M_{i+1}$: $g^{s_1 s_2 \ldots s_i}$, i =1, .. n-2

(2) $M_{n-1} \rightarrow$ ALL: $g^{s_1 s_2 \ldots s_{n-1}}$

(3) $M_i \rightarrow M_n$: $\{g^{s_1 s_2 \ldots s_{n-1} \hat{s}_i / s_i}\}_P$

(4) $M_n \rightarrow M_i$: $\{g^{s_1 s_2 \ldots s_n \hat{s}_i / s_i}\}_P$

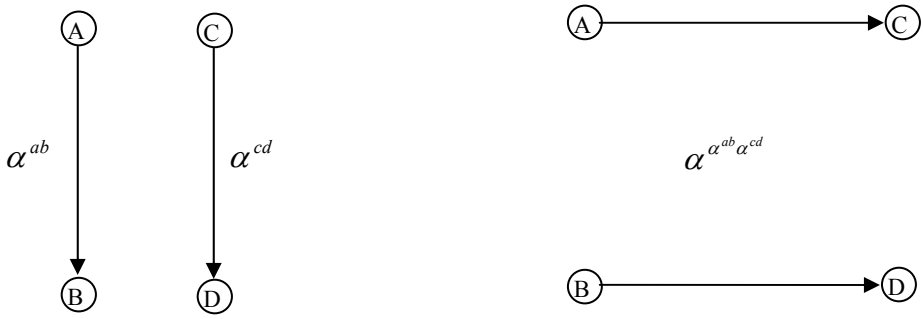
All above 4 stages are same as those in GDH.3, except stage 3. In stage 3, every member encrypts the revised intermediate key using share password *P* and sends it to M_n. In stage 4, instead of using multicast as in GDH.3, M_n unicasts the result to every member. It is suggested in the paper, the following step 5 may be used for verification. Some members broadcast the key message to make sure some other members decide the same key.

(5) $M_i \rightarrow$ ALL: M_i, $\{M_i, H(M_1, M_2, .. , M_n)\}_K$ for some i

This method will provide good forward secrecy. However, it requires a shared password *P*. To get such a password is a problem itself.

5.2.4 The Hypercube Protocol and extensions

Hypercube protocol was proposed by Becker and Wille [BW98]. It requires minimum number of rounds. The basic idea is as follows (Figure 18): suppose four nodes A, B, C, D are arranged in a square. Suppose a, b , c, d are their random share for 2-party DH. In the first round, A and B exchange keys in the usual way, resulting in $K_{AB}=\alpha^{ab}$. C and D do the same, resulting in $K_{CD}=\alpha^{cd}$. In the next round, A exchanges keys with C, using a 2-party DH. So do B and D. After two rounds, all four members will have the same key $K=\alpha^{\alpha^{ab}\alpha^{cd}}$.



Round 1 Round 2

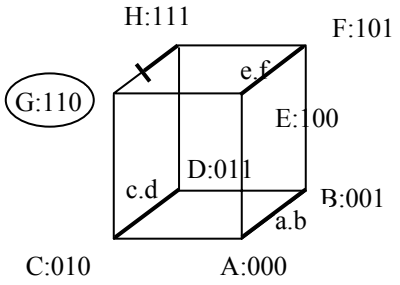
Figure 18: 2^2-cube key exchange

Now suppose there are 2^3 participants. They are arranged as the vertices in 3-dimensional hypercube. In the first 2 rounds, each participant can get a 4-party DH using the above method. In the 3rd round, each participant of a square (face) exchanges with peer of opposite square(face) using the achieved 4-party secret keys as the exponents. This process continues for 2^4 participants and so on. That is, in jth round, each participant performs a two–party DH with its peer on the jth dimension using the key of the j-1th round as its secret exponent. After d round, 2^d participants will have the same secret.

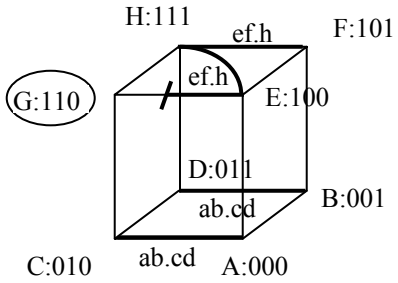
For the case where the number of participants n is not a power of 2, Becker and Wille present a protocol called *Octopus protocol* . If $2^d < n < 2^{d+1}$, the first 2^d participants play the role of central controllers. The rest of the participants form *wards* that are attached to one of the central nodes. First, the controllers do Diffe-Hellman key exchange with the wards. Then, the controllers perform 2^d-cube exchange using the products of the ward keys gathered in the first stage. Finally the key derived in the second stage is distributed to the wards.

The Octopus protocol works well for adding new member to the group. However, it's very inefficient when a member leaves the group. Another problem of Octopus protocol is that a two-party key exchange may fail. Becker and Wille didn't make clear how to deal with the problem. Asokan and Ginzboorg [AG00] provided a solution to deal with such faults.

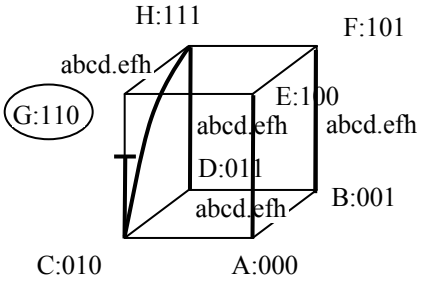
In Asokan and Ginzboorg's method, if a node finds its peer partner in a given round is faulty, it uses a distributed algorithm to find a suitable non-faulty partner. The partner selection algorithm tries the closer partner first in term of Euclidean distance between the node address. If there are 2^d participants, every node has a d-bit address. For any round *k*, the number of potential partners for a given node is bounded by 2^{k-1}. In this round, a given node's potential partners should have the same first *d-k* bits but different *k*th bit. Among these potential partners, a given node (suppose *N*) will try to do Diffie-Hellmen exchange with the node that has the least distance of address from the given node first (i.e., the least Hemming distance between the two node's addresses). If that fails, *N* tries the node with second least Euclidean distance. If there is more than one node having the second least Euclidean distance, the node with higher bit set has priority. If those attempts fail again, *N* will try the node with the third least Euclidean distance, and so on. The process is illustrated using an example in Figure 19:



Round 1 pairwise exchanges



Round 2 pairwise exchanges



Round 3 pairwise exchanges

Figure 19: Fault tolerant 8-party key exchange

Participants have 3-bit addresses and are labeled A to H. Their share contributions are represented as the corresponding lower case letters. The bold solid line represents DH exchange between two participants. Suppose G:110 is faulty. In round k =1, H:111 will select a node with the same first 2 bits but different last bit address, i.e. G:110, as partner to exchange share. This attempt fails. Because there is only one (i.e., 2^{1-1}) candidate, H does nothing more. In round k = 2, E:100's potential partners are those nodes having the same 1st bit but different 2nd bit address. There are two candidates, G:110 and H:111. G:110 has the less Euclidean distance from E:100 than H:111. E:100 will select G:110 as partner first. This attempt will fail. Then E will try H:111 as partner. This attempt will succeed. In round 3, C:010's potential partners are those nodes having different 1st bit address from C. C:010 selects G:110, which has the least Euclidean

distance from C, as partner and fails. Then C tries H:111, which has second least distance from C, as partner and will succeed.

A comparison of these protocols is given below:

Protocol	**messages**	**message size**	**total exponentiations**	**exchanges***	**rounds**
Generic multi-party	n+1	2n-1	- -	n+1	3
Burmester et al's protocol	2n (broadcast)	2n	n^2+2n	2n	2
GDH.2	n	(n-1) (n/2+2) -1	(n+3)n/2-1	n	n
GDH.3	2n-1	3(n-1)	5n-6	2n-1	n+1
Hypercube	$n \log_2 n$	$n \log_2 n$	$n(1+\log_2 n)$	$0.5n \log_2 n$	$\log_2 n$
Octopus	$3n+2^d$ (d-3)	$3n+2^d$ (d-3)	$4n+2^d$ (d-3)	$2n+2^{d-1}$ (d-4)	d+2

Table 7: A comparison of the performance

Here, for d, $2^d < n < 2^{d+1}$

* exchange means a DH exchange by two parties simultaneously or an exchange from one party to another at one time.

6. Conclusion

Key establishment is a key issue in secure communication. In this paper, we reviewed a variety of key management protocols for group communication in wired and wireless networks. We analyzed these protocols for security vulnerabilities and also discussed the pros and cons of these protocols and gave performance comparisons among related approaches. However, there are different requirements for different applications and environments and no panacea exists to solve all the problems. For specific application, the most suitable protocol may be chosen to implement.

References

[AD94] A. Aziz and W. Diffie, "Privacy and Authentication for Wireless Local Area Networks," *IEEE Personal Communications*, vol. 1, pp. 25-31, 1994

[AG00] N. Asokan and Philip Ginzboorg. Key Agreement in Ad-hoc Networks. *Elsevier Preprint*, 2000. To appear.

[ASW00] Michel Abdalla, Yuval Shavitt, and Avishai Wool. Key Management for Restricted Multicast using Broadcast Encryption. *IEEE/ACM Trans. On Networking*, 8(4), pp. 443--454, August 2000

[BCY91] M. J. Beller, L.F. Chang, and Y. Yacobi, "Privacy and Authentication on a Portable Communications System," in *Proceedings of GLOBECOM'91*, pp. 1922-1927, IEEE Press, 1991

[BCY92] M. J. Beller, L.F. Chang, and Y. Yacobi, "Security for Personal Communication Services: Public-key vs. Private Key Approaches," in *Proceedings of Third IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'92)*, pp. 26-31, IEEE Press, 1992

[BCY93] M. J. Beller, L.F. Chang, and Y. Yacobi, "Privacy and Authentication on a Portable Communications System," *IEEE Journal on Selected Areas in Communications*, vol. 11, pp. 821-829, Aug. 1993

[BY93] M. J. Beller and Y. Yacobi, "Fully-Fledged two-way Public Key Authentication and Key Agreement for Low-Cost Terminals," *Electronics Letters*, 29, pp. 999-1001, May 1993

[BD94] Mike Burmester and Yvo Desmedt. A Secure and Efficient Conference Key Distribution System. *Proc. Advances in Cryptology-UROCRYPT'94*, May 1994

[BM92] Steven M. Bellovin and Michael Merrit. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 1992

[BM98] C.Boyd and A. Mathuria. Key Establishment Protocols for Secure Mobile Communications: A Selective Survey. *Information Security and Privacy*, LNCS 1438, Springer-Verlag, 1998, pp.344-355

[BP98] C.Boyd and D.-G..Park. Public Key Protocol for Wireless Communications. in *The 1st International Conference on Information Security and Cryptology(ICISC'98)*, pp.47—57, 1998.

[BQ95] P.Beguin and J.J.Quisquater. Fast server-aided RSA signatures Secure Against Active Attacks. In *Crypto'95*, pp. 57-69, 1995.

[BR00] D. Bruschi., E. Rosti Secure multicast in wireless networks of mobile hosts: protocols and issues. to appear in *ACM-Baltzer MONET Journal, special issue on Multipoint Communication in Wireless Mobile Networks*, 2000.

[BW98] K. Becker and U. Wille. Communication Complexity of Group Key distribution. *Proc. Fifth ACM Conf. Computer and Comm. Security*, 1998

[CE99] I. Chang, R. Engel, D. Kanlur, D. Pendarakis, D. Saha. Key Management for Secure Internet Multicast using Boolean Function Minimization Techniques. *IEEE Infocomm 1999*.

[Dee88] Stephen E. Deering. Multicast Routing in Internetworks and Extended LANs. In *Proceedings of ACM SIGCOMM '88*, August 1988.

[FN94] Amos Fiat and Moni Naor. Broadcast Encryption. In *Advances in Cryptology – CRYPTO '93, LNCS 773*, pages 480-491, 1994.

[H98] G. Horn et al. Trialling secure billing with trusted third party support for UMTS applications. *Proceedings of the 3rd ACTS Mobile Communications Summit*, Rhodes, June 1998, pp574-579.

[HMR97] Harney Hugh, Carl Muckenhirn, Thomas Rivers. Group Key Management Protocol Architecture, Request for comments (RFC) 2093, Internet Engineering Task Force, March 1997

[HP98] G.Horn and B. Preneel. Authentication and Payment in Future Mobile Systems, *Technical Report ESAT-COSIC Report 98-2*, Department of Electrical Engineering, Katholieke Universiteit Leuven, Feb., 1998.

[Iolus] Suvo Mittra. Iolus: A Framework for Scalable Secure Multicasting. In *proceedings of ACM SIGCOMM '97*, 1997.

[K53] M. Karnaugh. The Map Method for Synthesis of Combinational Logic Circuits. *Transaction AIEE, Communications and Electronics*, Vol. 72, pp. 593-599, Nov, 1953.

[KPT00] Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In S. Jajodia, editor, *7th ACM Conference on Computer and Communications Security*, pages 235–244, Athens, Greece, Nov. 2000. ACM Press.

[KPT01] Y. Kim, A. Perrig, and G. Tsudik. Communication-efficient group key agreement. In *Information Systems Security, Proceedings of the 17th International Information Security Conference IFIP SEC'01*, 2001.

[LHYC98] Seungwon Lee, Seong-Min Hong, Hyunsoo Yoon and Yookun Cho. Accelerating Key Establishment Protocols for Mobile Communication.

[LNN01] Jeff Lotspiech, Moni Naor, and Dalit Naor. Subset-Difference based Key Management for Secure Multicast. *Internet Draft*, *draft-irtf-smug-subsetdifference-00.txt*, July 2001

[M95] C. Meadows. Formal Verification of Cryptographic Protocols: A Survey. in *Advances in Cryptology – ASIACRYPT'94*(J. Pieprzyk and R. Safavi-Naini, eds.), vol. 917 of Lecture Notes in Computer Science, pp.135-150, Springer-Verlag, 1995. Invited lecture

[MOV96] Alfred J.Menezes, Paul C. van Oorschot and Scott A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, Oct. 1996, pp.544

[NNL01] Dalit Naor, Moni Naor, and Jeff Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. To appear in *Crypto 2001*, 2001

[P97] C.S.Park, "On Certificate-Based Security Protocols for Wireless Mobile Communication Systems", *IEEE Network*, pp.50-55, Sept./Oct. 1997,

[STW00] Michael Steiner, Gene Tsudik and Michael Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 11, No. 8, August 2000

[STW96] M. Steine, G. Tsudik, and M. Waidner. Diffie-Hellman Key distribution extended to groups. *Third ACM Conf. Computer and Comm. Security*, pp.31-37, Mar 1996

[WCSWP99] Marcel Waldvogel, Germano Caronni, Dan Sun, Nathalie Weiler, Bernhard Plattner, The VersaKey Framework: Versatile Group Key Management, *IEEE Journal on Selected Areas in Communications*, 17(9), pp 1614-1631, September 1999

[WGL97] C. K. Wong, M. Gouda. Secure Group Communications Using Key Graphs. S. S. Lam, Department of Computer Sciences, University of Texas at Austin, Technical Report 97-23, July 28, 1997

[WS98] D. A. McGrew, A. T. Sherman. Key , Feb, 1999.

[YS89] Y.Yacobi and Z.Shumuely. On Key Distribution Systems. *Advances in Cryptography-Crypto'89*, Springer-Verlag, pp.344-355.