A MODIFIED CLIQUES KEY AGREEMENT PROTOCOL FOR SECURE MULTICAST

By

Xiaoyan Chen

**Bachelor of Electronic and Information System
Sun Yat-Sen University, Guangzhou, China, 1996**

A thesis

Presented to Ryerson University

In partial fulfillment of the
Requirements for the degree of
Master of Applied Science

In the Program of
Computer Networks

Toronto, Ontario, Canada, 2006

©Xiaoyan Chen 2006

UMI Number: EC53486

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform EC53486

Copyright 2009 by ProQuest LLC

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Author's Declaration

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Abstract

Xiaoyan Chen,

A Modified CLIQUES Key Agreement Algorithm for Secure Multicast

M.A.Sc, Computer Networks, Ryerson University, 2006

In a secured multicast application, members may join or leave frequently. Hence, key management is one of the most challenging problems. A practical and scalable key management requires high security features, efficient key distribution, low key-storage cost, and small processing overhead. CLIQUES [1-5] scheme was proposed to handle contributory key agreement. It extends the two-party Diffie-Hellman [6] algorithm to allow a group of members to “agree” upon a symmetric group key. Among the existing key management approaches, CLIQUES has the lowest cost in establishing the common session key. It also imposes less processing overhead in the user machine. The main drawback of CLIQUES is that a large number of re-key messages have to be exchanged among members when there is a membership change. The number of messages exchanged is proportional to the size of the membership. Consequently, CLIQUES is not very scalable and cannot support large group of members.

In this thesis, we proposed a Static CLIQUES, in which, a static group controller is introduced to reduce the complexity of status synchronization process within the group when there is a membership change. The number of keys stored in each member is smaller than that of the original CLIQUES. In addition, the use of static group controller provides member privacy protection since individual member does not have direct contact with other members. We also proposed a hierarchical CLIQUES structure to support a large number of members by sub-grouping them logically into a hierarchical key tree. It is more scalable than dynamic CLIQUES. The size of re-key message being distributed is found to be comparable with hierarchical key tree approach [7-11].

Acknowledgements

Hereby I would like to express my gratitude to Dr. Bobby Ma and Dr. Cungang Yang, for their tremendous guidance and assistance throughout this research.

I also want to give my gratitude to my husband, Guangsheng Fu, who always supports and encourages me.

Table of Contents

Author's Declaration	ii
Abstract.....	iii
Acknowledgements	iv
Table of Contents	v
List of Abbreviations.....	vii
List of Tables.....	viii
List of Figures.....	ix
Chapter 1: Introduction	1
1.1 Background.....	1
1.2 Key management Issues in Multicast Security	2
1.2.1 Scalability issues.....	4
1.2.2 Security issues.....	5
1.3 Current Multicast Key Management Mechanisms	6
1.3.1 Key Tree-based Approaches	6
1.3.2 Contributory Key Agreement supported by DH algorithm.....	8
1.3.3 Secure Multicast Framework.....	11
1.4 Conclusion	11
Chapter 2: M-CLIQUEs: Modified CLIQUEs Algorithm.....	13
2.1 Static CLIQUEs.....	13
2.1.1 Static Group Controller	13
2.1.2 Single Member Join	14
2.1.3 Mass Join	16
2.1.4 Member Leave.....	18
2.1.5 Key Refresh	19
2.2. Hierarchical CLIQUEs.....	20
2.2.1 GC and SGC	20
2.2.2 Member Join	27
2.2.3 Member Leave.....	32
2.2.4 Key Refresh	32
Chapter 3: Efficient Key Agreement Protocol.....	33
3.1 Optimal Key Tree Structure	33

3.2 M-CLIQUE Key Agreement Protocol Based on Balanced Tree	39
3.2.1 Assumptions	39
3.2.2 Concurrency Control and Interval-Based Re-keying.....	40
3.2.3 Pseudo-codes of Protocol	43
Chapter 4: Implementation of M-CLIQUE	56
4.1 M-CLIQUE Key Agreement Protocol Implementation.....	56
4.1.1 Classes	56
4.1.2 State machines.....	59
4.1.3 Packet and message format	61
4.2 GUI	64
4.3 Re-keying time comparison	67
4.3.1 Scenario 1	67
4.3.2 Scenario 2	69
4.3.3 Scenario 3	70
4.3.4 Comparison.....	72
4.4 Comparison to Other Key management Approaches.....	74
Key Tree	74
Chapter 5: Conclusion	76
References	79

List of Abbreviations

ACK	Acknowledge
AES	Advanced Encryption Standard
API	Application Programming Interface
CDH	Computational DH assumption
DDH	Decisional DH assumption
DES	Data Encryption Standard
DH	Diffie-Hellman key exchange algorithm
GC	Group Controller
G-CDH	Group Computational DH assumption
G-DDH	Group Decisional DH assumption
GS	Group Security Intermediary
GUI	Graphical User Interface
JCE	Java Cryptography Extension
JDK	Java Development Kit
MAC	Message Authentication Code
MD5	Message-Digest algorithm
MTU	Maximum Transmission Unit
NAT	Network Address Translation
PM	Pseudo-Member
RSA	Rivest-Shamir-Adleman (public key encryption scheme)
SGC	Sub-Group Controller
SHA	Secure Hash Algorithm

List of Tables

Table 2. 1 Partial key set, private exponent of the PMand subgroup key of each SGC and GC.....	26
Table 2. 2 Member's keys (After subgroup key calculation, partial keys are removed)	27
Table 4. 1 Packet Types.....	62
Table 4. 2 Fields in Data packet.....	62
Table 4. 3 Fields in Control packet	64
Table 4. 4 Sub-fields in KeySetInfo field	64
Table 4. 5 A comparison of key management schemes.....	74

List of Figures

Figure 1. 1 DH exchange algorithm.....	3
Figure1. 2 Hierarchical key tree: member join and leave.....	7
Figure 1. 3 An example of member join in CLIQUES	9
Figure1. 4 An example of member leave in CLIQUES	10
Figure 2. 1 An example of single member join.....	16
Figure 2. 2 An example of mass join	18
Figure 2. 3 An example of member leave.....	19
Figure 2. 4 An example of key refresh	20
Figure2. 5 Relationship of PM, SG, and SGC.....	22
Figure 2. 6 An example of member join	29
Figure 2. 7 Members in $SG(1,1,1)$ need to update $K(1,1,1)$, $K(1,1)$ and $K(1)$	30
Figure 2. 8 Member in $SG(1,1,2)$ need to update $K(1,1)$ and $K(1)$	30
Figure 2. 9 Members in $SG(1,2,1)$ and $SG(1,2,2)$ need to update $K(1)$	31
Figure 3. 1 An example of member distribution to $N^{1/2}$ SGCs	36
Figure3. 2 An example of member distribution to less than $N^{1/2}$ SGCs.....	36
Figure 3. 3 Re-keying per member	41
Figure3. 4 Re-keying per Subgroup.....	41
Figure 3. 5 An example of Single Member Join	47
Figure 3. 6 An example of single member leave	50
Figure 3. 7 An example of multiple members join and leave	52
Figure 4. 1 State machine for Group Controller (GC).....	59
Figure 4. 2 State machine for Subgroup Controller (SGC)	60
Figure 4. 3 State machine for Member	61
Figure 4. 4 GUIs for GC	65
Figure 4. 5 GUIs for SGC	66

Figure 4. 6 GUIs for member.....	67
Figure 4. 7 A height of 1 tree with maximum number of member: $1 \times 64 = 64$	68
Figure 4. 8 Re-keying time vs. number of existed members when a new member joined the group of one level tree structure	68
Figure 4. 9 Re-keying time vs. number of existing members when a member left the group of one level tree structure	69
Figure 4. 10 A height of two tree with maximum number of member: $8 \times 8 = 64$	69
Figure 4. 11 Re-keying time vs. number of existing members when a new member joins the group of two level tree structure	70
Figure 4. 12 Re-keying time vs. number of existing members when a member leaves the group of two level tree structure	70
Figure 4. 13 A height of three tree with maximum number of member: $4 \times 4 \times 4 = 64$	71
Figure 4. 14 Re-keying time vs. number of existing members when a new member joins the group of three level tree structure	71
Figure 4. 15 Re-keying time vs. number of existing members when a member leaves the group of three level tree structure	72
Figure 4. 16 Member join re-keying time in trees of height 1, 2 and 3	73
Figure 4. 17 Member leave re-keying time in trees of height 1, 2 and 3	73

Chapter 1: Introduction

1.1 Background

Multicast is an efficient way of delivering data from a sender to multiple receivers. The main advantage of multicast is computer resource and network bandwidth saving. Many existing applications, such as video conferencing, Internet radio, pay-per-view TV, and stock-real-time-quoting, take advantages of multicast. These applications, which require sending data to a group of members, use multicast to reduce the transmission overhead of both sender computer and network. As a result of increasing multicast applications, the concerns of Multicast security become more and more important and imperative. Data confidentiality is one of the challenging problems in secured multicast, which means that only authorized users can share the content of a message. Intruders should not be able to reveal the content even they sniff on the transmission units or medium.

In unicast environment, data confidentiality is achieved by delivering encrypted data over the medium. There are similar concerns and requirements about data security in multicast systems. Multicast data must be encrypted before being transmitted over the media. Hence, a shared secret key, refer to as a “group key”, must be secretly distributed to every member in the multicast group before it can be used to decrypt the multicast data. Since members may join or leave the multicast group frequently, the group key must also be changed frequently to achieve forward and backward secrecy [12]. Forward secrecy means a previous member should not be able to decrypt the data sent after he leaves the group, while backward secrecy means a new member could not decrypt the data sent before he joins the group. In order to securely deliver or derive the group key, many intermediate keys may need to be introduced, such as the encryption keys used to encrypt the group key. Consequently, an effective key management mechanism is required to manage key generation, delivery, renewal and storage. Current researches on key

management mechanism are focusing on the issues of system scalability, implementation complexity, and security. These issues will be explained in detail in section 1.2.

This thesis concentrates on developing a scalable and easy-to-implement key management approach of secured Multicast. As a result, an efficient protocol called Static CLIQUES protocol is proposed and studied. Based on Static CLIQUES, a hierarchical version of it, called Hierarchical CLIQUES, is proposed. The Hierarchical CLIQUES is more scalable.

1.2 Key management Issues in Multicast Security

In terms of data encryption, there are two categories of cryptography: symmetric and asymmetric. In a symmetrical system, such as DES [13] and AES [14], a single secret key is used to encrypt and decrypt data. The key is shared secretly between sender and receiver. In an asymmetric system, such as RSA [15], encryption and decryption are handled separately. Sender encrypts data using the receiver's public-key; receiver decrypts data using its own private exponent – hence it is called 'asymmetric'. Symmetric-key algorithm is less complex and can process data quite rapidly but the key must be delivered or derived secretly among parties. Asymmetric-key algorithm is more processing-intensive and slower. In practice, the two are often combined: the asymmetric side is used to exchange a shared secret key; the symmetric side is used to perform bulk data encryption using the exchanged shared secret key. For example, the sender encrypts a shared secret key by RSA using receiver's public key and sends encrypted shared secret key to the receiver. The receiver decrypts the shared secret key using its own private exponent. Then, the sender encrypts data by using the shared secret key. The receiver can decrypt the data by using the same key.

Diffie-Hellmen (DH) key exchange algorithm provides an alternate approach to derive the secret key between two parties. Bellow is the general description of the protocol:

- Alice and Bob agree to use a prime number p and base g .
- Alice picks a random natural number a and sends $g^a \bmod p$ to Bob.
- Bob picks a random natural number b and sends $g^b \bmod p$ to Alice.
- Alice computes $(g^b)^a \bmod p$
- Bob computes $(g^a)^b \bmod p$

Both Alice and Bob have arrived at the same value, because g^{ab} and g^{ba} are equal. Note that only a , b and $g^{ab} = g^{ba}$ are kept *secret*. All the other values could be sent out without encrypted. Therefore, we called a and b the private keying material, g^a and g^b the public keying material. Once Alice and Bob compute the shared secret key they can use it for encryption, known only to them, for sending messages across the same open communications channel. Figure 1.1 illustrates the DH exchange.

Figure 1. 1 DH exchange algorithm

The above key-generation approaches are working fine in two parties communication. Things are different in multicast system. First of all, asymmetric-key algorithm could not be used in data encryption. Because for each public key, there is only one private key and vice versa. There is no “one public key to multiple private keys” algorithm. Hence, the data encryption cipher in multicast should be symmetric-key

algorithm. However, how to securely deliver the shared secret key to every member of the multicast group becomes a new issue. Asymmetric cipher could be used to send the shared secret key. The sender keeps all receivers' public keys and unicasts the shared secret key encrypted to every receiver by using each individual's public key. Therefore, the number of public keys stored in the sender and the overhead of key-renew increases quickly as the group size grows. In addition, if membership changes frequently, the overhead of establishing new private key will also increase significantly. Hence, many researches are focusing on finding a low-cost and efficient secret key distribution and management approach, such as Key Tree-based approach and DH Contributory Key Agreement. These approaches will be introduced in chapter 3.

In the following section, some issues that should be addressed in key management of secured multicast will be introduced.

1.2.1 Scalability issues

Scalability refers to reducing key storage and re-keying cost of every element in a large and highly dynamic multicast group. The following criteria [5] are usually used to evaluate the scalability of the system.

- The number of keys stored in each member and control server

In order to decrypt the multicast data, each member must receive the group key (refer to the shared secret key mentioned before) and store it locally. If asymmetric-key cipher is used to deliver the group key, in order to communicate securely with the control server (it could be reside in the sender's computer), each member may have an individual private key, and the control server must store all the public keys of members. Some key management schemes introduce another type of key such as encryption key of subgroup key, which is used to reduce the total number of re-key messages. All of those keys introduce storage overhead. Apparently, a large key pool not only requires extensive memory but also increases the complexity of key management.

- The number of keys updated upon membership change

All group key and other keys have to be updated upon membership change and sent to the corresponding members. Such a burst of key renewal may be processing-intensive. This is especially critical for a low-power device. Also, the re-key messages include those updated keys. If re-key messages are too large, they may incur other overheads, such as key re-transmission. Because the re-key message contains a number of updated keys, when its size is larger than the MTU of the transmission media, e.g. 1500bytes on Ethernet, it will be fragmented into several smaller packets. If one of the fragmented packets is lost during transmission, the entire re-key message must be re-transmitted. Consequently, the larger the re-key message, the higher possibility of re-transmission will be.

- The cost of establishing a secured multicast system

There are two types of costs: the cost of control servers and the cost of user machines. If the key management scheme could reduce the computation cost and key storage on the member side, the user machine needs less power, less memory and less storage space. Lower cost of user machine will potentially increase the number of subscribers.

1.2.2 Security issues

- Forward secrecy and backward secrecy

A secured multicast system should provide both forward secrecy and backward secrecy. Forward secrecy means a member cannot retrieve the multicasted data after it left the group. Backward secrecy means a member cannot decrypt the multicasted data it cached before it joins the group. This is to allow users to know the group key only when they are granted access rights to the multicast group.

- Knowledge of group members

During the re-key process, members will need to update the group keys and some intermediate keys. It is advised that members should not know too much about re-keying mechanism, such as key generation and key distribution. Because the sender's

organization has no security control of user machines and the security system of user machine could be weak and easy to be hijacked, it would make attacking the multicast system easier if the member application in the user machine knows the mechanisms involved in the key management system.

- Member privacy

In some multicast systems, members may not want to know each other's existence. The secure multicast key management scheme needs to protect members' privacy. Only control server knows members' identities.

1.3 Current Multicast Key Management Mechanisms

1.3.1 Key Tree-based Approaches

The hierarchical key tree is a logical tree structure used to derive secret key in a multicast group. The members are at the leaves of the tree and the root and the internal nodes are representing the keys. Figure 1.2 shows an example of the key tree. There are three types of keys. The first type is the group key. The group key is at the root node, in the upper part of Figure 1.2 it is K_{1-9} , means member M_1 - M_9 share this key. The second type is subgroup key, at the internal node. K_{1-3} , K_{4-6} , and K_{7-9} in the upper part of Figure 1.2 are the subgroup keys. Member M_1 - M_3 share K_{1-3} , Member M_4 - M_6 share K_{4-6} , respectively. The third type is the member's individual key, at the leaf of the tree. K_1 - K_9 in the upper part of Figure 1.2 are members' individual keys and not shared by other members. A control server is managing this tree. It is responsible for keys generation and distribution. A member holds its individual key, the group key and corresponding subgroup keys. For example, in the upper part of Figure 1.2, M_1 holds K_1 , K_{1-3} , and K_{1-9} .

Figure1. 2 Hierarchical key tree: member join and leave

When a membership change occurs, all keys held by the leaving member or joining member must be refreshed in order to achieve forward and backward secrecy. As an example, suppose M_6 is new member joining group, as shown in Figure 1.2. K_{1-9} and K_{4-5} must be changed to K'_{1-9} and K_{4-6} . Since neither K_{1-9} nor K_{4-5} is known by M_6 , the control server can encrypt the new group key K'_{1-9} with the old group key K_{1-9} and multicasts them to the old members. The control server then encrypts the new subgroup key K_{4-6} with old subgroup key K_{4-5} and multicasts it to M_4 and M_5 . Finally, the control server encrypts K'_{1-9} and K_{4-6} with M_6 's individual key K_6 and unicasts it to M_6 .

On the other hand, suppose M_6 leaves the group, K'_{1-9} and K_{4-6} must be changed to K_{1-9} and K_{4-5} . The control server encrypts new subgroup key K_{4-5} with K_4 and K_5 and

unicasts them to M_4 and M_5 , respectively. It then encrypts the new group key $K_{1,9}$ with each subgroup key, $K_{1,3}$, $K_{4,5}$, and $K_{7,9}$, and multicasts them to each subgroup respectively.

If the key tree is a balanced full tree with total number of members of the group equals to N , and the total number of children of each subgroup equals to m , the size of re-key messages per join and leave are proportional to the height of the tree, $\log_m N$. As the re-key message contains the keys that are changed, the size of re-key message is proportional to the number of changed keys being transmitted. For each join, there are two re-key messages to be multicasted. One includes the updated keys to the old members encrypted by the old keys. The other includes the updated keys to the new member. Hence the size of re-key messages is $2\log_m N$. For each leave, since the leaving members know the old keys, every updated subgroup keys needs to be encrypted by each lower level subgroup's keys and sent to each lower level subgroup. Hence, the size of re-key messages is $m\log_m N$.

In a simple solution of key management scheme, the size of re-key message of a simple solution of key management scheme equals to N (the size of the group), because the new group key has to be encrypted member by each member's individual key and unicasted to the member. Comparing to this kind of solution, the key tree-based approach reduces the size of re-key messages. However, since the control server is managing all of the keys, when the group size increases, the processing time for updating subgroup keys will be slower. So, it is an approach suitable for a mildly large multicast group.

1.3.2 Contributory Key Agreement supported by DH algorithm

CLIQUE is a contributory key agreement approach which extends DH key exchange algorithm. It focuses on the issue of "key agreement" among members instead of generating and distributing keys by a control server. The idea of CLIQUE comes from the following equations:

$$g^{abc} = (g^{ab})^c = (g^{bc})^a = (g^{ca})^b$$

Hence, if there are three members A, B and C in the group, if A knows g^{bc} , it can calculate the group key g^{abc} , same as B and C. g^{ab} , g^{bc} and g^{ca} are called partial keys of the group, while a , b , and c are called private exponents. Researchers believe that nobody else can compute the shared key g^{abc} in a reasonable amount of time even though they know all of the partial keys. The algorithm can be extended to multiple members. The group key contains every member's contribution, e.g. a , b and c . If each member gets its partial key that contains all other members' contributions, then it can calculate the group key. The size of re-key message is proportional to the size of the group, for both member-join and member-leave. Consequently, it is good for a small size group.

Figure 1.3 gives an example of a member joining the group. Figure 1.4 gives an example of a member leaving the group.

Figure 1.3 An example of member join in CLIQUES

Stage1: M4 deletes M2's partial key $g^{s_1s_3s_4}$

$$\text{New group key: } \overset{\text{M1}}{g^{(s_2s_3s_4')s_1}} = \overset{\text{M3}}{g^{(s_1s_2s_4')s_3}} = \overset{\text{M4}}{g^{(s_1s_2s_3)s_4'}} \quad \overset{\text{M2}}{\text{Null}}$$

Figure1. 4 An example of member leave in CLIQUES

In CLIQUES, the last member of the group is acting like a group controller responsible for re-key message transmission. It stores all of the partial keys. Other members only store their own partial key and private exponent. When a new member joins, the last member sends all of the old partial keys to the new member. After the new member adding its private exponent into each partial key and send back to the old members, the new member becomes the last member of the new group. Hence it takes over the role of group controller. The last member is also responsible for updating partial keys when member leaving and key refreshing. If the last member leaves the group, a new group controller needs to be selected among the remaining members. The dynamic group controller mechanism increases the scheme complexity and reduces the system's security, as all members know how the entire protocol works. The requirement to the user's machine is high as well, as every member has chances to take role of group controller. Moreover, implementing such a dynamic group controller mechanism is not easy in reality. Because the current group controller can be in a user machine on a company's LAN, usually the company won't allow connection initiated from outside to a user machine due to security concerns and lack of public IP addresses (private IP addresses are used within the company, NAT is performed when accessing Internet). Consequently, a new member may not be able to connect to a dynamically assigned

group controller within a company's LAN.

1.3.3 Secure Multicast Framework

The Iolus Framework [16] is one of the secure multicast framework schemes [16-19]. It is a distributed system. The group is divided into several subgroups. The rule of group division can be based on geographic region. The members are physically within subgroups instead of being grouped logically as they are in the above approaches. A group security intermediary (GSI) is an agent in each subgroup. The group GSI encrypts data with a random key, encrypts this random key with its subgroup key, and appends the encrypted random key into the data packets. The downstream GSI decrypts the random key received from the upper-level GSI, re-encrypts the random key with its own subgroup key and multicast the encrypted data and encrypted random key to its downstream subgroup. This is called "key translation".

When a member joins a subgroup, GSI generates a new subgroup key. The new subgroup key can be encrypted with the old subgroup key and sent to the subgroup, and be encrypted with the new member's individual key and sent to the new member. For a member leave operation, the new subgroup key has to be encrypted by each member's individual key and unicasted to corresponding members.

This approach is good for wide-area multicast group because it breaks the group into smaller subgroups. However, since the GSIs must perform key translation, it introduces packet delay. Therefore it is not working very well in real-time multicast systems, especially when the group is very large.

1.4 Conclusion

The key management approaches introduced above have their own advantages as well as drawbacks. In the key tree-based approach, the controller generates and stores a large number of keys, hence it experiences heavy key re-generation overhead upon

membership change. CLIQUES key agreement scheme is not scalable. Secure framework is not suitable for real-time multicast applications.

In this thesis, a more efficient key management scheme, M-CLIQUES, is proposed. M-CLIQUES is a modification of CLIQUES introduced in section 1.3.2. The M-CLIQUES algorithm is introduced in chapter 2. Chapter 3 analyzes the optimal structure of the hierarchical CLIQUES key tree and illustrates the procedures of constructing a M-CLIQUES system. Chapter 4 demonstrates the implementation of M-CLIQUES in JAVA, including classes' description, diagrams of state machines, and message formats. Also, results from the implementations were collected and analyzed. Chapter 5 is the conclusion of this thesis. M-CLIQUES is compared with key-tree based approach and dynamic CLIQUES on several aspects.

Chapter 2: M-CLIQUE: Modified CLIQUES Algorithm

As demonstrated above, CLIQUES is a scheme that uses the contributory key agreement approach. It extends the two-party DH algorithm to allow a group of members to “agree” upon a symmetric group key thus imposes less decryption overhead in the user machine. But the dynamic group controller mechanism increases the scheme complexity. The other drawback of CLIQUES is that a large number of re-key messages have to be exchanged among members when there is a membership change. The number of messages exchanged is proportional to the size of the membership. Consequently, CLIQUES is not very scalable. In this thesis, a more scalable key management mechanism M-CLIQUE, which is also relatively simpler to implement, is proposed. The proposed scheme is a modification of CLIQUES. The modification consists of two stages. In section 2.1, Static CLIQUES is introduced, in which a static group controller (GC) is used to distribute partial keys. In section 2.2, we propose Hierarchical CLIQUES, which employs a hierarchical structure, to make the key management mechanism more scalable.

2.1 Static CLIQUES

2.1.1 Static Group Controller

CLIQUES relies on Group Controller (GC) to distribute partial keys to the members. In current implementation of CLIQUES, GC is dynamically assigned [20]. Essentially, the last-joined member becomes the GC. If the current GC leaves the group, another member will take over the GC’s role. CLIQUES assumes that the underlying communication system provides a timely and consistent membership view to all group members, which is referred as membership view synchrony.

In this paper, we introduce a static group controller (GC) that provides central controls of all membership change activities. GC is also the first member of the group,

identified as M_0 . Comparing with dynamic CLIQUES, Static CLIQUES is found more secure for key storage, less complex for not requiring status synchronization, less processing requirement in the user machine, and easier to provide member privacy protection. A GC can be a process residing in the same machine with the multicast application server (usually the multicast sender), if any, or can be running in a separate high-performance server. If GC runs on a separate machine and there is a multicast application server, it should have reliable network connection between them. A secure channel must be setup between GC and multicast application server before GC can provide services for the multicast group. Since this can be done using existing secure unicast methods, it is not covered in this thesis.

2.1.2 Single Member Join

We denote $g^{R\Pi(S_p | p \in [0, n+1] \wedge p \neq j)}$ for member j , M_j , as its partial key P_j . When a new member wants to join the group, the group GC and the new member must first authenticate each other. The initial authentication is needed, since DH algorithm, which is used in the CLIQUES to generate the group key, is vulnerable to the Man-in-the-Middle attack [21]. The authentication before DH key exchange is used to prevent this kind of attack. After the authentication, the member join process follows the following step:

1) GC generates a random number R , calculates

$$g^{R\Pi(S_p | p \in [0, n] \wedge p \neq j)}, \text{ for } j \in [0, n]$$

and,

$$g^{R\Pi(S_p | p \in [0, n])}$$

where S_p is the private exponent generated by the member p ; g is the base used in DH exchange algorithm, and n is the number of current members, excluding the new member, in the group. GC sends the calculated values to the new member.

2) The new member, M_{n+1} , generates its private exponent, S_{n+1} , and computes a new partial key set by adding S_{n+1} into every old partial keys it received:

$$g^{R\prod (S_p | p \in [0, n+1] \wedge p \neq j)}, \text{ for } j \in [0, n+1]$$

Note that the new partial key set contains partial key for each member including M_{n+1} 's partial key $g^{R\prod (S_p | p \in [0, n])}$. The partial key set is the collection of partial keys.

The calculated values above forms the new partial key set. The new member M_{n+1} sends this new partial key set back to GC and computes the new group key:

$$K_{new} = g^{R\prod (S_p | p \in [0, n]) S_{n+1}} = P_{n+1}^{S_{n+1}}$$

3) GC multicasts new partial key set to the other members. GC also calculates the new group key:

$$K_{new} = g^{R\prod (S_p | p \in [1, n]) S_0} = P_{0new}^{S_0}$$

4) When member M_j receives the partial key set, it calculates the new group key:

$$K_{new} = g^{R\prod (S_p | p \in [1, n+1] \wedge p \neq j) S_j} = P_{jnew}^{S_j}$$

Figure 2.1 gives an example of M_4 joins a multicast group.

$$g^{R_{5^2 1^3 2^3 3^4}} = g^{(R_{1^2 2^3 3^4})_{5^2}} = g^{(R_{0^2 2^3 3^4})_{5^2}} = g^{(R_{0^2 1^3 3^4})_{5^2}} = g^{(R_{0^2 1^2 2^4})_{5^2}} = g^{(R_{5^2 1^3 2^3})_{5^2}}$$

Figure 2. 1 An example of single member join

2.1.3 Mass Join

When there are multiple members M_{n+1} to M_{n+m} join the group simultaneously, mass join process takes place.

1) GC generates a random number R , calculates

$$g^{R \prod (S_p | p \in [0, n] \wedge p \neq j)}, \text{ for } j \in [0, n]$$

and,

$$g^{R \prod (S_p | p \in [0, n])}$$

GC sends the calculated values to M_{n+1} .

2) M_{n+1} generates its private exponent S_{n+1} and computes a new partial key set by adding S_{n+1} into every old partial keys it received:

$$g^{R \prod (S_p | p \in [0, n+1] \wedge p \neq j)}, \text{ for } j \in [0, n+1]$$

M_{n+1} sends this new partial key set back to GC.

3) GC relays $g^{R\prod (S_p | p \in [0, n+1] \wedge p \neq j)}$, for $j \in [0, n+1]$ to M_{n+2} .

4) M_{n+2} generates its private exponent S_{n+2} and computes a new partial key set by adding S_{n+2} into every partial keys it received:

$$g^{R\prod (S_p | p \in [0, n+2] \wedge p \neq j)}, \text{ for } j \in [0, n+2]$$

M_{n+2} sends this new partial key set back to GC.

5) The rest of new members generates their private exponents and add them into partial keys respectively, until M_{n+m} send the final new partial key set back to GC.

6) GC multicasts new partial key set to the other members. GC also calculates the new group key:

$$K_{new} = g^{R\prod (S_p | p \in [1, n]) S_0} = P_{0new}^{S_0}$$

7) When member M_j receives the partial key set, it calculates the new group key:

$$K_{new} = g^{R\prod (S_p | p \in [1, n+1] \wedge p \neq j) S_j} = P_{jnew}^{S_j}$$

Figure 2.2 gives an example when M_3 and M_4 join the group of two members.

Group key	GC	M1	M2	M3	M4
-----------	----	----	----	----	----

$$g^{R_{s0s1s2s3s4}} = g^{(R_{s1s2s3s4})s_0} = g^{(R_{s0s2s3s4})s_1} = g^{(R_{s0s1s3s4})s_2} = g^{(R_{s0s1s2s4})s_3} = g^{(R_{s0s1s2s3})s_4}$$

Figure 2. 2 An example of mass join

2.1.4 Member Leave

If M_d leaves the group, GC deletes the partial key that does not contain S_d , which is M_d 's contribution to the partial key. Then, GC generates a random number R , re-calculates all the remaining partial keys: $g^{R \prod_{(S_p | p \in [0, n] \wedge p \neq j)} S_p}$, for $j \in [0, n] \wedge j \neq d$, and multicasts them to the remaining members. The remaining members compute the new group key based on the new partial key set. It is no problem that the new group key still

contains M_d 's contribution S_d . Since M_d cannot get its public component (partial key) that includes GC's new exponent R from the renew-key message (because M_d 's partial key has been removed), it cannot compute the new group key.

If multiple members leave a subgroup at the same time, GC deletes every leaving members' partial keys. The subsequent process is the same as single member leave group.

Figure 2.3 gives an example of M_2 leave the group.

Figure 2. 3 An example of member leave

2.1.5 Key Refresh

Hackers may keep trying to calculate the group key from the collected partial keys, so a group key could not be used permanently. A proper lifetime of a group key should be defined. When the current key's lifetime is aged out, GC initializes the key refreshment process:

- 1) GC generates a random number R and a new private exponent, $S_{0_{new}}$, calculates new partial key set ($g^{RS_{0_{new}} \prod (S_p | p \in [0, n] \wedge p \neq j)}$, for $j \in [1, n]$, and $g^{R \prod (S_p | p \in [1, n])}$), multicasts it to all other members, and computes $K_{new} = g^{R \prod (S_p | p \in [1, n]) S_{0_{new}}}$.
- 2) Member M_j calculate $K_{new} = g^{RS_{0_{new}} \prod (S_p | p \in [0, n] \wedge p \neq j) S_j}$.

Figure 2.4 gives an example of key refresh.

Figure 2. 4 An example of key refresh

2.2. Hierarchical CLIQUES

Since CLIQUES is not very scalable and cannot support a large group of members, a Hierarchical CLIQUES algorithm that groups members into a hierarchical key tree, is proposed. The number of messages and times of key recalculation in Hierarchical CLIQUES is comparable to the Key Tree-Based Approach.

2.2.1 GC and SGC

In Hierarchical CLIQUES, the members of the multicast group are divided into a number of subgroups. Its Sub-group Controller, SGC, which has the similar function as GC in Static CLIQUES, controls key exchange in a subgroup. The members and SGCs of different subgroups can be re-grouped to form an upper level subgroup, and eventually a hierarchical tree. We denote a q^{th} level subgroup as $SG(G_1, G_2, \dots, G_q)$, where G_i is the subgroup ID at the i^{th} level, $G_i \in [0, n]$ (n is the maximum number of members or pseudo-members in a subgroup), for $i \in [2, q]$, and $G_1 \equiv 1$.

Figure 2.5 shows the hierarchical structure with 4 levels. Note that there is only

one subgroup in level 1, which is $SG(1)$. GC is the root of the hierarchical tree. It controls join/leave activities in $SG(1)$. The shared secret key of $SG(1)$ is the group key of the entire multicast group. Each SGC controls a lower level subgroup. SGC is always the first member (member 0) of the subgroup it controls. Elements in a subgroup can be normal multicast members (Ms) or pseudomembers (PMs), which represent lower-level subgroups. PM and subgroup are the same in definition from this point. Ms and PMs from a given subgroup agree upon a shared secret key for the subgroup. The shared secret key for the subgroup is referred as a subgroup key. The private exponent of PM is not randomly generated; in fact, it is the subgroup key of the subgroup that this PM represents. The reason is, a subgroup key is unknown by members which do not belong to this subgroup, therefore it can be a private exponent of the PM in upper level subgroup. The subgroup key changes when subgroup membership changes. Because of this, the private exponent of PM changes as well. In the view of upper level subgroup, it is equivalent to a PM initiates a key refresh process.

Figure2. 5 Relationship of PM, SG, and SGC

There is a partial key set associated with each subgroup held by each SGC. M or PM within the subgroup holds the subgroup key and its private exponent, similar to Static CLIQUES. When there is a membership change within the subgroup, Ms or PMs get their corresponding updated partial keys from the SGC, calculate the new subgroup key by adding their private exponents into their own new partial keys for this subgroup. The new subgroup key becomes the new private exponent of the upper level PM . So Ms and PMs can calculate the new upper level subgroup key by doing exponential function with their partial keys in the upper level as base and the new private exponent, level by level. Literally, Ms and PMs can compute the group key by themselves.

Let member j in $SG(G_1, G_2, \dots, G_q)$ be denoted as $M(G_1, G_2, \dots, G_q, j)$. We consider that $M(G_1, G_2, \dots, G_q, j)$ also belongs to the

subgroups $SG(G_1, G_2, \dots, G_{q-1}), SG(G_1, G_2, \dots, G_{q-2}), \dots, SG(G_1, G_2)$, and $SG(G_1)$. We call $SG(G_1, G_2, \dots, G_q)$ as the immediate subgroup of the member. The other subgroups the member belongs are called its upper subgroups. For example, in Figure 2.5, $M(1,1,1,1)$ belongs to $SG(1,1,1)$, $SG(1,1)$, and $SG(1)$; $SG(1,1,1)$ is the immediate subgroup; $SG(1,1)$ and $SG(1)$ are the upper subgroups.

Each member holds its private exponent in the immediate subgroup, all the subgroup keys (upper level PMs' private exponent), and partial keys of the upper level PMs. We denote the private exponent as $S(G_1, G_2, \dots, G_q, i)$, the partial key as $P(G_1, G_2, \dots, G_q, i)$, and the subgroup key as $K(G_1, G_2, \dots, G_q)$, where i means the member is the i^{th} member in its immediate subgroup $SG(G_1, G_2, \dots, G_q)$. There are:

$$\begin{aligned} P(G_1, G_2, \dots, G_q, i) &= g^{\prod (S(G_1, G_2, \dots, G_q, p)) | p \in [0, n]^q, p \neq i} ; \\ K(G_1, G_2, \dots, G_q) &= g^{S(G_1, G_2, \dots, G_q, i) \prod S(G_1, G_2, \dots, G_q, p) | p \in [0, n]^q, p \neq i} \\ &= P(G_1, G_2, \dots, G_q, i)^{S(G_1, G_2, \dots, G_q, i)} \end{aligned}$$

Let $PM(G_1, G_2, \dots, G_q)$ be the j^{th} member of its immediate subgroup $SG(G_1, G_2, \dots, G_{q-1})$; it also can be denoted as $PM(G_1, G_2, \dots, G_{q-1}, j)$ because in this case $G_q = j$. Let the private exponent of $PM(G_1, G_2, \dots, G_{q-1}, j)$ be $S(G_1, G_2, \dots, G_{q-1}, j)$. Note that

$$S(G_1, G_2, \dots, G_{q-1}, j) = K(G_1, G_2, \dots, G_{q-1}, j);$$

where $K(G_1, G_2, \dots, G_{q-1}, j)$ is subgroup key of subgroup $SG(G_1, G_2, \dots, G_{q-1}, j)$.

Also, let $P(G_1, G_2, \dots, G_{q-1}, j)$ be the partial key of $PM(G_1, G_2, \dots, G_{q-1}, j)$ in subgroup $SG(G_1, G_2, \dots, G_{q-1})$, $P(G_1, G_2, \dots, G_{q-1}, j)$ is given as:

$$P(G_1, G_2, \dots, G_{q-1}, j) = g^{\prod (S(G_1, G_2, \dots, G_{q-1}, p)) | p \in [0, n] \wedge p \neq j}$$

Using the partial key and the private exponent, $PM(G_1, G_2, \dots, G_{q-1}, j)$ can compute the subgroup key of $SG(G_1, G_2, \dots, G_{q-1})$:

$$\begin{aligned} K(G_1, G_2, \dots, G_{q-1}) &= g^{S(G_1, G_2, \dots, G_{q-1}, j) \prod (S(G_1, G_2, \dots, G_{q-1}, p)) | p \in [0, n] \wedge p \neq j} \\ &= P(G_1, G_2, \dots, G_{q-1}, j)^{S(G_1, G_2, \dots, G_{q-1}, j)} \\ &= P(G_1, G_2, \dots, G_{q-1}, j)^{K(G_1, G_2, \dots, G_{q-1}, j)} \quad \text{---(1)} \end{aligned}$$

Note that the newly derived subgroup key becomes the private exponent of $PM(G_1, G_2, \dots, G_q)$. Also note that since a member holds the partial keys of its upper subgroups, it can use equation (1) iteratively to generate the subgroup keys of all upper subgroups, including $SG(G_1)$, or more precisely, $SG(1)$. The subgroup key of $SG(1)$ of course is the group key of the entire multicast group.

Eventually, the group key K can be calculated as:

$$K = K(1) = g^{S(1,1) \prod (S(1,p)) | p \in [0, n] \& p \neq 1} = P(1,1)^{S(1,1)}$$

In general, a j^{th} member of an immediate subgroup $SG(G_1, G_2, \dots, G_q)$ holds its private exponent and $P(G_1, G_2, \dots, G_q, j)$, $P(G_1, G_2, \dots, G_q)$, ..., $P(G_1, G_2)$; it also holds: $K(G_1, G_2, \dots, G_q)$, $K(G_1, G_2, \dots, G_{q-1})$, ..., $K(G_1, G_2)$, $K(G_1)$. ($K(G_1)$, which is equivalent to $K(1)$, is the subgroup key of subgroup $SG(1)$, hence it is the group key of the entire multicast group.) These subgroups keys are used to generate new subgroup keys for all its upper subgroups whenever there is a membership change in the whole multicast group, demonstrated above.

Besides holding the keying material required as a member of its immediate subgroup, SGC will also hold the partial key sets of all its upper subgroups. The partial key sets are used by the SGC to compute new partial key sets to be multicasted to all the members whenever an event of member join or member leave or key refresh occurs in the

subgroup.

Based on the example shown in Figure 2.5, Table 2.1 shows the partial key set, private exponent of the PM, and the subgroup key of each SGC and GC in each level. Table 2.2 shows the partial key, private exponent of the PM, and the subgroup key of each member in each level. In a member machine, the partial keys are used for calculating the subgroup keys. After calculation, the partial keys are removed from member machine's memory. When there is a membership change or key refresh event in the group, the member can extract the appropriate level partial key from the key renew message, and do DH calculation using that level's private exponent. Not saving partial keys helps to reduce the cost of key storage in member machines.

Note that SGC can be a physical server, or can be a functional module residing in the physical server of GC or other SGCs. Moreover, each physical server may further split and merge into multiple logical servers according to the size of the group, though we have not considered it in our proposed protocol yet.

Level	Keys	GC	
1	Partial Key Set	$P(1,0), P(1,1), P(1,2)$	
	Node's private exponent	$S(1,0)$	
	Subgroup Key	$K(1)$	
2	Partial Key Set		
	Node's private exponent		
	Subgroup Key		
3	Partial Key Set		
	Node's private exponent		
	Subgroup Key		
Level	Keys	$SGC(1,1)$	$SGC(1,2)$

1	Partial Key Set	$P(1,0), P(1,1), P(1,2)$	$P(1,0), P(1,1), P(1,2)$		
	Node's private exponent	$S(1,1)-K(1,1)$	$S(1,2)-K(1,2)$		
	Subgroup Key	$K(1)$	$K(1)$		
2	Partial Key Set	$P(1,1,0), P(1,1,1), P(1,1,2)$	$P(1,2,0), P(1,2,1), P(1,2,2)$		
	Node's private exponent	$S(1,1,0)$	$S(1,2,0)$		
	Subgroup Key	$K(1,1)$	$K(1,2)$		
3	Partial Key Set				
	Node's private exponent				
	Subgroup Key				
Level	Keys	$SGC(1,1,1)$	$SGC(1,1,2)$	$SGC(1,2,1)$	$SGC(1,2,2)$
1	Partial Key Set	$P(1,0), P(1,1), P(1,2)$	$P(1,0), P(1,1), P(1,2)$	$P(1,0), P(1,1), P(1,2)$	$P(1,0), P(1,1), P(1,2)$
	Node's private exponent	$S(1,1)-K(1,1)$	$S(1,1)-K(1,1)$	$S(1,2)-K(1,2)$	$S(1,2)-K(1,2)$
	Subgroup Key	$K(1)$	$K(1)$	$K(1)$	$K(1)$
2	Partial Key Set	$P(1,1,0), P(1,1,1), P(1,1,2)$	$P(1,1,0), P(1,1,1), P(1,1,2)$	$P(1,2,0), P(1,2,1), P(1,2,2)$	$P(1,2,0), P(1,2,1), P(1,2,2)$
	Node's private exponent	$S(1,1,1)=K(1,1,1)$	$S(1,1,2)=K(1,1,2)$	$S(1,2,1)=K(1,2,1)$	$S(1,2,2)=K(1,2,2)$
	Subgroup Key	$K(1,1)$	$K(1,1)$	$K(1,2)$	$K(1,2)$
3	Partial Key Set	$P(1,1,1,0), P(1,1,1,1), P(1,1,1,2)$	$P(1,1,2,0), P(1,1,2,1), P(1,1,2,2)$	$P(1,2,1,0), P(1,2,1,1), P(1,2,1,2)$	$P(1,2,2,0), P(1,2,2,1), P(1,2,2,2)$
	Node's private exponent	$S(1,1,1,0)$	$S(1,1,2,0)$	$S(1,2,1,0)$	$S(1,2,2,0)$
	Subgroup Key	$K(1,1,1)$	$K(1,1,2)$	$K(1,2,1)$	$K(1,2,2)$

Table 2. 1 Partial key set, private exponent of the PMand subgroup key of each SGC and GC

Level	Keys	$M(1,1,1,1)$	$M(1,1,1,2)$	$M(1,1,2,1)$	$M(1,1,2,2)$
1	Partial Key (unsaved)	$P(1,1)$	$P(1,1)$	$P(1,0), P(1,1), P(1,2)$	$P(1,0), P(1,1), P(1,2)$
	Node's private exponent	$S(1,1)-K(1,1)$	$S(1,1)-K(1,1)$	$S(1,1)-K(1,1)$	$S(1,1)-K(1,1)$
	Subgroup Key	$K(1)$	$K(1)$	$K(1)$	$K(1)$

2	Partial Key (unsaved)	$P(1,1,1)$	$P(1,1,1)$	$P(1,1,0), P(1,1,1), P(1,1,2)$	$P(1,1,0), P(1,1,1), P(1,1,2)$
	Node's private exponent	$S(1,1,1)=K(1,1,1)$	$S(1,1,1)=K(1,1,1)$	$S(1,1,2)=K(1,1,2)$	$S(1,1,2)=K(1,1,2)$
	Subgroup Key	$K(1,1)$	$K(1,1)$	$K(1,1)$	$K(1,1)$
3	Partial Key (unsaved)	$P(1,1,1,1)$	$P(1,1,1,2)$	$P(1,1,2,1)$	$P(1,1,2,2)$
	Node's private exponent	$S(1,1,1,1)$	$S(1,1,1,2)$	$S(1,1,2,1)$	$S(1,1,2,2)$
	Subgroup Key	$K(1,1,1)$	$K(1,1,1)$	$K(1,1,2)$	$K(1,1,2)$
Level	Keys	$M(1,2,1,1)$	$M(1,2,1,2)$	$M(1,2,2,1)$	$M(1,2,2,2)$
1	Partial Key (unsaved)	$P(1,0), P(1,1), P(1,2)$	$P(1,0), P(1,1), P(1,2)$	$P(1,0), P(1,1), P(1,2)$	$P(1,0), P(1,1), P(1,2)$
	Node's private exponent	$S(1,2)=K(1,2)$	$S(1,2)=K(1,2)$	$S(1,2)=K(1,2)$	$S(1,2)=K(1,2)$
	Subgroup Key	$K(1)$	$K(1)$	$K(1)$	$K(1)$
2	Partial Key (unsaved)	$P(1,2,0), P(1,2,1),$ $P(1,2,2)$	$P(1,2,0), P(1,2,1), P(1,2,2)$	$P(1,2,0), P(1,2,1), P(1,2,2)$	$P(1,2,0), P(1,2,1), P(1,2,2)$
	Node's private exponent	$S(1,2,1)=K(1,2,1)$	$S(1,2,1)=K(1,2,1)$	$S(1,2,2)=K(1,2,2)$	$S(1,2,2)=K(1,2,2)$
	Subgroup Key	$K(1,2)$	$K(1,2)$	$K(1,2)$	$K(1,2)$
3	Partial Key (unsaved)	$P(1,2,1,1)$	$P(1,2,1,2)$	$P(1,2,2,1)$	$P(1,2,2,2)$
	Node's private exponent	$S(1,2,1,1)$	$S(1,2,1,2)$	$S(1,2,2,1)$	$S(1,2,2,2)$
	Subgroup Key	$K(1,2,1)$	$K(1,2,1)$	$K(1,2,2)$	$K(1,2,2)$

Table 2. 2 Member's keys (After subgroup key calculation, partial keys are removed)

2.2.2 Member Join

When a member $M(G_1, G_2, \dots, G_i, n+1)$ wants to join subgroup $SG(G_1, G_2, \dots, G_i)$, it issues a join request to $SGC(G_1, G_2, \dots, G_i)$. This, in turn, triggers the member join process as described below.

1) The initial step of the process is similar to Static CLIQUES, where the new member, existing members, and SGC of subgroup $SG(G_1, G_2, \dots, G_i)$ compute new subgroup key $K_{new}(G_1, G_2, \dots, G_i)$. The new key also becomes the new private exponent

$S_{new}(G_1, G_2, \dots, G_i)$ of $PM(G_1, G_2, \dots, G_i)$. Note

that $K_{new}(G_1, G_2, \dots, G_i) = (K_{old})^{R_i S(G_1, G_2, \dots, G_i, n+1)}$, here, R_i is a random number generated by SGC when the new member requests join-group. If not adding a random number, the partial key of the new member $M(G_1, G_2, \dots, G_i, n+1)$ will be $g^{\prod_{(S(G_1, G_2, \dots, G_i, p))|p \in [0, n]}}$, which is the old subgroup key of $SG(G_1, G_2, \dots, G_i)$. Since the new partial key set will be multicasted to the whole subgroup later, an intruder can easily get it and compute the old group key to decrypt the previous information he cached. Hence, SGC must add a random number to all of the partial keys, for each upper level subgroup.

2) After receiving the updated partial keys of this subgroup from the new member, SGC of subgroup $SG(G_1, G_2, \dots, G_i)$ needs to update the partial key sets of all upper level subgroups. Let $PM(G_1, G_2, \dots, G_i)$ be the j^{th} PM in the $(i-1)^{th}$ level subgroup, it can be denoted as $PM(G_1, G_2, \dots, G_{i-1}, j)$ since from this point $G_i = j$. The old partial key set of $SG(G_1, G_2, \dots, G_{i-1})$ is denoted as $P_{old}(G_1, G_2, \dots, G_{i-1}, p)$, for $p \in [0, n]$. $SGC(G_1, G_2, \dots, G_{i-1}, j)$ then generates new partial key set of $SG(G_1, G_2, \dots, G_{i-1}, j)$:

$$\begin{aligned}
 P_{new}(G_1, G_2, \dots, G_{i-1}, j) &= K_{old}(G_1, G_2, \dots, G_{i-1}, j)^R \\
 &= g^{R \prod_{(S_{old}(G_1, G_2, \dots, G_{i-1}, p))|p \in [0, n]}} \\
 &= P_{old}(G_1, G_2, \dots, G_{i-1}, j)^{RS_{old}(G_1, G_2, \dots, G_{i-1}, j)} \\
 P_{new}(G_1, G_2, \dots, G_{i-1}, l) &= g^{RS_{new}(G_1, G_2, \dots, G_{i-1}, j) \prod_{(S_{old}(G_1, G_2, \dots, G_{i-1}, p))|p \in [0, n] \wedge p \neq l}} \\
 &= P_{old}(G_1, G_2, \dots, G_{i-1}, l)^{RS_{new}(G_1, G_2, \dots, G_{i-1}, j)}, \text{ for } l \in [0, n] \wedge l \neq j
 \end{aligned}$$

Eventually,

$$\begin{aligned}
 P_{new}(G_1, j) &= g^{R \prod_{(S_{old}(G_1, p))|p \in [0, n]}} \\
 &= P_{old}(G_1, j)^{RS_{old}(G_1, j)}
 \end{aligned}$$

$$= K_{old}(G_1, j)^R,$$

and,

$$P_{new}(G_1, l) = g^{RS_{new}(G_1, j)} \prod_{(S_{old}(G_1, p) | p \in [0, n] \wedge p \neq l)} \\ = P_{old}(G_1, l)^{RS_{new}(G_1, j)}, \text{ for } l \in [0, n] \wedge l \neq j$$

SGC then multicasts the updated partial key sets to the whole multicast group.

3) Once the group members receive the partial key sets multicasted by the SGC of the subgroup containing new member, they identify the lowest level of subgroup to which both they and the new member belong. Then, they compute the new subgroup keys of all the subgroups higher than this level, iteratively. For example, in Figure 2.6, $M(1,1,1,3)$ joins $SG(1,1,1)$, which means there are membership changes happened in $SG(1,1,1)$, $SG(1,1)$ and $SG(1)$, because the new member will belong to these subgroups. Hence $K(1,1,1)$, $K(1,1)$ and $K(1)$ will be updated, as well as the partial key set of $SG(1,1,1)$, $SG(1,1)$ and $SG(1)$.

Figure 2. 6 An example of member join

Let's examine which subgroup keys need to be updated by members:

- $M(1,1,1,1)$, $M(1,1,1,2)$ and $M(1,1,1,3)$ are in $SG(1,1,1)$, which means the lowest level of parent subgroup shared by them and new member $M(1,1,1,3)$ is $SG(1,1,1)$. So, $M(1,1,1,1)$, $M(1,1,1,2)$ and $M(1,1,1,3)$ need to update $K(1,1,1)$, $K(1,1)$ and $K(1)$. See Figure 2.7.

Figure 2. 7 Members in $SG(1,1,1)$ need to update $K(1,1,1)$, $K(1,1)$ and $K(1)$

- $M(1,1,2,1)$ and $M(1,1,2,2)$ belong to $SG(1,1,2)$, $SG(1,1)$ and $SG(1)$, the lowest level of parent subgroup shared by them and $M(1,1,1,3)$ is $SG(1,1)$, so $M(1,1,2,1)$ and $M(1,1,2,2)$ need to update $K(1,1)$ and $K(1)$. See Figure 2.8.

Figure 2. 8 Member in $SG(1,1,2)$ need to update $K(1,1)$ and $K(1)$

- $M(1,2,1,1)$ and $M(1,2,1,2)$ belong to $SG(1,2,1)$, $SG(1,2)$ and $SG(1)$, the lowest level of parent subgroup shared by them and $M(1,1,1,3)$ is $SG(1)$, so $M(1,2,1,1)$ and $M(1,2,1,2)$ only need to update $K(1)$. Similarly, $M(1,2,2,1)$ and $M(1,2,2,2)$ belong to $SG(1,2,2)$, $SG(1,2)$ and $SG(1)$, the lowest level of parent subgroup shared by them and $M(1,1,1,3)$ is $SG(1)$, so $M(1,2,2,1)$ and $M(1,2,2,2)$ only need to update $K(1)$. See Figure 2.9.

Figure 2. 9 Members in $SG(1,2,1)$ and $SG(1,2,2)$ need to update $K(1)$

Now the members know which subgroup keys they need to update. They retrieve the new partial key of their represented PM of the lowest level that needs to be updated, calculate the new subgroup key of the subgroup on this level. For example, $M(G_1, G_2, \dots, G_q)$ needs to upgrade subgroup keys of subgroup $SG(G_1, G_2, \dots, G_i)$ and upper subgroups. Its pseudo-member $PM(G_1, G_2, \dots, G_i, j)$ is the j^{th} PM in $SG(G_1, G_2, \dots, G_i)$. Then $M(G_1, G_2, \dots, G_q)$ retrieves the partial key of $PM(G_1, G_2, \dots, G_i, j)$, which is denoted as $P_{new}(G_1, G_2, \dots, G_i, j)$. Because there is no membership change in the subgroup that $PM(G_1, G_2, \dots, G_i, j)$ represents, the private secret of $PM(G_1, G_2, \dots, G_i, j)$ hasn't been changed. So, $M(G_1, G_2, \dots, G_q)$ calculates the new subgroup key of $SG(G_1, G_2, \dots, G_i)$ as:

$$K_{new}(G_1, G_2, \dots, G_i) = P_{new}(G_1, G_2, \dots, G_i, j)^{S_{old}(G_1, G_2, \dots, G_i, j)}$$

and, of course,

$$S_{new}(G_1, G_2, \dots, G_i) = K_{new}(G_1, G_2, \dots, G_i)$$

For the $(i-1)^{th}$ level subgroup, $SG(G_1, G_2, \dots, G_{i-1})$

$$K_{new}(G_1, G_2, \dots, G_{i-1}) = P_{new}(G_1, G_2, \dots, G_i)^{S_{new}(G_1, G_2, \dots, G_i)}$$

Eventually, the new group key can be computed

$$K_{new} = K_{new}(G_1) = P_{new}(G_1, G_2)^{s_{new}(G_1, G_2)}$$

4) The other SGCs compute K_{new} the same way as members do. In addition, SGCs copy the whole new partial key sets into their databases.

If there are multiple members join one subgroup at the same time, the mass join process of Static CLIQUES occurs in the immediate subgroup to compute the new partial key set of this subgroup. After that, the process is the same as from step (2).

2.2.3 Member Leave

When a member $M(G_1, G_2, \dots, G_i, j)$ wants to leave group, it sends the request to its SGC in $SG(G_1, G_2, \dots, G_i)$. This SGC initiates a member leave process in $SG(G_1, G_2, \dots, G_i)$, which is the same as the member leave process of the Static CLIQUES. The remaining members and SGC compute new subgroup key, which is also the new private exponent of $SG(G_1, G_2, \dots, G_i)$. With the new private exponent, the key-refresh process will be initiated as described in steps (2) to (4) in section 2.2.

Eventually, a new group key will be generated.

2.2.4 Key Refresh

When the current subgroup key's lifetime in $SG(G_1, G_2, \dots, G_i)$ is aged out, the SGC initializes the key refresh process in its subgroup and recomputes the partial key set, like key refreshment in Static CLIQUES. The remaining members and SGC compute new subgroup key, which is also the new private exponent of $SG(G_1, G_2, \dots, G_i)$. With the new private exponent, the key-refresh process will be initiated as described in steps (2) to (4) in section 2.2.

Eventually, a new group key will be generated.

Chapter 3: Efficient Key Agreement Protocol

3.1 Optimal Key Tree Structure

Theoretically, the M-CLIQUE algorithm itself does not restrict the number of members in a subgroup. The number of members in a subgroup could be arbitrary. Also, members can be under GC or any SGC, which means the children of a SGC could be members and SGCs. These are due to the flexible and expandable nature of the algorithm. However, we may still want to ask questions: Is there an optimal structure of M-CLIQUE tree? Will different types of multicast applications have their own optimal structure of tree? What is the optimal height of the tree? If those questions get answered, a more efficient multicast key agreement protocol based on the M-CLIQUE algorithm can be proposed.

Before addressing above questions, let's review the major goals of an efficient key agreement algorithm:

- 1) Small size of re-key messages (usually it is proportional to the number of keys that need to be updated when re-keying);
- 2) Less processing overhead while re-keying;
- 3) Low key-storage cost.

If a tree structure can achieve the above goals, then it is optimal. The first assumption of the optimal tree is: the optimal tree is a balanced tree. A balanced tree is such a tree that each parent has the same number of children and members are under the lowest level of SGCs. Before starting to prove this assumption, let's take a look into the number of partial keys to be updated during a membership change. In M-CLIQUE, the number of partial keys that are needed to be updated and the number of re-key calculations are different depending on the role of the node. Let define an immediate SGC as the SGC of subgroup within which a member-join or member-leave event occurs.

During the member-join process,

- The new member will update the immediate subgroup's partial key set and calculate all of the upper level subgroup keys;
- The immediate SGC will update all of the upper levels subgroups' partial key sets and send them to the whole group;
- Other SGCs and GC will save the corresponding level subgroups' partial key sets delivered by the immediate SGC, in the mean time, calculate all of the upper level subgroup keys;
- The existing members will calculate all of the upper level subgroup keys only.

During a member-leave process,

- The immediate SGC will update the immediate subgroup's partial key set and all of the upper levels subgroups' partial key set and send them to the whole group;
- Other SGCs and GC will save the corresponding level subgroups' partial key sets delivered by the immediate SGC, in the mean time, calculate all of the upper level subgroup keys;
- The remaining members will calculate all of the upper level subgroup keys only.

From the above analyses, we observe that when a membership change happens, GC, other SGCs and other members will have the same number of calculation; no matter it is a member-join or a member-leave. And the number of calculation depends on the number of levels, which is related to the height of the tree. The difference between a member-join and a member-leave is, in a member-join, the new member updates the partial keys of the immediate subgroup; when a member leaves, the immediate SGC updates the partial keys of immediate subgroup. The number of these partial keys depends on the number of children of the immediate SGC. However, the number of partial keys needed to be updated during member-join equals to the old number of partial keys, and the number of partial keys needed to be updated during member-leave equals to the old number of partial keys minus one. The difference is one partial key. Since this slight difference would not affect the result of deduction, we will only examine the

member-leave case in the following proof. For the member-join process, there would be the similar result.

Considering the circumstance of a member leaves a balanced tree with N members. After this member leaves, the total number of the tree becomes $(N-1)$. Since the number of partial keys depends on the number of children of immediate SGC, while the number of re-key calculation of other SGC and members depends on the height of the tree, we can assume the height of tree is fixed and examine the partial keys that need to be updated by the immediate SGC. Firstly, to simplify the analysis, let look at a height of 2 tree, which means there are 3 levels: GC-SGC(or member)-member. The total number of the members is N . When the tree is balanced, there are $N^{\frac{1}{2}}$ subgroups, and there are $N^{\frac{1}{2}}$ members in each subgroup. The number of partial keys that each SGC holds equals to $(N^{\frac{1}{2}} + 1) \times 2$. So, the total number of partial keys that the immediate SGC needs to update equals to $(N^{\frac{1}{2}} + 1) + N^{\frac{1}{2}} = (2N^{\frac{1}{2}} + 1)$. The first portion of the equation, $(N^{\frac{1}{2}} + 1)$, accounts for the number of children of GC. The second portion of the equation, $N^{\frac{1}{2}}$, accounts for the updated (after one member leave) number of children of the immediate SGC. Let's examine what the values of the number of partial keys will be, if the tree is unbalanced.

1) If the number of SGCs remains $N^{\frac{1}{2}}$, then in some subgroup there would be less than $N^{\frac{1}{2}}$ members while in other some subgroup there would be more than $N^{\frac{1}{2}}$ members, otherwise the tree will be balanced. So in the subgroup which has the maximum number of members, e.g. $(N^{\frac{1}{2}} + m)$ members where $m \geq 1$, its SGC holds $((N^{\frac{1}{2}} + 1) + N^{\frac{1}{2}} + m + 1)$ partial keys, which is greater than $(N^{\frac{1}{2}} + 1) \times 2$, the balanced tree's number. When a member leaves this subgroup, the number of partial keys that SGC needs to update equals to $((N^{\frac{1}{2}} + 1) + N^{\frac{1}{2}} + m)$, which is $(2N^{\frac{1}{2}} + m + 1)$, greater than

$(2N^{\frac{1}{2}} + 1)$. Figure 3.1 gives an example of 16 members.

Figure 3.1 An example of member distribution to $N^{\frac{1}{2}}$ SGCs

2) If the number of SGCs is less than $N^{\frac{1}{2}}$, to make it simpler, say $(N^{\frac{1}{2}} - 1)$ subgroups are under GC and each of the subgroup holds $N^{\frac{1}{2}}$ members. Then, there are $N^{\frac{1}{2}}$ members need to be allocated to a subgroup. Be noted that GC is also serving a subgroup. $N^{\frac{1}{2}}$ members will be distributed into $N^{\frac{1}{2}}$ subgroups $((N^{\frac{1}{2}} - 1)$ SGC and 1 GC). Hence, at least one subgroup will have $(N^{\frac{1}{2}} + 1)$ or more members. So in the subgroup which has the maximum number of members, e.g. $(N^{\frac{1}{2}} + m)$ members where $m \geq 2$, its SGC holds $(N^{\frac{1}{2}} + (N^{\frac{1}{2}} + m + 1))$ partial keys, which is greater than $(N^{\frac{1}{2}} + 1) \times 2$, the balanced tree's number. When a member leaves this subgroup, the number of partial keys that SGC needs to update equals to $(N^{\frac{1}{2}} + (N^{\frac{1}{2}} + m - 1))$, which is $(2N^{\frac{1}{2}} + m)$, greater than $(2N^{\frac{1}{2}} + 1)$. Figure 3.2 gives an example of 16 members.

Figure3. 2 An example of member distribution to less than $N^{\frac{1}{2}}$ SGCs

3) If the number of SGCs is more than $N^{\frac{1}{2}}$, for example, $(N^{\frac{1}{2}} + 1)$ subgroups are under

GC, then N members need to be allocated to $(N^{\frac{1}{2}} + 1)$ subgroups, hence, at least one subgroup will have more than $\frac{N}{N^{\frac{1}{2}} + 1}$ members. When a member leaves that subgroup,

the number of partial keys that immediate SGC needs to update equals to $(N^{\frac{1}{2}} + 2 + \frac{N}{N^{\frac{1}{2}} + 1})$. It is equivalent to:

$$\frac{N + 3N^{\frac{1}{2}} + 2 + N}{N^{\frac{1}{2}} + 1} = \frac{2N + 2N^{\frac{1}{2}} + N^{\frac{1}{2}} + 2}{N^{\frac{1}{2}} + 1} = 2N^{\frac{1}{2}} + \frac{N^{\frac{1}{2}} + 2}{N^{\frac{1}{2}} + 1} > 2N^{\frac{1}{2}} + 1$$

If the height is larger than 2, we can still get the same result because the tree can be seen as a collection of several height of 2 sub-trees.

From the above observations, we know that a balanced tree is better than unbalanced tree, in terms of the number of partial keys stored in SGC and the number of partial keys that needs to update when there is a membership change.

Given the fact that a balanced tree is the optimal tree, there comes the second question: what is the optimal number of members in each subgroup? In other words, what is the optimal height of this balanced tree, given that the total number of members is fixed?

Let the maximum number of members in each subgroup (also called “order” of the balanced tree) be denoted as m , the height of the balanced tree be denoted as h , the total number of the group be denoted as N . Because it is a balanced tree, given the fixed total N members in the group, by calculation, $h = \log_m N$. Also, we know that a balanced tree would be at least a binary tree, which means a non-leaf node will have at least two children. So we can get the following equations:

The number of partial keys stored in SGC equals to:

$$(m+1) \times h = (m+1) \log_m N, \text{ where } 2 \leq m \leq N \quad \text{---(1)}$$

In (1), $(m+1)$ accounts for the number of partial keys of every level of subgroup.

There are h levels.

The number of partial keys that SGC needs to update when a member leaves the balanced tree equals to:

$$\begin{aligned} m + (m+1) \times (h-1) &= (m+1) \times h - 1 \\ &= (m+1) \log_m N - 1, \text{ where } 2 \leq m \leq N \quad \text{---(2)} \end{aligned}$$

In (2), m accounts for the number of new partial keys of immediate SGC, while $(m+1) \times (h-1)$, accounts for $(m+1)$ partial keys in each upper level subgroup, and there are $(h-1)$ upper level subgroups.

The number of subgroup keys that other members and SGCs should recalculate is no larger than $h = \log_m N$ ---(3).

The above question then can be described as the following problem: what is m 's value when function (1), (2) and (3) are minimized.

As for the number of subgroup keys that other members and SGCs should recalculate, apparently function (3) is smaller when m is larger.

As for the number of partial keys stored in SGC and the number of partial keys that SGC needs to update, which are function (1) and (2)'s values, see the following solution:

Because $2 \leq m \leq N$,

$$\therefore \log_m N \geq 1$$

$$\therefore (m+1) \log_m N \geq 2$$

$\therefore$ Functions (1) and (2) will have minimum value when $m=2$.

Hence, when the tree is a binary balanced tree ($m=2$), the number of partial keys stored in SGC and the number of partial keys that SGC needs to update when re-keying are minimum. However, when the tree is a binary balance tree, the result from step 1) tells us that the number of subgroup keys that other members and SGCs should

recalculate has the maximum value. Also, we need more SGCs if m is smaller. Consequently, we cannot find a value of m to make functions (1), (2) and (3) reach the minimum value at the same time. But we need to figure out a balance point in reality.

Looking back to the reasons why we introduce GC and SGCs into M-CLIQUEs on the first place, we find that the main tasks of SGC are to reduce the re-keying calculation of the members and the keys stored in the member. It is because SGC is a more powerful computer. It has memory so it can host more partial keys. It has powerful CPU and large memory so it can calculate new partial keys quickly. The members, on the other hand, sometimes are less powerful devices, e.g. a Digital Cable TV box. Based on the above discussion, we should choose a larger value of m .

3.2 M-CLIQUEs Key Agreement Protocol Based on Balanced Tree

3.2.1 Assumptions

From the analysis in the pervious section, we know that a balanced tree structure is efficient in terms of re-key messages and processing. In some multicast applications, such as Pay-per-View TV and stock price broadcasting (e.g. Reuters Stock), the total number of subscribers is known or predictable. In these cases, it is possible to construct a balanced tree because the total number of members can be approximated. By choosing the maximum number of members in each subgroup, which is called the order of the tree, the height of the entire tree could be easily calculated as $h = \log_m N$. Consequently, the number of SGCs could be determined. The optimal value of m has to be determined by considering the cost and the power of SGCs.

In this thesis, an efficient key agreement protocol of M-CLIQUEs, which is designed for the multicast applications of predictable or known total subscribers (within a certain period of time), is studied. We use the following assumptions:

- A M-CLIQUEs tree has a fixed maximum total number of members, N . The

tree will eventually become a balanced tree with a fixed order m and fixed height h .

- GC is the root of the M-CLIQUE tree. GC's children can only be SGCs. In practice, some SGCs can be residing in one computer. Each SGC runs an instance in the co-located computer.
- Members are the leaves of the lowest level of SGC's.
- Whenever a new member joins or an existing member leaves the group, authentication between the changing member and GC/SGC is required before re-keying process, as the DH algorithm is vulnerable to the man-in-the-middle attack. However, the thesis doesn't address multicast authentication. The proposed protocol assumes that a two-party authentication method is used in all initiation and re-keying process. Also, a method of multicast authentication could be integrated into the protocol in the future.

3.2.2 Concurrency Control and Interval-Based Re-keying

3.2.2.1 Concurrent Membership Change Control

In practice, there would be the situation that multiple members join and leave simultaneously or within a very short period of time. Since any SGC could do the re-keying, which means any SGC could multicast the updated partial key sets to the whole group when there is membership change within its subgroup, re-keying conflict will happen. For example, a member joins subgroup SG(1,1,1) and the other member joins SG(1,1,2) at the same time. SGC(1,1,1) and SGC(1,1,2) finish the calculation of new partial key sets at the same time as well, so they multicast their own updated partial key sets to the rest of the group. However, neither of two updated partial key sets contains both new members' contribution, which means neither of the partial key sets is completed. Consequently, the encryption key could not be synchronized correctly across the group. A central control mechanism should be developed to deal with concurrent membership change. GC is a good central point of control that oversees the status of the

whole group. It handles the member join and leave requests and coordinates the join/leave process among the SGCs.

In terms of the re-keying sequences of concurrent member join/leave requests, per-member re-keying is inefficient. Figure 3.3 illustrates four members joining a tree with order 3 and height 3. The group key is renewed 4 times respectively. Let T be the time of updating one partial key. The total time of updating partial keys equals to $4 \times (3 \times T + 3 \times T + 3 \times T + 3 \times T) = 48T$.

Figure 3. 3 Re-keying per member

Figure3. 4 Re-keying per Subgroup

Let's consider a different re-keying process of the same joining scenario as shown

in the above diagram. Instead of renewing the key one after another member join, SG(1,1,3), SG(1,3,1), SG(1,3,2) and SG(1,3,3) renew their own subgroups' partial keys simultaneously, shown as ① in Figure 3.4; then SG(1,1) and SG(1,3) renew their own subgroups' partial keys because PM(1,1,3) in SG(1,1), PM(1,3,1), PM(1,3,2) and PM(1,3,3) in SG(1,3) refreshed their private exponents, shown as ② , ③, and ④ in Figure 3.4; at last, SG(1) renew it's partial keys because PM(1,1) and PM(1,3) refreshed their private exponents, shown as step ⑤ and ⑥ in Figure 3.4. This way, the total time of updating partial keys equals to $[3xT + (3xT + 3xT + 3xT) + (3xT + 3xT)] = 18T$. It is less than per-member re-keying. Hence, per-subgroup re-keying is more efficient.

In addition to per-subgroup re-keying, within a subgroup where multiple PMs' private exponents have been updated, the SGCs which representing those PMs will renew partial keys in turn. For example, take a look at SG(1,3) in Figure 3.4. PM(1,3,1), PM(1,3,2) and PM(1,3,3)'s private exponents have been updated. Similar to Mass Join process of Static CLIQUUS, SGC(1,3,1), SGC(1,3,2) and SGC(1,3,3) update the partial keys of SG(1,3) in turn.

3.2.2.2 Interval-Based Re-keying

Usually, most of member join/leave events do not occur exactly at the same time. They occur within a shot period. So, in order to achieve better efficiency, Interval-based re-keying is introduced [24]. Which is, when GC receive a member join/leave request, it will not start the process for this single member immediately. GC will start a timer and wait for more members' join/leave requests. When the timer counts to a re-key interval, GC will start per-subgroup re-keying process for all the requesting members within this period of time. Any more requests will be queued. Until the current interval's re-key is finished GC will look at the queue and start the next re-key interval. If there is only one member join/leave request during the re-key interval, GC will start single member join/leave process.

The tradeoff of interval-based re-keying is, it will weaken the forward secrecy as

a result of delaying the member's leaving process. Hence, the re-keying interval should be defined as small as possible.

The main functions of GC are as follows:

- Maintains a membership database. It knows the current view of members' distribution.
- Accepts members join and leave requests, authenticates members, and redirects members to the corresponding SGC. When a member requires leaving the group, GC notifies SGC of this member leaving subgroup, and the SGC continues the re-keying of member leave. When a member requires joining the group, GC searches its database looking for a vacant leaf position of the tree by searching from left to right of the lowest level subgroup, and refers the new member to join the leftmost available SGC. This arrangement is to make members to line up from left to right. It could make tree subtraction easier when downgrading the height of the tree is necessary.
- Queues member join/leave requests and performs interval-based re-keying control.
- Manages key version to enhance the reliability of key multicast.

3.2.3 Pseudo-codes of Protocol

3.2.3.1 GC Initialization

3.2.3.1.1 Initialize the partial key set of the group, which is $SG(I)$

Since GC is the first member, $M(I,0)$, in group $SG(I)$, when it is getting started it needs to conduct the first partial key set of $SG(I)$ which contains only one partial key: $P(I,0) = g$. Also, it generates its own private secret exponent $S(I,0)$. And, it will calculate the group key using its own private exponent and public key: $DH(S(I,0); P(I,0))$.

3.2.3.1.2 Start serverices

- Start TCP service, waiting for SGC and member registration or resignation
- Start UDP service, listening to the key-set-renewal messages

- Start UDP service, ready for multicasting encrypted data. This is an optional service, which is for testing purpose

3.2.3.2 SGC Join Group

3.2.3.2.1 New SGC Registration

- New SGC sends join group request to GC or upper level SGC
- When GC or upper level SGC receives the join request from the new SGC, it will do the following:
 - Generate a NodeID for the new SGC
 - Updates partial key sets and subgroup key by adding a random exponent to every partial key and subgroup key
 - Send the join-accept message including the NodeID and updated partial key sets to the new SGC
- When new SGC receive join-accept message, it
 - Initialize the partial key set for the subgroup it controls, which contains only one partial key: SGC's own partial key = g . As the first member in this subgroup, it generates its own private secret exponent. And, calculate the subgroup key of its own subgroup using its own private exponent and public key: $DH(\text{private exponent of SGC}; \text{partial key of SGC})$
 - Update all of the upper level subgroup keys and partial key sets. Do the following calculations:
 - 1) Assign the current subgroup key as the private exponent of the PM which the current subgroup represents in the upper level subgroup.
 - 2) Go to the upper level.
 - 3) Retrieve this PM's partial key, replace it with the old updated subgroup key
 - 4) For all partial keys other than this PM's partial key, do $DH(\text{private exponent of PM}, \text{old partial key of other member})$

- 5) Repeat 1-4 until finishing recalculating the partial key set of the first level subgroup, $SG(1)$
 - Remove the subgroup keys from the new key set, only include the new partial keys into the key-renew message, multicast to the whole group, which then initiate the key renew process for each element of the whole multicast group
 - Send join-accept ACK back to the GC or upper level SGC it registered to
- New SGC starts services:
 - Start TCP service, waiting for member or SGC registration or resignation
 - Start UDP service, listening to the key-set-renewal messages

3.2.3.2.2 GC and other SGC renew key sets

When the GC or other SGC receive a key-renew message from the new SGC, it will do the following:

- 1) Save the partial key sets
- 2) Identify the lowest level subgroup that both the new SGC and itself belong to by comparing the sender's NodeID and its own NodeID. We call this subgroup as the immediate subgroup. Let current subgroup be the immediate subgroup
- 3) Identify the PM's ID it represents within the current subgroup
- 4) Retrieve the partial key of the PM, calculate the current subgroup key as $DH(\text{private exponent of PM, old partial key of PM})$
- 5) Assign the current subgroup key as the private exponent of the PM in the upper level subgroup.
- 6) Go to the upper level.
- 7) Repeat 3-6 until finishing recalculating the key of the first level subgroup, $SG(1)$.

This is the key of the whole group

3.2.3.2.3 Existing member renew subgroup keys

When an existing member receive a key-renew message, it will do the following:

- 1) Identify the lowest level subgroup that both the new SGC and itself belong to by comparing the sender's NodeID and it's own NodeID. We call this subgroup as the immediate subgroup. Let current subgroup be the immediate subgroup
- 2) Identify the PM's ID it represents within the current subgroup
- 3) Retrieve the partial key of the PM, calculate the current subgroup key as $DH(\text{private exponent of PM}, \text{old partial key of PM})$
- 4) Assign the current subgroup key as the private exponent of the PM in the upper level subgroup.
- 5) Go to the upper level.
- 6) Repeat 3-6 until finishing recalculating the key of the first level subgroup, SG(1).

This is the key of the whole group

3.2.3.3 Single Member Join

Figure 3.5 shows an example of single member join.

Figure 3. 5 An example of Single Member Join

3.2.3.3.1 New Member Registration

- New member sends join group request to GC
- When GC receives the join request from the new member, it will start re-keying timer. If within the re-keying interval there is no other member join/leave request received, GC will start single member join process and do the following:
 - Assign a subgroup to the new member
 - Generate a NodeID for the new member
 - Send join-redirect message that includes the NodeID of new member to the new member; redirect it to connect the SGC of the subgroup it is joining. Within join-redirect message there is a credential to identify the new member. The method of generating the credential is, GC generates a nouns and calculate the hash value of the nouns, the new members' IP and NodeID using

either MD5 [22] or SHA [23].

- When the new member receives the join-redirect message from GC, it will connect the designated SGC with a join-request message which containing the credential from GC. The SGC will do the following:
 - Extract the credential from member's join-request message
 - Connect GC requesting this member's credential via a secured unicast channel. GC will also send the new key version to the SGC
 - Compare member's credential against the credential from GC. If they are different, send join-reject message to the new member. If they match, start preparing for the updated partial keys
 - Update partial key set and key of the subgroup it controls by adding a random exponent to every partial key and subgroup key
 - Send join-accept message including the updated partial key set to the new member
- When new member receive the join-accept message, it
 - Generates it's own private secret exponent.
 - For every partial key of its immediate subgroup other than its own partial key, do $DH(\text{private exponent of new member, old partial key of other member})$
 - Assign the old updated subgroup key as the new member's partial key , append it to the end of the partial key set
 - Send join-accept ACK that includes the new partial key set back to the SGC.
- New member starts services:
 - Start UDP service, listening to the key-set-renewal message
 - Start UDP service, listening to the encrypted multicast data
- When the SGC receives join-accept ACK, it will do the following:
 - Save the new partial key set of it's subgroup
 - Update all of the upper level subgroup keys and partial key sets. Do the

following calculations:

- 1) Assign the current subgroup key as the private exponent of the PM which the current subgroup represents in the upper level subgroup.
 - 2) Go to the upper level.
 - 3) Retrieve this PM's partial key, replace it with the old updated subgroup key
 - 4) For all partial keys other than this PM's partial key, do $DH(\text{private exponent of PM, old partial key of other member})$
 - 5) Repeat 1-4 until finishing recalculating the partial key set of the first level subgroup, $SG(1)$
- Remove the subgroup keys from the new key set, only include the new partial keys into the key-renew message, multicast to the whole group, which then initiate the key renew process for each element of the whole multicast group

3.2.3.3.2 GC and other SGC renew key sets

GC and other SGCs' key-renew process when a new member joins group is the same as they do in section 3.2.3.2.2.

3.2.3.3.3 Member renew subgroup keys

The old members' as well as the new member's key-renew process when a new member joins group is the same as they do in section 3.2.3.2.3.

3.2.3.4 Single Member Leave

Figure 3.6 shows an example of single member leave.

Figure 3. 6 An example of single member leave

3.2.3.4.1 Leaving Member Resignation

- The member which is leaving sends member-leave request to GC
- GC removes the member information from database and send leave-accept request back to the leaving member
- Leaving member sends leave-accept ACK to GC and closes TCP and UDP services
- GC sends the member-leave message to the SGC which controls the leaving member's subgroup and new key version
- The SGC will do the following:
 - Remove the partial key of this member from the partial key of the subgroup it

controls.

- Update partial key sets and keys by adding a random exponent to every partial key and subgroup key, and initiates the key refresh processing illustrated bellow
- Update all of the upper level subgroup keys and partial key sets. Do the following calculations:
 - 1) Assign the key of current subgroup as the private exponent of the PM which the current subgroup represents in the upper level subgroup.
 - 2) Go to the upper level.
 - 3) Retrieve this PM's partial key, replace it with the old updated subgroup key
 - 4) For every partial keys other than this PM's partial key, do $DH(\text{private exponent of PM, old partial key of other member})$
 - 5) Repeat 1-4 until finishing recalculating the partial key set of the first level subgroup, SG(1)
- Remove the subgroup keys from the new key set, only multicast out the new partial keys to the whole group, which then initiate the key renew process for each element of the whole multicast group

3.2.3.4.2 GC and other SGC renew key sets

GC and other SGC's key-renew process when a member leaves group is the same as they do in section 3.2.3.2.2.

3.2.3.4.3 Remaining member renew subgroup keys

The remaining members key-renew process when a member leaves group is the same as they do in section 3.2.3.2.3.

3.2.3.5 Multiple Member Join/Leave - Per-subgroup re-keying

Figure 3.7 shows an example of multiple members join and leave.

Figure 3. 7 An example of multiple members join and leave

3.2.3.5.1 GC starts per-subgroup re-keying process when re-key interval expires

- Allocate the new joining members from the left to the right into the tree, filling the vacancies of the leaving member first
- Send join/leave-accept message to the joining/leaving members
- Send member-leave message to the SGCs of the subgroup containing leaving members

3.2.3.5.2 SGCs that have membership change behavior

- After confirm the new members identities and member leaving, they update their subgroups' partial keys respectively:
 - The SGC have members leaving removes the leaving members' partial keys and update the remaining partial keys of it's subgroup

- The SGC have members joining send the partial keys to the new members one after another for adding their contributions to the partial keys of the subgroup
- The SGC have both members joining and leaving removes the leaving members' partial keys first, and then send the partial keys to the new members one after another for adding their contributions to the partial keys of the subgroup.
- Since these subgroup keys have been updated, which means the private exponent of the PM that represents those subgroups have been changed. Hence the partial keys of upper level of subgroup should be updated as well. The SGCs that belong to the same subgroup will update the partial keys in term. Consequently, SG(1)'s partial keys will be updated. For example, in Figure 3.7, SGC(1,1,3) updates partial keys of SG(1,1). SGC(1,3,1) updates partial keys of SG(1,3), and after that SGC(1,3,3) updates partial keys of SG(1,3) as well. Now it is time for updating SG(1)'s partial keys. SGC(1,1,3) updates partial keys of SG(1) first (SGC(1,1,3) represents SG(1,1)). And then SGC(1,3,3) updates partial keys of SG(1) (SGC(1,3,3) represents SG(1,3)).
- After all of the partial key sets being updated, the SGCs, which have membership change multicast the partial key sets, they are owing to the whole group.

3.2.3.5.3 GC and other SGCs' behavior

GC and other SGC will receive multiple partial key sets since each SGC that has membership change multicast it's own partial key set. Because every key-renew packet contains the same key version, the SGCs will save the partial key sets that are sent by the most related SGC. For example, in Figure 3.7, SGC(1,1,2) receives partial key sets from SGC(1,1,3), SGC(1,3,1) and SGC(1,3,3). It will save the partial key sets of SG(1,1) and SG(1) from SGC(1,1,3)'s packet, because both SGC(1,1,2) and SGC(1,1,3) belong to SG(1,1), while the common subgroup of SGC(1,1,2) and SGC(1,3,1), SGC(1,3,3) is SG(1). So, SGC(1,1,2) and SGC(1,1,3) are most related. Then, they will calculate the

subgroup keys and group key the same as they do in section 3.2.3.2.2.

3.2.3.5.3 Members' behavior

Members, including old members and new members, will find the most related SGC's upon receiving multiple partial key sets, similar to the above illustration. The members will not save the partial key sets. They will calculate the subgroup keys and group key the same as they do in section 3.2.3.2.3.

3.2.3.6 Key Refresh

3.2.3.6.1 Behavior of the SGC which subgroup key expires

Once the subgroup key lifetime reach to an end, the SGC of this subgroup initiates the key refresh processing:

- Send key-refresh request to GC. When it receives key-refresh-accept message including the new key version from GC, it will begin key refreshing.
- Updates partial key sets and keys by adding a random exponent to every partial key and subgroup key
- Update all of the upper level subgroup keys and partial key sets. Do the following calculations:
 - 1) Assign the key of current subgroup as the private exponent of the PM which the current subgroup represents in the upper level subgroup.
 - 2) Go to the upper level.
 - 3) Retrieve this PM's partial key, replace it with the old updated subgroup key
 - 4) For every partial keys other than this PM's partial key, do $DH(\text{private exponent of PM, partial key of other member})$
 - 5) Repeat 1-4 until finishing recalculating the partial key set of the first level subgroup, SG(1)
- Remove the subgroup keys from the new key set, only multicast out the new

partial keys to the whole group, which then initiate the key renew process for each element of the whole multicast group

3.2.3.6.2 GC and other SGC renew key sets

GC and other SGC's key-renew process is the same as they do in section 3.2.3.2.2.

3.2.3.6.3 Existing member renew subgroup keys

The existing members' key-renew process is the same as they do in section 3.2.3.2.3.

Chapter 4: Implementation of M-CLIQUEs

4.1 M-CLIQUEs Key Agreement Protocol Implementation

In this thesis, a Hierarchical CLIQUES protocol with a single member join, single member leave and key refresh is implemented using Java language by using JDK1.5.0. Java Cryptography Extension (JCE) is a cryptography library included in JDK1.5.0 (It was an independent package before JDK1.5.0). JCE provide a framework and implementations for encryption, key generation and key agreement (DH), and Message Authentication Code (MAC) algorithms. It supports encryption includes symmetric, asymmetric, block, and stream ciphers.

4.1.1 Classes

In JCE, `javax.crypto.KeyAgreement` Class to generate DH key pair, compute 1024bit DH key, and generate DES key from the calculated DH key for multicast data encryption. Java security package `java.security` Class is another JDK1.5.0 library which provides management and storage of cryptographic public key pairs, as well as a number of exportable cryptographic operations including those for message digest and signature generation. It provides APIs to store and manage the DH and DES keys.

Based on the above standard APIs, a proprietary DH class was developed to generate, compute, extract and store DH keys, which include private exponents, partial keys and subgroup keys. Also, a proprietary DES class was developed to encrypt and decrypt multicast data.

4.1.1.1 Class DH

Constructor Summary	
public	DH() Generate DH key pair (private exponent and public key, which are a

	and g^a) by the system using. Creates a DH KeyAgreement object using prior generated DH private exponent.
public	DH(DHPublicKey) Creates a DH KeyAgreement object with known DH private exponent. This constructor is used when setting the subgroup key as the private exponent of it's represented PM in the upper level subgroup.
Method Summary	
Key	getPublicKey() Extract public key from DH KeyAgreement object.
Key	getPrivateKey() Extract private exponent from DH KeyAgreement object.
Key	genPublicKey(Key) Compute and return secret key (g^{ab}) by specifying a partial key, in DH KeyAgreement object. This is used to calculate new subgroup key when a node gets it's updated partial key.
SecretKey	genDESKey(Key) Generate DES key from computed subgroup key. Because the subgroup key computed by the above genPublicKey(Key) method is a 1024bit DH key. From this DH key, we can generate secret keys for DES, 3DES, or IDEA algorithm. In this thesis, DES algorithm is used to encrypt multicast data, hence, the DES key is the actual group key. This method is to generate a 56bit DES key from the calculated 1024bit DH group key.

4.1.1.2 Class DES

Constructor Summary	
public	DES(int, SecretKey)

	Construct an encryption cipher or decryption cipher using the inputted DES key. When the first parameter equals to 1, construct an encryption cipher; when the first equals to 2, construct a decryption cipher.
Method Summary	
byte[]	encrypt(byte[]) Encrypt clear text in encryption DES object.
byte[]	decrypt(byte[] clearText) Decrypt ciphertext in decryption DES object.

4.1.2 State machines

The Figure 4.1, 4.2 and 4.3 illustrate the states of GC, SGC and member.

Figure 4. 1 State machine for Group Controller (GC)

Figure 4. 2 State machine for Subgroup Controller (SGC)

Figure 4. 3 State machine for Member

4.1.3 Packet and message format

There are two types of packets: data packet and control packet. The data packet contains the encrypted multicast data. The control packets are used for the key agreement, handling the re-keying process. The fields in a packet are shown in Table 4.1.

Fields	Field Length	Value
PacketType	1 byte	0x01 = Data package 0x02 = Control packet
Payload	Variable length	If type=0x01, then payload is encrypted data; If type =0x02, then payload is control packet

Table 4. 1 Packet Types

4.1.3.1 Data packet format

In a Data Packet there are three fields, shown in Table 4.2.

Fields	FieldLength	Value
KeyVersion	2 bytes	Key version number. It indicates the current version of group key. The member compares this with its group key's version. If they don't match, member will request its current partial key from the immediate SGC and recalculate the group key.
Length	16 bytes	Length of data
Data	Variable length	Encrypted multicast data

Table 4. 2 Fields in Data packet

4.1.3.1 Control packet format

Control packets are the messages being sent among GC, SGCs and Members when initializing and re-keying. The fields in a control packet is shown in Table 4.3.

Fields	Field Length	Value
MessageType	2 bytes	0x0001 = SGC join request 0x0002 = SGC join accept (send partial key)

		0x0003 = SGC join reject 0x0004 = SGC join accept ACK 0x0005 = SGC leave request 0x0006 = SGC leave accept (send partial key) 0x0007 = Member join request 0x0008 = Member join accept (send partial key) 0x0009 = Member join redirect (accept) 0x000A = Member join reject 0x000B = Member join accept ACK 0x000C = Member leave request 0x000D = Member leave accept 0x000E = Member leave notify 0x000F = Key set renew notify (send partial key) 0x0010 = Tree expansion request 0x0011 = Tree expansion accept 0x0012 = Tree expansion accept ACK
TreeHeight	2 bytes	The height of tree
SenderNodeID	Byte[TreeHeight]	Node ID identify the node in the tree. It is a byte array. The number of array elements equals to the value of Tree Height, e.g. "1-1-0-1", "1-3-2-0-1".
SenderNodeType	1 byte	This is the node type of message sender 0x01 = GC 0x02 = SGC 0x03 = Member

KeySetInfo	Variable length	Only used when MessageType are 0x0002, 0x0006, 0x0008, 0x000F
------------	-----------------	---

Table 4. 3 Fields in Control packet

Further in KeySetInfo field, it contains the following sub-fields shown in Table 4.4:

Fields	Field Length	Value
KeyVersion	2 bytes	Key version number of the message
Length	4 bytes	Length of key set
PartialKeySets	Variable length	The updated Partial Key sets

Table 4. 4 Sub-fields in KeySetInfo field

4.2 GUI

GUIs for GC, SGC and member are developed to run the applications, monitor re-keying process, and view the current DES key and key version. Figure 4.4, 4.5 and 4.6 shows those GUIs.

Figure 4. 4 GUIs for GC

Subgroup Controller

Start **Stop**

Local Information	Registration Information
Host Name: master	Server Address: 10.3.244.128
Local Address: 10.3.244.128	Port Number: 5000
Listening Port: 4000	My Node ID: 1-1-0-0

Summary

Current key: 9D:B3:40:6D:6D:AE:EF:B5

Key generation time(ms): 541

Key version: 2

Message Box

Handle >received key set from GC
 new des key = 4C:62:E6:BA:DA:5D:F7:7F
 construct local keysets completed
 SGC 1-1-0-0 multicast key set
 Start >register to GC successfully
 Start >starting TCP service ...
 Start >starting UDP service ...
 Receiver >UDP service listening ...
 received keyset request from Member
 Handle >handle member join request
 new des key = 9D:B3:40:6D:6D:AE:EF:B5
 SGC 1-1-0-0 multicast key set
 Receiver >receive a datagram package
 Handle >handle renew keyset

Figure 4.5 GUIs for SGC

Figure 4.6 GUIs for member

4.3 Re-keying time comparison

In order to verify the re-keying time of M-CLIQUE, experiments were done in three scenarios. Each scenario had a different type of tree. For every scenario, the re-keying times of new member were collected when it joined a group. The group size was changing from 1 to 63 existed members. Also, the re-keying times of the SGC were collected when a member left the group. The group size was changing from 2 to 64 members.

4.3.1 Scenario 1

Scenario 1 is under the tree structure of one level, which is actually the Static CLIQUES structure. There is no subgroup. All of the members are within one group. Figure 4.7 shows the tree structure.

Figure 4. 7 A height of 1 tree with maximum number of member: $1 \times 64 = 64$

From the results of scenario 1, shown in Figure 4.8 and 4.9, we can see that the re-keying times for joining and leaving are similar, which proves that the size of re-key message and cost of re-keying processing for both member join and leave are proportional to the size of the group. In scenario 2 and 3, the re-keying times of joining and leaving are similar as well.

Figure 4. 8 Re-keying time vs. number of existed members when a new member joined the group of one level tree structure

Figure 4.9 Re-keying time vs. number of existing members when a member left the group of one level tree structure

4.3.2 Scenario 2

Scenario 2 is done under the tree structure of two levels, which is a height of 2, order of 8 Hierarchical CLQUES structure. Figure 4.10 shows the tree structure.

Figure 4.10 A height of two tree with maximum number of member: $8 \times 8 = 64$

From the results of scenario 2, shown in Figure 4.11 and 4.12, we can see that the re-keying times of joining and leaving are similar. The re-keying time vs. the total number of existing members of the group is not linear, instead, it is repeating every 8 members. This is because the height of tree and the size of each subgroup are fixed. The re-keying time is proportional to the size of the subgroup because the membership change in a subgroup will not affect other subgroup's secret key. When a subgroup is full, the next new member will be assigned to another subgroup, this causes the curve repeats.

Figure 4. 11 Re-keying time vs. number of existing members when a new member joins the group of two level tree structure

Figure 4. 12 Re-keying time vs. number of existing members when a member leaves the group of two level tree structure

4.3.3 Scenario 3

Scenario 3 is done under the tree structure of two levels, which is a height of 3, order of 4 Hierarchical CLQUES structure. Figure 4.13 shows the tree structure.

Figure 4. 13 A height of three tree with maximum number of member: $4 \times 4 \times 4 = 64$

The results of scenario 3, shown in Figure 4.15 and 4.16, are similar to the results of scenario 2. The differences are, the curve is repeating every 4 members and the maximum time is less than that in scenario 2.

Figure 4. 14 Re-keying time vs. number of existing members when a new member joins the group of three level tree structure

Figure 4. 15 Re-keying time vs. number of existing members when a member leaves the group of three level tree structure

4.3.4 Comparison

By comparing re-keying times of member join in three scenarios, and comparing re-keying times of member leave in three scenarios, as shown in Figures 4.16 and 4.17, we can see that given the same total number of members (the same group size) is fixed, the higher the tree, the lower cost of number of partial keys stored in SGC and the number of partial keys that SGC needs to update. It proves the analysis in Chapter 3, for equation (1) and (2). But still, the re-calculation costs in the other members are proportion to the height of the tree. We did not do the scenario to collect this data. Consequently, the height and order of tree should be carefully selected.

Figure 4. 16 Member join re-keying time in trees of height 1, 2 and 3

Figure 4. 17 Member leave re-keying time in trees of height 1, 2 and 3

From the data of the scenarios, the re-keying time seems long. It is probably due to the fact that Java language is not good for scientific calculation. Its speed is slower than C language. The reason why Java is chosen because it is easier to develop our protocol and our purpose is to implementing the protocol to verify its workability. So performance is not the main concern in the implementation. The protocol can be developed using another efficient computer language in the future.

4.4 Comparison to Other Key management Approaches

Comparisons of Key Tree-Based approach, CLIQUES, Static CLIQUES, and Hierarchical CLIQUES have been done on the number of keys stored in the server and members, re-key messages for join and leave, re-key processing for join and leave, and the cost of implementation. The comparisons are summarized in Table 4.5.

	Key Tree	CLIQUES	Static CLIQUES	Hierarchical CLIQUES
Number of keys stored in a controller	$\frac{N-1}{m-1}$	N	N	$m \log_m N$
Number of keys stored in each member	$\log_m N$	N for member as controller; 1 for other members	1	1 (or $\log_m N$, depending on the implementation)
Size of re-key messages on a join	$2 \log_m N$	N	N	$m \log_m N$
Size of re-key messages on a leave	$m \log_m N$	N	N	$m \log_m N$
Number of re-key processing by a new member on a join	$\log_m N$	N	N	$m \log_m N$
Number of re-key processing by each old/ remaining member on a join/leave	1 for the best; $\log_m N$ for the worst	1	1	1 for the best; $\log_m N$ for the worst
Requirement of memory in a user machine	medium	high	low	low-medium
Requirement of processing power in a controller	high	high	high	medium
Knowledge of member machine	low	high	low	low
Member privacy protection	high	low	high	High
Cost of system design and implementation	medium	high	low	medium

Table 4. 5 A comparison of key management schemes
(N is the group size, m is the number of children of the subgroup)

From Table 4.5, we can see that for CLIQUES, the number of keys stored in the controller and the size of re-key messages are the highest among all of the compared schemes. Therefore the larger memory and higher processing power on the user machine are required since the member could be acting as controller. Because of this, CLIQUES is not suitable for a large multicast group, which means it is not scalable. Also, a dynamic

controller being dynamical assigned increases the risk of security as the members' machines have the knowledge of all the functions. Once a member becomes the controller, it will be aware of all of the current members. Hence, the member privacy protection could not be achieved. At last, the controller's role will be switched from one member to the other as the members can join and leave the group dynamically and frequently. The mechanism of handling controller's changing makes the design and implementation of the scheme more complicated.

Static CLIQUES has a smaller cost in terms of system design and implementation and lower requirement of member's machine than CLIQUES. Also it is good for member privacy protection and more secure as the static controller is static. However, it is not scalable.

Based on Static CLIQUES, Hierarchical CLIQUES reduces the size of key storage in controllers, which are comparable to key tree-based approach. In Hierarchical CLIQUES, the size of re-key message upon member leave is comparable to key tree-based approach as well. But the size of re-key message upon member join is more than key tree-based approach. Actually, the size of re-key message is the same for both join and leave in Hierarchical CLIQUES. At the member side, Hierarchical CLIQUES needs smaller key storage than key tree-based approach. In terms of the requirement of memory in a member machine and requirement of processing power in a controller, Hierarchical CLIQUES is slightly better than key tree-based approach.

In this thesis, other Key management methods were not implemented, since this thesis focused on studying and implementing M-CLIQUES only.

Chapter 5: Conclusion

In this thesis, several current multicast key management approaches are introduced, such as Key Tree-based approach and Contributory key agreement algorithm – CLQUES. A Modified-CLQUES key agreement algorithm has been proposed in order to create a scalable and efficient multicast key management scheme. Modified-CLQUES, which in short is called M-CLQUES, includes two stages: Static CLQUES and Hierarchical CLQUES. Static CLQUES introduces a static group controller to store and distribute the partial keys when a membership change occurs. Hierarchical CLQUES splits the members into subgroups and “recursively” splits each subgroup to form multi-level subgroups. Members in each subgroup “agree” on a subgroup key based on Static CLQUES. A subgroup key is the private exponent of the subgroup in the upper level subgroup. By constructing such a subgroup key tree structure, the total number of partial keys stored and distributed by the subgroup controllers is significantly reduced.

Hierarchical CLQUES protocol with a single member join, single member leave and key refresh is implemented using Java language. Scenarios have been performed to show the re-keying time under different height of tree and various key management schemes has been compared in Chapter 4.

In addition, several security issues related to the proposed M-CLQUES scheme are considered.

- 1) All partial keys are transmitted over the network without encryption. It is quite easy for a sniffer to collect partial keys and try to figure out the subgroup key. There are two assumptions of multi-party DH: the Group Decisional DH assumption (G-DDH) [7] and Group Computational DH assumption (G-CDH) [25]. The G-DDH problem has been proved that if the two-party DDH problem is hard, then the multi-party DDH (G-DDH) problem is hard as well [7]. Though the G-CDH problem has not been solved, it is believed to be a “hard” problem [25], too. Based on the security of multi-party DH algorithm, researchers believe that it is safe to transmit partial keys by plain format. In addition, the subgroup key can

be refreshed periodically in the algorithm. Hence no one else can compute the shared key in a reasonable amount of time even though they know all of the partial keys. In this thesis, the strength of multi-party DH algorithm is not a studying topic. Multi-party DH algorithm is assumed strong enough during a reasonable key refresh interval.

- 2) When there is a membership change occurs, GC/SGC multicasts re-key messages to the group. In order to avoid re-key spoofing, the re-key message must accompany with the sender's signature. When implement the M-CLIQUE protocol, existing signature algorithm can be selected. In this thesis, we assumed a proper function of validating signature has been imbedded into the process during implementation.
- 3) Because the transmission protocol for multicast is UDP, which is connectionless, hence, the re-key message's transmission may not be reliable. Members may not receive some re-key messages. If this happens, the group key will be out-of-sync. In order to achieve successful key synchronization, we introduce a key version field in multicast data packets, which can help multicast for re-key messages more reliable. If the received key version is newer than the current key version on the member, or the member notices that the data cannot be decrypted correctly, it is aware that its current key is invalid. It will inquire its new public component from its SGC and compute the new group key.
- 4) As mentioned before, authentication between GC/SGC and the member is required before member joins or leaves the group. It is because DH algorithm is vulnerable to Man-in-the-Middle attack. In this thesis, I assumed a proper function of member authentication is assumed being imbedded into the process during implementation.

In this thesis, the proposed M-CLIQUE protocol assumes that the maximum group size N is fixed. However, in reality, it is highly possible that the number of

subscribers grows fast and exceeds the estimated maximum group size. In some cases the group may need to be downsized. Therefore, methods of tree expansion and reduction are two of the future research topics. Also, the implementation of multiple-member join and leave processes needs to be completed. The codes may need to be re-written using other languages such as C++.

References

- [1] M. Steiner, G. Tsudik, and M. Waidner, "Key Agreement in Dynamic Peer Groups", IEEE Trans. Parallel and Distributed Systems, vol. 11 no. 8, Aug. 2000, pp. 769-80
- [2] M. Just and S. Vaudenay, "Authenticated Multi-Party Key Agreement", Proc. Advances in Cryptology – ASIACRYPT: Int'l. Conf. Theory and Application of Cryptology and Info. Security, LNCS, Springer-Verlag, 1996, pp. 36-49
- [3] M. Burmester and Y. Desmedt, "A Secure and Efficient Conference Key Distribution System", Proc. Advances in Cryptology – EUROCRYPT'94, Wksp. Theory and Application of Cryptographic Techniques, Springer-Verlag, 1995, pp. 275-86
- [4] G. Ateniese, M. Steiner, and G. Tsudik, "New Multi-Party Authentication Services and Key Agreement Protocols", IEEE JSAC, vol. 18, no. 4, Apr. 2000, pp. 628-39
- [5] K. -C. Chan and S. -H. G. Chan, "Key Management Approaches to Offer Data Confidentiality for Secure Multicast", IEEE Network, vol. 17, Issue 5, Sept.–Oct. 2003, pp. 30-39
- [6] W. Diffie and M. Hellman, "New Directions in Cryptography", IEEE Trans. Info. Theory, vol. IT-22, Nov. 1976, pp. 644-54
- [7] D. M. Wallner, E. J. Harder, and R. C. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, June 1999
- [8] C. K. Wong, M. Gouda, and S. S. Lam, "Secure Group Communication Using Key Graphs", IEEE/ACM Trans. Net., vol. 8, no. 1, Feb. 2000, pp. 16-29
- [9] R. Canetti et al., "Multicast Security: A Taxonomy and Some Efficient Constructions", Proc. INFOCOM'99, 1999, vol. 2, pp. 708-16
- [10] I. Chang et al., "Key Management for Secure Internet Multicast using Boolean Function Minimization Techniques", Proc. IEEE INFOCOM'99, Conf. Comp. Commun., vol. 2, 1999, pp. 689-98
- [11] D. Balenson, D. A. McGrew, and A. T. Sherman, "Key Establishment in Large Dynamic Groups Using One-Way Function Trees and Amortized Initialization", Internet Draft, 1999
- [12] W. Diffie, "Authenticated Key Exchange and Secure Interactive Communication", Proc. SECURICOM, (Paris, France), 1990, pp. 399-406
- [13] NIST: FIPS PUB 46-3 for Data Encryption Standard (DES), Oct. 1999

- [14] NIST: FIPS PUB 197 for Advanced Encryption Standard (AES), Nov. 2001
- [15] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Commun. ACM, Feb. 1978,, pp. 120-26
- [16] S. Mitra, "Iolus: A Framework for Scalable Secure Multicasting", Proc. ACM SIGCOMM'97, 1997, pp. 277-88
- [17] T. Hardjono, B. Cain, and I. Monga, "Intra_Domain Group Key Management Protocol", Internet Draft, IETF, 1998
- [18] T. Hardjono, B. Cain, and N. Dorawamy, "A Framework for Group Key Management for Multicast Security", Internet Draft, IETF, 1999
- [19] M. Waldvogel et al., "The VersaKey Framework: Versatile Group Key Management", IEEE JSAC, vol. 17, no. 9, Sept. 1999, pp. 1614-31
- [20] K.-C. Chan and S.-H. Gray Chan, "Distributed Servers Approach for Large-Scale Secure Multicast", IEEE Journal on Selected Areas in Communications, vol.20, no.8, Oct. 2002, pp.1500-1510
- [21] Y. Amir, Y. Kim, C. Nita-Rotaru, J. Stanton, J. Schultz and G. Tsudik, "Secure Group Communication Using Robust Contributory Key Agreement", IEEE Trans. Parallel and Distributed Systems, vol. 15, no. 5, May 2004, pp. 468-480
- [22] R. Rivest, "The MD5 Message-Digest Algorithm", RFC 1321, Apr. 1992
- [23] NIST: FIPS PUB 180 for "Secure Hash Standard", May, 1993
- [24] Patrick P. C. Lee, John C. S. Lui, and David K. Y. Yau, "Distributed Collaborative Key Agreement and Authentication Protocols for Dynamic Peer Groups", IEEE/ACM Trans. Networking, vol. 14, no.2, April 2006, pp. 263-276
- [25] E. Bresson, O. Chevassut, D. Pointcheval, "Group Diffie-Hellman Key exchange Secure against Dictionary Attacks", ASIACRYPT 2002, pp. 497-514