P2P-HGKM: An Efficient Hierarchical Group Key Management Protocol for Mobile Ad-Hoc Networks

by

Peng Peng

B.Sc., Peking University, 2000

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
Master of Science
in
THE FACULTY OF GRADUATE STUDIES
(Department of Computer Science)

we accept this thesis as conforming
to the required standard

The University of British Columbia

July 2004

© Peng Peng, 2004
In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Name of Author (please print)
PENG PENG

Date (dd/mm/yyyy)
20/08/2004

Title of Thesis:
P2P-HGKM: An Efficient Hierarchical Group Key Management Protocol for Mobile Ad-Hoc Networks

Degree:
Master of Science

Year:
2004

Department of
Computer Science

The University of British Columbia
Vancouver, BC Canada
Abstract

Wireless networks are growing rapidly in recent years and research in wireless technology is gaining increasing attention, especially in the area of security. Our thesis addresses an interesting security problem in wireless ad-hoc network: the P2P dynamic secure group key establishment. To secure group communication in an ad-hoc network, a group key shared by all group members is required. This group key should be updated when there are membership changes in the group, e.g. when a new member joins or a current member leaves. In this thesis, we propose a novel efficient P2P hierarchical group key management protocol for ad-hoc networks. We introduce a two-level hierarchical structure and a new scheme of group key update. The idea is to divide the group into subgroups; each maintains its subgroup key and links with other subgroups in a ring structure. A salient feature of our scheme is that group key gets updated and managed in a P2P manner. By introducing subgroups, public encryptions, and unicasts in key updates, will be limited in one subgroup and computation load is distributed to many hosts. Both theoretical analysis and experiment results show that our protocol performs well for the key establishment problem in ad-hoc network in terms of both computing cost and bandwidth overhead.
Contents

Abstract.........................................................................................................................ii

Contents.......................................................................................................................iii

List of Figures ...............................................................................................................vi

List of Tables.................................................................................................................. viii

Acknowledgements...................................................................................................... x

Chapter 1  Introduction .................................................................................................1
  1.1  Motivation ..............................................................................................................1
  1.2  Overview of P2P-HGKM Protocol ......................................................................4
  1.3  Thesis Contribution .............................................................................................6
  1.4  Synopsis ...............................................................................................................7

Chapter 2  Background and Related Work................................................................. 8
  2.1  Multicast Routing in Ad-Hoc Network ...............................................................8
  2.2  Public Key and Symmetric Key Encryption .......................................................10
  2.3  Group Key Establishment Protocols ..................................................................13
      2.3.1  Key Agreement Protocols .........................................................................13
      2.3.2  Tree-Based Key Management Protocols ..................................................16
Chapter 3 Design and Implementation ........................................... 22

3.1 Overview of the P2P-HGKM System ........................................ 22

3.2 Initialization of P2P-HGKM System ........................................ 26
  3.2.1 Multicast the BUILD_REQUEST and Reply (Step 1 and 2) ...... 28
  3.2.2 Dividing Group into Subgroups (Step 3) .............................. 30
  3.2.3 Initialize and Distribute the Subgroup Key (Step 4) ............. 32
  3.2.4 Initialize and Distribute the Group Key (Step 5) ................. 33

3.3 Key update at Join Event ..................................................... 36

3.4 Key Update at Leave Event .................................................. 39
  3.4.1 Non-Leader Node Leave ............................................... 39
  3.4.2 Leader Node Leave – Merge ....................................... 41
  3.4.3 Leader Node Leave - Select a new subgroup leader ............. 45

Chapter 4 P2P-HGKM Performance and Evaluation .......................... 49

4.1 P2P-HGKM Security Analysis .............................................. 50

4.2 P2P-HGKM Experiment Overview ........................................ 51

4.3 Computational Cost ......................................................... 52
  4.3.1 Comparing with other protocols ................................. 52
List of Figures

Figure 1.1 802.11 Wireless LAN ...................................................... 2
Figure 1.2 Ad-hoc networks............................................................ 2
Figure 1.3 Members of multicast group are divided into subgroups........... 5
Figure 1.4 Subgroups link in a ring structure...................................... 5
Figure 1.5 $K_{\text{group}}$ propagates along the subgroup ring.................... 6
Figure 2.1 ODMRP........................................................................ 9
Figure 2.2 Public key encryptions..................................................... 11
Figure 2.3 Diffie-Hellman key exchange........................................... 13
Figure 2.4 GDH 2.0 algorithm......................................................... 15
Figure 2.5 GDH 2.0 works in a group with 4 members......................... 15
Figure 2.6 Tree-Based key management protocol............................... 17
Figure 2.7 Step 2 of AT-GDH algorithm.......................................... 19
Figure 2.8 Step 2 of AT-GDH algorithm.......................................... 20
Figure 3.1 Multicast group is divided into subgroups.......................... 23
Figure 3.2 Subgroups are linked by joint leaders into a ring structure........ 24
Figure 3.3. Key update at Leave Event .................................................. 25
Figure 3.4 Software layer ................................................................. 26
Figure 3.5 Broadcast the BUILD_SUBGROUP_REQUEST .......................... 28
Figure 3.6 Build subgroups ............................................................... 30
Figure 3.7 Initialize and distribute subgroup and group key ....................... 33
Figure 3.8 Distribute $K_{group}$ along the subgroup ring ............................ 34
Figure 3.9 Key update at Join Event .................................................... 38
Figure 3.10 Non-leader node leave ...................................................... 40
Figure 3.11 Leader node leave and subgroup merge ................................ 44
Figure 3.12 Leaver node leave and a new leader is selected ....................... 47
Figure 4.1 Experiment result: public key encryption/decryption .................. 56
Figure 4.2 Experiment result: symmetric key encryption .......................... 57
Figure 4.3 Experiment result: symmetric key decryption .......................... 58
Figure 4.4 Experiment result: number of unicast .................................. 61
Figure 4.5 Experiment result: message size of unicast ............................ 63
Figure 4.6 Experiment result: number of unicast .................................. 64
Figure 4.7 Experiment result: message size of unicast ............................ 65
Figure 5.1 Complex structure of subgroups ........................................ 69
List of Tables

Table 2.1 DES, 3DES and AES ................................................................. 11
Table 2.2 Communication and computation cost of GDH 2.0....................... 15
Table 2.3 Communication/Computation cost of Tree-Based key management .... 18
Table 2.4 Communication/Computation cost of GDH 2.0 and AT-GDH ............ 20
Table 3.1 Communication cost of start-up .................................................... 35
Table 3.2 Computation cost of start-up ......................................................... 35
Table 3.3 Computation cost at Join Event ...................................................... 38
Table 3.4 Communication cost at Join Event .................................................. 38
Table 3.5 Computation cost at Leave Event (non-leader node) ....................... 41
Table 3.6 Communication cost at Leave Event (non-leader node) .................... 41
Table 3.7 Computation cost at Leave Event (merge) ..................................... 44
Table 3.8 Communication cost at Leave Event (merge) .................................. 45
Table 3.9 Computation cost at Leave Event (replicate a new leader) ............... 47
Table 3.10 Communication cost at Leave Event (replicate a new leader) ......... 48
Table 4.1 A summary of computation cost ..................................................... 52
Table 4.2 Computation cost comparison

Table 4.3 Summary of communication cost

Table 4.4 Communication cost comparison
Acknowledgements

I am greatly indebted to my supervisor, Dr. Son Vuong. Completing this thesis would have been difficult without his inspiration and encouragement. I learnt a lot from Dr. Vuong in my years at UBC. I am also grateful to Dr. Konstantin Beznosov for being my second reader.

Peng Peng

The University of British Columbia

August 2004
1.1 Motivation

Wireless networks are growing rapidly in recent years. Wireless technology is gaining more and more attention from both academia and industry. Most wireless networks used today, e.g. the cell phone networks and the 802.11 Wireless LAN, are based on the wireless network model with pre-existing wired network infrastructures. Packets from source wireless hosts are received by nearby base stations, then injected into the underlying network infrastructure and then finally transferred to destination hosts, as shown in Figure 1.1. [13]
Another wireless network model, which is actively researched, is the ad-hoc network. This network is formed by only mobile hosts and requires no pre-existing network infrastructure. Hosts with wireless capability form an ad-hoc network instantly and packets can be delivered to any host in the network. Since there is no base station and no underlying network infrastructure in ad-hoc networks, some mobile hosts work as routers to relay packets from source to destination, as shown in Figure 1.2. It's very easy and economic to form an ad-hoc network in real time. Ad-hoc network is ideal in situations like battlefield or rescuer area where fixed network infrastructure is very hard to deploy.
Group-oriented network applications can be easily conducted in this new network environment. For example, in a conference room, users can form an ad-hoc network instantly with their wireless devices, e.g. notebook computers, PDAs, or even cell phones, without requiring any pre-installed cables or base stations. They can use this fast setup ad-hoc network for conducting a videoconference, sharing files or even playing interactive games.

Before the ad-hoc network concept can be widely accepted, several issues need to be resolved. For example, security is a major challenge. To secure group communication over an ad-hoc network, a group key \(K_{\text{group}}\) shared by all members of the group is needed. Packets will be encrypted with a group key \(K_{\text{group}}\) by the sender and decrypted with the same \(K_{\text{group}}\) by receivers. The group key \(K_{\text{group}}\) should always be kept confidential and should be updated timely whenever there are membership changes in the group. For example, if a new host joins the group, we should update the \(K_{\text{group}}\) so all past communication encrypted with the old \(K_{\text{group}}\) is kept secret to the new member. If a member node leaves, \(K_{\text{group}}\) should also be updated to keep future communications secret to the leaving node.

Many group key establishment protocols, e.g. [1], [2], [3], [4], [5], [6] and [17] have been proposed. Protocols in [1], [2], [3] and [17] are designed for group key establishment problem in general, and protocols in [4], [5] and [6] are proposed for group key establishment problem in ad-hoc networks. Unfortunately, none of these protocols are adaptive to the key establishment problem in ad-hoc networks. Key management protocols proposed in [1], [2], [3] and [17] depends on a reliable central key server or key management nodes, where as key agreement protocols in [4], [5] and [6] require
exponentiation computation. Besides, they are not communication efficient. A common problem of these protocols is that by design, they did not take into account the unique features of a wireless ad-hoc network. For example, we know that wireless devices, e.g. PDAs, cell phones and notebooks, are usually lightweight, powered by batteries and do not have strong computing ability. So by design, computational load should be distributed among multiple hosts. Heavy computation work on a single host should be avoided. Secondly, radio signal used in wireless communication propagates in every direction in the air. So in wireless network, a local broadcast to all neighbors within the radio range uses no significant more time and resource than a unicast. A well-designed group key establishment protocol should take these features into consideration and try its best to be adaptive to the ad-hoc network environment.

1.2 Overview of P2P-HGKM Protocol

The goal of our work is to propose a communication and computation efficient group key establishment protocol in ad-hoc network. The idea is to divide the multicast group into several subgroups, let each subgroup have its subgroup key shared by all members of the subgroup. Each subgroup has two leader nodes, and each leader is a joint leader of two subgroups. For example, in Figure 1.3, all member nodes are divided into three subgroups – $S_1$, $S_2$ and $S_3$, and $M_{19}$ is the joint leader of $S_1$ and $S_3$. We can assume that all subgroups are linked in a ring structure, as shown in Figure 1.4.
Figure 1.3 Members of multicast group are divided into subgroups.

Figure 1.4 Subgroups link in a ring structure

Without the concept of subgroup, an update $K_{\text{group}}$ would have to be encrypted and unicast to each individual host. This is neither computational nor communication
efficient. By introducing subgroups and subgroup keys, updated $K_{\text{group}}$ can be encrypted with subgroup keys and multicast to the corresponding subgroups. For example, in Figure 1.5, $K_{\text{group}}$ is updated by $M_{ij}$, encrypted with subgroup key $(K_i)$ of subgroup $S_i$ and multicast $(K_{\text{group}})K_i$ to $S_i$. When $M_8$ - the other leader node of $S_i$, receives this message, $M_8$ decrypts $(K_{\text{group}})K_i$ and gets $K_{\text{group}}$. Since $M_8$ is also the leader of $S_2$, $M_8$ will then encrypt $K_{\text{group}}$ with $K_2$ and multicast $(K_{\text{group}})K_2$ to $S_2$. In this way, $K_{\text{group}}$ can be propagated along the subgroup ring and finally received by all members, as shown in Figure 1.5.

![Figure 1.5 $K_{\text{group}}$ propagates along the subgroup ring](image)

#### 1.3 Thesis Contribution

- We propose a new efficient method for solving the group key management problem in ad-hoc network. Our protocol provides efficient and reliable key management service and is well adaptive to the mobile environment of ad-hoc network.
• We introduce the idea of subgroup and subgroup key, and we uniquely link all subgroups into a ring structure. Our design eliminates the central key server. Instead all hosts work in a peer-to-peer fashion. We use P2P-HGKM as the name of our protocol. It stands for P2P Hierarchical Group Key Management Protocol.

• We design and implement P2P-HGKM protocol using Java and conduct extensive experiments and theoretical analysis to evaluate the performance of our protocol.

1.4 Synopsis

The rest of the thesis is organized as follows. Chapter 2 introduces some background knowledge, including multicast routing algorithms in ad-hoc network, the public key and symmetric key encryption system, and some related group key establishment protocols. Chapter 3 discusses the design and implementation details of the protocol. In Chapter 4, we evaluate the performance of the protocol and compare P2P-HGKM with several existing key establishment protocols. Finally in Chapter 5 we present concluding remarks and offer suggestions for possible future work.
Chapter 2

Background and Related Work

2.1 Multicast Routing in Ad-Hoc Network

The routing problem in ad-hoc network has been an active topic for many years. Some unicast routing algorithms, e.g. [7], and multicast routing algorithms, e.g. [8] [9] and [10], have been proposed earlier. Cordeiro et.al. summarized current multicast protocols for ad-hoc networks in [11]. Our key management protocol uses multicast to send protocol messages. Hence an efficient multicast routing algorithm is the basis of our protocol. We used the On-Demand Multicast Routing Protocol proposed in [10] due to its simplicity, efficiency and its support for both multicast and unicast.

ODMRP (On-Demand Multicast Routing Protocol) is a mesh-based protocol that uses a forwarding group concept (only a subset of nodes forward the multicast packets instead of using system wide broadcast). The multicast routes are established on demand
when a group member has packets to send but there does not exist a route to the multicast group, the requesting node first broadcasts a Join-Query packet to the entire network, as shown in Figure 2.1. When an intermediate node receives this Join-Query packet, it stores the upstream node ID into its routing table. It also stores the source ID and the sequence number in its message cache to detect any potential duplicate. If the Join-Query message already exists, it is discarded; otherwise, the Join Query message is rebroadcast to neighboring nodes.

Besides re-broadcasting the Join-Query packet when a multicast group member receives a Join-Query packet, they also broadcast a Join-Reply packet containing the upstream ID to its neighbors. When a neighbor node receives this Join-Reply packet, it checks if the upstream ID in Join-Reply packet matches its own node ID. If it does, the node realizes that it is on the path to the source and thus it belongs to the forwarding group (FG). The node sets its forwarding group flag (FG_FLAG) and then creates and

Figure 2.1 ODMRP. Source node broadcasts Join-Query and waits Join-Reply from receivers
broadcasts its own Join-Reply packet. So in this way, forwarding group members propagate the Join-Reply until it reaches the source node via the selected (shortest) path.

After establishing the forwarding group (FG), members of the multicast group can multicast packets to the whole group via FG, and only nodes in the FG will rebroadcast data packets received. As long as the source node has data to send, it will periodically send Join-Query packets to refresh the forwarding group and the routes.

As we will see in Chapter 3, most key management messages exchanged in our protocol are multicast messages, so we use ODMRP as the supporting multicast routing protocol for our system. Another reason for choosing ODMRP is because ODMRP supports unicast [12]. Hence we don’t need to include another unicast routing protocol in our system, which greatly simplifies our design.

We also combine ODMRP with our protocol by using some key management messages in our protocol for routing discovery. This further reduces the number of messages required in key update.

2.2 Public Key and Symmetric Key Encryption

Modern cryptography systems include symmetric key algorithms and public key algorithms. In symmetric key algorithm, the same key is used for encryption and decryption. Many symmetric key algorithms have been developed and widely used, e.g. DES, Triple DES and AES. We summarize these algorithms in Table 2.1.
Key Length (bits) | Plaintext (bits) | Cipher Text (bits) | Strong for use?
---|---|---|---
DES | 56 | 63 | 64 | No longer
3DES | 112 (2x 56) | 64 | 64 | Yes
AES | 128 | 128 | 128 | Yes
| 128 | 256 | 128 | Yes

Table 2.1 DES, 3DES and AES

Symmetric Key encryption algorithms, e.g. AES and 3DES, are widely used in massive data encryption because of their efficiency – they are much faster than most public key algorithms. Symmetric key algorithms require all parties to possess the secret key before communication. Thus we need a way to secretly distribute the key. This has always been a weak link. In 1976, two researchers at Stanford University, Diffie and Hellman, proposed a radically new kind of cryptosystem - the public key cryptosystem, to solve this problem [14]. In public key cryptosystem, a pair of keys – a public key and a private key is presented. The public key is openly known to all communication parties, while the private key is always kept confidential by the key owner. Deducing the private key from the public key should be exceedingly difficult. Figure 2.3 depicts how public key encryption works. If A wants to securely send a plain text $P$ to B, A first acquires B’s public key $E_b$ from B or from a trusted third party. Then A will encrypt $P$ with $E_b$ and send $(P)E_b$ to B. By receiving the encrypted message, B decrypts $(P)E_b$ with its private key $D_b$ and extracts the correct content of the message $P = ((P)E_b)D_b$.

![Figure 2.2 Public key encryptions](image)

Figure 2.2 Public key encryptions
One of the widely used public cryptosystem is the RSA system, developed by Rivest, Shamir and Adleman in 1978 [15]. The RSA algorithm is based on the difficulty of factoring large numbers. The public and private key is at least 1024 bits in the RSA system for good security. Each block of plaintext for encryption and decryption is also 1024 bits long, versus 64 bits cipher in DES and 128-bits cipher in AES. Despite its strong security, there is a drawback of RSA system. The RSA algorithm takes more time for encryption and decryption than symmetric key algorithms. So in practice, RSA system is usually used for key distribution, instead of massive data encryption.

Both symmetric key and public key encryption is used in our key management protocol. In our design, members of the multicast group $G$ have group key - $K_{\text{group}}$. $K_{\text{group}}$ is a symmetric key (128 bits AES key) and is for data encryption. Each member node also has the subgroup key(s) of the subgroup(s) they are in. For example, if $M_i$ is the joint leader of both $S_i$ and $S_2$, then $M_i$ will have subgroup key $K_i$ and $K_2$. Subgroup key is also a 128 bit symmetric key, which is used to encrypt the updated key $K_{\text{group}}$. A subgroup key is shared by all members of the subgroup. As a result when there is a membership change in $S_n$, $K_i$ is also updated.

Each node in our system (both group members and non-group members) also has a pair of public key (1024 bits RSA key): for node $M_i$, $E_i$ is its public and $D_i$ is its private key. Public key encryption is used to encrypt updated $K_{\text{group}}$ or subgroup keys when there is no valid subgroup/group key to use. For example, when a node joins the multicast group, this new member does not have any subgroup key or group key, thus we use its public key for encryption. The key update at Leave Event also uses public encryption. We will describe this in detail in Chapter 3.
2.3 Group Key Establishment Protocols

In general, key establishment protocols can be classified into two categories: key management (key distribution) and key agreement. In key management protocols, group key is usually created and updated by a central key server, and then securely distributed to all members. In key agreement protocols, each node contributes a fraction of the group key and the creation and update of group key is the joint work of all members.

2.3.1 Key Agreement Protocols

In Key agreement protocols, [2], [16] and [17], each member contributes a fraction of the group key and the group key is constructed by the joint work of all members. All key agreement protocols are extensions of Diffie-Hellman key exchange [14]. Hence we will first discuss Diffie-Hellman key exchange. Then we will briefly discuss GDH 2.0 (Group Diffie-Hellman) [2] as an example of key agreement protocol.

Diffie-Hellman key exchange was first proposed in 1976 by Diffie and Hellman to solve the problem of establishing a shared secret key via an insecure channel. The key exchange protocol works like this:

\[
\begin{align*}
A & \text{ picks } x \\
& \text{ A picks } x
\\
B & \text{ picks } y \\
& \text{ B picks } y
\\
A & \text{ computes } g^x \mod n \\
& \text{ A computes } g^x \mod n
\\
B & \text{ computes } (g^y \mod n)^x \mod n \\
& \text{ B computes } (g^y \mod n)^x \mod n
\\
& \text{ n = } g^y \mod n
\\
& \text{ n = } g^y \mod n
\\
\end{align*}
\]

Figure 2.3 Diffie-Hellman key exchange
If node A and B want to establish a shared key via an un-secure channel, they first agree on two large numbers, \( n \) and \( g \) where \( n \) is a prime and \( (n-1)/2 \) is also a prime. \( n \) and \( g \) can be public. Then A picks a large number \( x \) and keeps it secret, and B also picks a secret number \( y \).

A computes \( (g^x \mod n) \) and sends this to B. Similarly, B sends \( (g^y \mod n) \) to A. By receiving \( (g^x \mod n) \) from B, A computes \( ((g^x \mod n)^y \mod n) \). Similarly, B computes \( ((g^y \mod n)^x \mod n) \) where \( (g^x \mod n) \) is from A. After the key exchange, both A and B have \( (g^xy \mod n) \) and this is the established shared key. \( ((g^x \mod n)^y \mod n) = (g^x \mod n) = (g^y \mod n) \).

The security of Diffie-Hellman key exchange lays in the fact that, deducing \( x \) from \( (g^x \mod n) \) is hard. So even if an eavesdropper has \( (g^x \mod n) \) and \( (g^y \mod n) \) and \( g \) and \( n \), he still cannot deduce either \( x \) or \( y \), and thus cannot compute the shared key \( (g^xy \mod n) \). Despite its elegance, Diffie-Hellman key exchange is vulnerable to man-in-the-middle attack.

Michael Steiner, Gene Tsudik and Michael Waidner extended DH key exchange to N-Party key agreement – GDH 2.0 in [2]. Figure 2.4 is the GDH 2.0 algorithm and Figure 2.5 depicts how GDH 2.0 works in a group with 4 members.
Group Diffie-Hellman Algorithm (GDH 2.0)

Round i (1 ≤ i ≤ n - 1)

1. Member $M_i$ selects a random integer $r_i \in \mathbb{Z}_q^*$

2. $M_i$ sends $M_{i+1}: \alpha^{\prod_{k=1}^{i} r_k}$ and $\alpha^{\prod_{k=1}^{i} r_k/r_j}$, ∀ 1 ≤ j ≤ n

Round n:

1. Member $M_n$ selects a random $r_n \in \mathbb{Z}_q^*$

2. $M_n$ sends each $M_i$ a number $y_i = \alpha^{\prod_{j=1}^{n} r_j/r_i}$.

3. Each member $M_i$ then computes the final key as

$$S_n = y_i^{r_i} = \alpha^{\prod_{j=1}^{n} r_j}.$$
GDH.2 performs well in wired network. But applying GDH.2 directly to the group key establishment problem in ad-hoc network may not yield the same result.

Group key in key agreement protocols is a combination of contributions from all member nodes. For this reason, each membership change will lead to a change of the group key and will cause all members to conduct exponentiation computations. Generally, key agreement protocols are not scalable in large networks because

- The cost for updating $K_{\text{group}}$ at Join Event is same as Leave Event for key agreement protocols. But we know the key update at Join Event could be complete with much less cost if we use the existing group key ($K_{\text{group, old}}$) to encrypt the updated $K_{\text{group}}$.

- As shown in Figure 2.5, the last node in GDH 2.0 needs to do $(N-1)$ exponential computations, while the first node only needs two exponential computations. The unbalanced work load will make the last node a bottleneck of the key agreement protocol and a vulnerable node of the system.

2.3.2 Tree-Based Key Management Protocols

In key management protocols, a central key management server is usually present in the system, and group key are initialized, updated and distributed by the key management server. Wong et.al. proposed a Tree-Based key management protocol in [1]. We will briefly introduce Wong's protocol as an example of key management protocol here.
Assuming a multicast group has \( n \) members, \( M_1 \) through \( M_n \), and there is a central key server in the system that manages membership and keys. The central key server stores a key tree, as shown in Figure 2.6. Each node also stores all the keys from the leaf up to the root in the key tree. For example, in Figure 2.6, \( M_9 \) stores \( k_9 \), \( k_c \) and \( k_{root} \).

![Figure 2.6 Tree-Based key management protocol](image)

When a new node joins the group, the key server adds this new member as a leaf node in the key tree, updates all the keys from the new node up to the \( K_{root} \) in the key tree and securely distributes them. For example, in Figure 2.6, if \( M_9 \) is the new member, then \( K_C \) and \( K_{root} \) in the key tree should be updated. Updated \( K_C \) will be encrypted with \( K_{C,old} \) and multicast to \( M_7 \) and \( M_8 \) \( ((K_C)K_{C,old} \rightarrow \{M_7, M_8\}) \). Updated \( K_{root} \) will be encrypted with \( K_{root,old} \) and multicast to the whole group \( ((K_{root})K_{root,old} \rightarrow G) \). Updated \( K_{root} \) and \( K_C \) will be grouped together, encrypted with \( k_9 \) and unicast to \( M_9 \) \( ((K_{root}, K_C)K_9 \rightarrow M_9) \).

Key update at node leaving is more complicated. All the keys possessed by the leaving node should be updated in order to secure future communication. For example, in Figure 2.6, if \( M_9 \) is the leaving node, then \( K_C \) and \( K_{root} \) should be updated because they
are held by $M_9$. $K_A$ and $K_B$ are not affected by the leaving node and can be used to encrypt the updated key(s). Updated $K_{root}$ is encrypted with $K_A$ for $M_1, M_2, M_3$ ($(K_{root})K_A \Rightarrow \{M_1, M_2, M_3\}$), and encrypted with $K_B$ for $M_4, M_5, M_6$ ($(K_{root})K_B \Rightarrow \{M_4, M_5, M_6\}$). Updated $K_{root}$ and $K_C$ are grouped together, encrypted with $k_7$ for $M_7$, and encrypted $k_8$ with for $M_8$. $((K_{root}, K_C)K_7 \rightarrow M_7 ; (K_{root}, K_C)K_8 \rightarrow M_8)$

<table>
<thead>
<tr>
<th></th>
<th>Join</th>
<th>Leave</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Cost</td>
<td>$h$ multicast + 1 unicast</td>
<td>$(h-1) \times (d-1)$ multicast</td>
</tr>
<tr>
<td>Computation Cost</td>
<td>Request node: $(h-1)$ symm decr</td>
<td>Request node: 0</td>
</tr>
<tr>
<td></td>
<td>Non-Req node: $n \times d(l-1)$ symm decr</td>
<td>Non-Req node: $n \times d(l-1)$ symm decr</td>
</tr>
<tr>
<td></td>
<td>Key server: $2 \times (h-1)$ symm encr</td>
<td>Key server: $d \times (h-1)$ symm encr</td>
</tr>
</tbody>
</table>

* $h$ – height of key tree; $d$ – degree of key tree
symm decr – symmetric decryption symm encr – symmetric encryption

Table 2.3 Communication/Computation cost of Tree-Based key management

The tree-based key management works efficiently in wired networks with a well-selected parameter $d$. But the protocol may not work well in general ad-hoc networks because the protocol requires a powerful and reliable key server. This requirement is very hard to satisfy in an ad-hoc network. Wireless hosts in ad-hoc networks are mobile and connections are not reliable. Also, as mentioned before, they are usually lightweight devices like PDAs and cannot be a key server for the whole system. Heavy computation load may totally weight down this wireless device and fail the whole key management protocols.
2.4 Related Work

Protocols in [4], [5] and [6] are proposed for the key establishment problem in ad-hoc network and they are all key agreement protocols. Here we briefly discuss the AT-GDH protocol in [5].

The “Efficient Key Agreement for Ad-hoc Networks” (AT-GDH) proposed by Maarit in [5] is an extension of GDH 2.0 in ad-hoc network. The protocol has two steps. At the first step, a spanning tree is constructed, each member contributes a fraction to the group key and the contributions are gathered from leaf nodes up to the root, as shown in Figure 2.7. At the second step, the root node combines these key fractions and sends back the results to all nodes in the spanning tree.

a) Step 1.a

\[ K(3; 0) = k_3 \]
\[ K(3; 1) = \alpha^{k_3} \cdot \omega(K(3; 0)) \]
\[ K(3; 2) = \alpha^{k_3} \cdot \omega(K(3; 1)) \]
\[ k_3 = \omega(K(3; 2)) \]
\[ 3 \rightarrow e : \alpha^{k_3} \]

b) Step 1.b

Figure 2.7 Step 1 of AT-GDH algorithm. At step 1, contributions gather at the root node.
b) Step 2.b

Figure 2.8 Step 2 of AT-GDH algorithm. At step 2, root sends back the combination of contributions to all nodes.

<table>
<thead>
<tr>
<th>Communication Cost</th>
<th>GDH 2.0</th>
<th>AT-GDH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N-1 Unicast + 1 Multicast</td>
<td>2x N Unicast</td>
</tr>
<tr>
<td></td>
<td>N Round</td>
<td>2k x (log_2 N) Round</td>
</tr>
<tr>
<td>Computation Cost</td>
<td>O(N^2) Exponentiation.</td>
<td>O(N log_2 N) Exponentiation</td>
</tr>
</tbody>
</table>

Table 2.4 Communication/Computation cost of GDH 2.0 and AT-GDH

Unicast in AT-GDH can be parallel and exponentiation computation is distributed with the introduction of tree structure. From Table 2.4 we can see that AT-GDH requires fewer rounds of message exchanges and less exponential computations than GDH 2.0. Despite this improvement in performance, AT-GDH still suffers from the following problems:

- The cost of key update at Join Event is the same as Leave Event.
- Root node in the tree structure does heavy computation work and failure of the root node will collapse the key agreement services. The root node becomes a single failure node in the system.

- The number of unicast and exponentiation computation is still at the order of $O(N)$ and $O(N \log_{10} N)$. 

3.1 Overview of the P2P-HGKM System

Our system is composed of many wireless hosts that form an ad-hoc network instantly, as discussed in Chapter 1. Some (or all) of these hosts are members of a multicast group $G$ and they share a group key ($K_{\text{group}}$) for secure group communication.

To update $K_{\text{group}}$ at Join Event, we can use the $K_{\text{group}}$ before update to encrypt the updated $K_{\text{group}}$, and multicast $(K_{\text{group}})K_{\text{group-old}}$ to all group members($(K_{\text{group}})K_{\text{group-old}} \Rightarrow G)$.

Updating $K_{\text{group}}$ at leave event is more complicated and is the key question in key establishment problem. Because the leaving node also holds the old group key ($K_{\text{group-old}}$), so we cannot encrypt $K_{\text{group}}$ with $K_{\text{group-old}}$ or any key known to the leaving node, as we did at the key update of Join Event. A generic way to update $K_{\text{group}}$ is to encrypt the
updated $K_{\text{group}}$ with the public key ($E_i$) of each member ($M_i$) and unicast encrypted $(K_{\text{group}})E_i$ to $M_i$. The generic algorithm requires $(N-1)$ public key encryptions and $(N-1)$ unicasts and this is very inefficient.

So we introduce redundant keys and more complex structure in order to reduce the communication and computation cost for updating $K_{\text{group}}$ at Leave Event. Our idea is to divide $G$ into several subgroups based on geographic location, and let each subgroup have its subgroup key. Nodes in the same subgroup are close to each other and each subgroup has roughly the same number of nodes ($M$), as shown in Figure 3.1 (We assume all the nodes are evenly distributed in the system). We will discuss how we divide group $G$ into subgroups in more details in Chapter 3.2.2.

![Figure 3.1 Multicast group $G$ is divided into subgroups.](image)
Each subgroup has two leader nodes, which are the key management nodes and membership management nodes of the subgroup. Each leader node is also a joint leader of another subgroup, e.g. in Figure 3.1. $M_8$ is the joint leader of $S_1$ and $S_2$. All the subgroups are linked by joint leader nodes into a ring structure, as shown in Figure 3.2.

![Figure 3.2 Subgroups are linked by joint leaders into a ring structure](image)

We chose the two-level ring structure because the ring structure is simple and easy to manage. And with the ring structure, updated key can be propagated simultaneously in two directions along the ring, as shown in Figure 3.3.

By introducing subgroup and subgroup key, public encryption and unicasts at Leave Event are limited within one subgroup. For example, in Figure 3.3, if $M_8$ is the leaving node, then both $K_{\text{group}}$ and $K_I$ should be updated. In our protocol, the subgroup leader with a smaller ID is responsible for the key update in this case (see more detailed discussion in Chapter 3.4.1). Here in our example, $M_8$ is this leader node with a smaller ID ($M_8$'s ID < $M_{19}$'s ID), so $M_8$ will be responsible for updating $K_{\text{group}}$ and $K_I$. $M_8$ first
updates $K_i$, encrypts $K_i$ with $E_i$ and $E_{i9}$ (the public key of $M_i$ and $M_{i9}$) and unicasts $(K_i)E_i$ and $(K_i)E_{i9}$ to $M_i$ and $M_{i9}$ individually.

After updating $K_i$, $M_8$ updates $K_{\text{group}}$, encrypts $K_{\text{group}}$ with $K_2$ and $K_{i7}$ and propagates encrypted $K_{\text{group}}$ along the subgroup ring, as shown in Figure 3.3. When $M_{i7}$ - the other leader node of $S_2$, receives this message, $M_{i7}$ decrypts $(K_{\text{group}})K_2$ and gets $K_{\text{group}}$. Since $M_{i7}$ is also a leader of $S_3$, $M_{i7}$ will then encrypt $K_{\text{group}}$ with $K_3$ and multicast $(K_{\text{group}})K_3$ to $S_3$. In this way, $K_{\text{group}}$ can be securely propagated along the subgroup ring and finally received by all members.

Figure 3.3. Key update at Leave Event

We implement our P2P-HGKM protocol and the supporting ODMRP protocol in Java. As shown in Figure 3.4, ODMRP provides multicast and unicast routing for both group oriented applications and our P2P-HGKM protocol. P2P-HGKM provides reliable and efficient group key management service to applications. Below ODMRP is the MAC layer of ad-hoc networks.
We know each wireless host has a unique identifier to identify itself from other hosts, such as the MAC address of network device. We can use this unique identifier as the ID in our system. In the following discussion, we still use 1 to 22 to number nodes in our system instead of actual MAC address for the reason of simplicity.

In the remaining sections, we will discuss the specification of our P2P-HGKM protocol, including the start-up phase, key update at Join Event and key update and Leave Event.

3.2 Initialization of P2P-HGKM System

At start-up, the P2P-HGKM system does not have a group key nor any subgroups. So the first thing the system needs to do is divide the group into subgroups, initialize the
group key and subgroup keys, then distribute these keys securely to all members. This phase has five steps:

1) $M_s$, the member node with the largest ID, is picked as the start node by running the bully election algorithm among all member nodes [18].

$M_s$ multicasts a BUILD_SUBGROUP_REQUEST.

$M_s \Rightarrow G$: BUILD_SUBGROUP_REQUEST.

2) Each member node $M_j$, receives the BUILD_SUBGROUP_REQUEST, and sends back a reply REPLY_BUILD_SUBGROUP together with its public key $(E_j)$ to $M_s$.

$\forall M_j, M_j \in G, i \neq j, M_j \rightarrow M_s: <\text{REPLY\_BUILD\_SUBGROUP, } E_j>$

3) $M_s$ divides all member nodes into subgroups based on REPLY_BUILD_SUBGROUP messages received. Then $M_s$ multicasts the node-list of each subgroup to the group.

$M_s \Rightarrow G$: <$S_c>$, for all valid $c$.

4) By receiving the SUBGROUP_INFO messages from STEP 3, each member node stores the subgroup structures if it’s in that subgroup. Then subgroup leaders are selected and they will initialize the subgroup keys and securely unicast the subgroup key to member of the subgroup.

5) The start node – $M_s$, initializes the group key $(K_{\text{group}})$, encrypts $K_{\text{group}}$ and propagates $K_{\text{group}}$ along the subgroup ring to all group members, as shown in Figure 3.3.
Details of each step are presented in the following sub sections.

3.2.1 Multicast the BUILD_REQUEST and Reply (Step 1 and 2)

![Diagram of multicast process]

Figure 3.5 Broadcast the BUILD_SUBGROUP_REQUEST

1) $\exists M_i, M_j \in G$. $M_i$ is selected as the start node.

$M_i$, the member node with the largest ID, is picked as the start node by running the bully election algorithm among all member nodes [18]. $M_i$ will be responsible for multicasting BUILD_SUBGROUP_REQUEST (step 1), dividing multicast group $G$ into subgroups (step 3), and initializing the group key (in step 5).

28
2) $M_s \rightarrow G$: BUILD_SUBGROUP_REQUEST

$M_s$ broadcasts BUILD_SUBGROUP_REQUEST. (Only messages along path 19->13->16->9->20->21 is shown in Figure 3.5 to make the figure more readable.). The request message is also used as the ODMRP multicast route discovering message to further reduce the traffic.

3) $\forall M_i, M_j \in G, s \neq j, M_j \rightarrow M_s$: {REPLY_BUILD_SUBGROUP, $E_j$}

As discussed in Chapter 2.1, by receiving the REQUEST message, each node will keep forwarding it to its neighbors. Each node also saves messages in a cache memory so that duplicate message will be discarded. Members of the multicast group ($M_j \in G$) will send back a REPLY_BUILD_SUBGROUP message, together with its public key ($E_j$) to $M_s$, (as shown in Figure 3.5, $M_9$ replies to $M_{10}$)

The REPLY_BUILD_SUBGROUP message goes backwards to the start node, similar to the way JOIN-REPLY message in ODMRP propagates. The REPLY message also brings back member nodes’ IDs along the return path (in our example, the return path from $M_9$ to $M_{10}$ is $<M_9, M_{16}>$). So REPLY message from $M_9$ has two fields: public key of $M_9$ - $E_9$, and list of member nodes in the return path - $<M_9, M_{16}>$. We will see how $M_s$ uses the information in REPLY messages to divide $G$ into subgroups in 3.2.2.
3.2.2 Dividing Group into Subgroups (Step 3)

With the REPLY messages in Step 1, $M_i$ can divide $G$ into subgroups. First, $M_i$ inserts all member nodes in a tree structure rooted at $M_i$ based on the return path in all REPLY messages. As shown in Figure 3.6, $M_{19}$ is the root of the tree. Since the REPLY from $M_9$ includes $<M_9, M_{16}>$, which means $M_9$ comes after $M_{16}$ in the return path, we will insert $M_9$ as the child node of $M_{16}$, and $M_{16}$ as child node of $M_{19}$ into the tree structure.

Next, we will walk through the tree structure and divide all nodes into subgroups. We pre-define an average size of subgroup ($M$) based on the total number of members estimated in $G$. The value of $M$ will affect the performance of the protocol. A large subgroup means more public key encryption and more unicast during key update, while a small subgroup means a high possibility of subgroup leader being the leaving node. We will have discussion further about this in Chapter 4.
The traversal of the tree is in pre order, taking every $M$ node as a subgroup. The last node in each subgroup becomes the 2nd leader as well as the 1st leader for the next subgroup (e.g. $M_8$ is the 2nd leader of $S_1$ and the 1st leader of $S_2$). Also, the 1st of the first subgroup automatically becomes the 2nd leader of the last subgroup (e.g. $M_{19}$ is the 2nd leader of $S_3$). For example, in Figure 3.5, with $M = 4$, all nodes are divided into 3 subgroups:

Subgroup 1: $< M_{19}, M_1, M_3, M_8 >$. $M_{19}$ and $M_8$ are subgroup leaders.

Subgroup 2: $< M_8, M_9, M_9, M_{17} >$. $M_8$ and $M_{17}$ are subgroup leaders.

Subgroup 3: $< M_{17}, M_{14}, M_{10}, M_{12}, M_{19} >$. $M_{17}$ and $M_{19}$ are subgroup leaders.

As mentioned before, all subgroups form a ring structure with subgroup leaders as the joint nodes, as shown in Figure 3.2.
3.2.3 Initialize and Distribute the Subgroup Key (Step 4)

1) \(M_i \rightarrow G: \{<S_c>\}\) for all subgroups.

After dividing \(G\) into subgroups, \(M_i\) multicasts how it made the division (the member list of each subgroup) to \(G\). For example, in Figure 3.6, \(<S_i> = <M_{19}, M_3, M_8>\)

2) \(\forall M_j \in G\), if \(M_j \in S_c\), \(M_j\) stores \(<S_i>\).

By receiving the message from 1), \(M_j\) stores the subgroup information \(<S_c>\), if \(M_j \in S_c\).

3) \(M_i \rightarrow S_c: \{((K_c)_{E_p})\}\), for \(\forall M_p \in S_c, p \neq i\)

If \(M_i\) is the 1st leader node of \(S_c\), \(M_i\) will initialize \(K_c\), encrypt \(K_c\) with the public key of subgroup members, combine all the encrypted \(K_c\) in one message and multicast it to \(S_c\) (Here we combine all encrypted keys into one message and use one multicast instead of multiple unicasts to save time and communication cost).

4) \(M_p \in S_c, p \neq i\), \(((K_c)_{E_p})_{D_p}) = K_c\)

If \(M_p\) is a member of \(S_c\), \(M_p\) decrypts \((K_c)_{E_p}\) with \(D_p\) and get \(K_c\).
For example, in Figure 3.7, $M_{19}$ initializes $K_f$ for $S_1$, encrypts $K_f$ with the public key of $M_1$, $M_2$ and $M_8$ and multicasts $<(K_f)E_1, (K_f)E_2, (K_f)E_8>$ to $S_1$. When $M_1$ receives this message, $M_1$ decrypt $(K_f)E_1$ with $D_1$ and get the subgroup key $K_1 = ((K_f)E_1)D_1$.

### 3.2.4 Initialize and Distribute the Group Key (Step 5)

The last step of start-up phase is to initialize the group key ($K_{\text{group}}$) and distribute encrypted $K_{\text{group}}$ along the subgroup ring.

1) $M_s$ initializes $K_{\text{group}}$
2) $S_i, M_i \in S_i, M_i \Rightarrow S_i : (K_{\text{group}})K_i$

$M_i$ encrypts $K_{\text{group}}$ with $S_i$'s subgroup key and multicast to $S_i$.

3) When a leader node $M_j (M_j \in S_i)$ receives $(K_{\text{group}})K_i$, it decrypts $(K_{\text{group}})K_i$ with $K_i$ and gets $K_{\text{group}}$, where $K_{\text{group}} = ((K_{\text{group}})K_i)K_i$. (This applies to all subgroup leaders.)

4) Because $M_j$ is the joint leader of another subgroup, $\exists S_k, M_j \in S_k$ and $M_j \in S_i, M_j \Rightarrow S_k : (K_{\text{group}})K_k$

Joint leader works as a bridge of two subgroups and propagates $K_{\text{group}}$ along the subgroup ring structure to the whole group $G$, as shown in Figure 3.8.

![Figure 3.8 Distribute $K_{\text{group}}$ along the subgroup ring](image-url)
<table>
<thead>
<tr>
<th>Num of Unicast</th>
<th>( N )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of Unicast Msg</td>
<td>( N \times (1 + [E] + N \times [ID]) )</td>
</tr>
<tr>
<td>Num of Multicast</td>
<td>( 3 \times T + 1 )</td>
</tr>
<tr>
<td>Size of Multicast Msg</td>
<td>( 1 + T \times ([K] + M \times [K] + 1 + M \times [E] + M \times [ID]) )</td>
</tr>
<tr>
<td>Round</td>
<td>( T + 4 )</td>
</tr>
</tbody>
</table>

\([E]\) – key length of public key, \([E] = 1024\) Bits = 128 Bytes  
\([K]\) – key length of symmetric key (\(K_{subgroup}\) and \(K_{group}\)), \([K] = 128\) Bits = 16 Bytes  
\([ID]\) – length of MAC address, \([ID] = 12\) Bytes  

\(T\) is the total number of subgroups, \(T = \lceil \frac{N}{M} \rceil\)  

Table 3.1 Communication cost of start-up

<table>
<thead>
<tr>
<th>Public Encryption</th>
<th>Symmetric Encryption</th>
<th>Symmetric Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>((M-1) \times T)</td>
<td>(T)</td>
<td>(N)</td>
</tr>
</tbody>
</table>

Table 3.2 Computation cost of start-up

(* In our system, public key encryptions and public key decryptions always come in pairs. This means the number of public key encryptions is always equal to the number of public key decryptions in our system. So here, and in the rest of the thesis, we will not analysis the cost of public decryption.)

The construction of subgroups requires a lot of communication/computation resources, as shown in Table 3.1 and 3.2. Fortunately, we only need to construct and initialize subgroups at the start-up of the system. We dynamically balance the size of subgroups (e.g. in Chapter 3.4.2) and rarely do re-initialization.
3.3 Key update at Join Event

There are 11 steps of the key update process at Join Event:

1) $M_r \rightarrow G$: \{JOIN_REQUEST\}

Requesting member - $M_r$ multicasts a JOIN_REQUEST message to $G$.

2) For $\forall M_j \in FG \& M_j \notin G$: $M_j$ forwards the JOIN_REQUEST

($FG$ is the forwarding group for multicast routing. $G$ is the multicast group. See Appendix for definition.)

For $\forall M_j \in G, M_j$ is subgroup leader : $M_r \rightarrow M_j$: \{JOIN_REPLY\}, $M_j$ does not forward JOIN_REQUEST.

By receiving the JOIN_REQUEST, members of $FG$ will forward JOIN_REQUEST to neighbours. Subgroup leaders will reply JOIN_REPLY message to the requesting node, but they do not forward JOIN_REQUEST.

3) $M_r \rightarrow M_i$: \{JOIN_CONFIRM, $E_r$\}

Assuming the first JOIN_REPLY is from $M_i$ ($M_i$ is a subgroup leader), then $M_r$ unicasts a JOIN_CONFIRM to $M_i$ together with $M_i$'s public key $E_r$.

4) After $M_i$ receives the JOIN_COMFIRM, $M_i$ add $M_r$ into subgroup $S_c$ (Assume $M_i \in S_c$)

5) $M_i$ updates $K_c$ and $K_{group}$.
6) \( M_t \rightarrow S_c : \{ (K_c)K_{c,old}, <M_r, E_r> \} \)

\( M_t \) encrypts updated \( K_c \) with \( K_{c,old} \), combines \((K_c)K_{c,old}\) with \(<M_r, E_r>\) in one message, and multicasts it to \( S_c \).

7) \( M_t \in S_c \), by receiving \((K_c)K_{c,old}, <M_r, E_r>\), adds \(<M_r, E_r>\) to \( S_c \) and updates \( K_c \) by decrypting \((K_c)K_{c,old}\) with the \( K_{c,old} \).

8) \( M_t \rightarrow M_r : \{ (K_c)E_r, (K_{group})K_{old} <S_r> \} \)

\( M_t \) encrypts \( K_c \) with \( E_r \) and \( K_{group} \) with \( K_{c,old} \). \( M_t \) then combines \((K_c)E_r, (K_{group})K_{c,old}\) and \(<S_r>\) in one message and unicasts it to \( M_r \).

9) \( M_r \), receives \((K_c)E_r, (K_{group})K_{c,old} <S_r>\), then stores \(<S_r>\) and decrypts \((K_c)E_r, (K_{group})K_{c,old}\) to get \( K_c \) and \( K_{group} \).

10) \( M_t \rightarrow G : (K_{group})K_{group,old} \)

\( M_t \) encrypts \( K_{group} \) with \( K_{group,old} \) and multicasts this to the whole group.

11) \( M_t \in G, k \neq r \), by receiving \((K_{group})K_{group,old}\), updates \( K_{group} \) by decrypting \((K_{group})K_{group,old} \) with \( K_{group,old} \).
Figure 3.9 Key update at Join Event

<table>
<thead>
<tr>
<th>Event</th>
<th>Public Encryption</th>
<th>Symmetric Encryption</th>
<th>Symmetric Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join</td>
<td>1</td>
<td>3</td>
<td>( M + N + 1 )</td>
</tr>
</tbody>
</table>

Table 3.3 Computation cost at Join Event

<table>
<thead>
<tr>
<th>Num of Unicast</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of Unicast Msg</td>
<td>([E] + 2\times[K] + M\times([E] + [ID]))</td>
</tr>
<tr>
<td>Num of Multicast</td>
<td>2</td>
</tr>
<tr>
<td>Size of Multicast Msg</td>
<td>(2\times[K] + [E] + [ID])</td>
</tr>
<tr>
<td>Round</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3.4 Communication cost at Join Event
3.4 Key Update at Leave Event

There are two types of Leave Event – the leaving node is a non-leader node, or the leaving node is a leader node. For non-leader node Leave Event, the group key and subgroup key are updated and distributed by one of the subgroup leaders. For leader node leave, a new leader will be selected to replace the leaving leader, or if both subgroups are too small, they merge into a new subgroup.

3.4.1 Non-Leader Node Leave

Assume:

1) $M_l$ is the leaving node, and $M_l \in S_c$

2) $M_a$ and $M_b$ are leaders of $S_c$, and $a < b$

The key update at non-leader node leave event has 5 steps:

1) $M_l \rightarrow M_a: \{\text{LEAVE\_REQUEST}\}$

$M_l$ unicasts a LEAVE\_REQUEST to the subgroup leader of $S_c$ with a smaller ID (here it is $M_a$), then $M_l$ leaves the group.

2) $M_a$ removes $M_l$ from $S_c$.

3) $\forall M_j \in S_c, M_j \neq M_a, M_a$ encrypts $K_c$ with $E_{j}(K_c)E_j$

$$M_a \rightarrow S_c: \{(j)E_j, M_l\} \text{ for } \forall M_j \in S_c, j \neq a$$
$Ma$ updates $K_c$. For each member $M_j$ in $S_c$, $Ma$ encrypts $K_c$ with the public key $E_j$, combines all the encrypted $K_c$ together with $M_j$'s ID into one message $\{(K_c)E_j, M_j\}$, and multicasts to $S_c$.

4) By receiving $\{(K_c)E_j, M_j\}$, $M_j (M_j \in S_c, j \neq a)$ decrypts $(K_c)E_j$ with $D_j$ and get updated $K_c = ((K_c)E_j)D_j$.

$M_j$ also removes $M_j$ from its subgroup member list.

5) $Ma$ updates $K_{group}$ and multicasts $K_{group}$ along the subgroup ring structure; this is same as the $K_{group}$ distribution we discussed in 3.3.2.

![Figure 3.10 Non-leader node leave](image)
### 3.4.2 Leader Node Leave - Merge

When the leaving node is a subgroup leader, we will either select a node as the new leader, or we will merge the two subgroups into a new subgroup if both subgroups have too few members.

Assume the leaving subgroup leader is $M_i$, and we know $M_i$ is the joint node of two subgroups - $\exists S_a$ and $\exists S_b$, $M_i \in S_a$ and $M_i \in S_b$, and $a < b$.

If $[S_a] < a \times \text{SUBGROUP\_SIZE}$ and $[S_b] < 1/3 \times \text{SUBGROUP\_SIZE}$, or $[S_a] = 2$ or $[S_b] = 2$, ($0 < a < 1$, $a$ can be configurable. In our implementation, $a = 1/3$). We merge $S_a$ and $S_b$ to a new subgroup - $S_a$. Otherwise, we select a new joint leader (see the discussion in 3.4.3).

The subgroup merging and key update has 4 steps:
1) $M_t \Rightarrow S_a, S_b$: \{LEADER_LEAVE, $<a, [S_a]>$, $<b, [S_b]>$\}

The leaving node $M_t$ multicasts a LEADER_LEAVE request to both $S_a$ and $S_b$ with ID and size of both subgroups.

2) $M_j, j \neq t$, if $M_j \in S_a, M_j$, remove $M_t$ from member list of $S_a$.

   if $M_j \in S_b, M_j$, remove $M_t$ from member list of $S_b$.

By receiving the Leader_Leave request, members of $S_a$ and $S_b$ remove $M_t$ from their subgroup node list.

3) $\exists M_p, M_p \in S_a$, $M_p$ is the leader of $S_a$ and $M_p \neq M_t$.

   $\exists M_q, M_q \in S_b$, $M_q$ is the leader of $S_b$ and $M_q \neq M_t$.

$M_p \Rightarrow S_b$: $<S_a>$, $p$, $a$

$M_q \Rightarrow S_a$: $<S_b>$, $q$, $b$

Since each subgroup has two subgroup leaders, we can find the remaining leader, $M_p$ ($M_p \neq M_t$) for subgroup $S_a$, and $M_q$ for $S_b$. $M_p$ multicasts $<S_a>$, ID of $M_p$ and subgroup ID of $S_a$ to subgroup $S_b$. When members of $S_b$ receives this message, they add the member list of $S_a$ to their subgroup, replaces $M_t$ with $M_p$ as the new leader and sets $a$ as the new ID for the subgroup.

$M_q$ also sends $<S_b>$, $q$, $b$ to $S_a$. When members of $S_a$ receive $<S_b>$, $q$, $b$, they set $M_q$ as the new leader, and adds $<S_b>$ to their member list.
After this message exchange, all members from former $S_a$ and $S_b$ have \{<S_a> U <S_b>\} as their member list, and have $<M_p, M_q>$ as the pair of subgroup leaders and the same subgroup ID $- a$. So they are merged to a new subgroup $- S_a$.

4) The leader node with smaller ID updates $K_a$ and $K_{group}$.

If $M_p$ and $M_q$ is the leader nodes of $S_a$ and $p < q$, then $M_p$ updates $K_a$ and $K_{group}$ and securely distributes $<K_a, K_{group}>$. This step is same as step 3, 4 and 5 in Non-Leader node leave 3.4.1.
Figure 3.11 Leader node leave and subgroup merge

Table 3.7 Computation cost at Leave Event (merge)
Num of Unicast | 0  
| Size of Unicast Msg | 0  
| Num of Multicast | $T + 4$  
| Size of Multicast Msg | $[ID] + 2 \times a \times M \times ([E] + [ID]) + 2 \times a \times M \times [K] + T \times [K]$  
| Round | $T + 3$  

Table 3.8 Communication cost at Leave Event (merge)

### 3.4.3 Leader Node Leave - Select a new subgroup leader

If the leaving node is a leader and the two subgroups do not satisfy the merging condition as discussed in 3.4.2, we select a new leader from the subgroup with more members. This node will be the new joint leader of both subgroups and will update the subgroup and group keys. There are 10 steps to select a new leader and update keys:

1) 2) is same as Step 1) and 2) in 3.4.2.

3) Assume $[S_a] > [S_b]$, then the new leader will be selected from $S_a$ (if $[S_b] > [S_a]$, then the new leader will be selected from $S_b$)

In step 1, the leaving node will send a leaving request together with the ID and size of both subgroups $S_a$ and $S_b$. Assume $M_p$ is the leader of $S_a$ and $M_q$ is the leader of $S_b$. By receiving the leave request from step 1, if $[S_a] > [S_b]$, then $M_p$ will select the new leader, otherwise, if $[S_a] < [S_b]$, then $M_q$ will select the new leader.

$\exists M_h, M_h \in S_a, M_h \neq M_i$ and $M_h \neq M_p$ (Here we assume $M_p$ is the leader of $S_a$ and $M_q$ is the leader of $S_b$), $M_p$ randomly picks a node $M_h$ from $S_a$ as the new leader.
4) $M_p \rightarrow S_a, S_b : \langle h, E_h, a \rangle$

$M_p$ multicast to $S_a$ and $S_b$ the selected new leader $M_h$ and its public key $E_h$.

5) $\forall M_j \in S_a$, by receiving the message from Step 4, $M_j$ set $M_h$ as the new leader.

6) $\forall M_j \in S_b$, by receiving the message from Step 4, $M_j$ adds $\langle M_h, E_h \rangle$ to $S_b$'s subgroup member list and sets $M_h$ as the new leader.

7) $M_q \rightarrow M_h : \langle S_b \rangle$ ($M_q \in S_b$ and $M_q$ is leader node.)

$M_q$ - the leader node of $S_b$ uncasts $\langle S_b \rangle$ to $M_h$.

8) By receiving messages from step 7, $M_h$ creates and adds $\langle S_b \rangle$ as $S_b$'s member list.

9) $M_h$ updates $K_a$ and multicasts it to $S_a$. This step is the same as step 3 and 4 in Non-Leader Node Leave. (3.4.1)

10) $M_h$ updates $K_b$ and multicasts it to subgroup $S_b$. This step is the same as steps 3 and 4 in Non-Leader Node Leave. (3.4.1)

11) $M_h$ updates $K_{group}$ and multicasts encrypted $K_{group}$ to $G$ along the subgroup ring. This step is the same as step 5 in Non-Leader Node Leave (3.4.1).
(1) $M_{fg} \Rightarrow S_1, S_2: \{M_{14}, E_{16}, 1\}$ (Step 4)
(2) $M_{8} \Rightarrow M_{14}: \{<S_2>\}$ (Step 7)
(3) $M_{6} \Rightarrow S_2: \{K_j \ E\}$ (Step 9)
(4) $M_{fg} \Rightarrow S_3: \{K_j \ E\}$ (Step 10)

Figure 3.12 Leaver node leave and a new leader is selected

<table>
<thead>
<tr>
<th>Event</th>
<th>Public Encryption</th>
<th>Symmetric Encryption</th>
<th>Symmetric Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leader Leave (Replicate)</td>
<td>$2 \times M$</td>
<td>$T$</td>
<td>$N$</td>
</tr>
</tbody>
</table>

Table 3.9 Computation cost at Leave Event (replicate a new leader)
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Num of Unicast</strong></td>
<td><strong>1</strong></td>
</tr>
<tr>
<td><strong>Size of Unicast Msg</strong></td>
<td><strong>$M \times ([E] + [ID])$</strong></td>
</tr>
<tr>
<td><strong>Num of Multicast</strong></td>
<td><strong>$T + 4$</strong></td>
</tr>
<tr>
<td><strong>Size of Multicast Msg</strong></td>
<td><strong>$2 \times [ID] + [E] + 2 \times M \times [K] + T \times [K]$</strong></td>
</tr>
<tr>
<td><strong>Round</strong></td>
<td><strong>$T + 4$</strong></td>
</tr>
</tbody>
</table>

Table 3.10 Communication cost at Leave Event (replicate a new leader)
Chapter 4

P2P-HGKM Performance and Evaluation

In this chapter, we compare the performance of P2P-HGKM with some existing key establishment protocols, including GDH 2.0, AT-GDH and Tree-Based key management protocol. Theoretical analysis shows that our P2P-HGKM outperforms GDH 2.0 and AT-GDH in computation and communication cost. Performance of P2P-HGKM is also close to optimum key management protocols like Tree-Based key management protocol, and our P2P-HGKM shows more advantage in ad-hoc networks. We also study the relationship between the two parameters ($M$ and $N$) and the computation/communication cost. We did extensive experiments and the results justify our prediction.
4.1 P2P-HGKM Security Analysis

Our P2P-HGKM protocol is secure at passive attack such as eavesdropping. This is because the algorithms (AES and RSA) and the key length (128 bits symmetric key and 1024 bits public key) we choose are considerably strong encryption algorithms. As far as the secret keys (the symmetric key for AES and the private key for RSA) are kept confidential, it’s very hard for eavesdroppers to deduce any key from the encrypted packets.

Active attack (falsification of data) from non-member nodes can affect the performance of P2P-HGKM, but cannot break the security of our protocol (At this step, we assume the active attacks are from non-member nodes, and all member nodes can be trusted. We will discuss the case of malicious member nodes right after this). As shown in Figure 1.2, intermediate nodes are required to relay packets from source to destination in ad-hoc networks. If an intermediate node is malicious, it can modify the packets before it rebroadcast. The solution to man-in-the-middle attack is to have all messages signed by source nodes. In our protocol, Join and Leave Request message (see Chapter 3.3 and 3.4) are sent and signed by the requesters. All other messages are sent and signed by subgroup leaders. In our system, if all member nodes maintain a list of valid subgroup leaders, then any falsified message from non-member nodes or the leaving node (at Leave Event) will be detected and reported.

Attack from member nodes is very hard to detect and prevent. If the system detects some member node becomes malicious, it can immediately expel the malicious member from the group and update all the keys held by the leaving node.
4.2 P2P-HGKM Experiment Overview

We have designed and constructed the P2P-HGKM protocol as well as the test bed for experiment using Java. ODMRP is also implemented as the supporting routing protocol.

At startup of the system, 50 nodes are created and are assumed to be randomly scattered in a 1000m x 1000m area. Radio range of wireless signal is 250 m. Then we randomly select \( N \) \( (N < 50) \) nodes as the group members of \( (G) \). A start node – \( M_s \) is elected by running bully algorithm among all member nodes. This \( M_s \) is responsible for building up the subgroup structure and distributing group and subgroup keys, as discussed in Chapter 3.2. In the following discussion, we use \( q \) to represent the percentage of Leave Events out of all membership change events. For example, in our simulation, every test run has a total of 20 membership change events – 10 Join Events and 10 Leave Events, so the percentage of Leave Event \( q = 0.5 \).

There are two parameters – \( M \) and \( N \) that can affect the computation and communication cost of our protocol. We desire to study the link between these parameters and the protocol performance. We do theoretical analysis and extensive experiments to prove our predictions (see section in 4.3.2 and 4.4.2). \( M \) is the average size of subgroup. In our experiments, we increase \( M \) from 6 to 16 at the step of 2 to see how \( M \) affects the performance. We record the number of public encryption/decryption and symmetric encryption/decryption as the computation cost. We also record the number and message size of unicast/multicast as the communication cost of key update. We
repeat each case 50 times and use the average of the results to plot graphs (see 4.3 and 4.4 for the graphs.)

\[ N \] is the number of group member. We assume \( G \) is always divided into roughly \( N^{1/2} \) subgroups and each subgroup has roughly \( N^{1/2} \) members. In our experiments, \( N \) increases from 16 to 40 by 4 at each step. Again, we record the computation and communications cost, repeat and plot graphs using the average of the results.

### 4.3 Computational Cost

An approximate measure of the computation costs is the number of key encryptions and decryptions required by a join/leave request. We summarize the computation cost from Chapter 3 and tabulate it in Table 4.1.

<table>
<thead>
<tr>
<th>Event</th>
<th>Public Key Encryption</th>
<th>Symmetric Key Encryption</th>
<th>Symmetric Key Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join</td>
<td>1</td>
<td>3</td>
<td>( M + N + 1 )</td>
</tr>
<tr>
<td>Non-Leader Leave</td>
<td>( M )</td>
<td>( T )</td>
<td>( N )</td>
</tr>
<tr>
<td>Leader Leave (Merge)</td>
<td>( M )</td>
<td>( T )</td>
<td>( N )</td>
</tr>
<tr>
<td>Leader Leave (Replicate)</td>
<td>( 2 \times M )</td>
<td>( T )</td>
<td>( N )</td>
</tr>
</tbody>
</table>

Table 4.1 A summary of computation cost

### 4.3.1 Comparing with other protocols
<table>
<thead>
<tr>
<th></th>
<th>P2P-HGKM</th>
<th>GDH 2.0</th>
<th>AT-GDH</th>
<th>Tree-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Join</strong></td>
<td>1 public enc +</td>
<td>( O(N^2) ) exp.</td>
<td>( O(N \log N) ) exp.</td>
<td>( 2 \times (h-1) ) symm enc ( (h-1)+ N \times d / (d-1) ) symm dec</td>
</tr>
<tr>
<td></td>
<td>1 public dec +</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 symm enc +</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( M + N + 1 ) symm dec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Overall:</strong></td>
<td>( O(N) ) symmetric decryption</td>
<td>( O(N^2) ) exp.</td>
<td>( O(N \log N) ) exp.</td>
<td>( O(N) ) symmetric decryption</td>
</tr>
<tr>
<td><strong>Leave</strong></td>
<td>( M ) public enc +</td>
<td>( O(N^2) ) exp.</td>
<td>( O(N \log N) ) exp.</td>
<td>( (d-1) \times h ) symm enc + ( N \times d / (d-1) ) symm dec</td>
</tr>
<tr>
<td></td>
<td>( M ) public dec +</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \lceil N/M \rceil ) symm enc +</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( N ) symm dec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Overall:</strong></td>
<td>( O(N) ) symmetric decryption + ( O(M) ) public encryption</td>
<td>( O(N^2) ) exp.</td>
<td>( O(N \log N) ) exp.</td>
<td>( O(d \times h) ) symmetric encryption ( O(N) ) symmetric decryption</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* enc – encryption; dec – decryption; exp - exponentiation

\( h \) – height of the tree in Tree-Based protocol
\( d \) – degree of the tree in Tree-Based protocol

Table 4.2 Computation cost comparison. Comparing computation cost with other key establishment protocols

We summarize the computation cost of P2P-HGKM, GDH 2.0, AT-GDH and the Tree-Based key management protocol in Table 4.2. From Table 4.2 we can see Diffie-Hellman key agreement protocols, e.g. GDH2.0 or AT-GDH, requires \( O(N \times \log N) \) or \( O(N^2) \) exponential computation at each Join or Leave request. Our P2P-HGKM protocol outperforms GDH 2.0 and At-GDH with only 2 public encryption/decryption and \( O(N) \)
symmetric decryptions at Join Event; and $O(M)$ public encryption/decryption and $O(N)$ symmetric decryptions at Leave Event. This is because our protocol limits public key encryption/decryption within one subgroup.

Key management protocols, e.g. Tree-Based Key management protocol, have a better computation cost performance than our P2P-HGKM protocol, as shown in Table 4.2. This is because Tree-Based protocol assumes that a reliable central key server is presented in the system. Our subgroup ring structure is not the optimal structure, but its performance is close to Tree-Based key management protocol. More importantly, our P2P-HGKM protocol works in a P2P fashion, thus avoiding the role of a single key server in our system. So our protocol is more applicable and reliable for ad-hoc networks than Tree-Based key management protocol.

4.3.2 Performance

Among all Leave Events, non-leader node leaving happens at the possibility of $(M-1) / M$ and leader node leaving happens as the possibility of $1 / M$. In 3.4.2, we merge two small subgroups into one subgroup to balance subgroups and improve overall performance. The merging only happens when the leaving node is a leader node and both subgroups have too few members. This is a rare case from the experience of experiments, especially when Join Event happens as often as Leave Event. Also, the computation and communication cost at the case of “Subgroup Merge” is very close to the case of “Select a New Leader”. So, for simplicity, we combine the case of “Subgroup Merge” with the case of “Select a New Leader” when we count the computation and communication cost.
In this section and in 4.3.2, we will first predict how $M$ and $N$ can affect the performance of P2P-HGKM. Then we use experiment data to plot graphs and testify our prediction.

1) Number of Public Encryption:

$$1 \times (1-q) + M \times ((M-1)/M) \times q + 2 \times M \times (1/M) \times q$$

$$= 1 + q \times M$$

$$= 1 + q \times N^{1/2}$$

<table>
<thead>
<tr>
<th></th>
<th>Number of Public Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M \uparrow$</td>
<td>$\uparrow q \times M$</td>
</tr>
<tr>
<td>$N \uparrow$</td>
<td>$\uparrow q \times N^{1/2}$</td>
</tr>
</tbody>
</table>

![Graph showing the relationship between public key encryption and members per subgroup](image)

a) $M$
2) Number of Symmetric Encryption:

\[2 \times (1-q) + \frac{(N/M) \times q}{M}\]

\[= 3 \times (1-q) + \frac{(N \times q)}{M}\]

\[= 3 \times (1-q) + q \times N^{1/2}\]

<table>
<thead>
<tr>
<th></th>
<th>Number of Symmetric Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M \uparrow)</td>
<td>(\downarrow \frac{(N \times q)}{M})</td>
</tr>
<tr>
<td>(N \uparrow)</td>
<td>(\uparrow q \times N^{1/2})</td>
</tr>
</tbody>
</table>

Figure 4.1 Experiment result: public key encryption/decryption
3) Number of Symmetric Decryption:

\[ (M + N) \times (1-q) + N \times q \]

\[ = N \times [(1-q) + (1-q) \times M] \]

\[ = n \times [1 - q + q \times M] \]
<table>
<thead>
<tr>
<th>Number of Symmetric Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M \uparrow$</td>
</tr>
<tr>
<td>$N \uparrow$</td>
</tr>
</tbody>
</table>

![Graph showing the relationship between the number of members per subgroup and symmetric decryption rate.](image)

**a) $M$**

![Graph showing the relationship between the number of group members and symmetric decryption rate.](image)

**b) $N$**

Figure 4.3 Experiment result: symmetric key decryption
4.4 Communication Cost

Approximate measure of communication cost in our key managements includes: the number of unicast, the number of multicast, and total number of rounds required for a Join or Leave request. We summary and tabulate the result from Chapter 3 in Table 4.3.

<table>
<thead>
<tr>
<th></th>
<th>Join</th>
<th>Non-Leader Leave</th>
<th>Leader Node Leave (Merge)</th>
<th>Leader Node Leave (Replicate)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Num of Unicast</strong></td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Size of Unicast Msg</strong></td>
<td>$[E] + 2\times[K] + M\times([E] + [ID])$</td>
<td>$[ID]$</td>
<td>0</td>
<td>$M\times([E] + [ID])$</td>
</tr>
<tr>
<td><strong>Num of Multicast</strong></td>
<td>2</td>
<td>$T + 1$</td>
<td>$T + 4$</td>
<td>$T + 4$</td>
</tr>
<tr>
<td><strong>Size of Multicast Msg</strong></td>
<td>$2\times[K] + [E] + [ID]$</td>
<td>$T\times[K] + M\times([ID] + [ID]) + 2\times M\times[K] + T\times[K]$</td>
<td>$[ID] + 2\times M\times[K] + T\times[K]$</td>
<td>$2\times[ID] + [E] + 2\times M\times[K] + T\times[K]$</td>
</tr>
<tr>
<td><strong>Round</strong></td>
<td>4</td>
<td>$T + 2$</td>
<td>$T + 3$</td>
<td>$T + 4$</td>
</tr>
</tbody>
</table>

Table 4.3 Summary of communication cost

4.4.1 Comparing with other protocols
As we can see from Table 4.4, our P2P-HGKM protocol outperforms GDH 2.0 and AT-GDH in the number of round and the number of messages. P2P-HGKM is not as good as Tree-Based key management protocol, but the performance is close and P2P-HGKM is more applicable and reliable in ad-hoc networks, as discussed before.

### 4.4.2 Performance

Like in 4.2.2, we will study the effect of parameter $M$, $q$ and $N$ on communication cost.

1) **Number of Unicast:**

<table>
<thead>
<tr>
<th></th>
<th>P2P-HGKM</th>
<th>GDH 2.0</th>
<th>AT-GDH</th>
<th>Tree-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Join</strong></td>
<td>2 Unicast</td>
<td>(N-1) Unicast + 1 Multicast</td>
<td>2×N Unicast</td>
<td>$h$ multicast + 1 unicast</td>
</tr>
<tr>
<td></td>
<td>2 Multicast</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 Round</td>
<td>$N$ Round</td>
<td>$2k \times (\log_2 N)$ Round</td>
<td>$h$ Round</td>
</tr>
<tr>
<td><strong>Leave</strong></td>
<td>2 Unicast</td>
<td>(N-1) Unicast + 1 Multicast</td>
<td>2$N$ Unicast</td>
<td>$(h-1) \times (d-1)$ multicast</td>
</tr>
<tr>
<td></td>
<td>4+ $T$ Multicast</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 + $T$ Round</td>
<td>$N$ Round</td>
<td>$2k\times(\log_e N)$ Round</td>
<td>$h$ Round</td>
</tr>
</tbody>
</table>

Table 4.4 Communication cost comparison Comparing communication cost with other key establishment protocols
\[2(1-q) + q(M-1)/M + (1/M) \times q = 2-q\]

<table>
<thead>
<tr>
<th></th>
<th>Number of Unicast</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M \uparrow)</td>
<td>Unchanged ↔</td>
</tr>
<tr>
<td>(N \uparrow)</td>
<td>Unchanged ↔</td>
</tr>
</tbody>
</table>

Figure 4.4 Experiment result: number of unicast
2) Size of Unicast

\[
\begin{align*}
([E] + 2[K] + M \times [E] + M \times [ID]) \times (1-q) &+ ((M-1)/M) \times ([ID]) \times q + M \times ([E] + [ID]) \times (1/M) \times q \\
&= ([E] + 2[K] \times (1-q) + 2 \times [ID] \times q) + ([E] + [ID]) \times (1-q) \times M \times [ID] \times q / M \\
&= ([E] + 2[K] \times (1-q) + 2 \times [ID] \times q) + ([E] + [ID]) \times (1-q) \times N^{1/2} - [ID] \times q / N^{1/2}
\end{align*}
\]

| Message Size of Multicast | \\
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( M \uparrow )</td>
</tr>
<tr>
<td>( N \uparrow )</td>
</tr>
</tbody>
</table>

![Graph](image_url)

a) \( M \)
Figure 4.5 Experiment result: message size of unicast

3) Number of Multicast

\[ 2 \times (1-q) + \frac{(N/M) \times ((M-1)/M) \times q + (4+N/M) \times (1/M) \times q}{N} \]

\[ = (2 - q) + q \times \frac{(N + 3)}{M} \]

\[ = (2 - q) + q \times \left( \frac{N^{1/2} + 3}{N^{1/2}} \right) \]

<table>
<thead>
<tr>
<th></th>
<th>Number of Multicast</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M \uparrow )</td>
<td>( \downarrow q \times \frac{(N + 3)}{M} )</td>
</tr>
<tr>
<td>( N \uparrow )</td>
<td>( \uparrow q \times \left( \frac{N^{1/2} + 3}{N^{1/2}} \right) ) when ( N &gt; 3 )</td>
</tr>
</tbody>
</table>
Figure 4.6 Experiment result: number of multicast

4) Size of Multicast:

\[
(2[K]+[E]+[ID]) \times (1-q) + \left( \frac{N}{M} \times [K] + M \times [ID] + M \times [K] \right) \times \left( \frac{(M-1)}{M} \right) \times q + (2 \times [ID] + [E]) \\
+ 2 \times M \times [K] + N/M \times [K]) \times (1/M) \times q
\]

\[
= (2-q) \times [K] + [E] \times (1-q) + [ID] \times (1-2q) + ([ID] + [K]) \times q \times M + q \times (N \times [K] + 2 \times [ID] + [E]) / M
\]
\[
= (2-q)x[K]+[E]x(1-q)+[ID]x(1-2xq) + ([ID]+[K])xqxN^{1/2} + qx(2x[ID]+[E]) / N^{1/2}
\]

<table>
<thead>
<tr>
<th>Message Size of Multicast</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M \uparrow)</td>
</tr>
<tr>
<td>(N \uparrow)</td>
</tr>
</tbody>
</table>

Figure 4.7 Experiment result: message size of multicast

a) \(M\)

b) \(N\)
5) Round:

\[ 4(1-q) + \frac{N}{M} + 2x((M-1)/M)xq + (N/M+4)x(1/M)xq \]

\[
= (4 - 2xq) + qx(N+2) / M
\]

\[
= (4 - 2xq) + qx(N^{1/2} + 2 / N^{1/2})
\]

<table>
<thead>
<tr>
<th></th>
<th>Round</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M \uparrow )</td>
<td>( \downarrow qx(N+2) / M )</td>
</tr>
<tr>
<td>( N \uparrow )</td>
<td>( \uparrow qx(N^{1/2} + 2 / N^{1/2}) ) when ( N &gt; 2 )</td>
</tr>
</tbody>
</table>
Chapter 5

Conclusion and Future Work

5.1 Conclusion

To support dynamic secure group communication in ad-hoc networks, a group key $K_{\text{group}}$, shared by all members of the group needs to be constantly updated whenever there is a membership change of the group, e.g. when a current member leaves or a new member joins the group. Many group key establishment protocols have been proposed and a few were proposed for ad-hoc networks, but they all have major shortcomings when applied to ad-hoc network environments. Wireless hosts in ad-hoc networks are mobile, and are usually lightweight, battery-powered devices. And using multicast in wireless network has advantage over multiple unicast - it takes less time and less resource.

In this thesis, we proposed an efficient P2P hierarchical group key management protocol for ad-hoc networks. We introduced the concept of subgroups, each maintaining
its subgroup key and links with other subgroups in a simple ring structure. By dividing group $G$ into subgroups, we limit unicast and public key encryption within one subgroup, thus greatly reducing the cost of group key update at node Leave Events. The distinguishing feature of our scheme lies in the fact that our scheme works in P2P, rather than a single key server.

We implement our protocol and do extensive theoretical analysis and experiments. Both theoretical analysis and experiment show that our protocol provides efficient and reliable group key management services, and outperforms many existing group key establishment protocol in ad-hoc network.

5.2 Future Work

As discussed in Chapter 3, subgroups are linked in a ring structure, and every joint leader connects two subgroups. Using a more complex structure of subgroups could be a viable future work. As shown in Figure 5.1, all the subgroups can be linked in a more complex structure, e.g. a cubic, and a subgroup leader can be a joint node of more than two subgroups. Updated $K_{\text{group}}$ can be sent in more direction at the same time. Such complex structure would offer advantages in efficiency and scalability, but of course, managing such structures would be more complex.
Our protocol can merge two subgroups into a new subgroup when there are too few nodes in both subgroups, as discussed in 3.4.2. This is a way to dynamically balance subgroups without re-initialization. We know a balanced subgroup ring has a better performance at group key updates than an unbalanced subgroup ring. As future work, we could develop a way to automatically find those “fat” subgroups (subgroups with too many members) and split them into smaller subgroups.

Our protocol handles key update at single node join or leave. It can be easily extended to multiple nodes cases by running the protocol multiple times. A more complex but efficient scheme for group key update at simultaneous multiple-node join/leave events can be explored in future research.
Appendix

Notations used in P2P-HGKM protocol:

\( K_{\text{group}} \): the group key shared by all members of the multicast group

\( K_{\text{group,old}} \): the group key before updating

\( K_i \): the subgroup key for subgroup \( i \), shared by all members of the subgroup

\( K_{i,\text{old}} \): the subgroup key of \( S_i \) before updating

\( M_i \): node with ID \( i \)

\( E_i \): public key of node \( i \) (\( M_i \))

\( D_i \): private key of node \( i \) (\( M_i \))

\( S_i \): Subgroup \( S_i \), \( i \) is ID of the subgroup.

\( <S_i> \): node list of subgroup \( S_i \)

\( M_i \rightarrow M_j : \{\text{Rekey Msg}\} \): \( M_i \) unicast \{Rekey Msg\} to \( M_j \)

\( M_i \rightarrow S_c : \{\text{Rekey Msg}\} \): \( M_i \) multicast \{Rekey Msg\} to subgroup \( S_c \)

\( M_i \rightarrow G : \{\text{Rekey Msg}\} \): \( M_i \) multicast \{Rekey Msg\} to multicast group \( G \)
$FG$: Forwarding Group, only members of the forwarding group will forward multicast packages.

$N$: The total number of node in the multicast group $G$

$M$: The average number of node in each subgroup

$T$: $T$ is the total number of subgroups, $T = \lceil \frac{N}{M} \rceil$
Bibliography


[13] Cisco System:
http://www.cisco.com/en/US/about/ac123/ac147/ac174/ac177/about_cisco_ipj_article_article09186a00800c83e4.html


