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

Library Authorization

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.

PENG PENG

Name of Author *(please print)*

20/08/2004

Date (dd/mm/yyyy)

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

List of Figures

List of Tables

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

Chapter 1

Introduction

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]

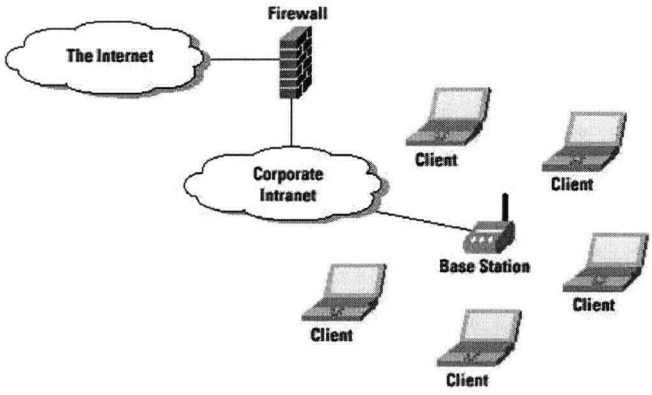


Figure 1.1 802.11 Wireless LAN

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.

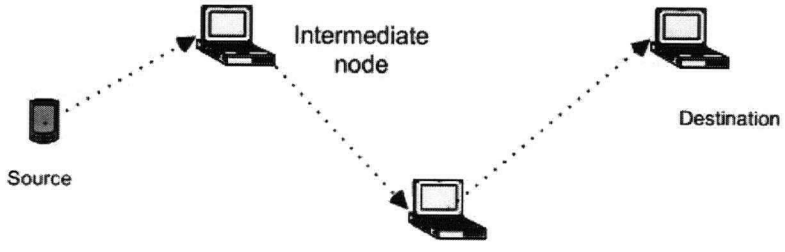


Figure 1.2 Ad-hoc networks. In ad-hoc network, intermediate nodes relay packets from source to destination

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_{group}) shared by all members of the group is needed. Packets will be encrypted with a group key K_{group} by the sender and decrypted with the same K_{group} by receivers. The group key K_{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_{group} so all past communication encrypted with the old K_{group} is kept secret to the new member. If a member node leaves, K_{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.

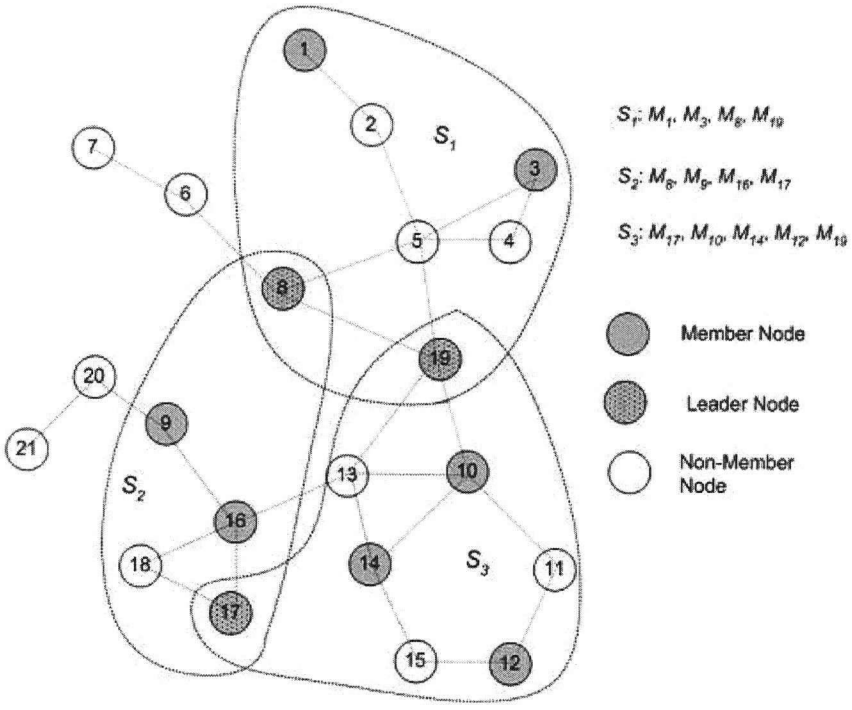


Figure 1.3 Members of multicast group are divided into subgroups.

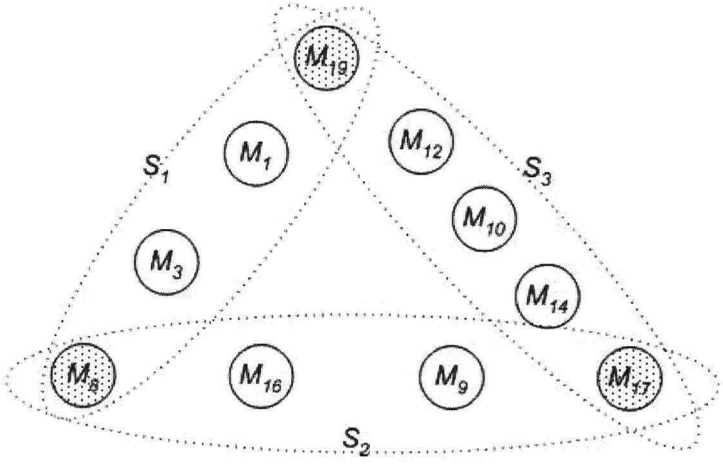


Figure 1.4 Subgroups link in a ring structure

Without the concept of subgroup, an update K_{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_{group} can be encrypted with subgroup keys and multicast to the corresponding subgroups. For example, in Figure 1.5, K_{group} is updated by M_{19}, encrypted with subgroup key (K_1) of subgroup S_1 and multicast $(K_{group})K_1$ to S_1. When M_8 - the other leader node of S_1, receives this message, M_8 decrypts $(K_{group})K_1$ and gets K_{group}. Since M_8 is also the leader of S_2, M_8 will then encrypt K_{group} with K_2 and multicast $(K_{group})K_2$ to S_2. In this way, K_{group} can be propagated along the subgroup ring and finally received by all members, as shown in Figure 1.5.

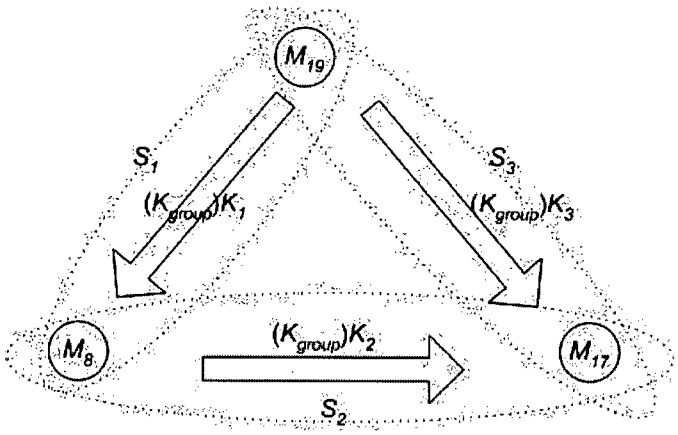


Figure 1.5 K_{group} propagates along the subgroup ring

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.

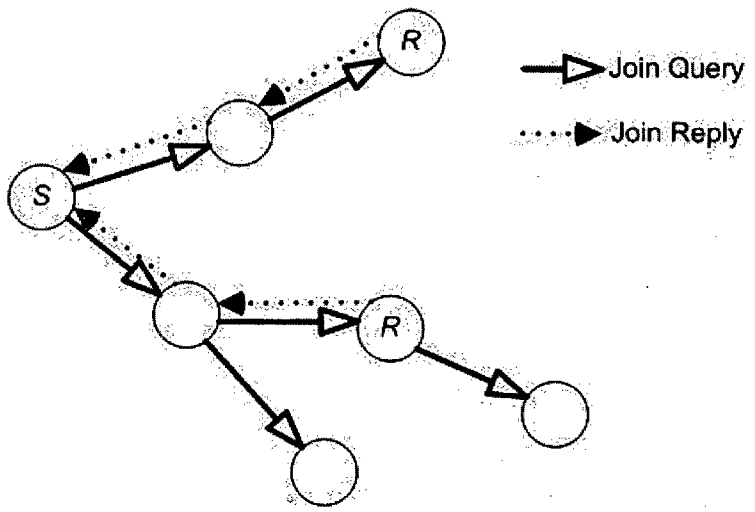


Figure 2.1 ODMRP. Source node broadcasts Join-Query and waits Join-Reply from receivers

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

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 $P = ((P)\ E_b)\ D_b$.

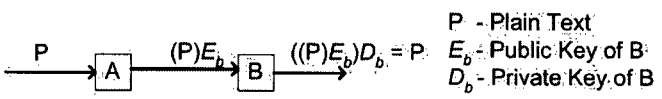


Figure 2.2 Public key encryptions

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_{group}. K_{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_1 is the joint leader of both S_1 and S_2, then M_1 will have subgroup key K_1 and K_2. Subgroup key is also a 128 bit symmetric key, which is used to encrypt the updated key K_{group}. A subgroup key is shared by all members of the subgroup. As a result when there is a membership change in S_i, 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_{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:

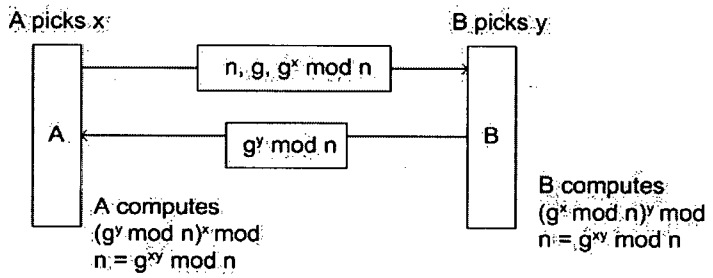


Figure 2.3 Diffie-Hellman key exchange

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

The security of Diffie-Hellman key exchange lays in the fact that, deducing x from $(g^x \bmod n)$ is hard. So even if an eavesdropper has $(g^x \bmod n)$ and $(g^y \bmod n)$ and g and n, he still cannot deduce either x or y, and thus cannot compute the shared key $(g^{xy} \bmod n)$. 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 ($1 \leq i \leq n-1$)

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

2. M_i sends M_{i+1}: $\alpha^{(\Pi^{i}_{k=1} r_k)}$ and $\alpha^{((\Pi^{i}_{k=1} r_k)/r_j)}$, $\forall\ 1 \leq j \leq n$

Round n:

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

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

3. Each member Mi then computes the final key as

$S_n = y_i^{\wedge} r_i = \alpha^{(\Pi^{n}_{j=1} r_j)}$.

Figure 2.4 GDH 2.0 algorithm

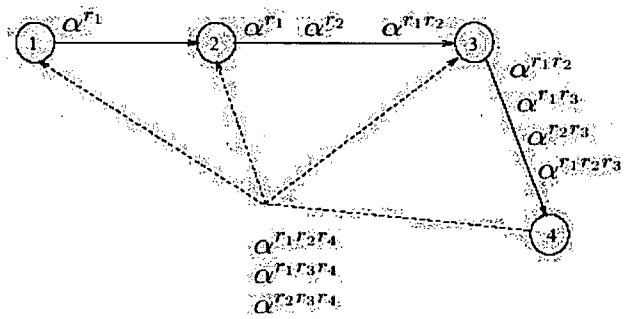


Figure 2.5 GDH 2.0 works in a group with 4 members.

Message	(n-1) unicast + 1 multicast
Round	n
Exponentiation	(i+1) for M_i ($i < n$) n for M_n Total number of exp. = $O(n^2)$

Table 2.2 Communication and computation cost of GDH 2.0

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_{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_{group_old}) to encrypt the updated K_{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}.

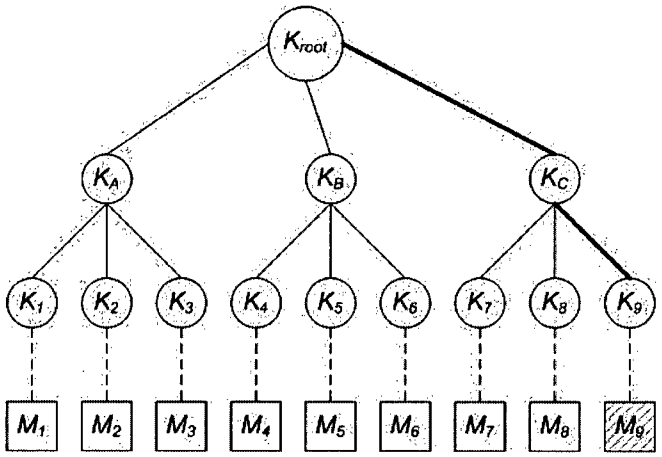


Figure 2.6 Tree-Based key management protocol

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. $((K_{root}, K_c)K_7 \rightarrow M_7 ; (K_{root}, K_c)K_8 \rightarrow M_8)$

	Join	Leave
Communication Cost	h multicast + 1 unicast	$(h\text{-}1) \times (d\text{-}1)$ multicast
Computation Cost	Request node: $(h\text{-}1)$ symm decr Non-Req node: $n \times d/(d\text{-}1)$ symm decr Key server: $2 \times (h\text{-}1)$ symm encr	Request node: 0 Non-Req node: $n \times d/(d\text{-}1)$ symm decr Key server: $d \times (h\text{-}1)$ symm encr

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

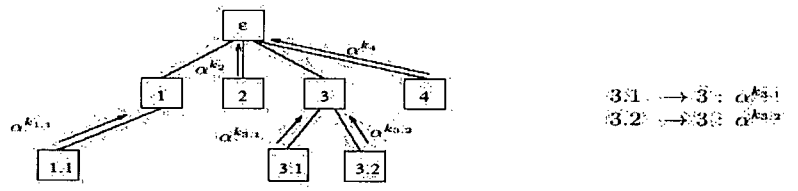
Table 2.3 Communication/Computation cost of Tree-Based key management

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.

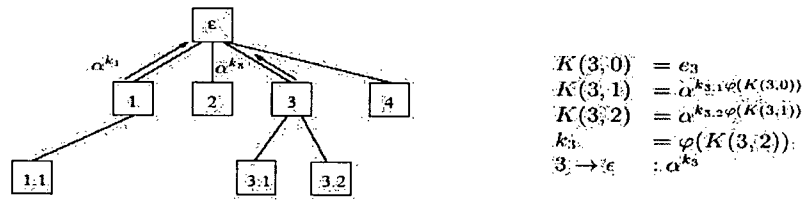
2.4 Related Work

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.

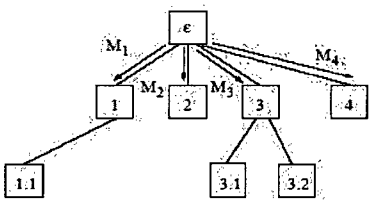


a) Step 1.a



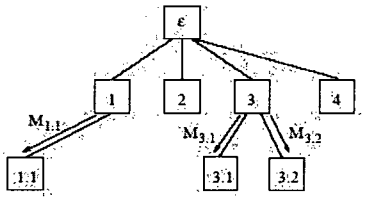
b) Step 1.b

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



$\epsilon \rightarrow 3 \quad : M_3$
$M_3 \quad = \langle -, \alpha^{\varphi(K(\epsilon,2))}, \alpha^{k_4} \rangle$

a) Step 2.a



$3 \rightarrow 3.1 \quad : M_{3.1}$
$M_{3.1} \quad = \langle M_3, \alpha^{\varphi(e_3)}, \alpha^{k_{3.2}} \rangle$
$3 \rightarrow 3.2 \quad : M_{3.2}$
$M_{3.2} \quad = \{M_3, \alpha^{\varphi(K(3,1))}\}$

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.

	GDH 2.0	AT-GDH
Communication Cost	N-1 Unicast + 1 Multicast	$2\times N$ Unicast
	N Round	$2k \times (log_k N)$ Round
Computation Cost	$O(N^2)$ Exponentiation.	$O(N\ log_k N)$ Exponentiation

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 $O(N \log_k N)$.

Chapter 3

Design and Implementation

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 shares a group key (K_{group}) for secure group communication.

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

Updating K_{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_{group_old}), so we cannot encrypt K_{group} with K_{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_{group} is to encrypt the

updated K_{group} with the public key (E_i) of each member (M_i) and unicast encrypted $(K_{group})E_i$ 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_{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.

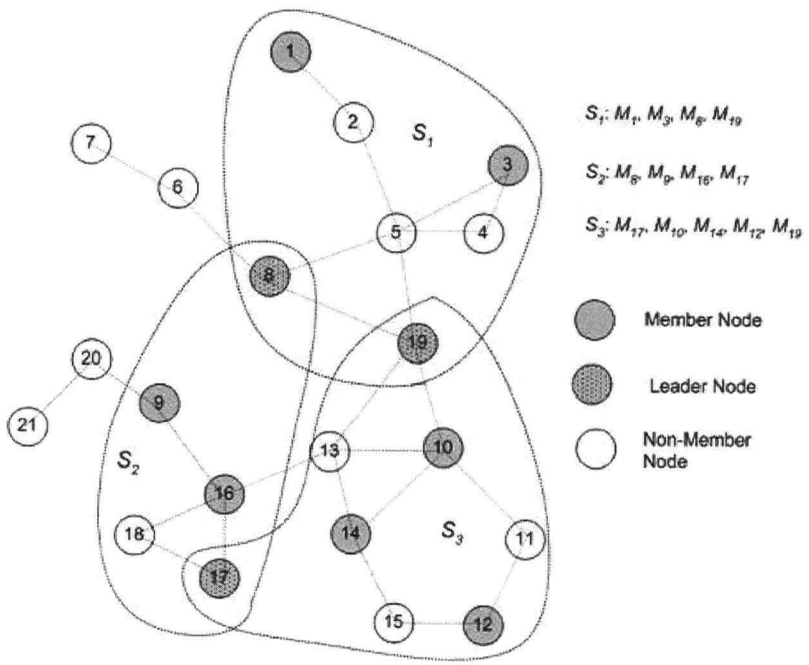


Figure 3.1 Multicast group G is divided into subgroups.

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

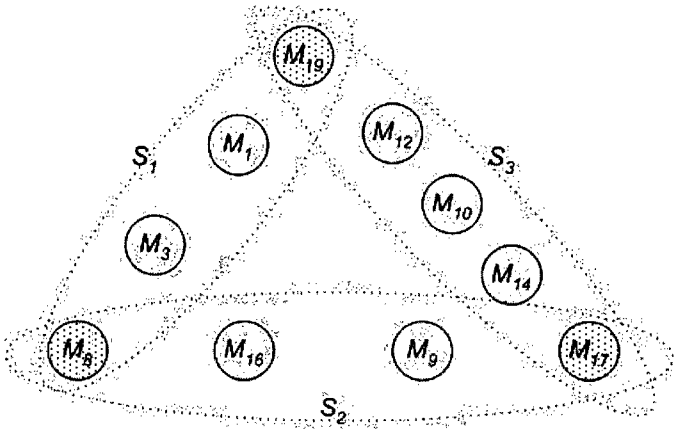


Figure 3.2 Subgroups are linked by joint leaders into a ring structure

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_3 is the leaving node, then both K_{group} and K_1 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_{group} and K_1. M_8 first

updates K_1, encrypts K_1 with E_1 and E_{19} (the public key of M_1 and M_{19}) and unicasts $(K_1)E_1$ and $(K_1)E_{19}$ to M_1 and M_{19} individually.

After updating K_1, M_8 updates K_{group}, encrypts K_{group} with K_2 and K_1 and propagates encrypted K_{group} along the subgroup ring, as shown in Figure 3.3. When M_{17} - the other leader node of S_2, receives this message, M_{17} decrypts $(K_{group})K_2$ and gets K_{group}. Since M_{17} is also a leader of S_3, M_{17} will then encrypt K_{group} with K_3 and multicast $(K_{group})K_3$ to S_3. In this way, K_{group} can be securely propagated along the subgroup ring and finally received by all members.

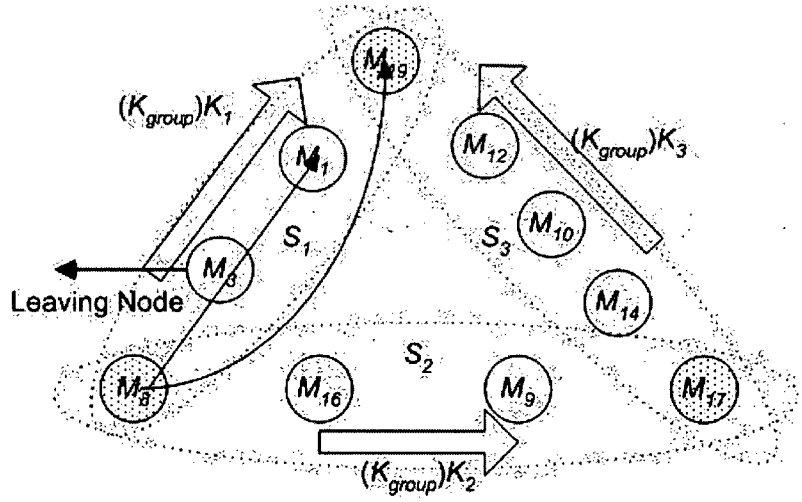


Figure 3.3. Key update at Leave Event

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.

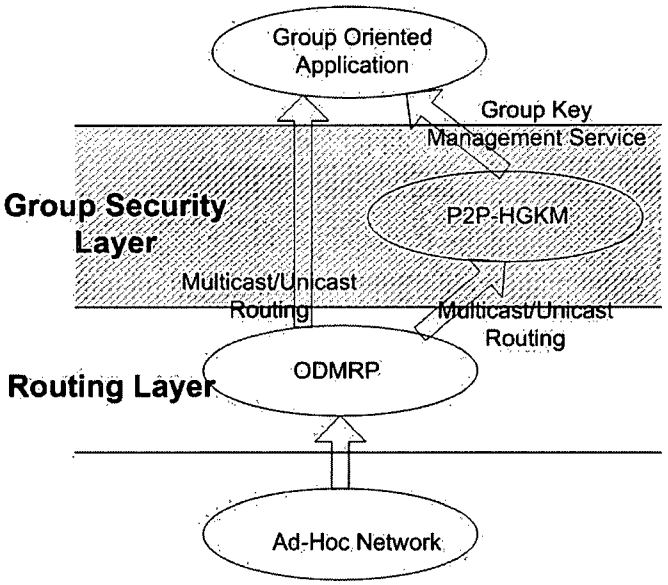


Figure 3.4 Software layer

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_i.

$\forall M_j, M_j \in G, i \neq j, M_j \rightarrow M_s$: <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.

$M_i \Rightarrow G$: $<S_c>$, 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_{group}), encrypts K_{group} and propagates K_{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)

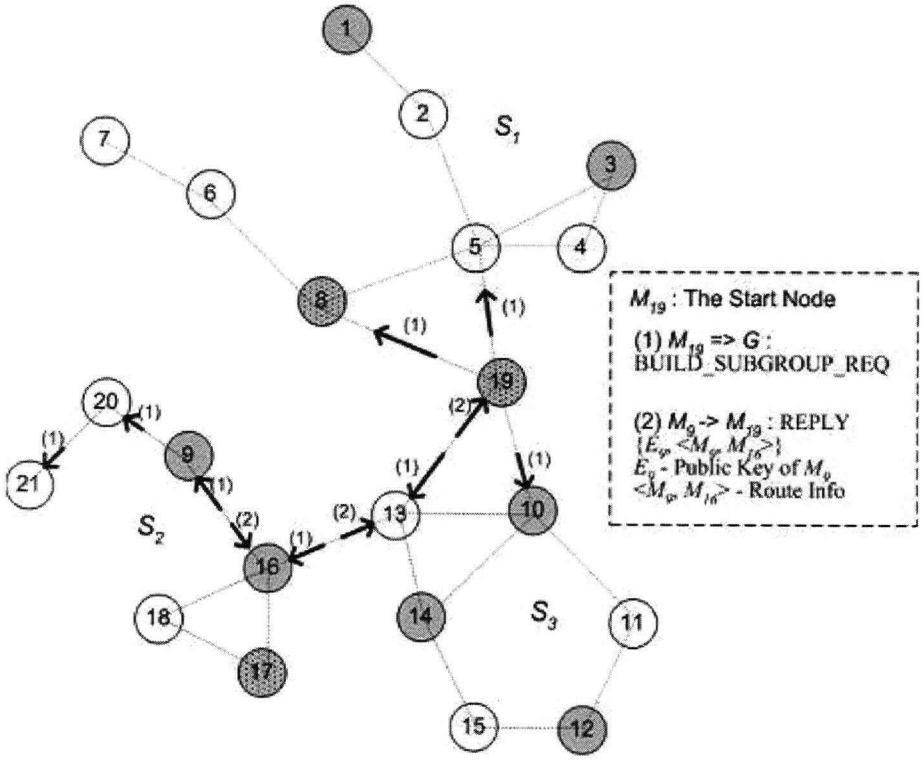


Figure 3.5 Broadcast the BUILD_SUBGROUP_REQUEST

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

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 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).

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

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_{19})

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_{19} 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)

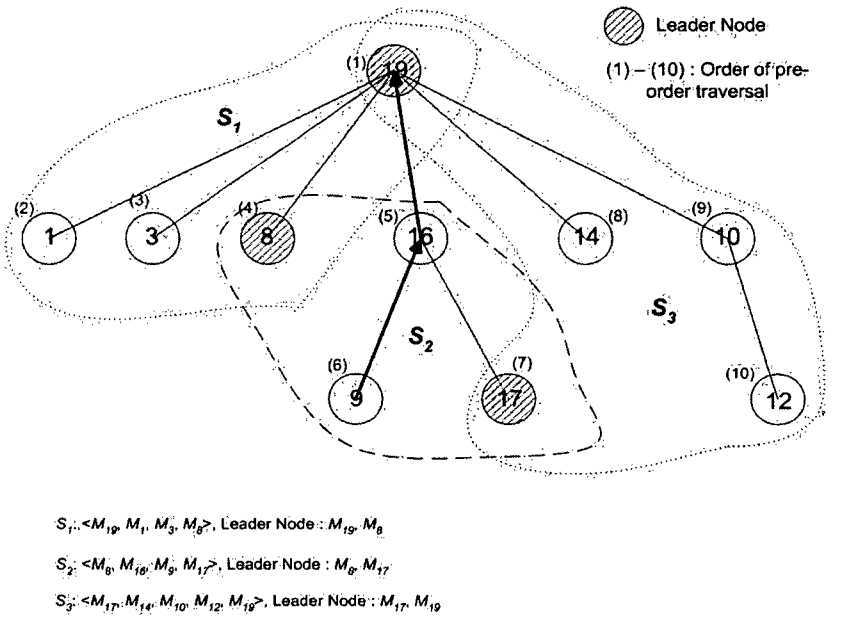


Figure 3.6 Build subgroups

With the REPLY messages in Step 1, M_s can divide G into subgroups. First, M_s inserts all member nodes in a tree structure rooted at M_s 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_{16}, 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) $M_s \Rightarrow G$: $\{<S_c>\}$ for all subgroups.

After dividing G into subgroups, M_s multicasts how it made the division (the member list of each subgroup) to G. For example, in Figure 3.6, $<S_1> = <M_{19}, M_3, M_8>$

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

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

3) $M_l \Rightarrow S_c$: $\{(K_c)E_p\}$, for $\forall M_p \in S_c$, $p \neq l$

If M_l is the 1st leader node of S_c, M_l 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) $M_p \in S_c$, $p \neq j$, $(((K_c)E_p)D_p) = K_c$

If M_p is a member of S_c, M_p decrypts $(K_c)E_p$ with D_p and get K_c.

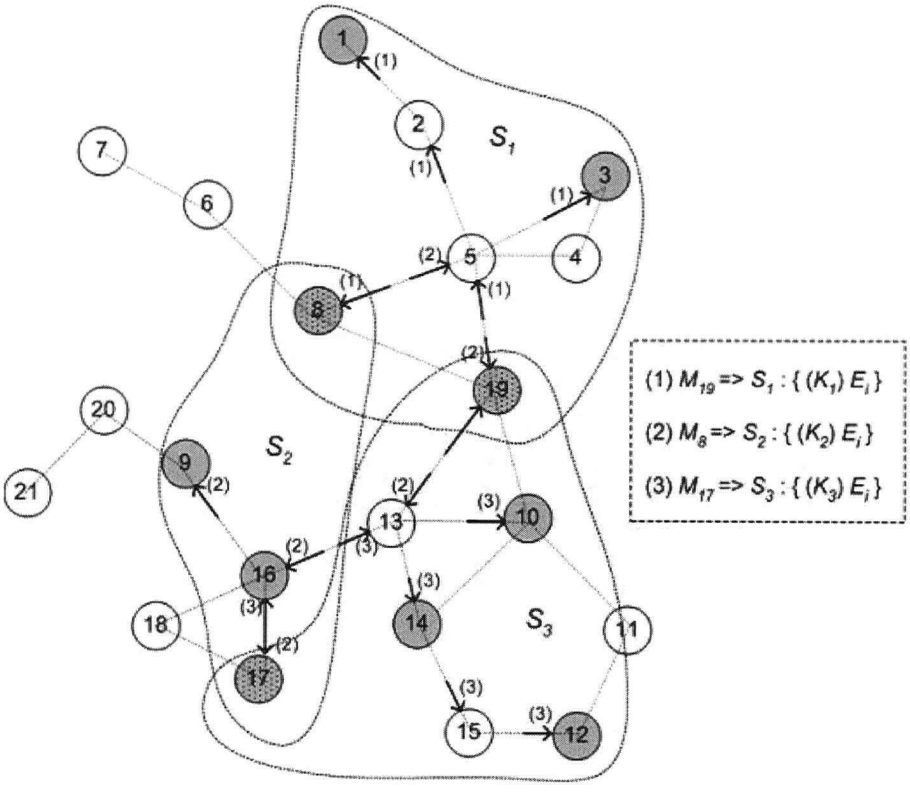


Figure 3.7 Initialize and distribute subgroup and group key

For example, in Figure 3.7, M_{19} initializes K_1 for S_1, encrypts K_1 with the public key of M_1, M_2 and M_8 and multicasts <$(K_1)E_1$, $(K_1)E_3$, $(K_1)E_8$> to S_1. When M_1 receives this message, M_1 decrypt $(K_1)E_1$ with D_1 and get the subgroup key $K_1 = ((K_1)E_1)\ D_1$.

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_{group}) and distribute encrypted K_{group} along the subgroup ring.

1) M_s initializes K_{group}

2) $S_i, M_s \in S_i.\ M_s \Rightarrow S_i : (K_{group})K_i$

M_s encrypts K_{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 $(K_{group})K_i$, it decrypts $(K_{group})K_i$ with K_i and gets K_{group}, where $K_{group} = ((K_{group})K_i)K_i$. (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 $M_j \in S_i, M_j \Rightarrow S_k$: $(K_{group})K_k$

Joint leader works as a bridge of two subgroups and propagates K_{group} along the subgroup ring structure to the whole group G, as shown in Figure3.8.

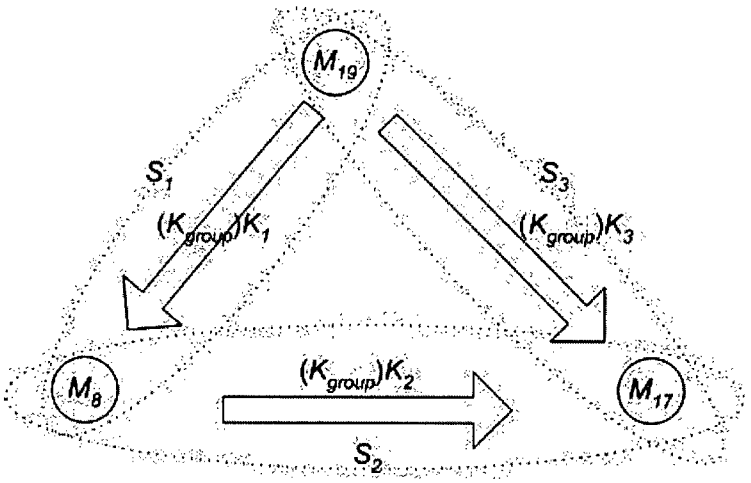


Figure 3.8 Distribute K_{group} along the subgroup ring

Num of Unicast	N
Size of Unicast Msg	$N\times (1 + [E] + N\times[ID])$
Num of Multicast	$3\times T + 1$
Size of Multicast Msg	$1 + T \times ([K] + M\times[K] + 1 + M\times[E] + M\times[ID])$
Round	$T + 4$

$[E]$ – 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, $T \approx \lceil N/M \rceil$

Table 3.1 Communication cost of start-up

Public Encryption	Symmetric Encryption	Symmetric Decryption
$(M\text{-}1) \times T$	T	N

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

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

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

5) M_t 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) $M_j \in S_c$, by receiving $\{(K_c)K_{c\_old}, <M_i\ E_i>\}$, 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) $M_t \rightarrow M_r$: $\{(K_c)E_r, (K_{group})K_c, <S_c>\}$

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

9) M_r, receives $\{(K_c)E_r, (K_{group})E_r, <S_c>\}$, then stores $<S_c>$ and decrypts $<(K_c)E_r, (K_{group})K_c>$ 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) $M_k \in G$, $k \neq r$, by receiving $(K_{group})K_{group\_old}$, updates K_{group} by decrypting $(K_{group})K_{group\_old}$ with K_{group_old}.

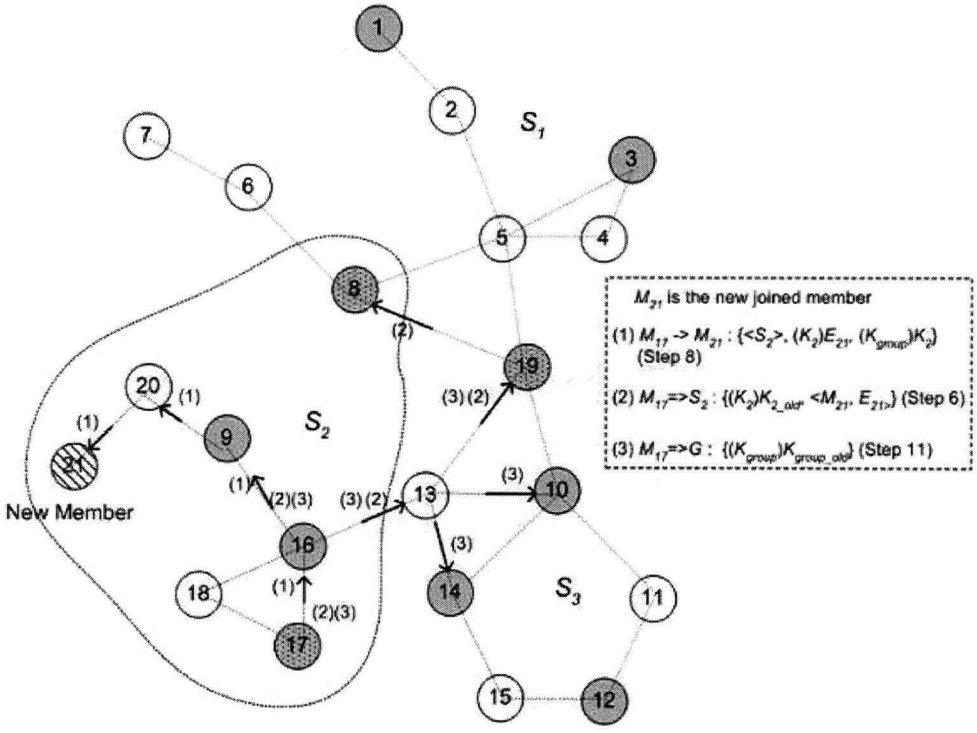


Figure 3.9 Key update at Join Event

Event	Public Encryption	Symmetric Encryption	Symmetric Decryption
Join	1	3	$M+N+1$

Table 3.3 Computation cost at Join Event

Num of Unicast	2
Size of Unicast Msg	$[E] + 2\times[K] + M\times([E] + [ID])$
Num of Multicast	2
Size of Multicast Msg	$2\times[K] + [E] + [ID]$
Round	4

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) $M_l \rightarrow M_a$: {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, $(K_c)E_j$

$M_a \Rightarrow S_c$: $\{\{(K_c)E_j\}, M_l\}$ for $\forall M_j \in S_c, j \neq a$

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

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

M_j also removes M_l from its subgroup member list.

5) M_a 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.

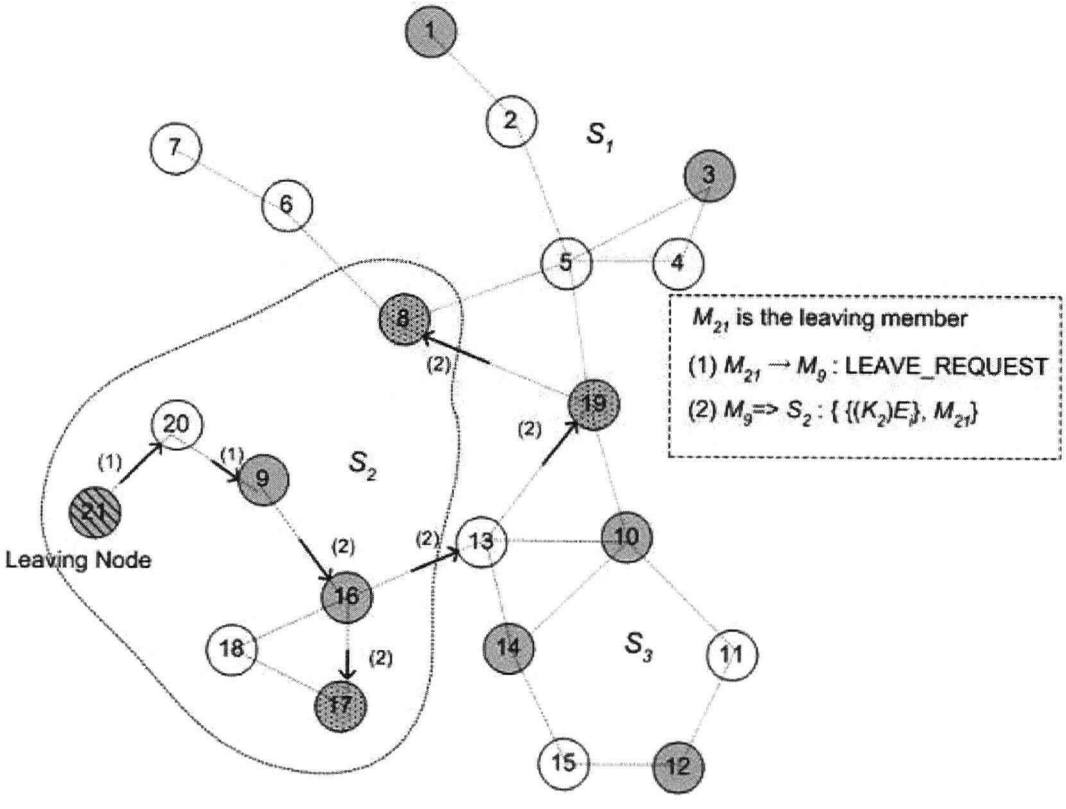


Figure 3.10 Non-leader node leave

Event	Public Encryption	Symmetric Encryption	Symmetric Decryption
Non-Leader Leave	M	T	N

Table 3.5 Computation cost at Leave Event (non-leader node)

Num of Unicast	*1*
Size of Unicast Msg	$[ID]$
Num of Multicast	$T + 1$
Size of Multicast Msg	$T \times [K] + M \times [ID] + M \times [K]$
Round	$T + 2$

Table 3.6 Communication cost at Leave Event (non-leader Node)

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_l, and we know M_l is the joint node of two subgroups - $\exists S_a$ and $\exists S_b$, $M_l \in S_a$ and $M_l \in S_b$, and $a < b$.

If $[S_a] < a \times$ SUBGROUP_SIZE and $[S_b] < 1/3 \times$ 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) $M_l \Rightarrow S_a, S_b$: {LEADER_LEAVE, <a, [S_a]>, <b, [S_b]>}

The leaving node M_l multicast a LEADER_LEAVE request to both S_a and S_b with ID and size of both subgroups.

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

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

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

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

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

$M_p \Rightarrow S_b$: {<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 ($M_p \neq M_l$) 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_l with M_p as the new leader and sets a as the new ID a 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 $\{<S_a> \cup <S_b>\}$ 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.

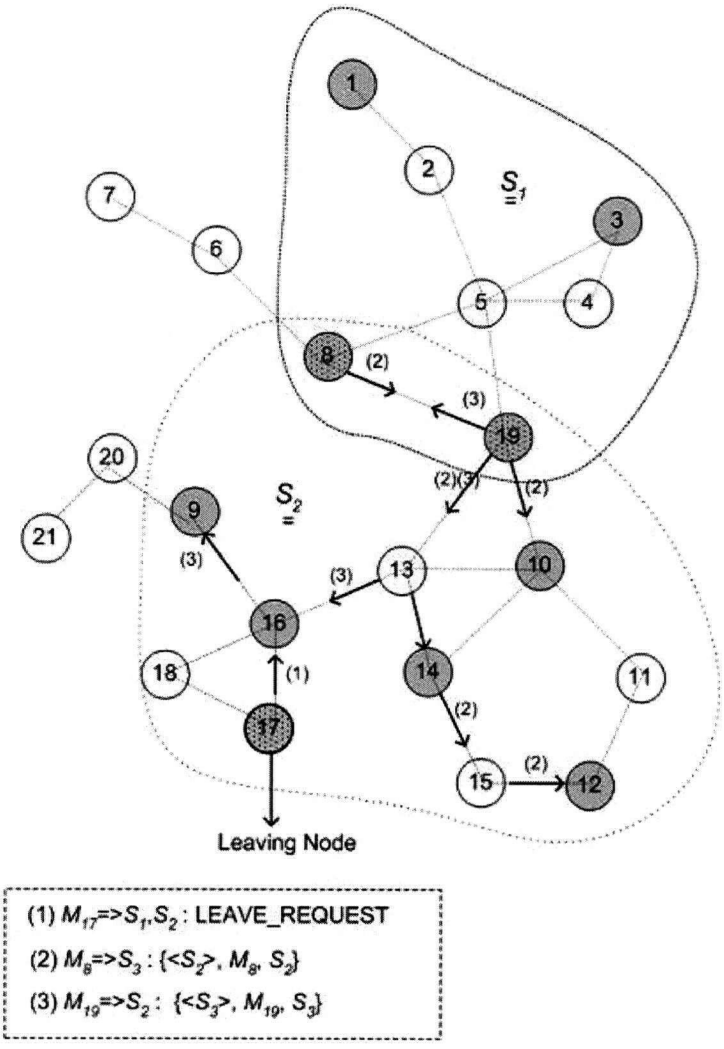


Figure 3.11 Leader node leave and subgroup merge

Event	Public Encryption	Symmetric Encryption	Symmetric Decryption
Non-Leader Leave	*M*	*T*	*N*

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) $M_p \Rightarrow S_a, S_b$: $<h, E_h, a>$

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 $< M_h, E_h>$ to S_b's subgroup member list and sets M_h as the new leader.

7) M_q -> M_h: $< S_b>$ ($M_q \in S_b$ and M_q is leader node.)

M_q – the leader node of S_b unicasts $<S_b>$ to M_h,.

8) By receiving messages from step 7, M_h creates and adds $<S_b>$ 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).

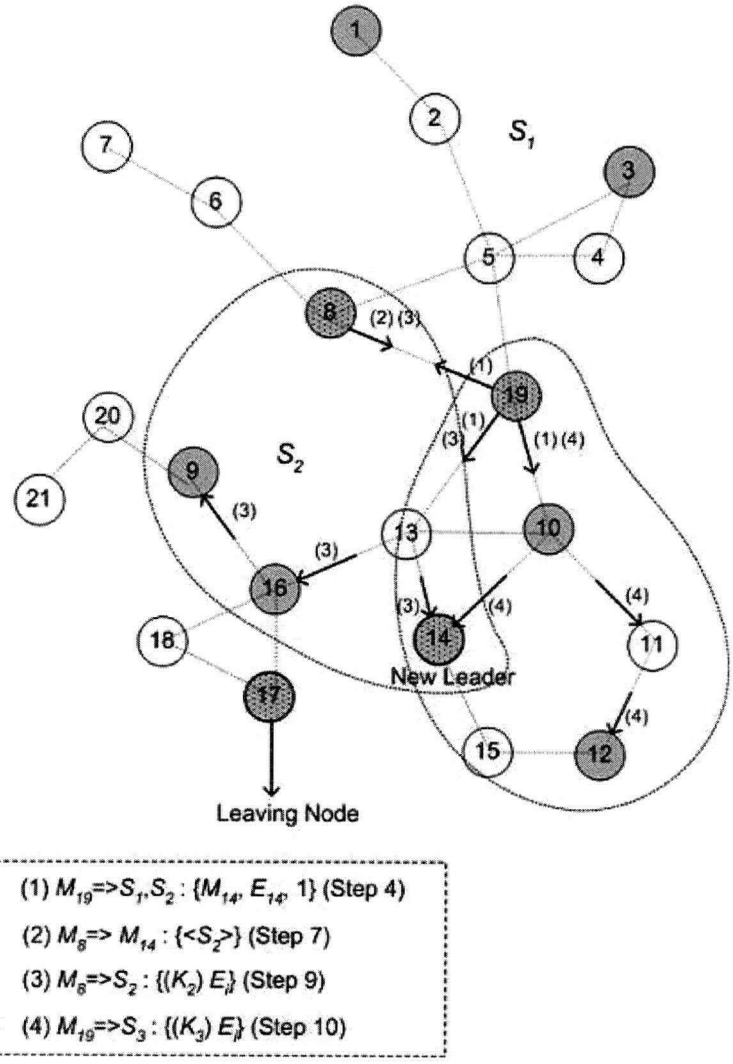


Figure 3.12 Leaver node leave and a new leader is selected

Event	Public Encryption	Symmetric Encryption	Symmetric Decryption
Leader Leave (Replicate)	$2\times M$	T	N

Table 3.9 Computation cost at Leave Event (replicate a new leader)

Num of Unicast	*1*
Size of Unicast Msg	$M\times([E] + [ID])$
Num of Multicast	$T + 4$
Size of Multicast Msg	$2\times[ID] + [E] + 2\times M\times[K]+ T\times[K]$
Round	$T + 4$

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 members 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 hold 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.

Event	Public Key Encryption	Symmetric Key Encryption	Symmetric Key Decryption
Join	1	3	$M+N+1$
Non-Leader Leave	M	T	N
Leader Leave (Merge)	M	T	N
Leader Leave (Replicate)	$2\times M$	T	N

Table 4.1 A summary of computation cost

4.3.1 Comparing with other protocols

	P2P-HGKM	GDH 2.0	AT-GDH	Tree-Based
Join	1 public enc + 1 public dec + 3 symm enc + $M+N+1$ symm dec	$O(N^2)$ exp.	$O(N \log_k N)$ exp.	$2 \times (h\text{-}1)$ symm enc $(h\text{-}1)+ N \times d/(d\text{-}1)$ symm dec
	Overall: $O(N)$ symmetric decryption	$O(N^2)$ exp.	$O(N \log_k N)$ exp.	$O(N)$ symmetric decryption
Leave	M public enc + M public dec + $\lceil N/M \rceil$ symm enc + N symm dec	$O(N^2)$ exp.	$O(N \log_k N)$ exp.	$(d\text{-}1) \times h$ symm enc + $N \times d / (d\text{-}1)$ symm dec
	Overall: $O(N)$ symmetric decryption + $O(M)$ public encryption	$O(N^2)$ exp.	$O(N \log_k N)$ exp.	$O(d \times h)$ symmetric encryption $O(N)$ symmetric decryption

* 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 $O(N \times \log_k N)$ 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 $(M\text{-}1) / M$ 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.

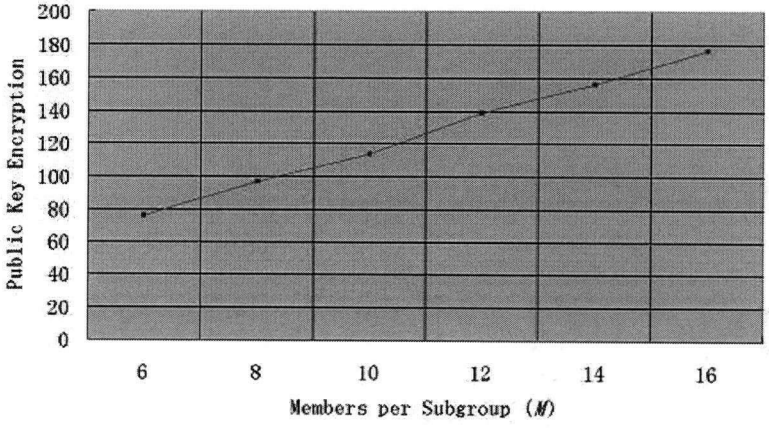
1) Number of Public Encryption:

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

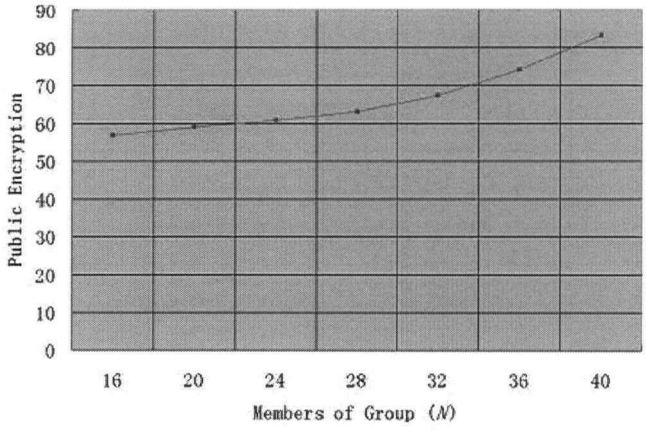
$$= \quad 1 + q \times \boldsymbol{M}$$

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

	Number of Public Encryption
$M \uparrow$	$\uparrow q \times \boldsymbol{M}$
$N \uparrow$	$\uparrow q \times \boldsymbol{N^{1/2}}$



a) M



b) N

Figure 4.1 Experiment result: public key encryption/decryption

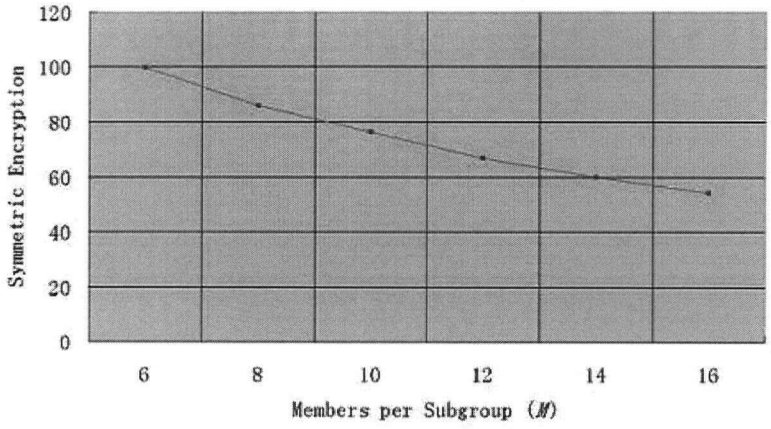
2) Number of Symmetric Encryption:

$$2 \times (1\text{-}q) + (N/M) \times q$$

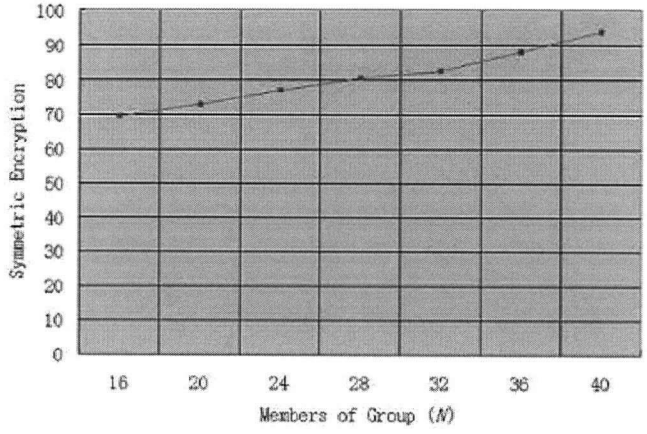
$$= \quad 3 \times (1\text{-}q) + (N \times q) / \boldsymbol{M}$$

$$= \quad 3 \times (1\text{-}q) + q \times \boldsymbol{N^{1/2}}$$

	Number of Symmetric Encryption
$M \uparrow$	$\downarrow (N \times q) / \boldsymbol{M}$
$N \uparrow$	$\uparrow q \times \boldsymbol{N^{1/2}}$



a) M



b) N

Figure 4.2 experiment result: symmetric key encryption

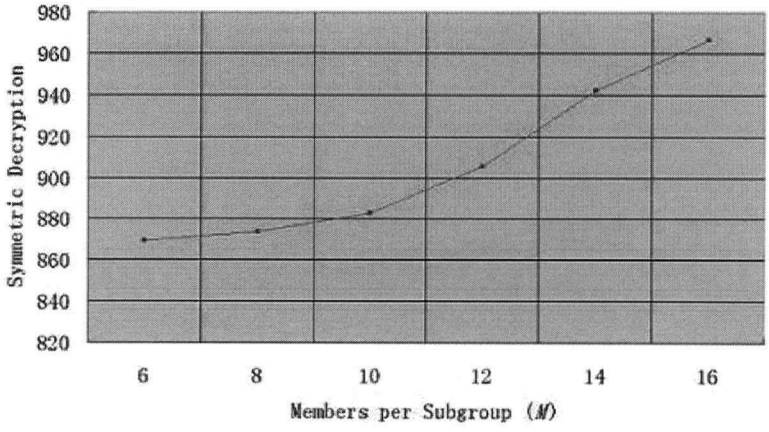
3) Number of Symmetric Decryption:

$$(M+N)\times(1\text{-}q)+N\times q$$

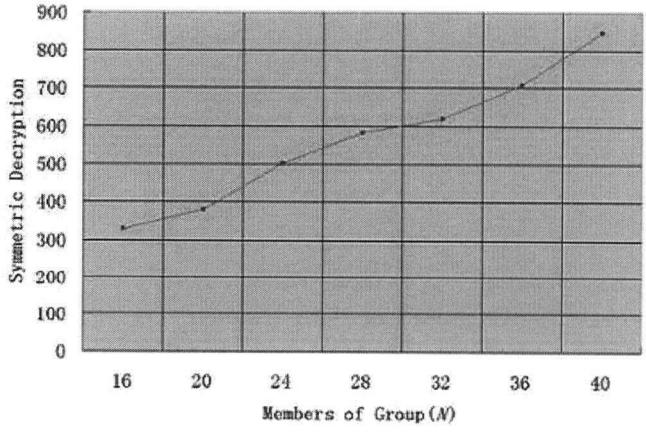
$$= \quad N+(1\text{-}q)+(1\text{-}q)\times \boldsymbol{M}$$

$$= \quad (1\text{-}q)+\boldsymbol{N}+(1\text{-}q)\times \boldsymbol{N}^{1/2}$$

	Number of Symmetric Decryption
$M \uparrow$	$\uparrow (1\text{-}q) \times \boldsymbol{M}$
$N \uparrow$	$\uparrow \boldsymbol{N} + (1\text{-}q) \times \boldsymbol{N}^{1/2}$



a) M



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.

	Join	Non-Leader Leave	Leader Node Leave (Merge)	Leader Node Leave (Replicate)
Num of Unicast	*2*	*1*	*0*	*1*
Size of Unicast Msg	$[E] + 2\times[K] + M\times([E] + [ID])$	$[ID]$	0	$M\times([E] + [ID])$
Num of Multicast	2	$T + 1$	$T + 4$	$T + 4$
Size of Multicast Msg	$2\times[K] + [E] + [ID]$	$T\times[K] + M\times[ID] + M\times[K]$	$[ID] + 2\times a\times M \times([E]+[ID]) + 2\times a\times M\times[K]+ T\times[K]$	$2\times[ID] + [E] + 2\times M\times[K]+ T\times[K]$
Round	4	$T + 2$	$T + 3$	$T + 4$

Table 4.3 Summary of communication cost

4.4.1 Comparing with other protocols

	P2P-HGKM	GDH 2.0	AT-GDH	Tree-Based
Join	2 Unicast 2 Multicast	(N-1) Unicast + 1 Multicast	$2\times N$ Unicast	h multicast + 1 unicast
	4 Round	N Round	$2k \times (log_k N)$ Round	h Round
Leave	2 Unicast 4+ T Multicast	(N-1) Unicast 1 Multicast	$2N$ Unicast	(h-1) $\times$ (d-1) multicast
	4 + T Round	N Round	$2k\times(log_k N)$ Round	h Round

Table 4.4 Communication cost comparison Comparing communication cost with other key establishment 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

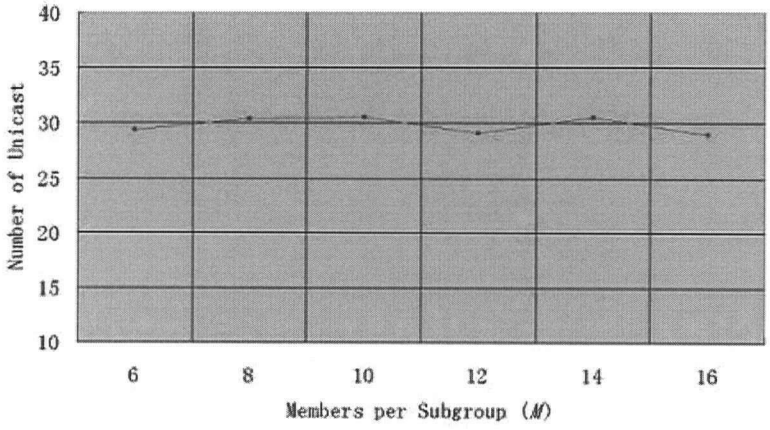
Like in 4.2.2, we will study the effect of parameter M, q and N on communication cost.

1) Number of Unicast:

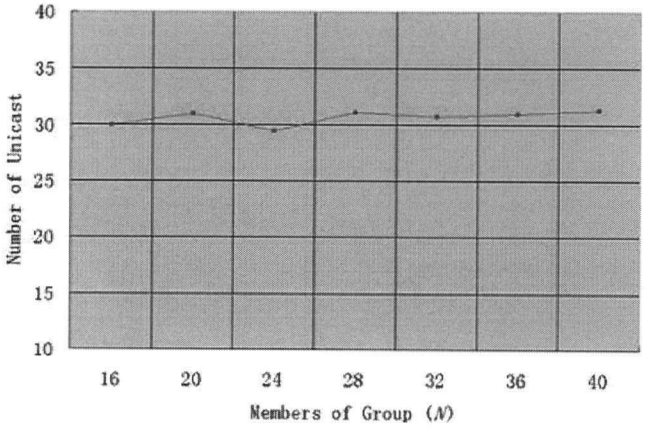
$$2\times(1-q) + q\times(M-1)/M + (1/M)\times q$$

$$= \quad 2-q$$

	Number of Unicast
$M \uparrow$	Unchanged $\leftrightarrow$
$N \uparrow$	Unchanged $\leftrightarrow$



a) M



b) N

Figure 4.4 Experiment result: number of unicast

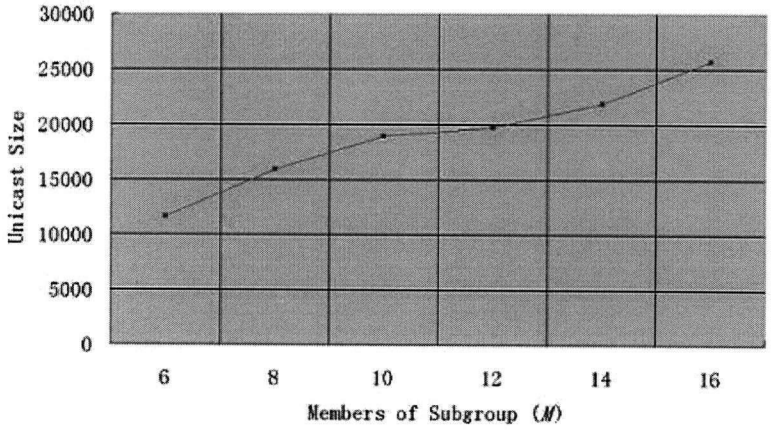
2) Size of Unicast

$$([E]+2[K]+M\times[E]+M\times[ID])\times(1-q) + ((M-1)/M)\times([ID]) \times q + M\times([E]+[ID])\times(1/M) \times q$$

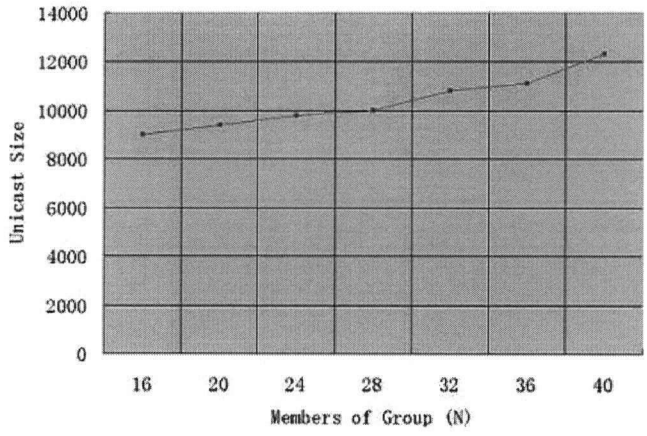
$$= \quad ([E]+2[K]\times(1-q) + 2\times[ID]\times q) + ([E]+[ID])\times(1-q)\times \boldsymbol{M} - [ID]\times q / \boldsymbol{M}$$

$$= \quad ([E]+2[K]\times(1-q) + 2\times[ID]\times q) + ([E]+[ID])\times(1-q)\times \boldsymbol{N^{1/2}} - [ID]\times q / \boldsymbol{N^{1/2}}$$

	Message Size of Multicast
$M \uparrow$	$\uparrow([E]+[ID])\times(1-q)\times \boldsymbol{M} - [ID]\times q / \boldsymbol{M}$
$N \uparrow$	$\uparrow ([E]+[ID])\times(1-q)\times \boldsymbol{N^{1/2}} - [ID]\times q / \boldsymbol{N^{1/2}}$



a) M



b) N

Figure 4.5 Experiment result: message size of unicast

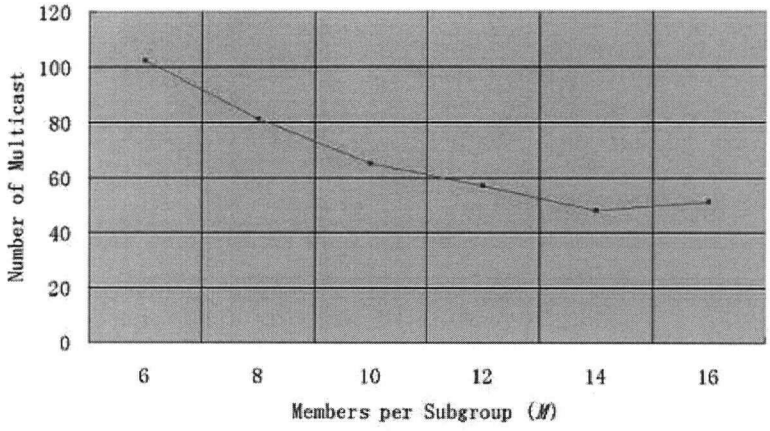
3) Number of Multicast

$$2\times(1-q) + (N/M)\times((M-1)/M)\times q + (4+N/M)\times(1/M)\times q$$

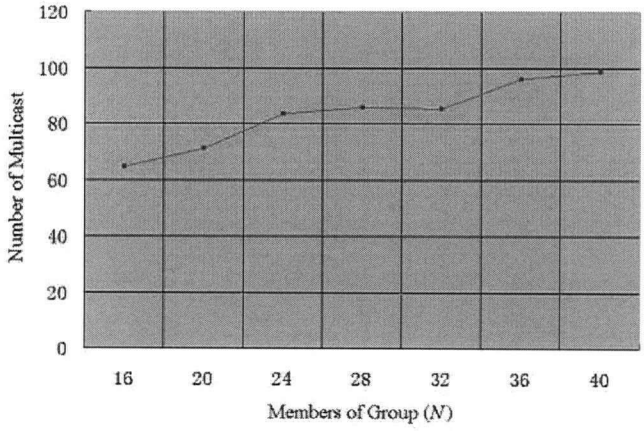
$$= (2 - q) + q\times(N + 3) / \boldsymbol{M}$$

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

	Number of Multicast
$M \uparrow$	$\downarrow q\times(N + 3) / \boldsymbol{M}$
$N \uparrow$	$\uparrow q\times(\boldsymbol{N^{1/2}} + 3 / \boldsymbol{N^{1/2}})$ when $N > 3$



a) *M*



b) *N*

Figure 4.6 Experiment result: number of multicast

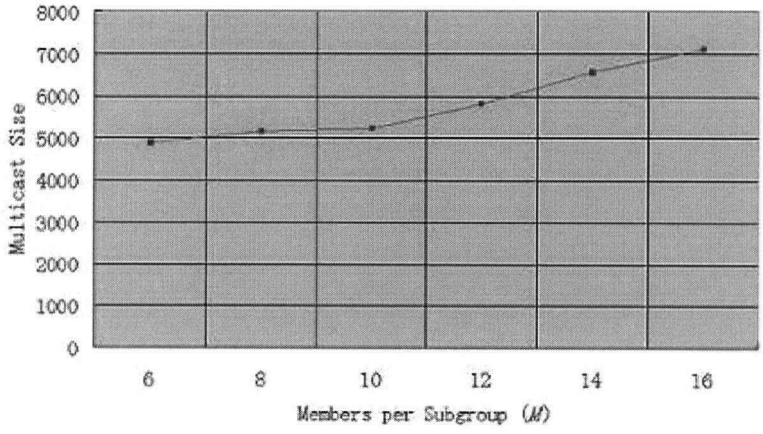
4) Size of Multicast:

$(2[K]+[E]+[ID])\times(1-q) + (N/M \times [K] + M\times[ID] + M\times[K])\times((M-1)/M)\times q + (2\times[ID]+[E] + 2\times M\times[K] + N/M \times [K]) \times (1/M) \times q$

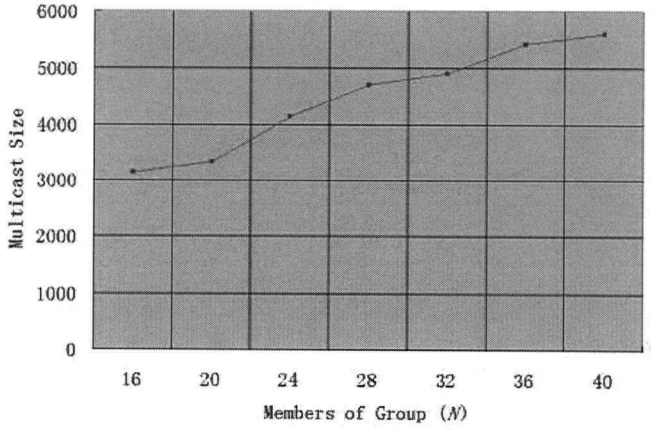
$= (2-q)\times[K]+[E]\times(1-q)+[ID]\times(1-2\times q) + ([ID]+[K])\times q\times \boldsymbol{M} + q\times(N\times[K]+2\times[ID]+[E])/\boldsymbol{M}$

$$= (2\text{-}q)\times[K]+[E]\times(1\text{-}q)+[ID]\times(1\text{-}2\times q) + ([ID]+[K])\times q\times \boldsymbol{N}^{1/2} + q\times(2\times[ID]+[E]) / \boldsymbol{N}^{1/2}$$

	Message Size of Multicast
$M \uparrow$	$\uparrow ([ID]+[K])\times q\times \boldsymbol{M} + q\times(N\times[K]+2\times[ID]+[E])/\boldsymbol{M}$
$N \uparrow$	$\uparrow ([ID]+[K])\times q\times \boldsymbol{N}^{1/2} + q\times(2\times[ID]+[E]) / \boldsymbol{N}^{1/2}$



a) M



b) N

Figure 4.7 Experiment result: message size of multicast

5) Round:

$$4\times(1-q) + (N/M + 2)\times((M-1)/M)\times q + (N/M+4)\times(1/M)\times q$$

$$= \quad (4 - 2\times q) + q\times(N+2) / \boldsymbol{M}$$

$$= \quad (4 - 2\times q) + q\times(\boldsymbol{N^{1/2}} + 2 / \boldsymbol{N^{1/2}})$$

	Round
$M \uparrow$	$\downarrow q\times(N+2) / \boldsymbol{M}$
$N \uparrow$	$\uparrow q\times(\boldsymbol{N^{1/2}} + 2 / \boldsymbol{N^{1/2}})$ when $N > 2$

Chapter 5

Conclusion and Future Work

5.1 Conclusion

To support dynamic secure group communication in ad-hoc networks, a group key K_{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_{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.

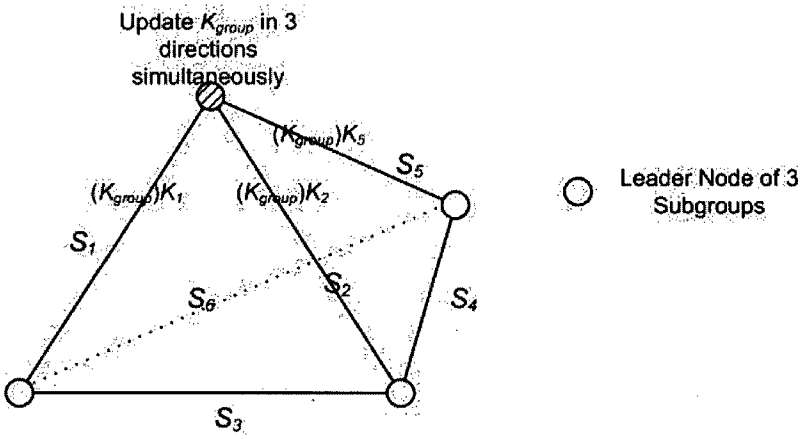


Figure 5.1 Complex structure of subgroups

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_{group} : the group key shared by all members of the multicast group

K_{group_old} : the group key before updating

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

K_{i_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 : provate 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

$M_i \rightarrow M_j$:{Rekey Msg} : M_i unicast {Rekey Msg} to M_j

$M_i \Rightarrow S_c$:{Rekey Msg} : M_i multicast {Rekey Msg} to subgroup S_c

$M_i \Rightarrow G$: {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, $T \approx \lceil N/M \rceil$

Bibliography

[1] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. "secure group communication using key graphs" *In Proceedings of ACM SIGCOMM '98*, September 2-4, 1998.

[2] M. Steiner, G. Tsudik and M. Waidner, "Diffie-Hellman Key Distribution Extended to Groups " *ACM CCS'96*, March 1996

[3] Suvo Mittra: "Iolus: A Framework for Scalable Secure Multicasting". *SIGCOMM 1997: 277-288*

[4] Xiang-Yang Li, Yu Wang, Ophir Frieder, "Efficient Hybrid Key Agreement Protocol for Wireless Ad Hoc Networks", *IEEE 11th International Conference on Computer Communications and Networks (ICCCN2002)*, Oct. 2002

[5] Maarit Hietalahti. "Efficient key agreement for ad-hoc networks". Master's thesis, Helsinki University of Technology, Department of Computer Science and Engineering, May 2001.

[6] Chun Zhang, Brian DeCleene, James F. Kurose, Donald F. Towsley: "Comparison of inter-area rekeying algorithms for secure wireless group communications". *Perform. Eval. 49(1/4): 1-20* (2002)

[7] Charles E. Perkins and Elizabeth M. Royer. "Ad hoc On-Demand Distance Vector Routing." *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, New Orleans, LA, February 1999, pp. 90-100.

[8] Elizabeth M. Royer and Charles E. Perkins. "Multicast Operation of the Ad hoc On-Demand Distance Vector Routing Protocol." *Proceedings of MobiCom '99*, Seattle, WA, August 1999, pp. 207-218.

[9] J.J. Garcia-Luna-Aceves and E.L. Madruga, "The Core Assisted Mesh Protocol", accepted for publication in *IEEE Journal on Selected Areas in Communications, Special Issue on Ad-Hoc Networks*, 1999.

[10] M. Gerla, T.J. Kwon and G. Pei, "On Demand Routing in Large Ad Hoc Wireless Networks with Passive Clustering", *Proceedings of IEEE WCNC 2000*, Chicago, IL, Sep. 2000.

[11] C. Cordeiro, H. Gossain, and Dharma P. Agrawal, "Multicast Over Wireless Mobile Ad Hoc Networks: Present and Future Directions," *IEEE Network, special issue on Multicasting: An Enabling Technology, Vol. 17, No. 1*, January/February 2003, pp. 52-59.

[12] Sung-Ju Lee, William Su, Mario Gerla, "Exploiting the unicast functionality of the on-demand multicast routing protocol", *WCNC 2000 - IEEE Wireless Communications and Networking Conference*, no. 1, September 2000 pp. 1317-1322

[13] Cicso System:
http://www.cisco.com/en/US/about/ac123/ac147/ac174/ac177/about_cisco_ipj_archive_article09186a00800c83e4.html

[14] Whitfield Diffie and Martin E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no.6, pp.644-654, 1976

[15] R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining digital Signatures and Public-Key Cryptosystems", *Communications of the ACM*, vol. 21(2), pp.120-126, 1978

[16] Klaus Becker and Uta Wille, "Communication complexity of group key distribution," in *5th ACM conference on Computer and Communication Security*, Nov. 1998.

[17] Yongdae Kim, Adrian Perrig, and Gene Tsudik. "Simple and fault-tolerant key agreement for dynamic collaborative groups." In *ACM Conference on Computer and Communications Security*. pp. 235-244, 2000

[18] Garcia-Molina, "Elections in Distributed computer Systems", *IEEE Transactionson Computers*, 1982, Vol C-31. No.1, pp 48-59