A Scalable Group Key Agreement protocol for multiparty conferencing over P2P Networks

Author

Alamzeb khan
MS-07 (Software Engineering)

Supervisor

Dr.Ghalib AsadUllah Shah
Assistant Professor

DEPARTMENT OF COMPUTER ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND ECHNOLOGY
ISLAMABAD.

July, 2009

A Scalable Group Key Agreement protocol for multiparty conferencing over P2P Networks.

Author

Alamzeb khan

MS-07 (Software Engineering)

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

MS (Computer Software Engineering)

Thesis Supervisor:

Dr. Ghalib AsadUllah Shah

Assistant Professor,

Department of Computer Engineering.

Thesis Supervisor Signature: ____

DEPARTMENT OF COMPUTER ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

JULY, 2009

ABSTRACT

A Scalable Group key Agreement protocol for multiparty conferencing over P2P Networks.

Alamzeb Khan

Thesis Supervisor: Dr.Ghalib AsadUllah Shah

Assistant Professor,

Department of Computer Engineering.

Up to now most of the group key agreement protocols (often tree-based) involve unnecessary delays because members with low-performance computer systems can join group key computation. These delays are caused by the computations needed to balance a key tree after membership changes. This work presents an efficient stack-based and tree-based group key agreement protocol (STGDH) and the results of its performance are better than other approaches used by researchers. The proposed approach to filtering of low performance members in group key generation is scalable and it requires less computational overhead than other conventional group key agreement protocols. STGDH protocol uses two approaches for efficient group key generation and distribution. STGDH Protocol uses tree-based DIffie-Hellman to manage the group members in the network and secondly it keeps the information of highest performance member in a stack data structure. The highest performance is called the 'group controller'. So the proposed approach implements tree-based and stack-based technique. STGDH Protocol provides efficient computation for group key generation by filtering low performance members from this process. It also maintain a stack for all group controllers in the tree so that to make the network reliable incase, any group controller leaves the network or is lost due to network fault. The main advantage of this approach is efficient group key generation and distribution in minimum possible time and keep information of all the preceding group controller that is the highest performance members. The experimental results of this approach are better than conventional group key agreement protocols.

Keywords: Group key agreement, group key generation and distribution, group controller

UNDERTAKING

I certify that research work titled "**A Scalable Group Key Agreement Protocol for multiparty conferencing over P2P networks**" is my own work. The work has not been presented elsewhere for assessment. Where material has been used from other sources it has been properly acknowledged / referred.

Alamzeb Khan,

REG NO: 2007-NUST-MS PhD-CSE (E)-07

ACKNOWLEDGEMENTS

When my course work gets completed, I was confused in selection of my thesis topic. It is an important stage in which any body needs help and guidance. For this purpose, I went to Dr.Ghalib AsadUllah Shah and discussed some topics in the field of computer networks. After selecting my thesis topic and supervisor, Dr.Ghalib guided me throughout my thesis work by properly guiding me, and giving me technical assistance in this area. I enjoyed my thesis work very much because of the friendly nature of supervisor. I would like to say thanks to my supervisor for his so nice, friendly supervision.

After my theoretical work, when I was doing simulation of my work. I got stuck in some technical problems in visual studio.NET 2005. I was in search of someone who helps me in some technical problems. I discussed some technical problems with one of my class fellow who helped me in it thoroughly. I would to like to say thanks to Mr. Sheraz for helping me in technical problems. Finally I am grateful to all those who helped me during my thesis work.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Statement of the Problem

Peer-to-peer (**P2P**) networking is a method of delivering computer network services in which the participants share a portion of their own resources, such as processing power, disk storage, network bandwidth, printing facilities. Such resources are provided directly to other participants without intermediary network hosts or servers. Peer-to-peer network participants are providers and consumers of network services simultaneously, which contrasts with other service models, such as traditional client-server computing where the clients only consume the server's resources. P2P networks has a lot of advantages due to which the world is turning towards P2P communication instead of client-server architecture, but P2P networks are also facing some problems and the most important is security. P2P networks are used for many applications but here we will discuss P2P networks from security point of view for multiparty conferencing.

Multi-Party Conferencing is a fully distributed multi-party video conferencing system. No central server is needed. Instead Multi-Party Conferencing participants are directly connected with each other. Multi-Party Conferencing has a mechanism to avoid security attacks. The mechanism is based on the philosophy that real world computer security is not only a matter of providing locks, but is also related to the comparison of value and costs. The main purpose of this work is to design multiparty conferencing softwares that provide efficient security mechanism. So far many multiparty conferencing tools have been designed like Skype, Digiparty, and Digimetro etc but they still lack in efficient security mechanism. The main emphasis of this work is to design efficient security protocol for multiparty conferencing. If the required security is achieved it could be used for many government, private sector organizations and especially for military use.

1.2 Introduction to P2P Networks

Nowadays P2P networks are evolving throughout the world in many areas such as audio-video conferencing, online chatting programs, games, file sharing etc but the main threat to peer-to-peer communication is security. Secure group communication is very important factors in peer-to-peer communication. Many researchers are trying to design efficient and secure group communication protocols. So far the work done in this area is appraised but most of the researchers ignored the performance factor.

A P2P network is a special type of computer network that exhibits self-organization, symmetric communication, and distributed control. The network is self-organizing in that there is typically no centralization of resources. As a result, link capacity is typically distributed throughout peers in the network, and as a result control is distributed, as well. Peer-to-peer systems have two dominant features that distinguish them from a more standard client-server model of information distribution: they are overlay networks that have unique namespaces. P2P systems link different, possibly heterogeneous systems as 'peers,' and allow them to interact on top of existing network configurations. It does this by defining relationships unique to that system, usually in the form of a topology by which systems are linked. The research on P2P system can be divided into four groups- search, storage, security and applications. Here in this work we have discussed security in detail and other groups are beyond the scope of this work.

1.2.1 P2P Security Issues and Risks

Since P2P systems inherently rely on the dependence of peers with each other, security implications arise from abusing the trust between peers. The security risks can be divided into three categories: security, legal and infrastructure risks. Here in this work we have focused on security risks. P2P protocols have been designed with a lot of features but it does not focus on security measurements. They take advantage of:

1. Decentralized data storage.
2. Unauthenticated access to data storage.
3. Operate in an environment of unstable connections.
4. Provide connection to temporary or unpredictable IP addresses.

5. Possess significant or total autonomy from central servers; And avoiding filtering and security policy control.

These objectives stand in a sharp contrast to network security requirements. Most P2P applications, both file sharing and instant messaging, have weak or easily cracked measures to protect user identities. There is no encryption of any communication sent or received via most P2P protocols. A malicious hacker who identifies a security flaw in one or more poorly written P2P clients could then launch a DoS attack against the P2P peer or use it as a part of Distributed Denial of Service (DDoS) against a third target. In a P2P network, attackers can make use of the querying nature of P2P networks to overload the network. In the case of the query flooding P2P network, the attack is straightforward: simply send a massive number of queries to peers, and the resulting broadcast storm will render portions of the network inoperable.

Most P2P applications are bundled with spyware, software that automatically installs with the P2P application being installed. Spyware is now considered even more dangerous than viruses because a virus can only damage data on victim's computer. Spyware on the other hand, not only can result in data corruption, but also identity theft.

P2P networks are a new and efficient way to spread worms and viruses. P2P based viruses and worms attach themselves to or disguise themselves as movie, music or software files that appeal to the entire P2P community and wait for the users to "pull" it from an infected node. The number of participating nodes in P2P networks and the number of files being exchanged every day guarantee an effective way of malware distribution.

Injecting a useless data (poison) in P2P network, since P2P networks implements a lookup service in some way. An attacker can inject large amounts of useless lookup key-value pairs into the index. Bogus items in the index could slow down query times or, worse, yield invalid queries results. Poisoning can also be used as fodder for DDoS attacks.

P2P networks also present privacy and identity issues. In respect to privacy, a peer's data stream may be compromised by fellow peers who assist in transmitting the data. A direct example is that of VoIP applications, such as Skype, which route traffic in a P2P fashion.

In P2P networks which distribute resources of dubious legality, the issue of lack of anonymity becomes apparent. For example, the Bit Torrent file sharing system directly exposes the IP address of peers to each other in a swarm. This would allow peers in the swarm to know the identity of other peers who are downloading certain resources.
An important issue that looms over P2P networks is blocking and throttling of P2P traffic. In next section we have discussed solutions to the security issues and risks faced by P2P networks.

1.2.2 Solution to P2P Security Issues

There are two straightforward approaches to securing P2P networks: encrypting P2P traffic and anonymizing the peers. By encrypting P2P traffic, the hope is that not only will the data be safely encrypted, but more importantly, the P2P data stream is encrypted and not easily detectable. With the actual connection stream completely encrypted, it becomes much harder for the P2P traffic to be detected, and, thus, attacked, blocked, or throttled. The first step in addressing the P2P application problem is to accurately recognize P2P traffic within the corporate network flow. Modern network monitoring tools with application layer classification capabilities are available to satisfy these requirements. Traffic analysis is an important first step to identify hosts running P2P applications. By anonymizing peers, the P2P network can protect the identity of nodes and users on the network, something that encryption only cannot ensure

1.3 Objectives

- To provide efficient group communication via P2P networks.
- To provide secure group communication via P2P networks.

1.4 Thesis Organization

This thesis is organized as follow:

- Chapter I: discusses problem statement, brief introduction to P2P networks, P2P security issues and risks, solution to P2P security issues and then objectives that needs to be achieved.

- Chapter II: introduction to group communication, group management, group key secrecy and then group key agreement protocols are discussed in detail.
- Chapter III: this chapter is about proposed approach, introduction to STGDH, membership operations (join, leave, partition and merge) and summary of the proposed approach.
- Chapter IV: discusses the experimental results of the proposed approach and comparison with other group key agreement protocols, then conclusion and future work in this area.
- Chapter V: Simulation tutorial
- References

CHAPTER 2

Literature Survey

2.1 Introduction to Group Communication

The proliferation of applications, protocols and services that rely on group communication prompts the need for group-oriented security mechanisms (in addition to the traditional requirements of fault tolerance, scalability, and reliability). Current group-oriented applications include IP telephony, video conferencing, collaborative workspaces, interactive chats and multi-user games. The security requirements of these applications are fairly typical, e.g., confidentiality, data integrity, authentication and access control. These are achieved through some form of group key management. The peer nature of many group applications results in certain unique properties and requirements. First, every member in a peer group is both a sender and a receiver. Second, peer groups tend to be small, with fewer than a hundred members. Also, peer groups have no hierarchy and all members enjoy the same status. Therefore, solutions that assign greater importance to some group members are undesirable, since privileged members might behave maliciously; they are also attractive targets of attacks. This essentially rules out the traditional key distribution paradigm as it calls for higher trust in the group member who generates and distributes keys. Finally, since all networks are prone to faults and congestion, any subset of group members must be prepared to function as a group in its own right. In other words, if a network partition splits the members into multiple subgroups, each subgroup must quickly recover and continue to function independently.

There are two kinds of group communications, one-to-many and peer-to-peer. One-to-many is client server based communication, for example, TV or radio broadcasting, Geographic Position System (GPS), and so forth. In peer-to-peer communication the group size is relatively small, less than 100 and there is no centralized controller. In this work, the term group refers to peer-to-peer group communication. Membership in a dynamic peer-to-peer group communication tends to change frequently. Networks are generally regarded as insecure because they are connected to each other and there is no central controller. A secure communication channel must be established in group communications to protect messages over an insecure network environment. Currently, a group key management protocol is being used for establishing a secure communication channel.

2.2 Group Key Management

Group communication arises in many different settings: from low-level network multicasting to conferencing and other groupware applications. In particular, group communication is often crucial in the battlefield. Regardless of the environment, security services are necessary to provide communication privacy and integrity. For secure communication, group members need a common group key to protect their messages while they are communicating to others. In this context, group key management is responsible for generating a group key and distributing it to each member securely over an insecure network environment, making key management the building block in group key management. Unless the communication channel is secure, the delivery of messages over the network to the right destination cannot be guaranteed. Group key management is used for establishing a secure channel.

Group key management can be classified in two categories; group key distribution and group key agreement. In group key distribution one member is designated as the key distribution center. He/she computes the group key and distributes it to each member in the group. This scheme is suitable for client-server environments like multicast. However, peer-to-peer group communication needs a different key generation and key distribution method due to the characteristics of peer-to-peer group communication such as dynamic, relatively small number of group size, network partition, and merging.

2.3 Group Key Secrecy

Group key management focuses on how to generate a secure group key efficiently. In doing that, the group key must be secure in order to build a secure communication; therefore, four security requirements are required: 1) group key secrecy, 2) backward secrecy, 3) forward secrecy, and 4) key independence.

Definition 1: Assume that a group key is changed *m* times and the sequence of successive group keys is $K = \{K_0, K_1 \ldots K_{m-1}, K_m\}$.

1. Group Key Secrecy guarantees that it is computationally infeasible for a passive adversary to discover any group key $Ki \in K$ for all *i*.
2. Forward Secrecy guarantees that a passive adversary who knows a contiguous subset of old group keys (say $\{K_0,K_1,\ldots,K_i\}$) cannot discover any subsequent group key K_j for all *i* and *j* where $j > i$..
3. Backward Secrecy guarantees that a passive adversary who knows a contiguous subset of group keys (say $\{K_i, Ki_{+1},\ldots, Kj\}$) cannot discover preceding group key K_l for all *l, j, k* where $l < i < j$.
4. Key Independence guarantees that a passive adversary who knows a proper subset of group keys $K_{subset} \in$ K cannot discover any other group key $K_i \in (K - K_{subset})$.

Before examining group key secrecy, possible security attacks must be defined. A group key is a common secret key which means one key can encrypt and decrypt the messages during communication, so it is a symmetric cipher. In a symmetric cipher, the most well known attack is a brute-force attack which is the process of enumerating through all of the possible keys until the proper key is found that decrypts a given cipher text into correct plain text. All symmetric encryption algorithms will eventually fall to brute-force attacks given enough time. It can be helpful for group key secrecy to see if the vulnerability of the algorithm for generating a group key is revealed through the usage of brute-force. If there are enough possible keys to slow down such an attack, then the algorithm can be considered secure. In a brute-force attack, the expected number of trials before the correct key is found is equal to half the size of the key space because it is the average of the best and worst trial cases to find the right key. Symmetric ciphers with keys 64 bits or less are vulnerable to brute force attacks. Therefore, a key size must be long enough to prevent an attack on symmetric ciphers.

The size of a group key in secure group communication is a 1,024 bit-long number which is known to be secure in the current technology, so a brute-force attack must try an average of $2^{1,024} / 2$ times to find the group key, which is computationally infeasible. Furthermore, one of the most important security requirements of group key agreement is called key freshness. A group key is changed regularly and irregularly. There is a certain time period to change a group key. In addition, whenever membership changes, a group key will be collaboratively regenerated. A group key is always refreshed after a membership change or after a certain period of time, so an adversary does not have a chance to use an old key. In addition, a group key in a group communication will not last long and it is generally used for only a few hours because group members have a short life cycle. Therefore, group key secrecy is secure enough for an outside attack.

2.4 Group Key Agreement Protocols

A **key agreement protocol** is a key establishment technique whereby a shared secret key is derived by two or more specified parties as a function of information contributed by, or associated with, each of these, such that no party can predetermine the resulting value.

The idea of group key agreement stems from the earlier work of two-party key management. In 1976, Diffie and Hellman introduced a two-party key exchange protocol, DH that allowed two participants to create a private key through the use of publicly exchanged messages. This protocol allowed two participants, without any prior shared secrets, to securely establish a shared session key. DH is now at the heart of many two-party secure communication protocols. With the prevalence of the Internet and networked technologies, there are applications that would benefit from a group key agreement protocol that provides the same protection as DH, but for groups of more than two participants. This list includes conference calls, distributed computation, whiteboards, distributed databases, Unmanned Aerial Vehicles, and battlefield communications, among many others. To ensure secure and reliable communication in these applications, there have been several attempts to create efficient group key agreement protocols for large and dynamic groups based on the DH algorithm. Currently, a group key management protocol is being used for establishing a secure communication channel. So far different group key agreement protocols have been designed by researchers but they are lacking in some aspects. Some of the protocols that we will discuss are: BD, STR, GDH, TGDH and ETGDH. Group Key Structures (GKS) in current group key agreements are briefly explained as follows, where K_i, $i = 1, 2, \ldots, n$, $1 < n < 100$.

2.4.1 BD (Burmester-Desmedt) Protocol

BD is a protocol based on Burmester-Desmedt variation of group Diffie-Hellman (DH). BD supports dynamic group operations. It has a relatively low computational overhead due to two modular exponentiations. However, it needs more message exchanges to generate group key. Furthermore, BD is completely decentralized and has no sponsors, controllers, or any other members charged with any special duties. The main idea in BD is to distribute the computation among members, such that each member performs only three exponentiations. The cost of BD roughly doubles as the group size grows in increment of the total number of machines. Its GKS can be determined as:

$$\text{GKS} = g^{K1K2+K2K3+\ldots+Kn\text{-}1Kn}$$

2.4.3 STR (Skinny Tree) Protocol

STR is a version of TGDH with the underlying key tree completely unbalanced or stretched out. In other words, the height of the key tree is always (n - 1), as opposed to log (n) in TGDH. All other features of the key tree are the same as in TGDH. After a partition, the sponsor is defined as the member corresponding to the leaf node just below the lowest leaving member. After deleting all leaving nodes, the sponsor refreshes its key share, computes all (key, blinded key) Group Key Agreement Protocols for Dynamic Peer Groups pairs up to the level just below the root node. Finally, the sponsor broadcasts the updated key tree thus allowing each member to compute the new group key. STR merge runs in two rounds. In the first round, each sponsor (topmost leaf node in each of the two merging tree) first refreshes its key share and computes the new root key and root blinded key. Then, the sponsors exchange their respective key tree views containing all blinded keys. The topmost leaf of the larger tree becomes the sole sponsor in the second round in the protocol. Using the blinded keys from the key tree it received in the first round, the sponsor computes every (key, blinded key) pair up to the level just below the root node. It then broadcasts the new key tree to the entire group. All members now have the complete set of blinded keys which allows them to compute the new group key.

STR protocol is modified to provide dynamic group operations. It has a relatively low communication overhead and is well suited for adding new group members. Robustness is easily provided. However, it is relatively difficult for member exclusion (O (*n*) modular exponentiations). Its GKS is:

$$\mathrm{GKS} = g^{K_n g^{K_{n-1} \dots g^{K_2 g^{K_1}}}}$$

2.4.3 GDH (Group Diffie-Hellman) Protocol

GDH is fairly computation-intensive requiring O (n) cryptographic operations upon each key change. Here the shared key is never transmitted over the network. Here the group shared key is generated by each member by using the partial keys of all other members. One member, who is the group controller, distributes this list. The protocol runs as follows. When a merge event occurs (new member/group joins), the current controller generates a new key token by refreshing its contribution to the group key and passes the token to one of the new members. When the new member receives the token, it adds its own contribution and passes the token to the next new member. Eventually, the token reaches the last new member.

When some of the members leave the group, the controller (who, at all times, is the most recent remaining group member) removes their corresponding partial keys from the list of partial keys, refreshes each partial key in the list and broadcasts the list to the group. Each remaining member can then compute the shared key. Note that if the current controller leaves, the last remaining member becomes the controller of the group. Its GKS is:

$$\mathrm{GKS} = g^{K1K2K3K4\dots Kn\text{-}1Kn}$$

2.4.4 TGDH (Tree-based Group Diffie-Hellman) Protocol

It is a tree-based group Diffie-Hellman. The key tree is organized as follows: each node is associated with a key K_v and a corresponding blinded key BK_v derived from the key. The root key represents the group key shared by all members, and a leaf key represents the random contribution by of a group member. Each internal node has an associated secret key and a public blinded key. The secret key is the result of a Diffie–Hellman key agreement between the node's two children. Every member knows all keys on the path from its leaf node to the root as well as all blinded keys of the entire key tree. The protocol relies on the fact that every member can compute a group key if it knows all blinded keys in the key tree.

Following every group membership change, each member independently and unambiguously modifies its view of the key tree. Depending on the type of the event, it adds or removes tree nodes related to the event, and invalidates all keys and blinded keys related with the affected nodes (always including the root node). As a result, some nodes may not be able to compute the root key by themselves. However, the protocol guarantees that at least one member can compute at least one new key corresponding to either an internal node or to the root. Every such member (called a sponsor) computes all keys and blinded keys as far up the tree as possible and then broadcasts its key tree (only blinded keys) to the group. If a sponsor cannot compute the root key, the protocol guarantees the existence of at least one member which can proceed further up the tree, and so on. After at most two rounds (in case of a merge) or log (n) rounds (in case of a worst-case partition), the protocol terminates with all members computing the same new group (root) key. After a partition, the protocol operates as follows. First, each remaining member updates its view of the tree by deleting all leaf nodes associated with the partitioned members and (recursively) their respective parent nodes. To prevent reuse of old group keys, one of the remaining members (the shallowest rightmost sponsor) changes its key share. Each sponsor computes all keys and a blinded key as far up the tree as possible and then broadcasts its view of the key tree with the new blinded keys. Upon receiving the broadcast, each member checks whether the message contains a new blinded key. This procedure iterates until all members obtain the new group key.

In addition, TGDH provides robustness, and its GKS is as follow:

$$\mathrm{GKS} = g^{g^{g^{K_1 K_2} g^{K_3 K_4}} \cdots g^{g^{K_{n-3} K_{n-2}} g^{K_{n-1} K_n}}}$$

2.4.5 ETGDH (Efficient Tree-based Group Diffie-Hellman) Protocol

An Enhanced Tree-based Group Diffie-Hellman (ETGDH) protocol is proposed that improves the existing TGDH, a tree-based group key agreement protocol. To generate a group key efficiently, there must be a group controller. The group controller is the last member to join the group in the Enhanced Tree-based Group Diffie-Hellman (ETGDH) protocol. The newest member always plays the role of the group controller. The group controller is responsible for managing group key generation. He/she initiates group key generation for all members by sending them his/her blind key. All members participate in group key generation. When the group key has been computed, the highest performance member computes the final group key and distributes it to all members, and then group communication begins. Whenever membership changes, the group key must be regenerated. The final group key G is calculated as follows:

$$G = g^{g^{g^{\left(g^{K\langle 3,0\rangle K\langle 3,1\rangle}\right)^{K\langle 2,1\rangle}\left({}^{K\langle 2,2\rangle^{K\langle 3,2\rangle K\langle 3,3\rangle}}\right)}}} \bmod p$$

It still requires more computational cost for group key generation. Each new member whether it is low performance or high performance becomes the group controller and decides which group members will participate in group key generation so if new member is low performance machine then it takes more time in computation. Also there is no idea that who will decide a set of group members for group key generation in case of leave operation.

2.5 Summary

Group key agreement is a fundamental building block for secure peer group communication systems. Several group key management techniques were proposed in the last decade, all assuming the existence of an underlying group communication infrastructure to provide reliable and ordered message delivery as well as group membership information. Despite analysis, implementation and deployment of some of these techniques, the actual costs associated with group key management have been poorly understood so far. This resulted in an undesirable tendency: on the one hand, adopting sub-optimal security for reliable group communication, while, on the other hand, constructing excessively costly group key management protocols. Our work is closely related to two distinct areas: group key management and reliable group communication. Research in securing group communication is relatively new.

CHAPTER 3

Proposed Approach

3.1 Introduction to Proposed Approach

STGDH (Stack and Tree-based Group Diffie-Hellman) protocol is a new group key agreement protocol; it solves all those problems that were faced by researcher before this work. Up to now there were two main problems faced by researchers: computational complexity of generating the group key in terms of computational cost and communication cast; another problem was tree maintenance whenever membership operation was performed because after membership operation, each group member updates its tree structure along the path. In this protocol both of these issues have been resolved very efficiently and the results are impressive. To solve first problem, this protocol always chooses the best performance group member as a group controller whenever membership operation is performed and for this purpose our new designed algorithm called GC Algorithm is executed to choose the highest performance member. To keep track of group controllers we push each group controller on to the stack, so that it can be used for final group key (G) generation. In STGDH, the group key is always generated by group controllers so group key generation process is achieved in minimum possible time due to their performance. For example we have a network of 25 machines in which 4 machines are selected as the group controller then group key is calculated like below:

$$G = GC_1 + GC_2 + GC_3 + GC_4$$

$$\text{Where } GC_1 = g^{K<l,\, v>} \bmod p$$

The top of the stack always contain the highest performance group controller of all for example:

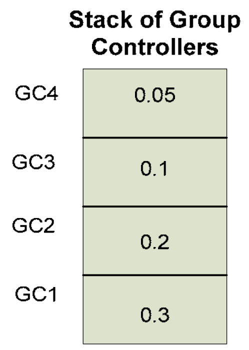


Figure No.01: Shows an example of group controllers based on their response time.

The number of group controllers in a network of group members increases the efficiency. The performance of group key (G) calculation depends on:

- Number of group controllers; the number of group controllers are directly proportional to efficient key generation process. if the number of group controllers is large the key generation process is evenly distributed among group controllers.
- Computational power of each group controllers; that is response time of each group controller.

In our approach, each group controller maintains two types of information. First, it maintains a stack of group controllers. Each group controller contains reference of each group member registered with that group controller. Second, it maintains a tree structure for all group members and group controllers. Now we come towards the second problem that is solved by our approach that is tree structure maintenance. The tree structure is generated by each group controller with the help of stack maintained by group controller and the updated tree structure (T*) is broadcast by the group controller. Group member is not required to updates its tree this work is done by each group controller. The detailed discussion of tree structure is given in section. STGDH protocol is the most efficient protocol as compared to other group key agreement protocols. Due to efficient group controllers each join operation results in O (log n) computations and as the number of group controller increases the performance of join operation becomes more efficient. In case of leave operation, there are two cases: 1 if leaving member is a normal group member, the results of join and leave operation are same and in case 2: if leaving member is a group controller the computation complexity begins to decrease from maximum value and generates a result graph like parabola. In our approach, partition and merge operations are very simple because every group controller has a set of registered group members and on the basis of these group controllers we can easily partition a group in to subgroups and merge subgroups into one group. The result of partition operation remains almost constant and does not decrease so much with the number of partitioned members. In section we will discuss the working of this proposed approach in detail.

3.2 STGDH Protocol

A Stack and Tree-based Group Diffie-Hellman (STGDH) protocol is a tree-based group key agreement protocol that provides maximum efficiency by always choosing the highest performance member as a group controller for group key generation and distribution. Under STGDH approach more efficiency is obtained because in this approach only group controller performs all the tasks and each time when a group member joins the group its response time is compared with the group controller if its response time is lesser than the group controller, the group controller is replaced by the new joined members. The newly joining member starts group key generation by sending his/her blind key to existing group controller. Each member $\mathbf{M_i}$ selects a random private number r_i, calculates $\mathbf{M_i = g^{ri} \bmod p}$ and response time $\mathbf{T_c\ (M_i)}$. The newly joined member forwards his blind key and response time to the group controller. The group controller also computes blind key and response time $\mathbf{T_c\ (GC)}$. Then GC algorithm decides the group controller based on response time. After GC algorithm, group controller broadcasts blind key to all group members to calculate their group key. The final group key is computed by all group controllers as discussed above and is distributed to all group members, and then group communication begins. Whenever membership changes the group key must be regenerated. The pseudo code of GC algorithms is given in subsequent section. The basic definitions that are used throughout the paper are given below in the Table No.01.

Symbol	Definition
n	The number of group members
G	A set of current group members
T	A key tree
T*	A modified tree after membership operations
GC_i	An ith group controller; i € [1,n]
M_{SGC}	A subgroup controller
M_D	A leaving member
M_i	An ith group member; i € [1,n]
p,g	A large prime number;
<l,v>	vth node at level l in a tree where $0 \le v \le 2^l-1$
$K_{<l,v>}$	<l,v>th node's random private key
F(x)	g^x mod p
$BK_{<l,v>}$	<l,v>th node's blind (public key)
$T_c(M_i)$	ith group member's response time
S_{GC}	Represents stack of group controllers

Whenever a new group controller is selected based on their performance it is pushed to the top of the stack S_{GC} and all those group members M_i who were compared with that group controller gets registration with that group controller. Every group controller has a set of group members registered with it; it may contain IP address or MAC Address of registered group members as shown in figure no.02. The procedure for generating the tree or updating tree is simple. For tree construction, the group controllers are choose from stack S_{GC} in a popping manner and are placed in the tree from left to right. For balancing the tree an intermediate node is created and all the registered group members for each group controllers are placed under intermediate node from left to right as shown in figure no.02. This approach is very efficient because stack of group controller provides ease in binary tree construction, and this approach is very simple to implement. Let us see how group key is generated, by ignoring group members in group key generation process. The table below shows that only high performance group controllers generate group key (G) and the group members are included in it. However each member M_i computes a secret key $K_{<l,v>}$ as well as blind key $BK_{<l,v>}$ and also response time. The key exchange process using Diffie-Hellman is only performed by group controllers GC_i in order to generate subgroup key. The final group key G is generated by the highest performance group controller that resides on the top of the stack and is distributed to all group members.

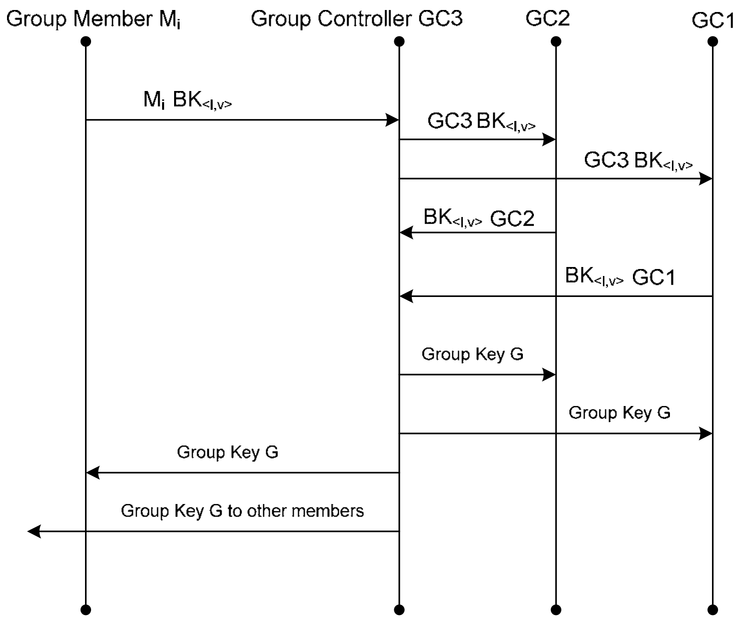


Figure No.02: shows Key Generation Process

The set of group controllers makes clusters of group members, and each cluster is controlled and managed by one group controller. The Figure No. 03 below shows clusters of group members controlled by group controllers.

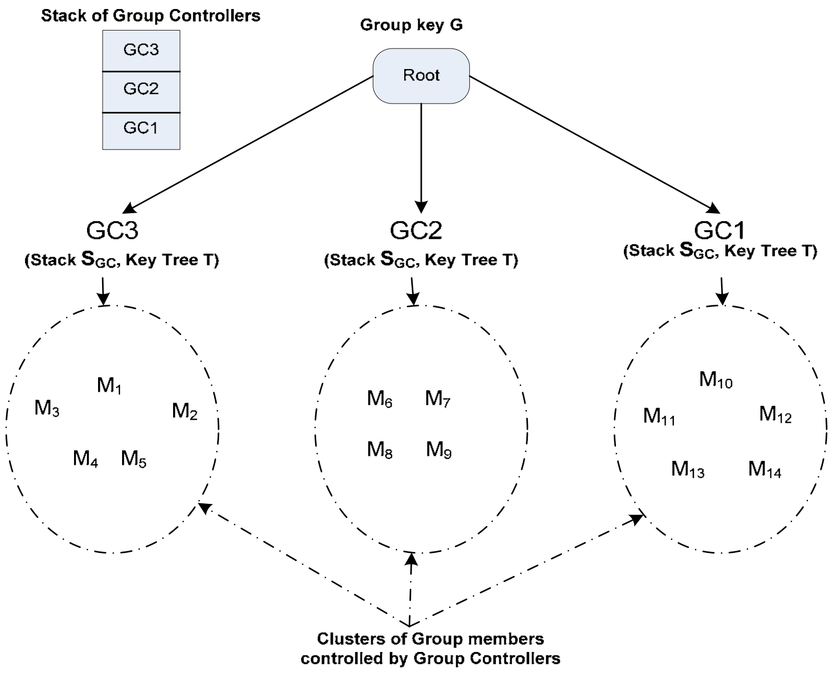

Stack of Group Controllers
GC3
GC2
GC1
Group key G
Root
GC3
(Stack S_{GC}, Key Tree T)
GC2
(Stack S_{GC}, Key Tree T)
GC1
(Stack S_{GC}, Key Tree T)
M_1 M_2 M_3 M_4 M_5
M_6 M_7 M_8 M_9
M_{10} M_{11} M_{12} M_{13} M_{14}
Clusters of Group members controlled by Group Controllers

Figure No. 03 Clusters of group members

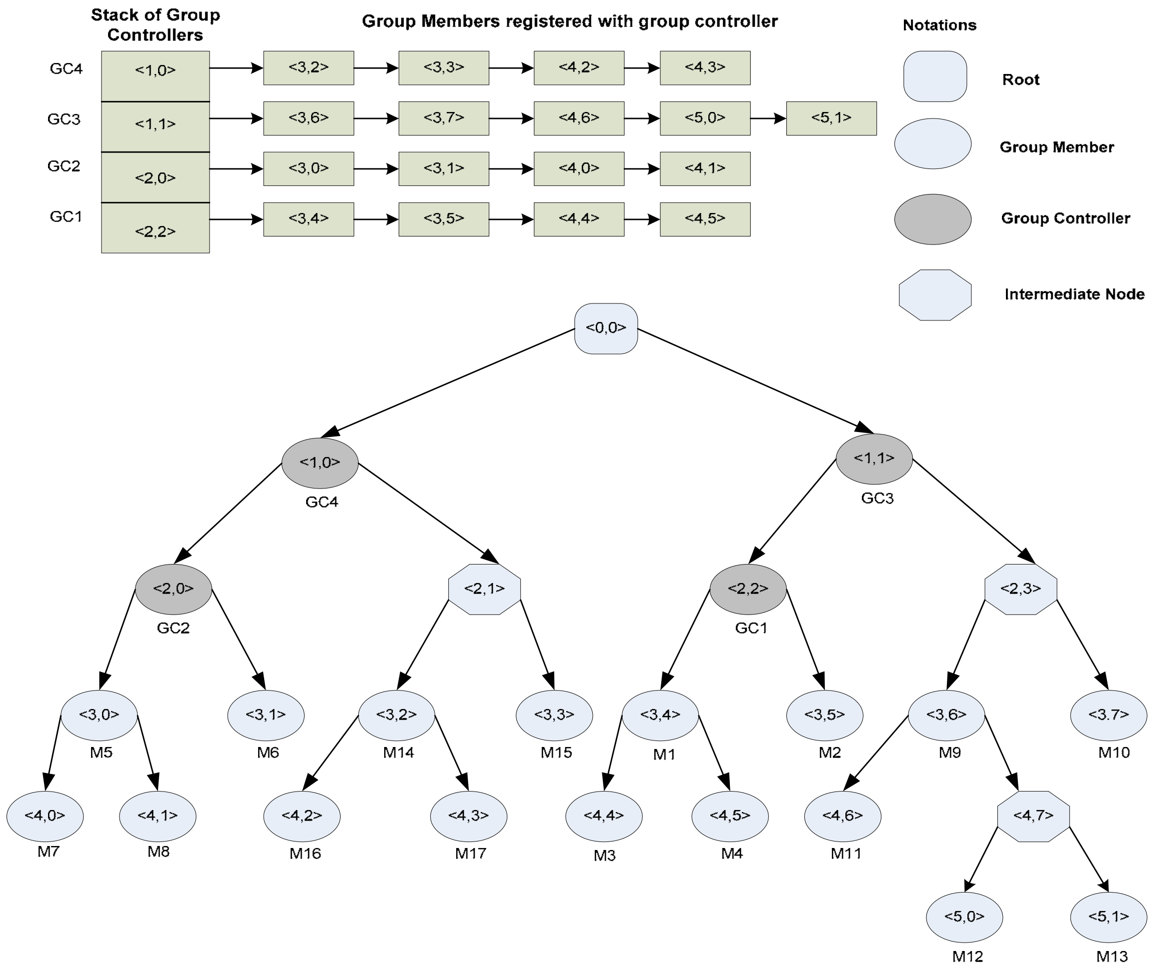


Figure No.04: Shows a stack-based binary key tree.

Figure no.05 shows the general overview of the secure group communication. Here whenever a membership changes, the highest performance member is selected as a group controller. The following flow chart in is identifying the numbers of steps involved in group key generation process and distribution. When a new member joins the group it is checked for whether it qualifies the criteria for to be a group controller. For this reason if it qualifies it takes part in group key generation and distribution process otherwise the existing group controllers does this process.

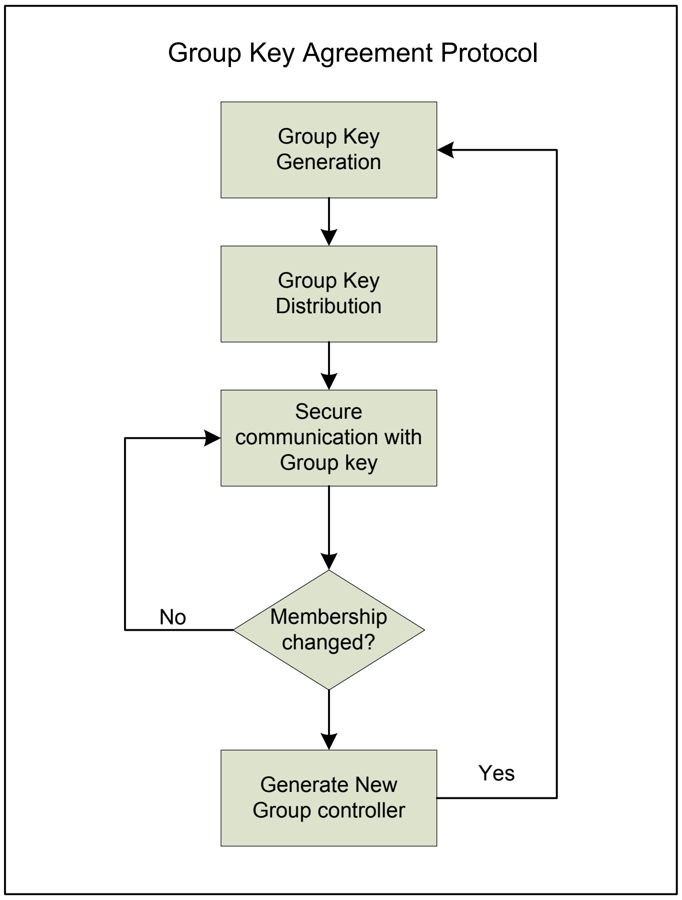

Group Key Agreement Protocol
Group Key Generation
Group Key Distribution
Secure communication with Group key
Membership changed?
No
Generate New Group controller
Yes

Figure No.05: Overview of secure group communication

In this paper we have designed an efficient algorithm for deciding which member must be the group controller based on their performance. The pseudo code for this algorithm is given below:

A Stack Based Group Controller (GC) Algorithm:

```
While limit of maximum group members in not reached that is up to 100 members do
    If a new member ($M_i$) joins the group then:
    Calculate: Its blind key, private key and its Response time $T_c$ ($M_i$) of newly joined member and Response time of Group Controller $T_c$ (GC)
    If response time of group controller $T_c$ (GC) is less than newly joined member $T_c$ ($M_i$)$T_c$ (GC) < Tc ($M_i$) then:
        1. Calculate Key using group key agreement protocol
        2. Distribute the Key using group key agreement protocol to all members.
    Else
```

```
        1. New member ($M_i$) becomes the group controller (GC).
        2. Calculate Key using group key agreement protocol including the new
           joined member.
        3. Distribute the group Key to all members using group key agreement
           protocol.
        4. Push the newly generated group controller (GC) on to the stack.
    End.
End.
```

3.2.1 Membership Operations

In STGDH group key agreement protocol, there are four membership operations:

- Join Operation: when a new member wants to join the group.
- Leave Operation: when an existing member leaves the group.
- Partition Operation: when the group is partitioned into subgroups, the subgroup is split from group communication.
- Merge Operation: A partitioned group is merged with the current group communication.

3.2.1.1 The Join Protocol in STGDH:

Suppose there are n numbers of group members $M_{1,}$ $M_{2,}$ $M_{3,}$ $M_{4,}$ $M_{5,}$ up to M_n where $n <= 100$.each time when a new member joins the group two main processes are performed; one is to decide new group controller and second each group member and group controllers updates its key tree along the path. The key tree generation process is discussed in detail in above section.

Suppose we have key tree as show below in the figure no.04 and new member joins the group. So after new membership the key tree structure will be one of the following on the basis of two cases: 1 if it is selected as a group controller then key tree will be like in the figure no.06 and 2.if it becomes a group member then it is shown in figure no.07

In this scenario of Figure No.06, when a newly joined member becomes a group controller:

1. The top of stack always contains the highest performance group controller, so it will become the left most node in the hierarchy that is new group controller in this case is node <1, 0> and represents group controller GC_3 .

2. Remaining group members and group controller are adjusted according to stack.

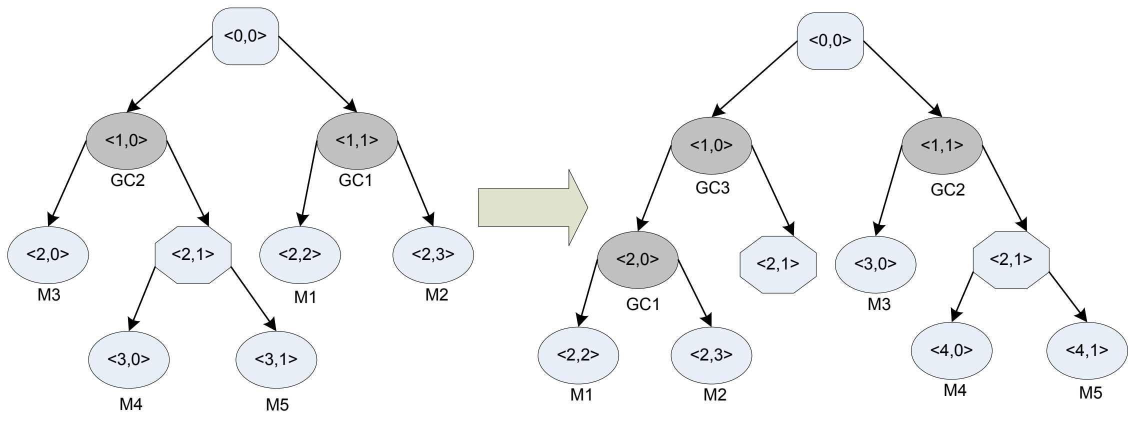


Figure No.06: Case 1: New member is selected as group controller

In this scenario of Figure No.07, when a new member joins the group:

1. Delete the intermediate node <2, 1>
2. Bring node <3, 0> to upper level and make it node <2, 1>
3. Member M_5 becomes the left most child of node <2, 1> that is member M_4
4. New group member becomes the right most member of node <2, 1> that is member M_4

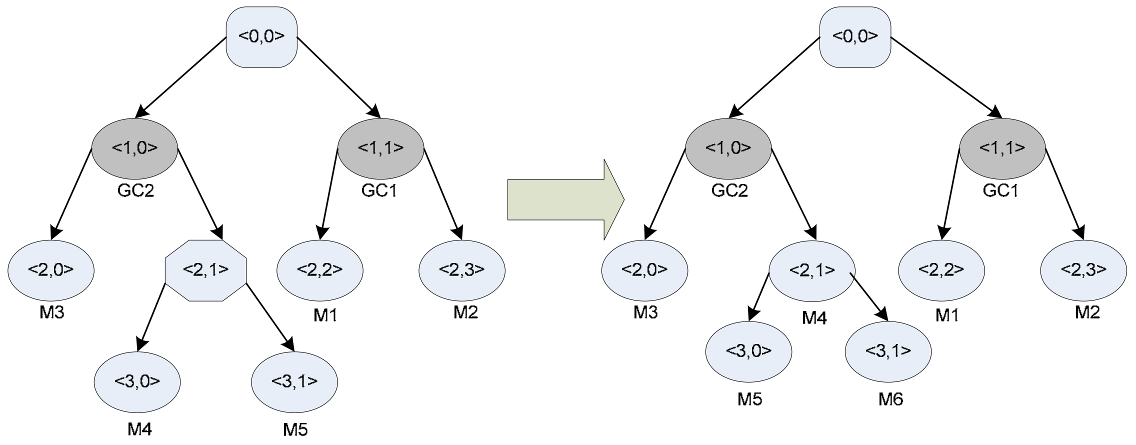


Figure No.07: Case 2: New member is selected as group member

Here we have given join operation algorithm in pseudo code form below, which gives a clear idea about the working of join protocol.

```
While limit of maximum group members in not reached that is up to 100 members Do
Step 1: Each new member sends a request to generate the group key by sending his blind
        key.
          • Newly joined member ($M_i$) sends his blind key to the group controller that
            resides on the top of the stack.
Step 2: The group controller (GC):
          • Generates a private key and a blind key and response time of calculating the
            key as well that is $T_c$ (GC).
Step 3: Execute: the Group Controller (GC) Algorithm.
Step 4: if new member ($M_i$) becomes the group controller then:
          • Push it to the stack $S_{GC}$.
          • Updates the key tree (T*)
          • Broadcasts blind key and key tree (T*) to all group controllers to update its
            key tree and to compute his/her keys.
          • Generate group key (G) and distribute it all group members and secure
            communication begins.
Step 5: Else:
          • New member ($M_i$) gets registration with the group controller by sending his
            blind key.
          • Group controller updates key tree (T*).
          • Broadcast blind key and key tree to all group controllers to compute his/her
            blind key and to update its key tree (T*).
          • Group controller generate group key (G) and distribute it to all group
            members.
          • Secure communication begins.
End.
```

3.2.1.2 The Leave Protocol in STGDH:

When any group member leaves the group, the group key must be re-computed for group secrecy. The leaving member can be a group member or it can be a group controller. In these two cases key tree (T*) is updated differently. If the leaving member is a group member then the key tree after update is given in Figure No.08 as shown below:

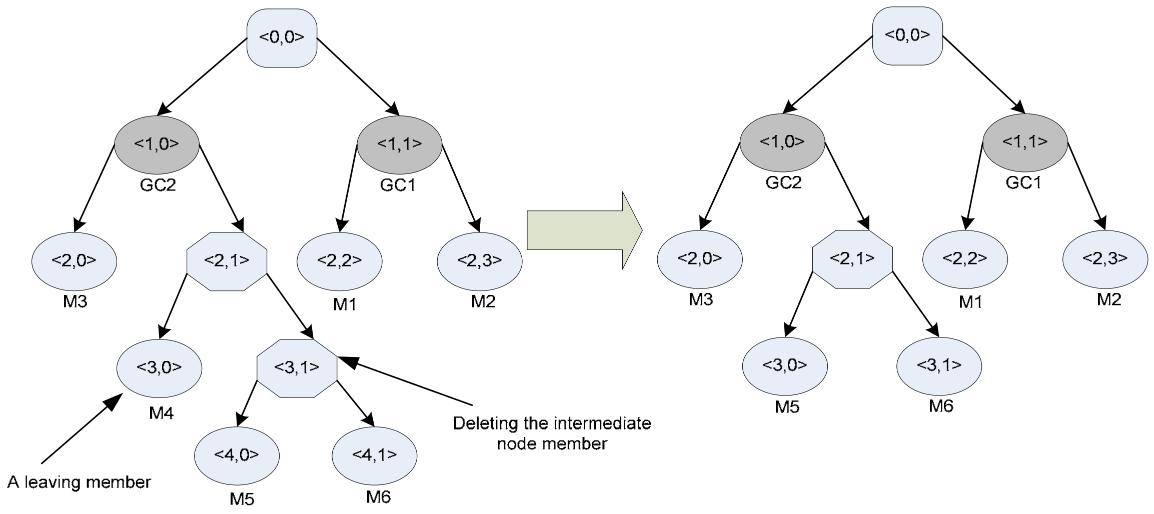


Figure No.08: A key tree (T*) after a group member leaves the group

In this scenario the following steps are performed:

1. Delete node <3, 0> and intermediate node <3, 1>
2. Rename node <4, 0> to <3, 0> and node <3, 1> to <4, 1>
3. Node <2, 1> becomes the parent of nodes <3, 0> and <3, 1>

If the leaving member is a group controller then the updated key tree (T*) is shown in Figure No.09 below:

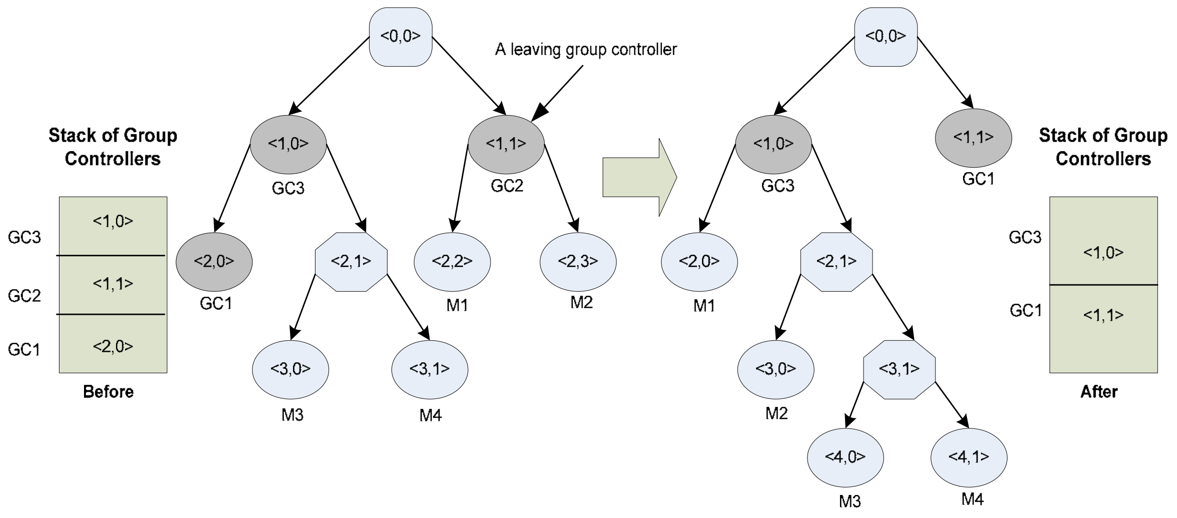


Figure No.09: A key tree (T*) after a group controller leaves the group

The pseudo code for leave protocol is given, which can give a good idea about the working of this protocol is given below:

While limit of maximum group members in not reached that is up to 100 members Do

If the leaving member (M_D) is **Not** a group controller Then:

- Sends a control message to his group controller in order to update the key tree (T*).
- The group controller (GC) then inform all other group controllers to update its key tree (T*).
- All members in the group update its tree and re-compute their keys.
- The group controller (GC) calculates group key (G) and distribute it to all group members.

Else

- The leaving group controller selects group controller from top of the stack (S_{GC}) and sends a control message.
- The selected group controller update its key tree (T*) using stack (S_{GC}) while ignoring the leaving group controller.
- All the group members registered with leaving group controller are now registered with the selected group controller.
- All group controllers update key tree (T*) and stack.

- All group members update key tree (T*) and computes keys.
- The final group key is generated by group controllers and is distributed to all group members.

End

3.2.1.3 The Partition Protocol in STGDH:

The group is divided in to several subgroups when a network problem is detected. So when there is a network problem, subgroups independently communicates by choosing a group controller for that subgroup and generates group key for secure communication. In this approach, if the partitioned members contain group controller then it becomes the subgroup controller of the subgroup and generates group key for that subgroup for secure communication. On the other hand if the subgroup doesn't contain any group controller then on the basis of response time for computing the group key, a new subgroup controller is chosen. In the following Figure No.10 we have just shown the first case when the partitioned members contain a group controller.

Suppose group controller $GC_{1,}$ group members M1 and M2 are partitioned from group controller $GC_{2,}$ and M3, M4, M5, and M6. The group controllers for each subgroup generates the key tree (T*) and computes group key G and distribute it to all group members. In our approach, no group member participates in group key generation process except when the member joins the group.

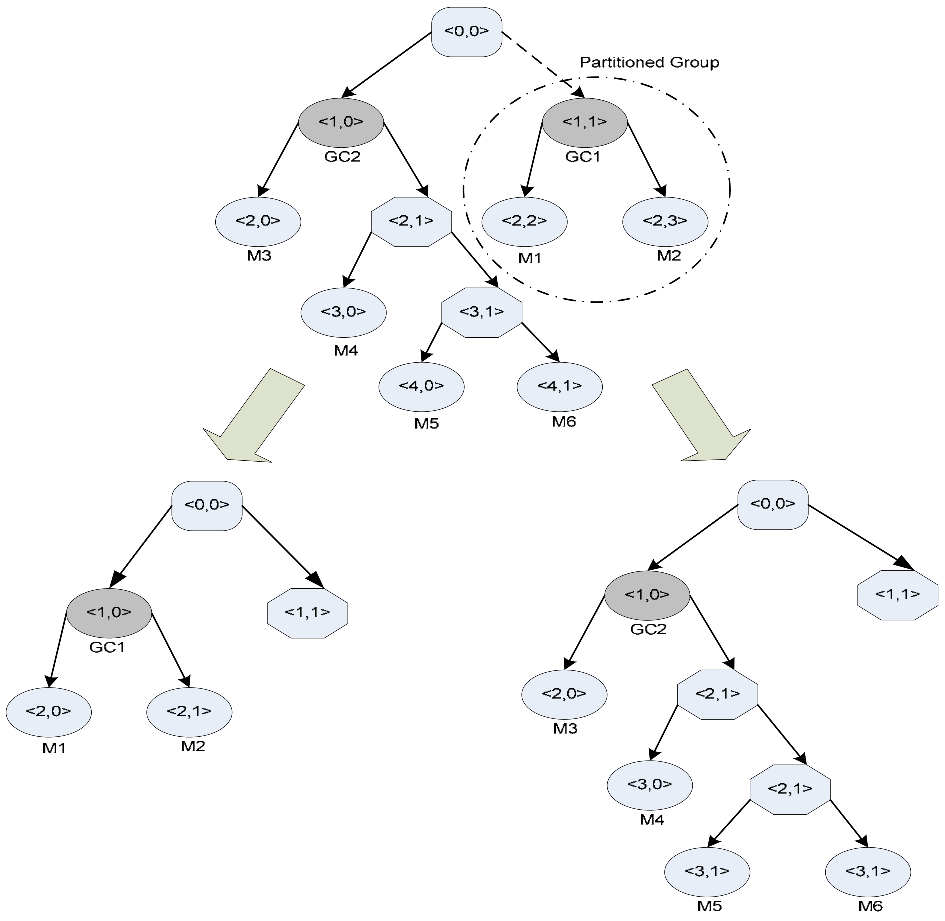


Figure No.10 shows the partition operation

Step 1: Every group controller (GC) in the sub key tree:

- Updates the sub key tree (T*) by removing all leaving member nodes and their parent nodes.
- Updates the stack if the leaving members contain the group controller.
- Calculate blind key, private key and update its key tree (T*).

Step 2:

- All group controllers generate a group key (G).
- Distribute group key (G) and updated tree (T*) to all group members and secure group communication begins.

3.2.1.4 The Merge Protocol in STGDH:

When a network recovers from fault, the subgroups can be merged into original group. The subgroup controller communicates with the group controllers of the group by sending his blind key. The group controllers update stack by pushing the subgroup's group controller on to the stack, key tree (T*) and compute a group key for the updated group. The group controllers then broadcast group key to all group members.

Suppose we have two subgroups as shown in Figure No.11, are merged in to one group. The tree is generated from the stack as it contains the set of group controllers; we have now two group controllers in stack. And each group controller has a set of group members. Group controller GC1 has two group members M3 and M4 and GC2 has two group members M1and M2.

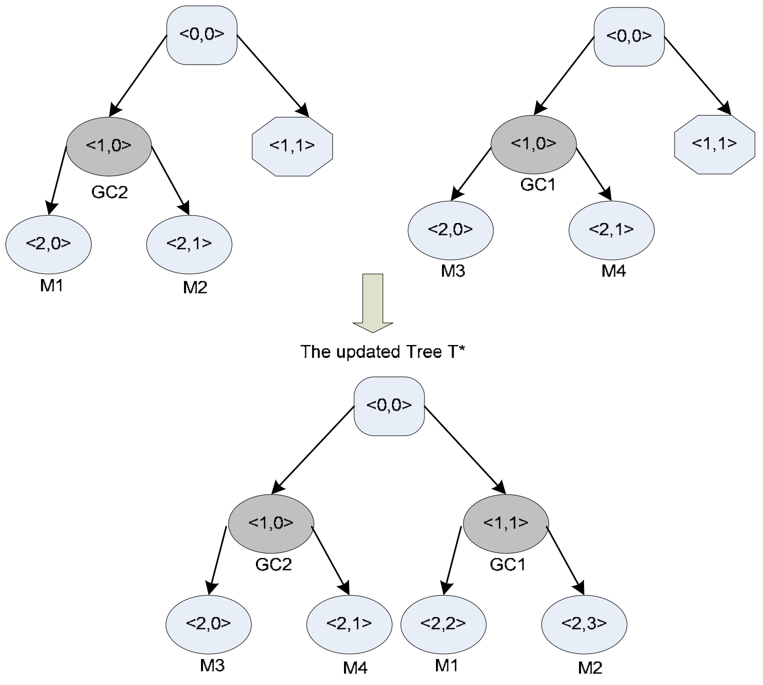


Figure No.11: shows a Merge operation

The pseudo code for merge operation is given below:

Step 1: The subgroup controller (M_{SGC}):

- Generates his blind key and requests group controllers for merger by sending his blind key to all group controllers.

Step 2: Each group Controller (GC):

- The group controllers updates stack and updates key tree (T*).
- Calculates his private key and blind key.
- Sends their blind key to the highest performance group controller (GC) that resides on the top of the stack.

Step 3:

- Generates Group key G and distribute it to all group controllers.
- The group controllers then send the group key G to the registered group members and secure communication begins.

3.3 Summary

Most of the group key agreement protocols lack in one of the membership operation but STGDH provides efficient algorithms for each membership operation which are easy to manage and implement. The two main issues that are faced by most of the group key agreement protocols are: efficient group key computation and efficiently maintaining the key tree after each membership operation. ETGDH is efficient in terms of computing the group key but it stills has the problem of key tree maintenance. STGDH solves both of these problems. For efficient group key generation only group controllers compute the final group key, and the group controllers are the highest performance group members. In our approach GC algorithm selects the highest performance member. For second problem of maintaining the key tree, we use stack of group controllers and again the key tree is managed by only group controllers. The procedure for key tree generation is very simple and efficient. After the group key, key tree and stack is generated by the group controllers, the group key is forwarded to group members. Each group controller has a set of group members registered with it and it makes clusters of group members as discussed above. The experimental results of STGDH are impressive and in next chapter its performance is evaluated.

CHAPTER 4

Experimental Results

4.1 Introduction

The overall performance for each membership operation can be analyzed by the total response time involved in computational and communication overhead for generating the group key. This chapter addresses the complexity analysis and experimental results of communication and computation costs to generate a group key.

4.2 Performance Evaluation

The simulation of this protocol has been done in Visual studio.NET 2005. The simulation has the ability to test this protocol for a LAN/WAN of 100 machines at least. In this simulation we have assumed different machines that could be PC, laptop, mobile devices and we computed the performance of these machines based on the following factors:

- CPU Response time
- Data storage (RAM size and HDD size) for maintaining stack of group controllers and Key tree.
- Communication cost (The number of uni-cast and multi-cast messages)
- Current traffic load.
- Number of allowable/possible group members registered with group controller.
- Maximum distance of a group member from the group controller.

In this section, we have compared our approach with other group key agreement protocols. Communication and computation costs for each membership operations (join, leave, merge, and partition) are analyzed in terms of T_c. The communication and computation costs represent the response times involving T_c for generating a group key. The communication costs are involved in the number of messages, network delays, and managing the key generation structures. On the other hand, the computation costs depend on the hardware capacity such as CPU type, clock pulse, main memory size, hard disk size, etc.

Other variables to be considered that would affect the performance of each machine include number of rounds, the total number of control messages, network overhead, and group Key Generation costs. The proposed protocols were compared to other group key agreement protocols including STR, GDH, TGDH, and ETGDH. The number of current groups, merging groups, leaving members, partitions and number of group controllers are denoted by *n*, *m*, *k* (*m* ≥ *k)*, *p* and *t* respectively. The height of the key tree constructed by the TGDH protocol is *h*. Table No.02 shows the communication and computation costs of these five protocols.

Protocol		**Communication Cost**		**Computation Cost**
		Rounds	**Messages**	**Exponentiations**
STGDH	**Join**	2	3t	Height of t
	Leave	1	2t	Height of t
	Partition	1	2t	Height of t /p
	Merge	2	2t	Height of t
	Group Controller Leave	1	2t	Height of t
ETGDH	**Join**	2	2n-2	3h/2
	Leave	1	2n-2	3h/2
	Partition	1	2n-2	3h
	Merge	2	2n-2	3h/2
TGDH	**Join**	2	3	3h - 3
	Leave	1	1	3h - 3
	Partition	min(log2 p, h)	2p	3h - 3
	Merge	$\log_2 k + 1$	2k	3h - 3
GDH	**Join**	4	n + 3	n + 3
	Leave	1	1	n – 1
	Partition	1	1	n - p
	Merge	m + 1	n + 2m + 1	n + 2m + 1
STR	**Join**	2	3	4
	Leave	1	1	3n / 2 +2
	Partition	1	1	3n / 2 +2
	Merge	2	k + 1	3m + 1

Table No.02 Communication and Computation Costs Summary

4.3 Comparison

Now let’s see how the proposed approach is compared with other group key agreement protocols for each membership operation.

4.3.1 Join Operation

The simulation results of our join protocol are given in two different graphs with group members of size n =25, 50 both of the graphs provide computational complexity of O ($\log_2$ t) where t represents the number of group controllers as shown in Figure No.12. The best thing in our approach is that if the number of group controllers increases the computational complexity decreases and as a result more efficient results are produced. This is the first approach in which with the increase of group size the computational complexity increases in a very small amount. In comparison with other approaches, GDH IS inefficient, because the join operation presents an increasing overhead when the group size increases. ETGDH, TGDH and STR are efficient comparatively because they use divide and conquer algorithm to compute the group key but the most efficient protocol is STGDH because at each join operation it chooses the highest performance member as a group controller. The top of the stack contains the highest performance group controller that is why group key is generated more efficiently.

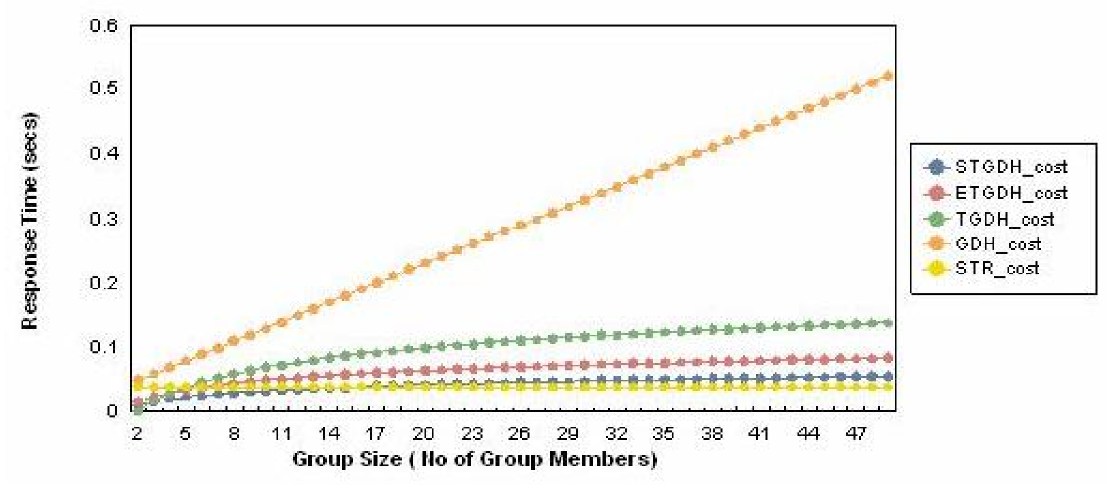


Figure No.12 Join protocol: Group size vs. Response time

4.3.2 Leave Protocol:

In our approach, Leave protocol has two different cases: if leaving member is a group member (M_i) then the results of join and leave protocol are almost the same, but in case when the leaving member is a group controller (GC), the simulation results are very well because the leaving group controller hands over his registered group members to the highest performance group controller on the top of the stack as shown below in Figure No.13. The computation cost for leave event in STR depends on the location of leaving member; therefore efficiency of leave cost is better than as compared to the join cost. The leave cost for GDH is linear, and leave cost of GDH is highest of all others. As ETGDH and TGDH uses divide and conquer algorithm so the efficiency of leave event is better but the best efficiency is achieved in STGDH because only the highest performance group controllers regenerates group key.

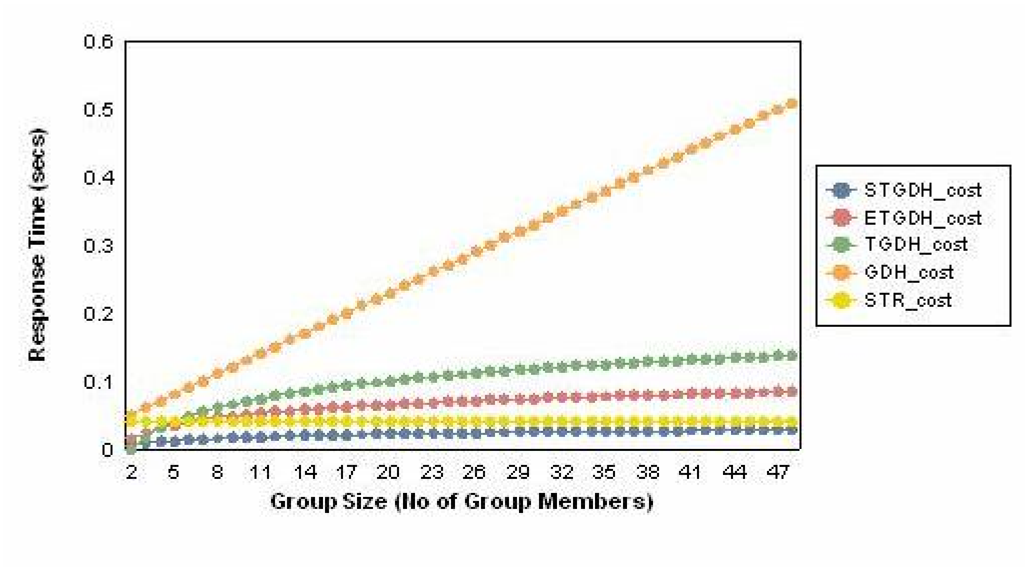
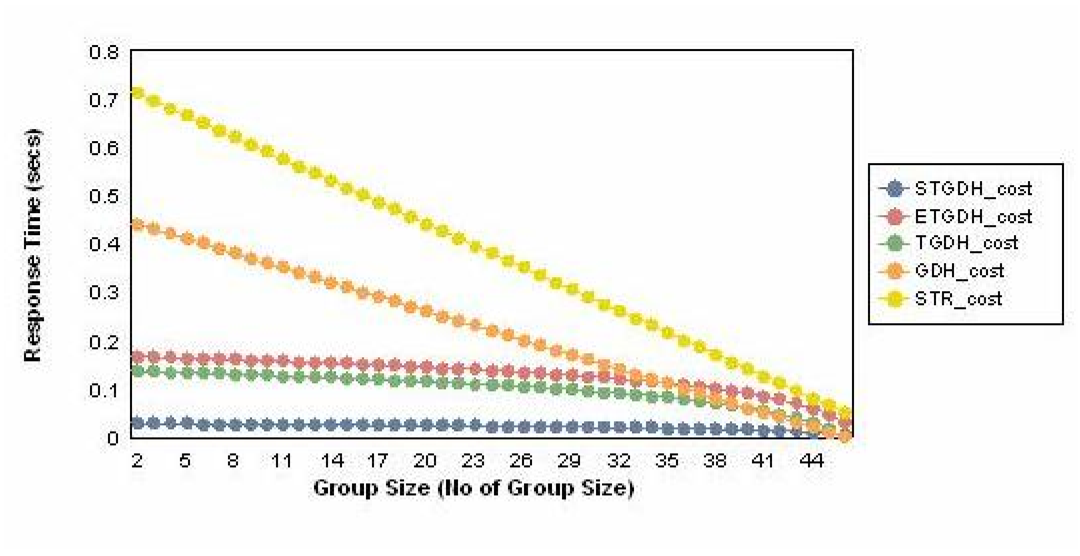


Figure No.13 Leave protocol: Group size vs. Response time

4.3.3 Partition Protocol:

The group is divided into subgroups whenever a network problem occurs. In our approach the partition operation has no effect on the performance because group key is generated by group controllers. The group key generation process for the remaining group members is almost constant, however in TGDH and ETGDH takes some time for managing the tree structure after that it begins to decrease. On the other hand, the cost of STR and GDH decreases linearly as show in Figure No. 14.



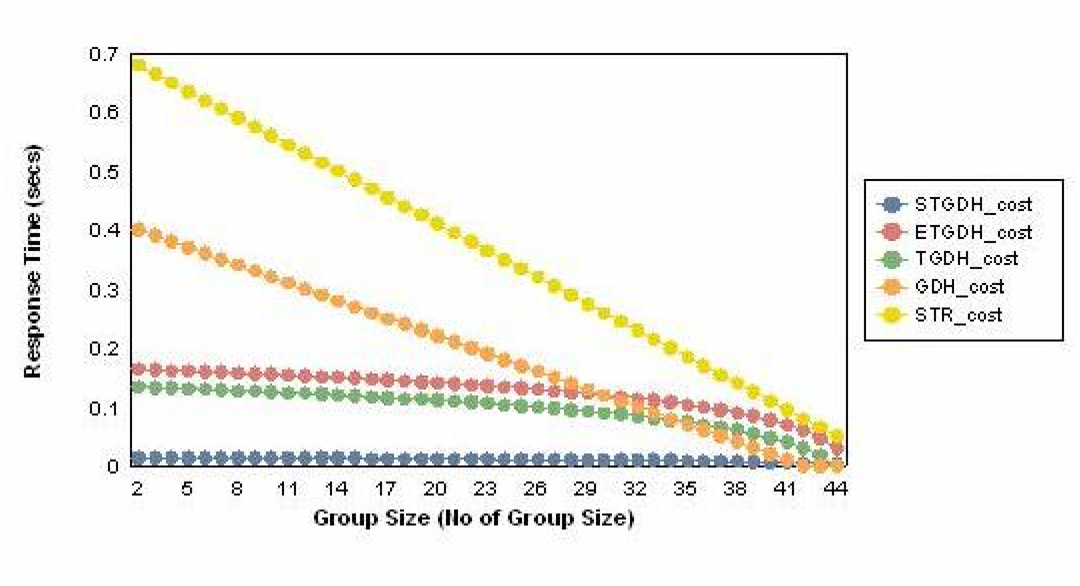


Figure No.14: Partition Protocol

4.3.4 Merge Protocol:

If the merging members are large the cost of STGDH increases accordingly and same is the case in ETGH, TGDH. The cost of GDH is increasing linearly, and it provides the worst performance for merge operation. STR takes constant time for merge operation; the group size does not affect the performance of STR for merge operation. Figure No. 15 shows the results of all protocols for merge operation.

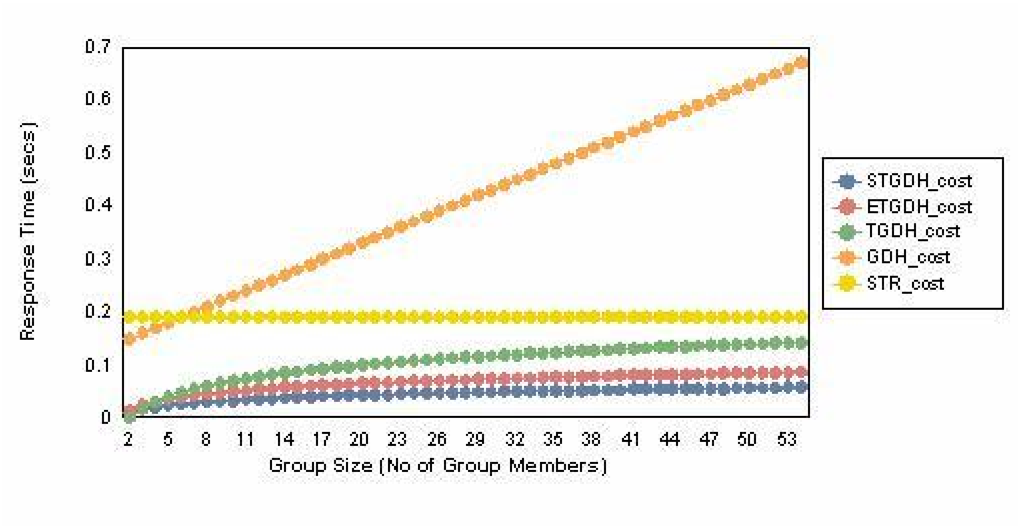


Figure No.15: Merge Protocol

4.4 Conclusion & Future Work

For efficient and secure group communication, the group key generation process must be reduced to maximum. Up to now the group key agreement protocols have emphasis on the computational and communication cost for all membership operations but most of them lacks in the ability of key tree maintenance and efficient group key generation process. STGDH protocol provides a best way to efficiently maintain a key tree, in this approach the group members do not participate in group key generation process and they don’t need to worry about key tree maintenance. STGDH always chooses the best performance member as a group controller. The performance of any machine is checked on the basis of some factors like CPU speed, data storage capacity, communication speed, current traffic load etc. The set of group controllers resides in the stack and performs the whole key generation process; these group controllers communicate with each other, maintain a stack of group controllers, a key tree and computes group key for communication. Each group controller has a set of registered group members and makes a cluster of group controller and group members as discussed earlier. Each cluster is monitored and controlled by one group controller. The group controller sends key tree along with the group key to all group members within the cluster. The experimental results of this protocol are far better than other approaches so far. STGDH performs with an efficiency of O ($\log_2 t$) which is very efficient as compared to other approaches. The only drawback of this protocol is that that it requires more data storage capacity.

In future, we need to design an efficient and secure group key agreement protocol for large networks because as the size of the network increases the computational complexity increases and key tree maintenance becomes difficult. Also we need to design an efficient group key agreement protocol to work in limited resources like mobile devices have limited computational power and limited memory.

CHAPTER 5

Simulation Tutorial

5.1 Introduction

The simulation of this protocol was performed in Visual studio.NET 2005 and a supporting tool of SQL Server for keeping important data in a database. This simulation can be tested for as many machines as you want. For using this simulation, it is very simple it just follows the steps of algorithm the way we have designed. In this chapter we have shown the number of steps involved for each membership operation. The general GUI of this simulation of is given below:

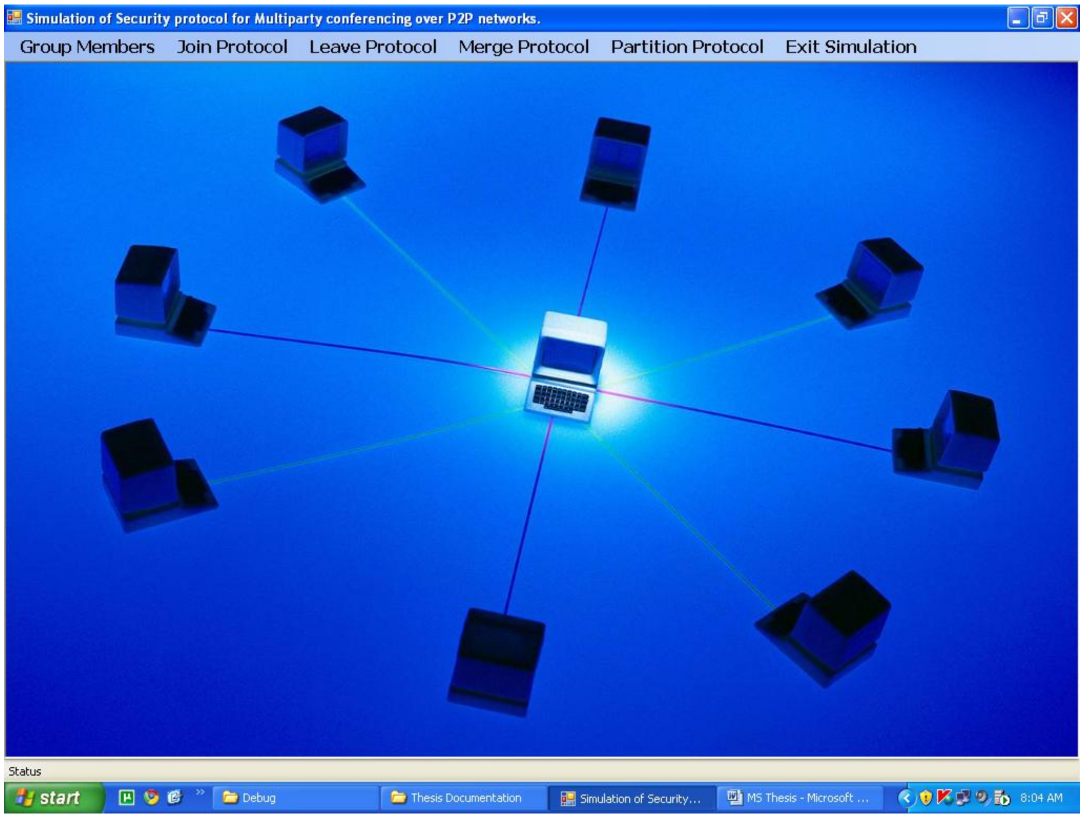
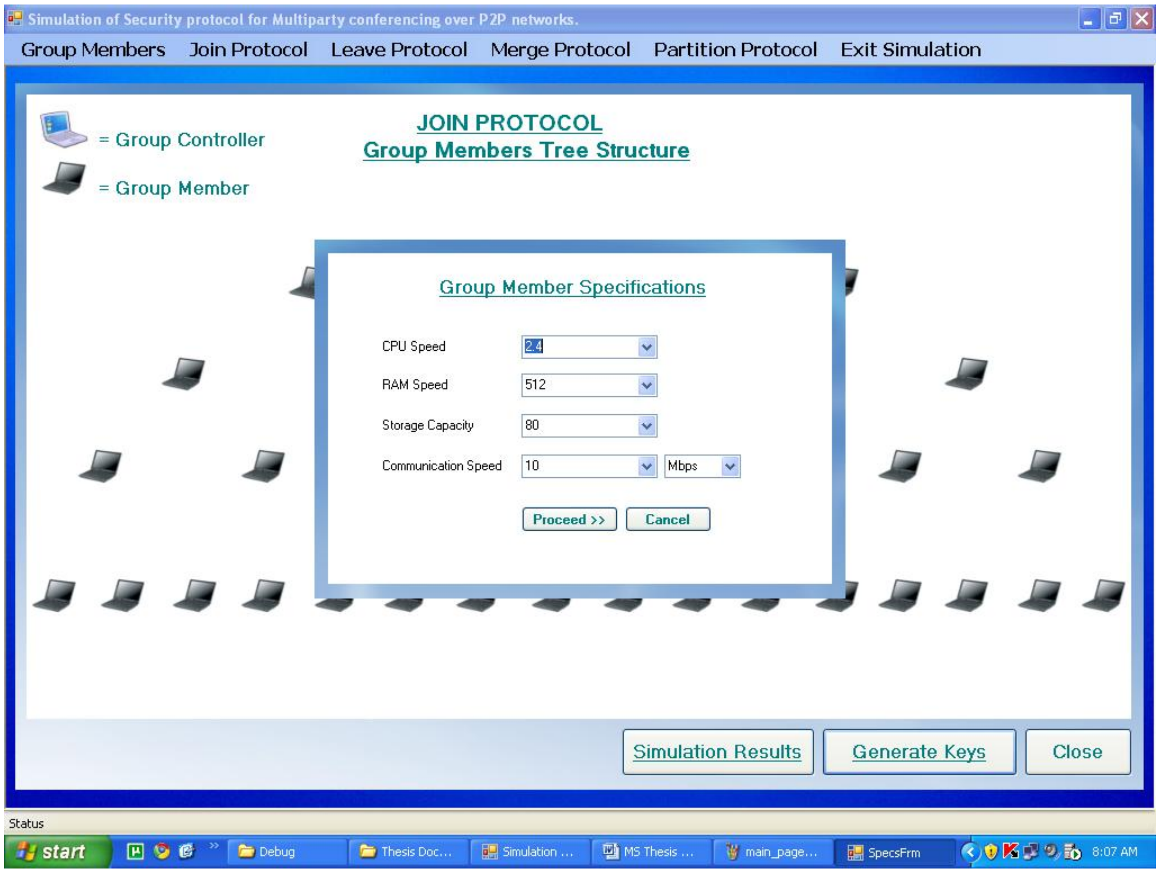


Figure No.16 General Overview of Simulation GUI

5.2 Join Operation

The general overview of this simulation shows that there are four main tabs representing each membership protocol (join, leave, partition and merge). Let's see the number of steps involved in join operation.



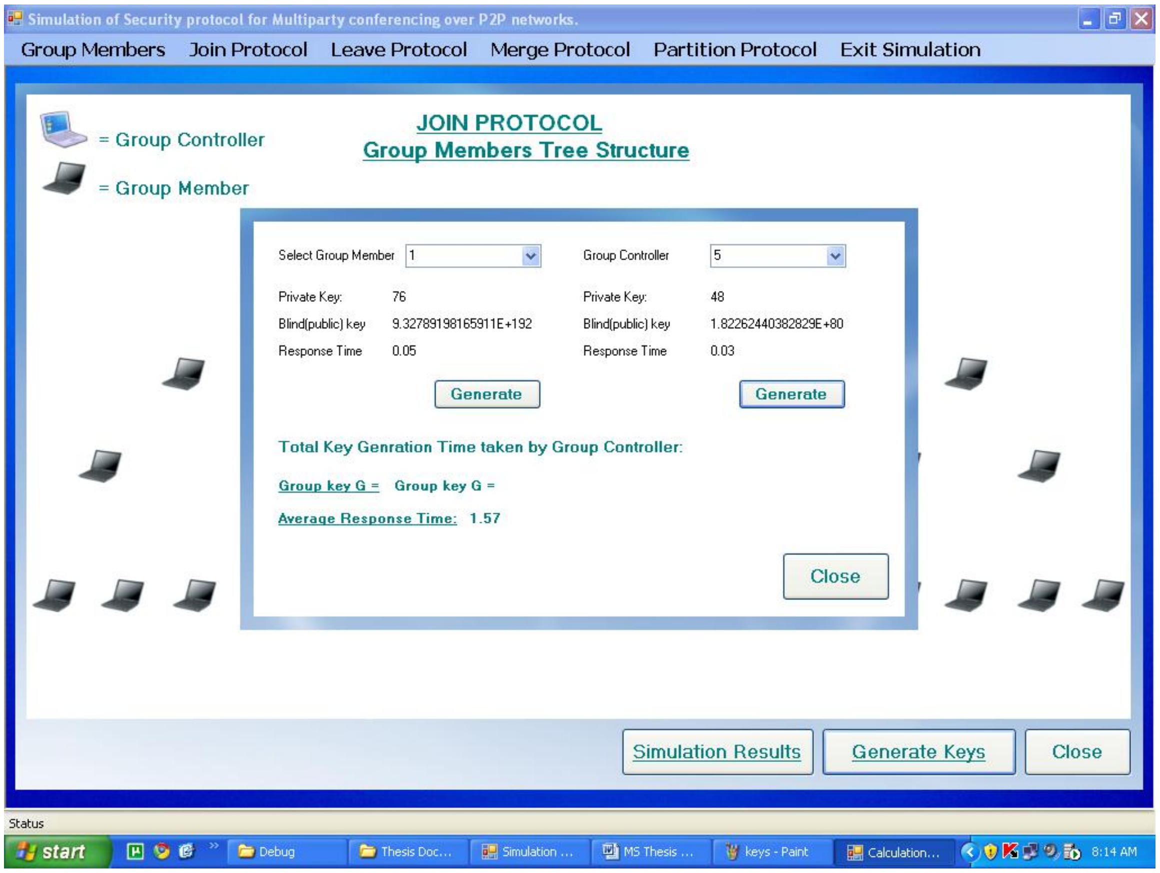
CPU Speed	2.4
RAM Speed	512
Storage Capacity	80
Communication Speed	10 Mbps

Figure No.17 Join Operation

Whenever a member wants to join the group, the following steps are performed:

1. Click on "Join protocol" tab, it displays the GUI as shown in Figure No.16.
2. In this GUI, there are 30 machines have been shown, on clicking any machine you first need to select member's specifications. The required specifications have been discussed in fourth chapter in detail

3. After choosing member's specs, the "proceed button" it brings you to next step of join operation. Actually our GC algorithm runs behind this button, on the member's specs it is compared with the current group controller.
4. Figure No.17 shows that how keys (private and blind) are generated for each member's join operation.



Select Group Member	1	Group Controller	5
Private Key:	76	Private Key:	48
Blind(public) key	9.32789198165911E+192	Blind(public) key	1.82262440382829E+80
Response Time	0.05	Response Time	0.03

Total Key Genration Time taken by Group Controller:

Group key G = Group key G =

Average Response Time: 1.57

Figure No.18 Group key generation

5. Figure No.17 Shows that we can generated keys for group controller. In the left side of this GUI for group controller 5, we have calculated private key, blind key and response time.
6. The "Generate button" done most of the operation, all group controllers calculates their own keys and at the end group key G is generated and is forwarded to group all group members registered with it.

7. The “Simulation Results button” compares it with other group key agreement protocols as shown in Figure No.18.

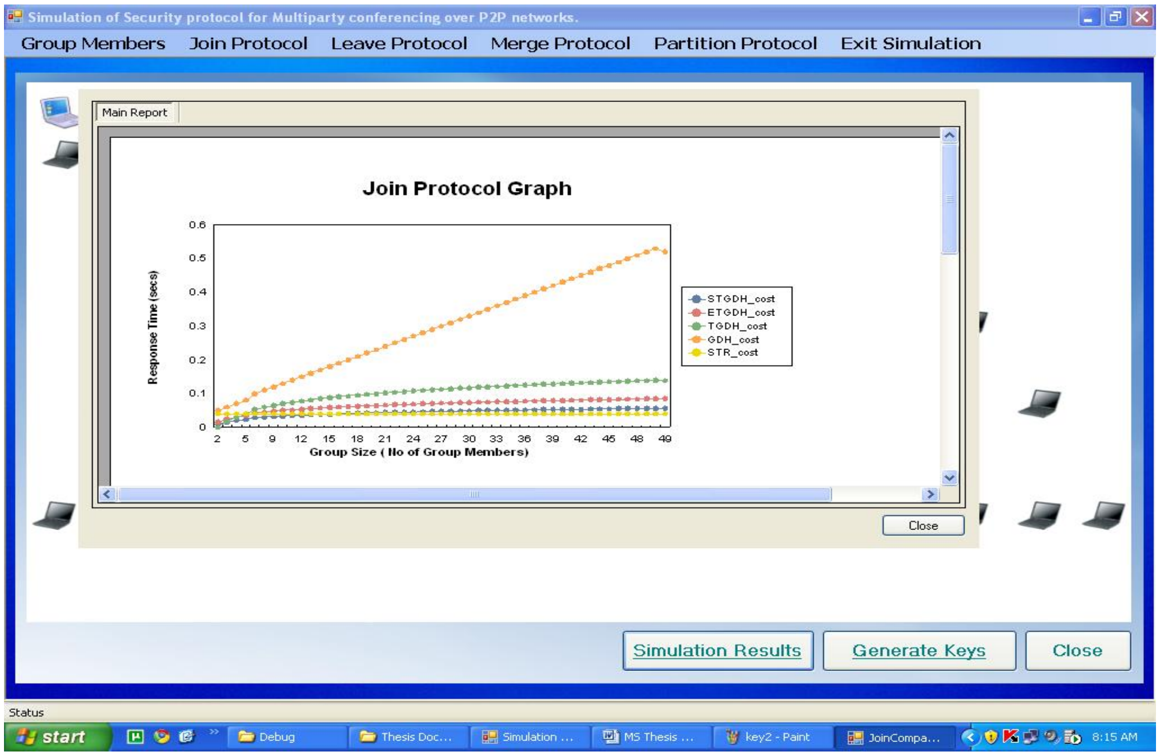


Figure No.19 Join Results

5.3 Leave Operation

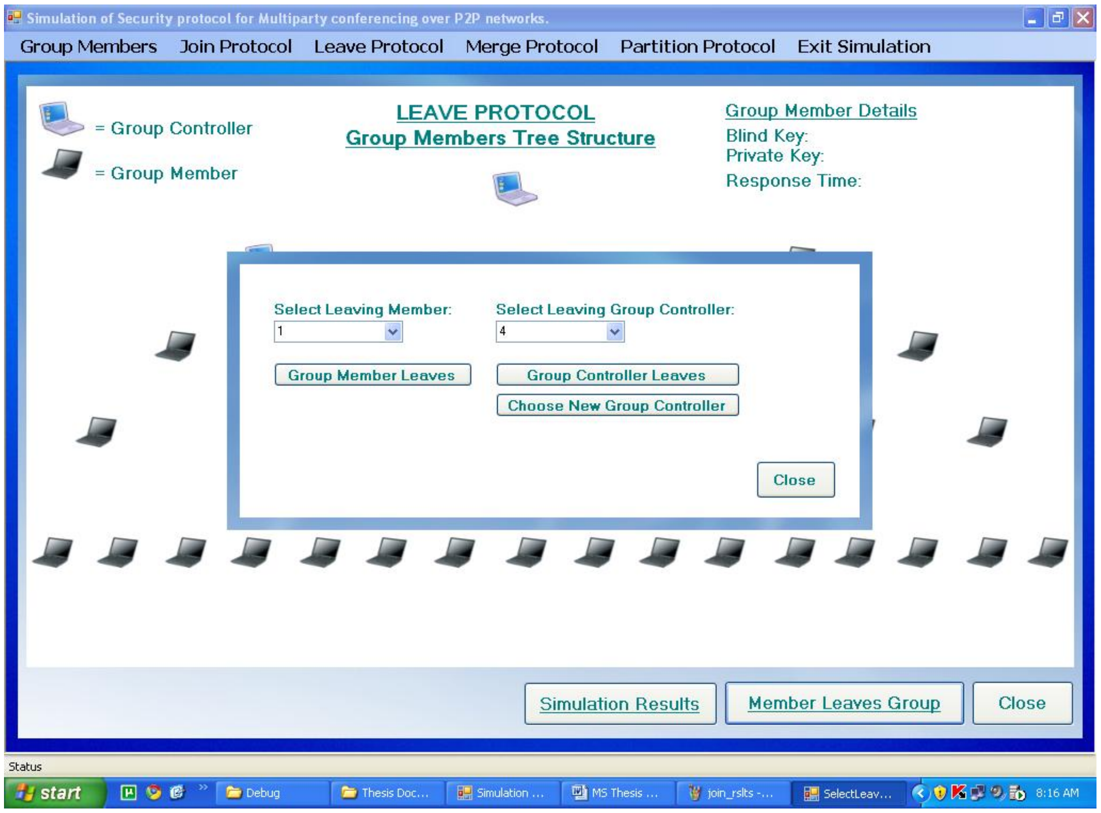


Figure No.20 Leave Operation

1. There are two options for leave operation, GUI in Figure No.19 shows that we can select a group member as a leaving member or we can select a group controller as a leaving member.
2. The leave algorithm runs behind two buttons that is “Group controller leaves button” and “Choose new group controller button”.
3. The “Simulation Results button” compares these results with other group key agreement protocols as shown in Figure No.20.

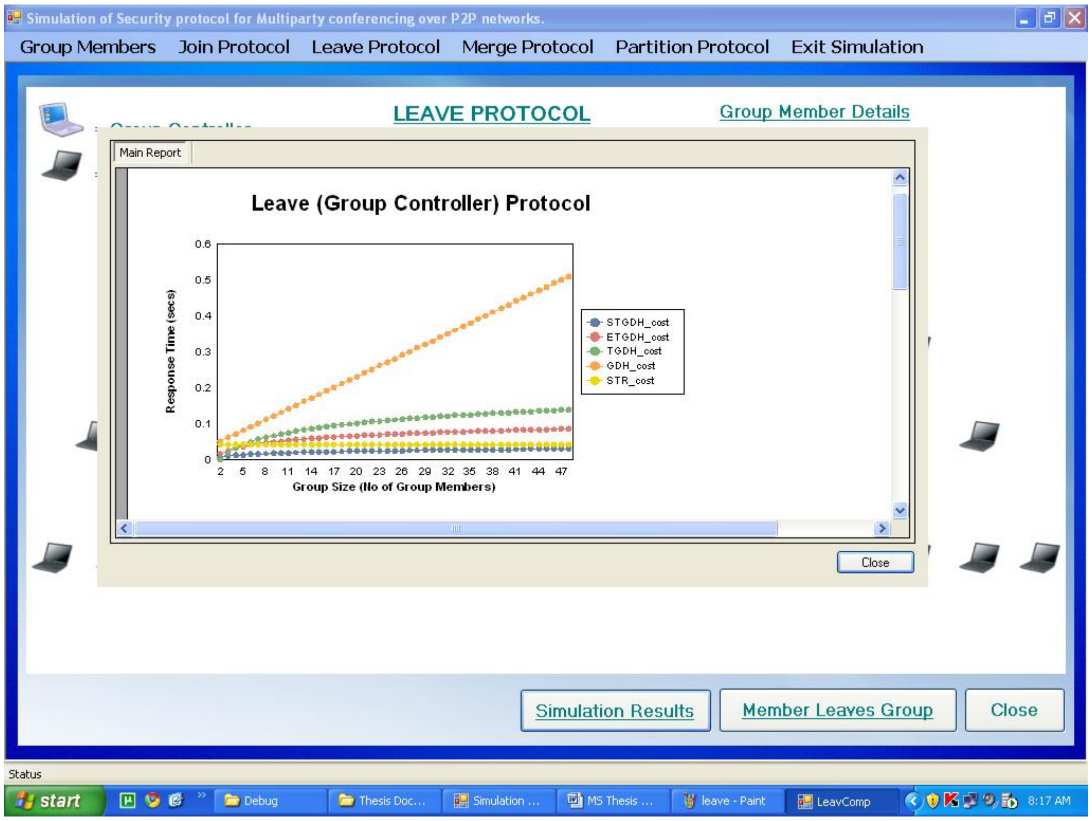


Figure No.21 Leave Operation Results

5.4 Partition Operation

1. In Figure No.21, we first choose the number of partitioned members and "Calculate results button" runs the partition algorithm.
2. "Simulation Results button" shows the results of partition operation as shown in Figure No.22.

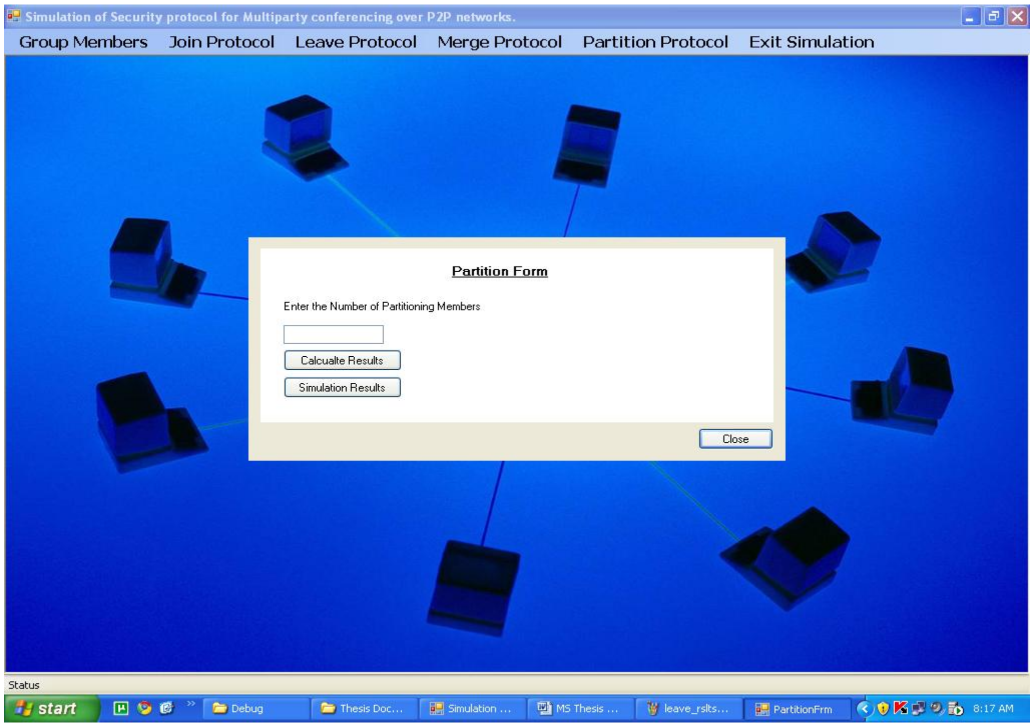


Figure No.22: Partition Operation

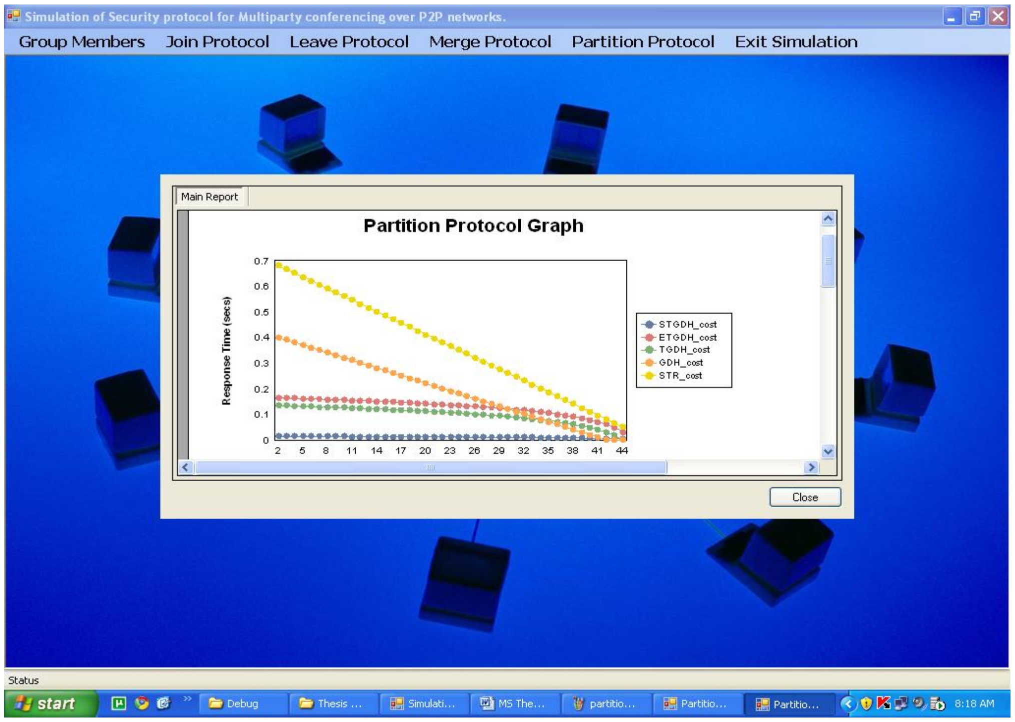


Figure No.23: Partition Operation Results

5.5 Merge Operation

1. First enter the number of merging groups, to be merged with actual group as shown in Figure No.23.
2. By clicking on “Calculates results button” runs merge algorithm.
3. By clicking on “Simulation Results button”, the results of merge operation are compared with other group key agreement protocols as shown in Figure No.24.

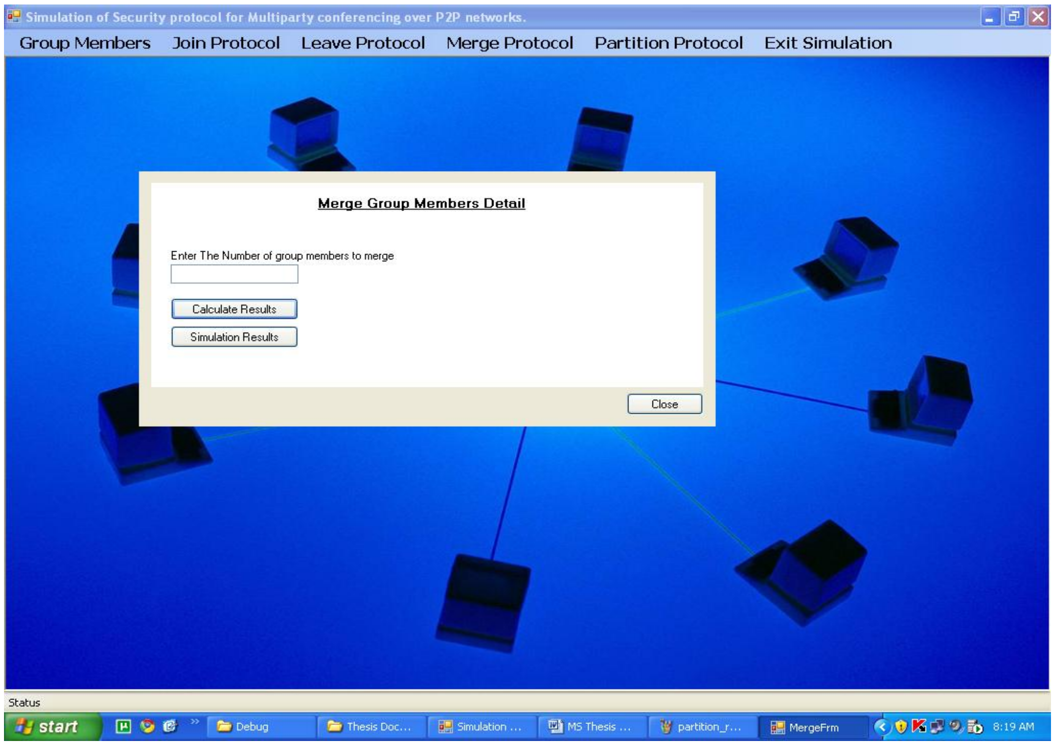


Figure No.24: Merge Operation

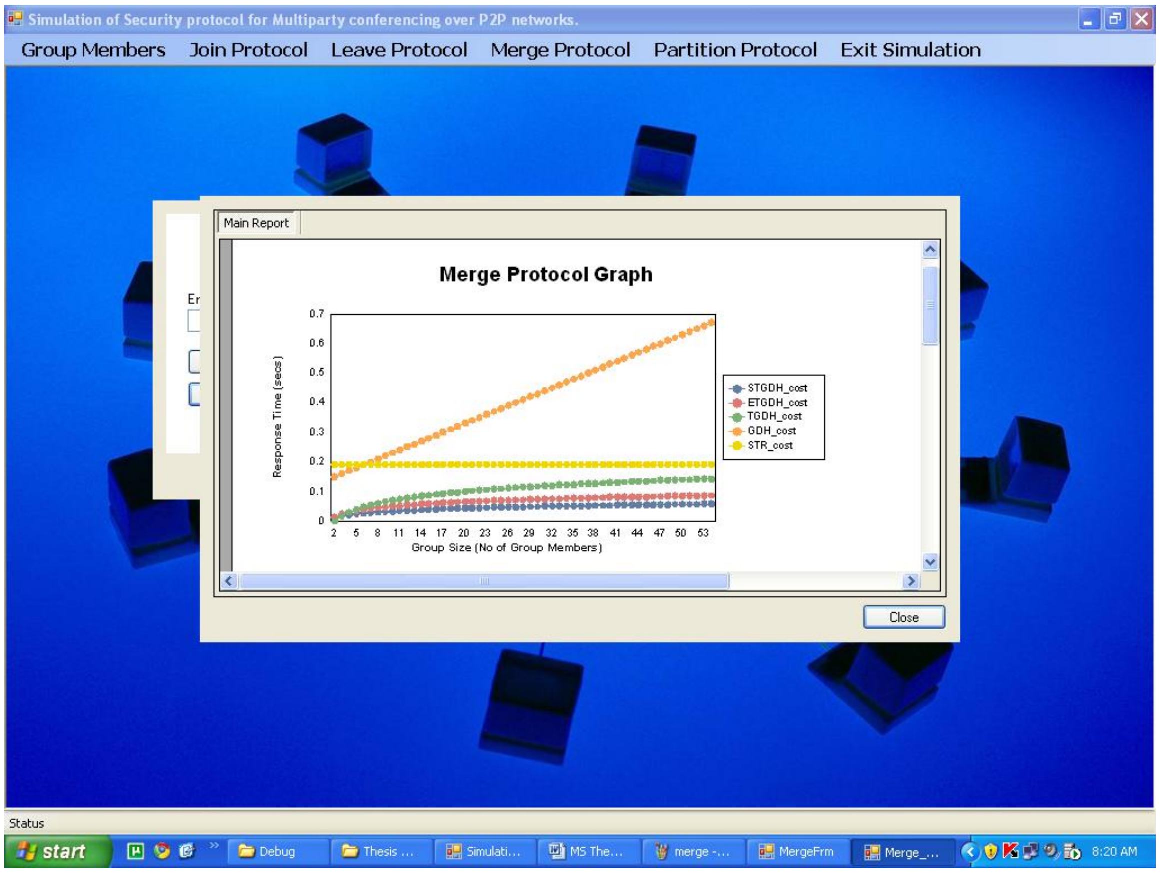


Figure No.25: Merge Operation Results

REFERENCES

[1] S. Hong,”Secure and Efficient Tree-based Group Diffie-Hellman Protocol”, April 2009.

[2] J.Li on”A survey of Peer-to-Peer Network Security Issues” 2008.

[3] A. Friedman on “Peer-to-Peer Security”.

[4]”Peer-to-Peer Network Protocols” Position Paper VERSION 1.0 April 2007.

[5] N.Jasapara, on”Group Key Agreement Protocols for Dynamic Peer Groups”.

[6] F.Otto, D.Patrick Mirembe on”A model for data management in Peer-to-Peer systems”.

[7] H.Harney, C. Muckenhirn, SPARTA, inc, July, 1997 on ”RFC2093 - Group Key Management Protocol (GKMP) Specification”

[8] Y.Kim, A.Perrig, G.Tsudik on”Communication-efficient group key agreement”

[9] F.Liu and H.Koenig, Brandenburg University of Technology Cottbus, Department of Computer Science on”A Secure P2P Video Conference System for Enterprise Environments”

[10] B.Eun Jung IEEE COMMUNICATIONS LETTERS, VOL. 10, NO. 2, FEBRUARY 2006 on”An Efficient Group Key Agreement Protocol”

[11] M. Steiner, G. Tsudik, and M. Waidner, Key agreement in dynamic peer groups, IEEE Trans. Parallel Distrib. Syst., vol. 11, pp. 769-780, Aug. 2000.

[12] K. Y. Choi, J. Y. Hwang, and D. H. Lee, Efficient ID-based group key agreement with bilinear maps, PKC04, Lecture Notes in Computer Science, vol.2947, 2004.

[13] F. Zhang and X. Chen, Attack on two ID-based authenticated group key agreement schemes from PKC 2004, In proc:Information Processing Lett., vol. 91, pp. 191-193, Aug. 2004.

[14] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik. On the Performance of Group Key Agreement Protocols. IEEE ICDCS’2002, July 2002.

[15] M. Burmester and Y. Desmedt, A secure and efficient conference key distribution system, Advances in Cryptology EUROCRYPT94, May 1994.

[16] Y.Kim, A.Perrig, G.Tsudik: Group Key Agreement Efficient in Communication. IEEE Trans. Computers 53(7): 905-921 (2004)
[17] Key distribution protocol for digital mobile communication systems by M.Tatebayashi, N.Matsuzaki, Matsushita Electric Industrial Co Ltd Japan and D.B. Newman, Jr, the Jeorge Washington University, Washington DC.
[18] www.wikipedia.org