

# **Distributed group key management using hierarchical approach with Diffie-hellam and symmetric algorithm**

Project Report submitted in partial fulfillment of the requirement for the degree of  
Bachelor of Technology.

in

**Computer Science & Engineering**

under the Supervision of

***Prof Satya Parkash Gharera***

By

***Veenu aggarwal***

**111313**

to



Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

# **CERTIFICATE**

This is to certify that project report entitled “Distributed Group Key Management using Hierarchical approach with Diffie Hellman and Symmetric Algorithm ”, submitted by Veenu Aggarwal in fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Prof Satya Parkash Ghrera  
Head Dept. of CSE

## ACKNOWLEDGEMENT

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to my final year project supervisor Prof. Dr. S.P.Ghrera whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project .

Furthermore I would also like to acknowledge with much appreciation the crucial role of the staff of Jaypee university of information technology, who helped me to assemble the parts and gave suggestion about the task "*Distributed group key management using hierarchical approach with diffie-hellman and symmetric algorithm*". Last but not least, I have to appreciate the guidance given by other supervisor as well as the panels especially in our project presentation that has improved our presentation skills thanks to their comment and advices.

Date:

Name of t he student : Veenu Aggarwal

# TABLE OF CONTENT

<b>TITLE</b>	<b>PAGE NO</b>
Certificate	ii
Acknowledgement	iii
List of Figures	v
List of Tables	vi
Abstract	vii
Introduction	1
1.1 Group communication protocols	2
1.1.1 Centralised protocols	2
1.1.2 Decentralised protocols	2
1.1.3 Distributed protocols	2
The purpose of crptography	5
Literature review	7
3.1 Centralised group key management	8
3.2 Decentralised group key management	8
3.3 Contributory group key agreement	9
3.4 Diffie hellamn algorithm	11
3.5 Symmetric algorithm	13
3.6 Asymmetric algorithm	13
Design principle	14
4.1 Logging into the system	18
4.2 Logging out from system	19
4.3 Use case diagram for login and logout	20
System requirements	21
5.1 Software requirements	21
5.2 Hardware requirements	21
Final design	22
6.1 Join	22
6.2 Leave	24
6.3 Simulation	26
6.3.1 First user join the group	26
6.3.2 Second user join the group	26
6.3.3 Third user joins the group	27
6.3.4 Fourth user joins the group	27
Real time implementation of distributed group key management	28

7.1join operation to login	28
7.2Leave operation to logout	29
7.3Simulation	30
7.3.1Html page to login	30
7.3.2First user login	30
7.3.3first user has been inserted into the tree	31
7.3.4Second user login	32
7.3.5Second user has been inserted into the tree	32
7.3.6If user enters wrong password or username	32
7.3.7User logout	32
Security requirements	34
Applications of group key management	35
9.1 mobile and ad hoc network	35
9.2multimedia security	36
9.3 wired networks	37
Conclusion	38
Appendix	40

## **LIST OF FIGURES**

<b>TITLE</b>	<b>PAGE NO</b>
2.1 DGKD join operation	6
2.2 DGKD leave operation	6
2.3 EDKAS join operation	7
2.4 EDKAS leave operation	7
3.1 DHSA parent binary code for 3.5 Diffie hellman	9 12
Member position discovery	
3.2 Flow chart for leave operation	10
3.3 Flow chart for join operation	11
4.1 DHSA join operation	14
4.2 DHSA leave operation	15
4.3 USE CASE diagram	20
LOGGING into system	18
Flowchart for logging into system	18
Flowchart for logging out system	19

## **LIST OF TABLES**

<b>TITLE</b>	<b>PAGE NO</b>
3.1 List of parent binary code and Associated public key	9
3.2 List of parent binary code and Associated public key	14
3.3List of parent binary code and Associated public key	16

## ABSTRACT

Secure and reliable group communication is an active area of research. Its popularity is caused by the growing importance of group-oriented and collaborative applications. Various network applications require sending data to one or many members within the network, maintaining security in the large groups is among the biggest obstacles for controlling access. Unfortunately, IP multicast does not provide any security over the group communication. Group key management is a fundamental mechanism for secured multicast and unicast. The most widely used technique in a network is group communication. This helps in the reduction of the bandwidth usage. The major concern in group communication is message security. Group key provides security of messages and hence proper group key management is a necessity in a group communication. While centralized methods are often appropriate for key distribution in large multicast-style groups, many collaborative group settings require distributed key agreement techniques. Ensuring secure communication in an ad hoc network is extremely challenging because of the dynamic nature of the network and the lack of centralized management. For this reason, key management is particularly difficult to implement in dynamic networks. Group key management is a fundamental building block for secure group communication systems. We will present an efficient many-to-many group key management protocol in distributed group communication. In this protocol, group members are managed in the hierarchical manner logically. Two kinds of keys are used, asymmetric and symmetric keys. The leaf nodes in the key tree are the asymmetric keys and all the intermediate node keys are symmetric keys assigned to each intermediate node and uses a simple rekeying procedure which is suitable for large and dynamic networks. For asymmetric key, a more efficient key agreement will be introduced. To calculate intermediate node keys, members use codes assigned to each intermediate node key tree. Group members calculate intermediate their own node keys rather than distributed by a sponsor member. The features of this approach are that, no keys are exchanged between existing members at join, and only one key, the group key, broadcasted to remaining members at leave. This work investigates a novel group key agreement approach which blends so-called key trees with Diffie-Hellman key exchange.



# 1.0 INTRODUCTION

The most widely used technique in a network is group communication . With rapid growth of wireless networks, the usage of group communication has become more popular from both application and academic points of view. These applications include private video conferencing , distributed interactive simulation, multi-partner military action, wireless sensor, , paying TV,transmission of audio and video, updating and downloading softwares, video games , mobiles and ad hoc networks,checking accounts online. Security , bandwidth management ,speed etc are the various concerns on group communication.If the communication between members is properly designed and managed, then it will lead to the effective usage of band width. Recently the focus is mainly on the security issues involved in the group communication. When the group uses the unicast communication, one sender is sending the data stream onto one group member. In multicasting, the group member is sending the data onto other group members. The most critical problem that has to be addressed in any group communication is the security of its messages .Group key management is the most important among all its security problems However , security and scalability are two important factors that need to be considered.

Multicast is an efficient technology that supports group communication. It helps in better utilization of network resources. Group key needs to be shared among all the members, to ensure security in group communication and also it needs to be maintained secure and fresh. All the group members should participate in the secure distribution, creation and revocation of the keys. This helps to ensure that only authorized users have group key. The communication session in group key management is managed by two entities: Group Controller (GC), responsible for key generation, distribution and rekeying for membership change and Key Server (KS), responsible for maintaining the keys and distributing the keys .The group key is renewed all the time as In the group communication, the members in the group are not fixed, members can join / members can leave the group. So we need to secure the sending message to be received by the group members at that instance. Every messages has to be encrypted with group key before transmitting. Thus outsiders or intruders are unable to interpret the messages even though they receive the encrypted message.

In any practical application, the network has to be scalable and dynamic. Frequent membership changes might exist in such networks. With every membership change, key management operation has to be performed to ensure security. Key distribution is required in secure multicast to ensure that only the current group members can send encrypted multicast datagram, and decrypt the received multicast datagram. In other words, the key distribution algorithm must ensure that an entity is only allowed to participate during those periods when it is authorized or allowed to do so. It means the securing multicast communication must provide the following features :

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

**Forward secrecy :** An entity should not be allowed to read multicast communication after it leaves the multicast group.

**Backward secrecy :** An entity should not be allowed to read multicast communication messages exchanged prior to the time when it joins the multicast group.

One of the solutions to support such requirements is the group data encryption with group key. A group key is the key that is shared by all the existing group members. In order to ensure security requirements, the group key must be renewed on each membership change and redistributed securely to only valid members. This process is called group re-keying or re-keying in short.

## **1.1 Group communication protocols are classified into three main categories:**

Group key management (GKM) is one of the most viable issues in secure group communication (SGC). The existing GKM protocols fall into three typical classes: centralized group key distribution (CGKD), decentralized group key management (DGKM), and distributed/contributory group key agreement (CGKA). major problems lies within these protocols, as they require central trusted entities such as group controller or subgroup controllers, relaying of messages (by subgroup controllers), member synchronization (for multiple round stepwise key agreement),

thus suffering from the single point of failure as if ceteral manager fails in the group the entire group destroys and attack, performance bottleneck, or misoperations in the situation of transmission delay or network failure. We proposed a new class of GKM protocols: distributed group key distribution (DGKD) . The new DGKD protocol solves the above problems and surpasses the existing GKM protocols in terms of simplicity, efficiency, scalability, and robustness.

**1.1.1 Centralized (one-to-many) protocols:** A key server is only responsible for managing and distributing group key to group members .

**1.1.2 Decentralized protocols:** The group is divided into multiple domains. Each domain is managed by a domain controller or subcontroller which is responsible for generating the keys for that domain .

**1.1.3 Distributed (many-to-many) protocols:** Instead of key server, group members collaborate with each other to establish the group key . The responsibility of the members is equal.

Based on application requirements, one of the above protocols is used. In distributed applications such as wireless sensor and ad hoc networks, there is no infrastructure or base station. To ensure group confidentiality, group members need to collaborate to share the group key securely on each membership change. This task involves high overhead. The main goal of secure distributed group communication is how to share a group key securely and efficiently in the group.

Several approaches have been proposed to reduce the overhead of secure group communication in distributed environment . Most of these approaches are based on different types of n-party Diffie-Hellman key agreement protocol. The main issue with such approaches is that the group members need synchronization to form parental keys from their two child keys. Moreover, the cost of modular exponentiation is higher than any other approaches. Once the calculation of a member is slow, the key agreement process gets delayed. Although the other approaches introduce different concepts of key management, the re-keying overhead has not been decreased largely.

The project will be based on efficient many-to-many group key management protocol in distributed group communication. In this protocol, group members are managed in the hierarchical manner logically. Two kinds of keys are used, asymmetric and symmetric

keys. The leaf nodes of logical tree are the asymmetric keys of the corresponding group members and all the intermediate node keys are symmetric keys assigned to each intermediate node. For asymmetric key, Diffie-Hellman key agreement is introduced which provides help in authentication. To calculate intermediate node keys, members use codes assigned to each intermediate node key tree. Group members calculate intermediate node keys rather than distributed by a sponsor member. The features of this approach are that, no keys are exchanged between existing members at join, and only one key, the group key, is delivered to remaining members at

## 2.0 THE PURPOSE OF CRYPTOGRAPHY

Cryptography is the science of writing in secret code and is an ancient art; the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphs in an inscription. Some experts argue that cryptography appeared spontaneously sometime after writing was invented, with applications ranging from diplomatic missives to war-time battle plans. It is no surprise, then, that new forms of cryptography came soon after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium, which includes just about *any* network, particularly the Internet.

Within the context of any application-to-application communication, there are some specific security requirements, including:

- **Authentication:** The process of proving one's identity. (The primary forms of host-to-host authentication on the Internet today are name-based or address-based, both of which are notoriously weak.)
- **Privacy/confidentiality:** Ensuring that no one can read the message except the intended receiver.
- **Integrity:** Assuring the receiver that the received message has not been altered in any way from the original.
- **Non-repudiation:** A mechanism to prove that the sender really sent this message.

Cryptography, then, not only protects data from theft or alteration, but can also be used for user authentication. There are, in general, three types of cryptographic schemes typically used to accomplish these goals:

1. Secret key (or symmetric) cryptography
2. Public-key (or asymmetric) cryptography

### 3. Hash functions

In all cases, the initial unencrypted data is referred to as *plaintext*. It is encrypted into *ciphertext*, which will in turn (usually) be decrypted into usable plaintext.

### 3.0 LITERATURE REVIEW

Consider a group of people who wish to establish secure ad-hoc communication using their mobile devices. The group is called dynamic if it allows to add and delete participants during the communication session, otherwise the group is static. We call an ad-hoc group heterogeneous if its members are equipped with different kinds of devices, and homogeneous if devices have similar performance properties.

In hierarchical approaches, the members of group are mapped with the leaves of a logical binary key tree. Each member maintains all the keys along the path from his/her leaf to the root. The root key is the group key. At join/leave, all the keys in the path set need to be changed to new ones. Based on the key management approach, the number of key generations, key encryptions, and key delivery differs. Typically, each member maintains  $O(\log n)$  keys which shows the height of the key tree where  $n$  is the number of members.

In order to establish a group key for secure distributed group communication, many approaches have been proposed. As stated before, most of these approaches are based on different types of n-party Diffie-Hellman key agreement protocol. The other approaches are based on other methods. The main purpose of these approaches is to reduce the overhead of group key management. The fact with all of them is that the evaluation measures of these approaches are not distinct. For example, they do not consider key generation, key encryption separately in their works.

With increasing communication services, users are often grouped in various applications. They normally form centralized or decentralized structures, capable of handling entities involved in functions ranging from web and mail to sensor networks, file sharing to databases, and so on. Applications of such groups are enormous, and so are the demands for secure and reliable communication in these groups.

The literature presents with several different approaches to group key management. We can divide them into three main groups in terms of efficiency in communication and computation, and scalability:

### **3.1 CENTRALISED GROUP KEY MANAGEMENT**

In CGKD schemes [1], [2], [3], [4], [5], [6], [7], [8], [9], there is a central trusted authority (called group controller (GC)) that is responsible for generating and distributing the group key i.e rekeying operations. Whenever a new member joins or an existing member leaves, the GC generates a new group key and distributes the new key to the group. A single entity is responsible for controlling the whole group, hence a group key management protocol seeks to minimize storage requirements, computational power on both client and server sides, and bandwidth utilization; The problems with the centralized schemes are the central point of failure, performance bottleneck, non-scalability, and the requirement of trustworthiness of the group controller by all members.

### **3.2 DECENTRALISED GROUP KEY MANAGEMENT:**

In DGKM schemes [10], [11], [12], [13], the group is divided into multiple distinct subgroups and every subgroup has a subgroup controller (SC) responsible for key management for its subgroup, trying to minimize the problem of clubbing the work in a single place. In addition, an SC has the key of its parental subgroup. When an SC receives a message from one subgroup, it decrypts the message, encrypts the message with the key of the other subgroup and sends to the other subgroup, i.e., relay of the message occurs. The problems with DGKM are that SCs can still be considered as central and trusted entities (at a smaller scale) and the messages undergo multiple relaying before they reach the entire group. Relaying of every data message puts huge burden on SCs.

### **3.3 CONTRIBUTORY GROUP KEY AGREEMENT:**

Contributory group key agreement (CGKA) protocols, originally designed for peer groups in local- and wide-area wired networks, can also be used in ad-hoc scenarios because of the similar security requirements and trust relationship between participants that excludes any trusted central authority (e.g., a group manager) from the computation of the group key. In CGKA schemes [14], [14], [15], [16], [17],[18], the group key is generated/agreed up by uniform contributions from all group members. These kind of schemes assume equality and uniform work load among all group members. There is no explicit KDC, and the members themselves do the key generation .All members can perform access control and the generation of the key can be either contributory, meaning that all members contribute some part of information to generate the group key, or done



by single member. They are generally executed in multiple rounds and require strict synchronization. The CGKA protocols are primarily different variations of the n-party Diffie-Hellman key agreement/exchange [14], [16], [19], [20], [17], [18]. The main problem with using this key exchange mechanism is that the group members need synchronization to iteratively form parental keys from their two children's keys. Once one member is slow or one rekeying packet is delayed, the key agreement process will be postponed or even can misoperate. Moreover, there are dependences among nodes' keys (i.e., a blinded node key is dependent on the secret node key and a parental key on its two child's keys).

TGDH [15] uses two-party Diffie-Hellman key agreement protocol in group. In addition, The concept of hierarchical key tree is used in this method. The leaves of the key tree represent users. Each member assigned to the leaf of the key tree maintains the key tree. Starting from leaf nodes, each intermediate node represents a key shared by its two child node keys computed by using single Diffie-Hellman key agreement protocol. TGDH has reduced the modular exponentiation from  $O(n)$  to  $O(\log n)$  during initial establishment and join/leave. However, the cost of modular exponentiation causes the protocol to delay because the protocol needs initiation after each membership change.

DGKD [16] ,uses the concept of sponsor and co-distributor in this method. This method is based on hierarchical tree structure. A sponsor initiates the key generation and rekeying process at join/leave. The sponsor is chosen based on the ID size. The selected sponsor is responsible to change the keys along the path, and co-distributor distributes new key . A co-distributor is the sponsor of a branch on other path. Since this is distributed method all the group members are equally capable and are mutually trusted. Depending on the relative location of joining/leaving member, any group member can have the potential sponsor. The disadvantage of this approach is that all affected intermediate keys in the path have to be generated by the sponsor node. Moreover, this approach uses asymmetric cryptosystem for sending the necessary keys from sponsor to co-distributors which is slower than symmetric cryptosystem.

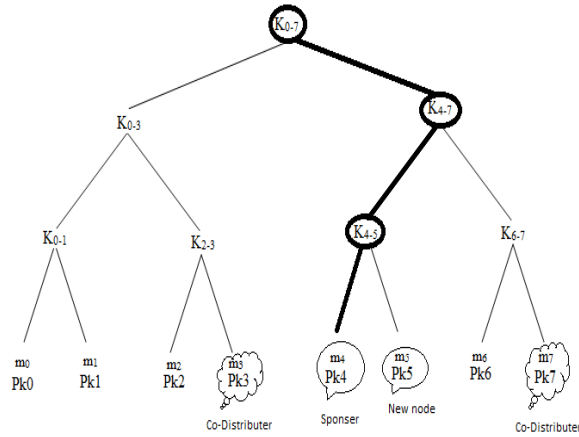


Figure 3.1 DGKD join operation

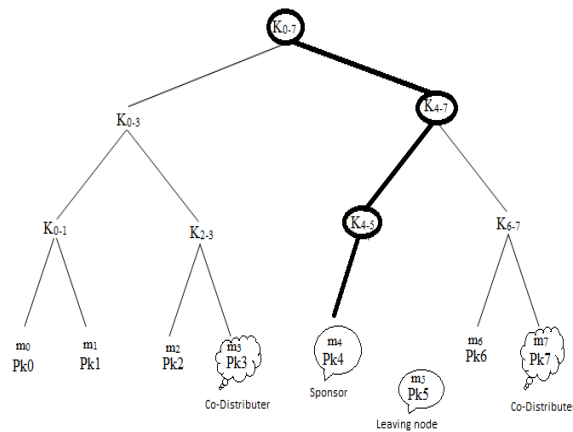


Figure 3.2 DGKD leave operation

EDKAS [17] This method is based on the concept of distributed one way function trees. it is a period based group rekeying approach. For each node a secret key and its corresponding blinded key is associated. The blinded keys are computed by applying a given one-way function(oft). Each member generates a unique secret key for itself by a secure pseudo random number generator (PRNG). In this way the secret key associated with the root node (known as group key) is shared by all the members. The key of an intermediate node is computed by the blinded keys of two child nodes using  $K_{i,j}=f(g(K_i), g(K_j))$  where  $f$  is a mixing function and  $g$  is a given one-way hash function. Each member maintains his/her own secret key and all the blinded keys of nodes that are sibling of the nodes from the path set. The disadvantages of this protocol are expensive maintenance of secure channels between members, and expensive communication cost as well as its message size cost.

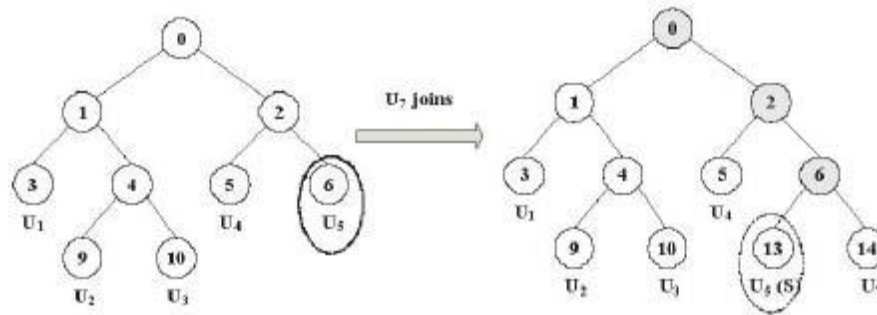


Figure 3.3 EDKAS join operation

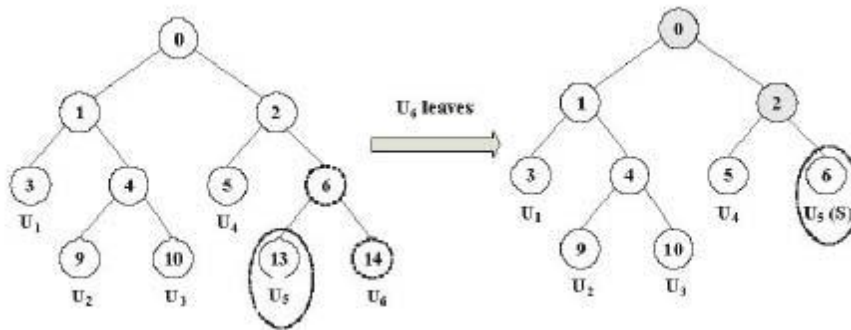


Figure 3.4 EDKAS leave operation

### 3.4 DIFFIE HELLMAN ALGORITHM FOR KEY EXCHANGE

The Diffie–Hellman key exchange algorithm solves the following dilemma. The two users named Alice and Bob want to share a secret key for use in a symmetric cipher, but their communication mean is insecure. Every piece of information that they exchange is observed by their Eve. How is it possible for Alice and Bob to share a key without making it available to Eve? At first glance it appears that Alice and Bob face cant share the key through this mean. It was a brilliant insight of Diffie and Hellman that the difficulty of the discrete logarithm problem for  $F * p$  provides a possible solution. The first step is for Alice and Bob to agree on a large prime number  $p$  and a nonzero integer  $g$  modulo  $p$ . Alice and Bob make the values of  $p$  and  $g$  public knowledge; for example, they might post the values on their web sites, so Eve knows them, too. For various reasons to be discussed later, it is best if they choose  $g$  such that its order in  $F * p$  is a large prime. The next step is for Alice to pick a secret integer  $a$  that she does not reveal to anyone, while at the same time Bob picks an integer  $b$  that he keeps secret.

Bob and Alice use their secret integers to compute

$A \equiv g^a \pmod{p}$  Alice computes this

and  $B \equiv g^b \pmod{p}$  Bob computes this

They next exchange these computed values, Alice sends A to Bob and Bob sends B to Alice. Note that Eve gets to see the values of A and B, since they are sent over the insecure communication channel.

Finally, Bob and Alice again use their secret integers to compute

$A_0 \equiv B^a \pmod{p}$  Alice computes this

$B_0 \equiv A^b \pmod{p}$  Bob computes this

The values that they compute,  $A_0$  and  $B_0$  respectively, are actually the same, since  $A_0 \equiv B^a \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \equiv B_0 \pmod{p}$ .

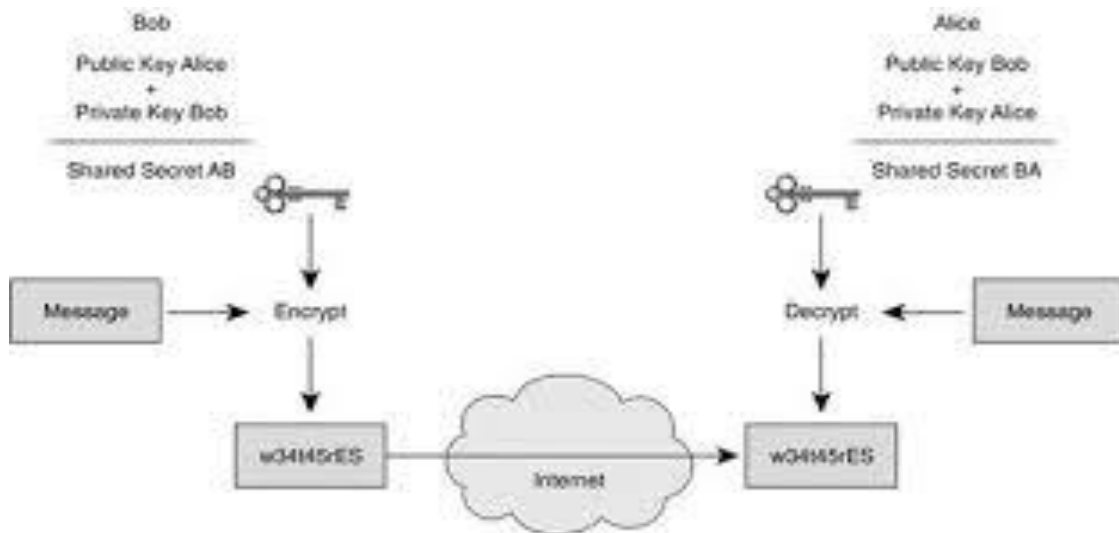


Figure 3.5 Diffie hellman key exchange

### **3.5 SYMMETRIC ALGORITHM FOR INTERMEDIATE NODES**

When using symmetric algorithms, both parties share the same key for encryption and decryption. To provide privacy, this key needs to be kept secret. Once somebody else gets to know the key, it is not safe any more. Symmetric algorithms have the advantage of not consuming too much computing power.

A few well-known examples are: DES, Triple-DES (3DES).

### **3.6 ASYMMETRIC ALGORITHM FOR LEAF NODES**

Asymmetric algorithms uses two pairs of keys. One pair is used for encryption and the other one for decryption. The decryption key is typically kept secretly, hence called "private key" or "secret key", while the encryption key is spread to all who might want to send encrypted messages, therefore called "public key". Everybody having the public key is able to send encrypted messages to the owner of the secret key. The secret key can't be reconstructed from the public key.

Asymmetric algorithms seem to be ideally suited for real-world use: As the secret key does not have to be shared, the risk of getting known is much smaller. Every user only needs to keep one secret key in secrecy and a collection of public keys, that only need to be protected against being changed. With symmetric keys, every pair of users would need to have an own shared secret key. Well-known asymmetric algorithms are RSA, DSA

## 4.0 DESIGN PRINCIPLE

We now present our efficient approach, DHSA, for distributed secure group communication. As name indicates, this distributed group key management approach uses Diffie-Hellman and symmetric algorithm along with the concept of logical hierarchical key tree. The main reason for this approach is to reduce re-keying overhead at join/leave. DHSA focuses on member collaboration for key calculation instead of key delivery by sponsor or co-distributor. For this purpose, we introduce three basic characteristics of DHSA:

- (1) The leaf key (users) in the logical tree is the public key of the corresponding group member, and all intermediate node keys are symmetric keys.
- (2) The public key of each member along with binary code the corresponding parent node is stored in a list shared by group members. This list will be updated each time on any membership change and periodically (leave/join).
- (3) All group members have the same capability and are equally trusted and responsible i.e no central authority.

The public key(key for leaf) of each member is generated by Diffie-Hellman key agreement. When  $p$  is a large prime number and  $g$  is the primitive element of multiplicative group  $Z^*_p$ , the public key of a member is obtained by  $g^{xi} \pmod p$ . This public key is used to create a share key with other members in the group. For example,  $ui$  can share a key with  $uj$  by calculating  $g^{(xa*xb)} \pmod p$ .

DHSA introduces two types of codes in its key tree, binary code for member position discovery, and decimal one for intermediate node key calculation.

**1. Binary Code:** Code will be used for member position discovery.

**2. Decimal Code:** Code will be used for intermediate node key calculation.

Figure illustrates a key tree with 8 members,  $\{u1, \dots, u8\}$ , and its corresponding binary code. The binary code of first level of each intermediate node from the bottom of the key tree, and the corresponding two members' public key are stored in a list. Each member uses this list to find the public key of any member who don't have sibling and with whom they can establish a connection. As stated before, this list is updated whenever there is a membership change and is broadcasted to other members by multicast. Usually, the

sibling member of affected branch is responsible to broadcast/multicast the updated information to other members. Table shows the management of binary code and its associated members public key in the list. As shown in this table, the public keys of  $u_1$  and  $u_2$  are  $gx_1$ ,  $gx_2$  respectively, and their associated parent binary code is  $000$ . Since there is no sibling member for  $u_3$ , the list just shows its public key,  $g^{x_3}$ , and the associated parent binary code,  $00$ .

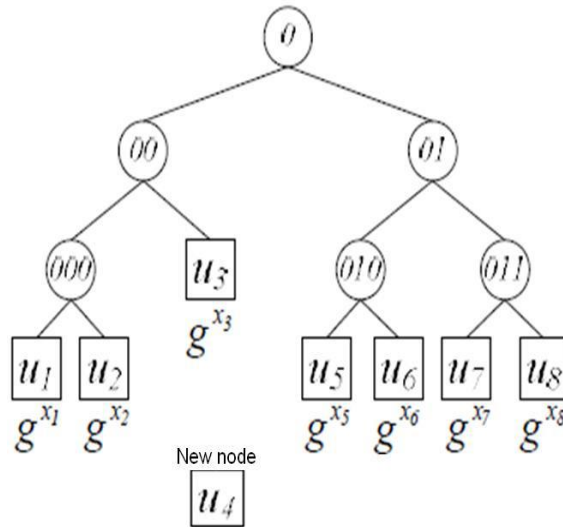


Figure 4.1 DHSA parent binary code for member position discovery

Table 4.1 List of parent binary code and associated member public key

Parent binary code	Member public key
000	$gx_1, gx_2$
00	$gx_3$
010	$gx_5, gx_6$
011	$gx_7, gx_8$

As stated above, the other code type in DHSA is decimal code. Decimal code is used just for intermediate node keys calculation, and is assigned to each intermediate node in the key tree. Root node will contain the group key. Intermediate node key is calculated using the below formulae.

$$\text{Key}_{\text{intermediate node}} = f(\text{Key}_{\text{group}} \oplus \text{Code}_{\text{intermediate node}})$$

$$\text{Code}_{\text{child\_node}} = (\text{Code}_{\text{parent\_node}} \parallel \text{Random digit}).$$

Figure 4.1 shows a hierarchical tree structure. When a new member wants to join a group it sends a join request message to the entire group. The node with no siblings or if everyone has sibling then message is replied by node with lowest id will reply. If there are multiple nodes having no siblings, then the node with smallest parent binary code value replies to the join request. On receiving this join request each member check if it has the smallest binary code value, if so then that node will be responsible for the group key management operations at this join. When  $f$  is a given one way hash function, and  $K_g$  is the previous group key, the new group key  $K'_g$  is calculated as follows.

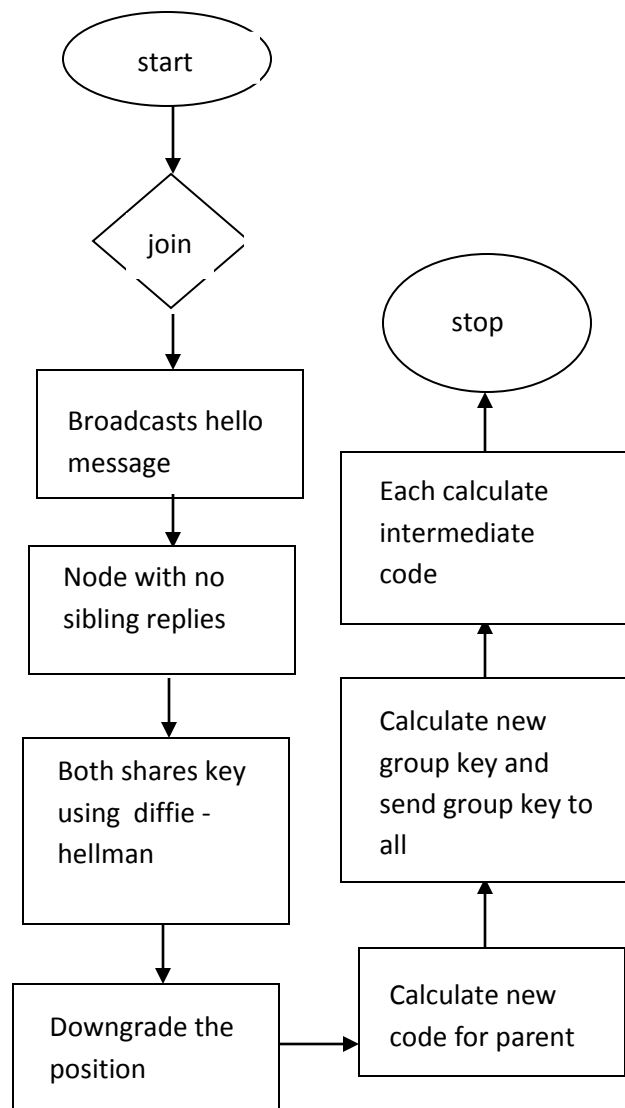


Figure 4.2 Flow chart for join operation



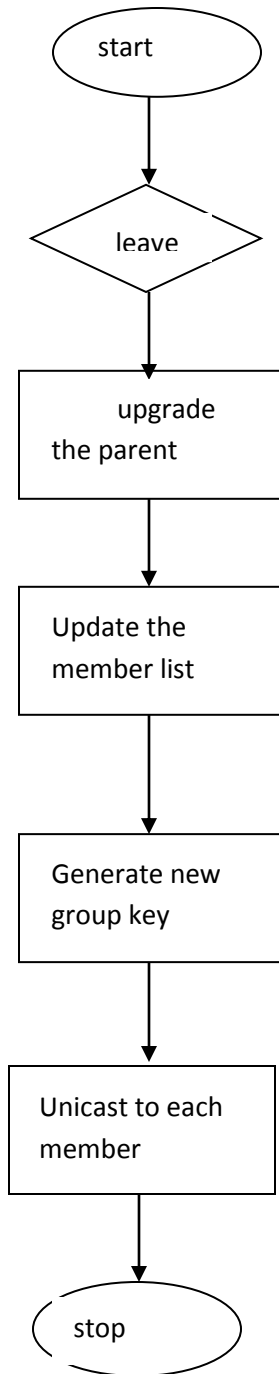


Figure 4.3 Flowchart for leave operation

## 4.1 LOGGING INTO THE SYSTEM

When a new member wants to join a group it enters the username and password. The node with no siblings or if everyone has sibling then message is replied by node with lowest id will reply. If there are multiple nodes having no siblings, then the node with smallest parent binary code value replies to the join request. On receiving this join request each member check if it has the smallest binary code value, if so then that node will be responsible for the group key management operations at this join.

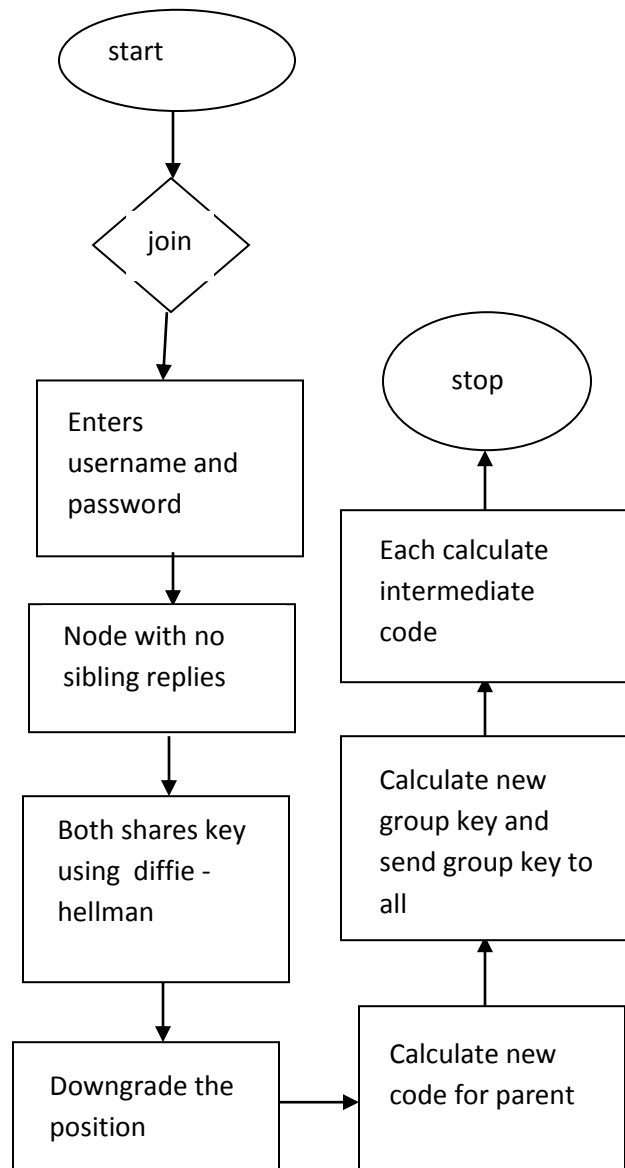


Figure 4.4 Flowchart for logging into the system

## 4.2 Logging out from the system

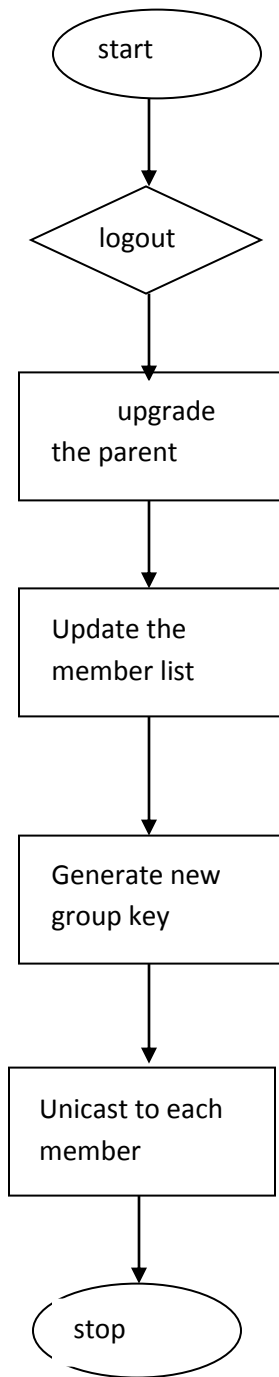


Figure 4.5 flowchart for logging out from system

### 4.3 USE CASE DIAGRAM FOR LOGIN SYSTEM

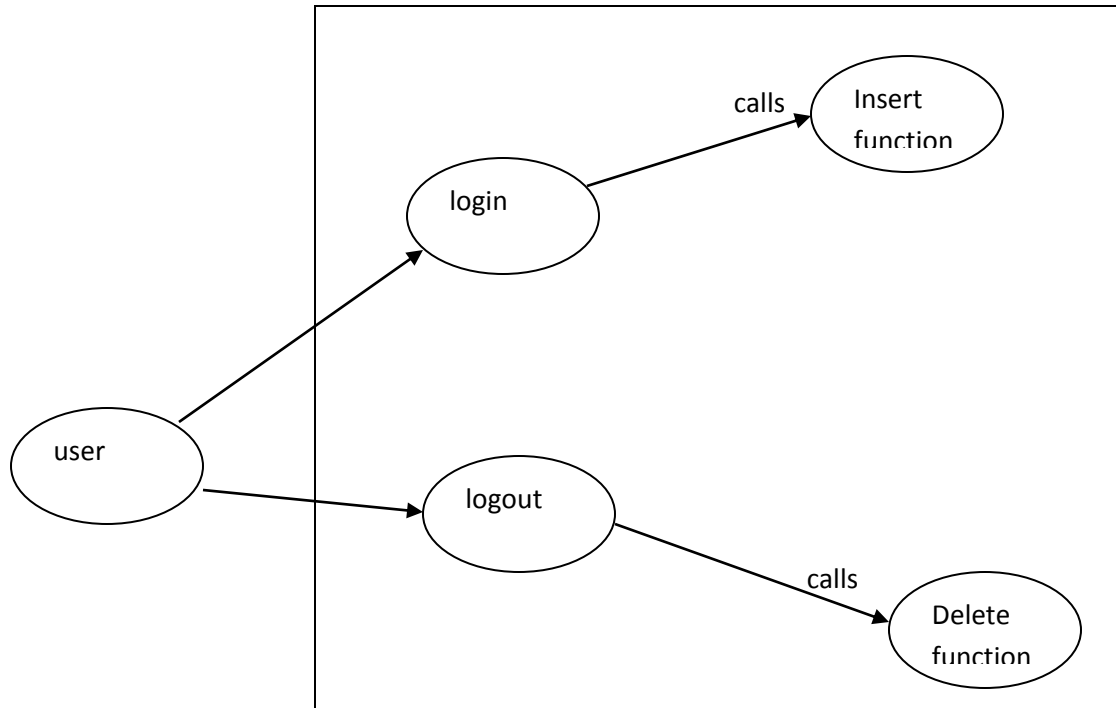


Figure 4.6 Usecase for logging in and out from system

## 5.0 SYSTEM REQUIREMENT

### 5.1 Software requirement

- Operating system
- Developing language (JAVA).
- Eclipse platform
- Glass fish server 4.1
- Xampp server

### 5.2 Hardware requirements

- HDD - To install the software at least 2 GB and the data storage is depending upon the organizational setup.
- PROCESSOR - Intel Pentium IV, 1GHZ or above
- RAM - 256MB or above
- VIDEO - 1024x768, 24-bit colors
- KEYBOARD - Standard 104 Keys(QWERTY)

## 6.0 FINAL DESIGN

To explain the detailed approach, consider our simple example with 8 members as illustrated in Figures 5.1 for join operation and Figure 5.2 for leave operation. Members decide a large prime number  $p$  and its primitive element  $g$  for each group. These values are public in group. Initially, this value is selected at initial mode of key tree establishment.

When a new member wants to join a group, it sends a hello message to discover the group members. Members, who receive the signal of this member, check the list to know which member does not have a sibling member. A member who does not have a sibling member in its branch replies to the hello message. But when each member has his/her corresponding sibling member, the member with lowest parent binary ID replies to that member. He/she exchanges the public key generated by Diffie-Hellman key agreement. Here, a member who replies is responsible to provide authentication to the member.

Once authentication operation is provided, the public key of new member and his/her corresponding parent binary code is stored in list and updates the list, and the updated information is broadcasted to existing members. Next, the current members as well as the new one can calculate the affected intermediate node keys by applying a given one-way hash function (OFT) to bitwise XOR of new group key and the intermediate node code.

### 6.1 JOIN OPERATION

Using Fig 5.1 a multicast group of 7 members,  $\{u1, u2, u3, u5, u6, u7, u8\}$  as current members when a new member  $u4$  joins the group (Fig. 5.1).

- (1)  $u4$  broadcasts a hello message for joining group.
- (2)  $u3$  who does not have a sibling node, replies to this message.
- (3)  $u3$  shares a key with  $u4$  by Diffie-Hellman key agreement.

This key is  $g^{x3x4} \pmod{p}$ .

- (4)  $u3$  downgrades its position from 00 to 001, updates the member discovery key by replacing the new parent binary code and new member's public key (Table 5.1).
- (5)  $u3$  calculates the new intermediate node code for its Parent as shown.

$Code_{K3,4} = (04 \parallel 6) = 046$ .

(6)  $u_3$  generates new group key .

$$K'g=f(Kg)$$

(7)  $u_3$  sends  $K'g$ , and the new node code to  $u_4$  being encrypted by the shared key between them.

$$U_3 \rightarrow (K'g, 046) \xrightarrow{g^{x_3 \times 4}}$$

(8) Existing members,  $\{u_1, u_2, u_3, u_5, u_6, u_7, u_8\}$ , renew the group key as describe .

(9) Then, the members in the affected path set calculate the intermediate node keys by applying one-way hash function to bitwise XOR of intermediate node codes and the new group key.

$$U_3, U_4: K_{3,4} = f(K'g(+), 046).$$

$$U_3 \dots U_4: K_{3,4} = (K'g(+), 04).$$

Table 6.1 List of parent binary code and associated member public key

Parent binary code	Member public key
000	$gx_1, gx_2$
001	$gx_3, gx_4$
010	$gx_5, gx_6$
011	$gx_7, gx_8$

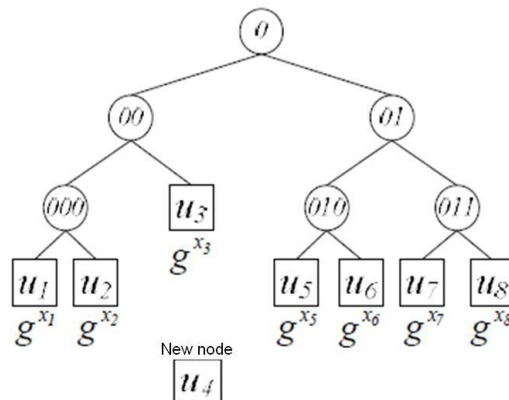


Figure 6.1 DHSA join operation

In this just one key is delivered to new member, which is an important feature for distributed group communication in wireless and dynamic network. Since members are mobile, in addition to dynamic join/leave, simultaneous join may occur in wireless networks. In order to solve such problem, the overload of rekeying operation must be minimized. The features of DHSA provide this task with just one key delivery reducing overhead of rekeying.

## 6.2 Leave operation

When a member leaves a multicast group, his/her node is deleted from the key tree. The sibling member on that branch moves up to its parent node position. And the sibling node is responsible to update the list and to transmit this information of the list to other members. After each leave, the group key and some intermediate node keys need to be updated. At leave operation, the key tree has divided into some parts. The number of these parts is equal to  $(\log n - 1)$  where  $n$  is the number of group members, new group key is generated by sibling and sends it to one of the member in each part. To do this the sibling node looks up the list and finds one of the available members in each part, shares a key with that member using its public key and send the group key for its via unicast. The member who receives multicast group key to his its branch members being encrypted with upper intermediate node which is not affected. Now the users are able to renew the affected intermediate node key.

Following are the steps to leave a group:

- (1)  $u_7$  is upgraded to its parent position.
- (2)  $u_7$  updates the member list by deleting the leaving node's public key, and changes its parent binary code.  $u_7$  also broadcast message to the other nodes about the updated information.
- (3)  $u_7$  generates new group key  $K''_g$  by using symmetric algorithm.
- (4)  $u_7$  looks up the list and use Diffie-Hellman key agreement to share a key with one of the member in each branch. Then, it unicast new group key to each member.

$$U_7 \rightarrow U_1: (K''_g)g^{x_1}x_7$$

$$U_7 \rightarrow U_5: (K''_g)g^{x_5}x_7$$



(5) now  $u1$  and  $u5$  multicast the received new group key  $G, K''$ , to members of their branch as follow:

$$U7 \quad U2 \dots U4: (K'' \oplus g) \rightarrow K_{1,4}$$

$$U7 \quad U6: (K'' \oplus g) \rightarrow K_{5,6}$$

(6) Finally the members in affected path calculate the code of the affected intermediate node by the formula below.

$$U5, U6, U7: K_{5,7} = f(K'' \oplus 08).$$

Table 6.2 List of parent binary code and associated member public key

Parent binary code	Member public key
000	$gx1, gx2$
001	$gx3, gx4$
010	$gx5, gx6$
01	$gx7$

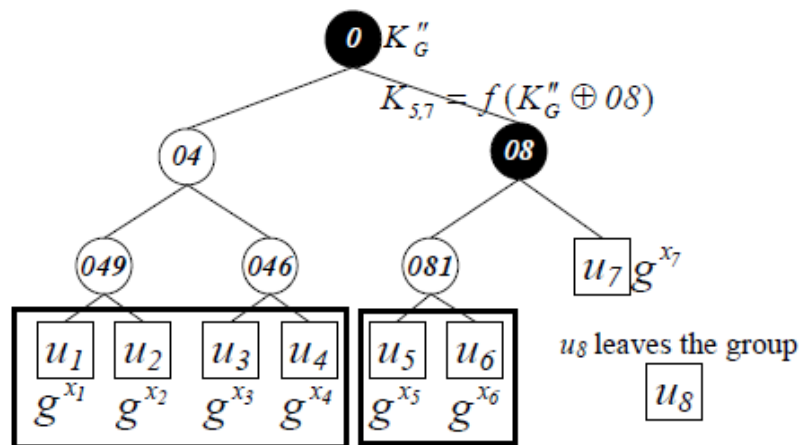


Figure 6.2 DHSA leave operation

## 6.3 SIMULATION:

### 6.3.1 First user joined the group

```
.....  
Enter first letter of show, insert, find, delete, or traverse: i  
Enter first letter of show, insert, find, delete, or traverse: s  
.....  
75  
.....  
Enter first letter of show, insert, find, delete, or traverse: |
```

### 6.3.2 Second user joined the group

```
.....  
Enter first letter of show, insert, find, delete, or traverse: i  
user77sends hello message  
user75replies message  
prime number1312670517633624847  
user75has key1030014197  
user77has key187201694  
sharedkey187201694  
intermediate code for node 0  
old group key-1  
new group key-1400787691  
Enter first letter of show, insert, find, delete, or traverse: s  
|.....  
75 0 77  
.....  
Enter first letter of show, insert, find, delete, or traverse:
```

### 6.3.3 Third user joined the group

```
.....
Enter first letter of show, insert, find, delete, or traverse: i
0
user79sends hello message
user75replies message
prime number1312670517633624847
user75has key1030014197
user79has key268171675
sharedkey268171675
intermediate code for node 7
old group key-1400787691
new group key595033851
Enter first letter of show, insert, find, delete, or traverse: s
|.....
|
|          0
|         00          77
|        75          79          --          --
|.....
```

### 6.3.4 Fourth user joined the group

```
.....
Enter first letter of show, insert, find, delete, or traverse: i
0
user81sends hello message
user77replies message
prime number1312670517633624847
user77has key187201694
user81has key100456815
sharedkey100456815
intermediate code for node 3
old group key595033851
new group key1482937450
Enter first letter of show, insert, find, delete, or traverse: s
.....
|
|          0
|         00          01
|        75          79          77          81
|.....
Enter first letter of show, insert, find, delete, or traverse:
```

## 7.0 REAL TIME IMPLEMENTATION OF DISTRIBUTED GROUP KEY MANAGEMENT

Members decide a large prime number  $p$  and its primitive element  $g$  for each group. These values are public in group. Initially, this value is selected at initial mode of key tree establishment.

When a new member wants to join a System, it enters a username and password to discover the authentication using the database. Members, who receive the signal of this member, checks the list to know which member does not have a sibling member. A member who does not have a sibling member in it's branch replies to the message. But when each member has his/her corresponding sibling member, the member with lowest parent binary ID replies to that member. He/she exchanges the public key generated by Diffie-Hellman key agreement.

Once the member replies, the public key of new member and his/her corresponding parent binary code is stored in list and updates the list, and the updated information is broadcasted to existing members. Next, the current members as well as the new one can calculate the affected intermediate node keys by applying a given one-way hash function(OFT) to bitwise XOR of new group key and the intermediate node code

### 7.1 JOIN OPERATION WHILE LOGGING INTO THE SYSTEM

A multicast group of 7 members,  $\{u1, u2, u3, u5, u6, u7, u8\}$  as current members when a new member  $u4$  joins the group (Fig. 5.1).

- (1)  $u4$  logs into the system through the login the page.
- (2)  $u3$  who does not have a sibling node, replies to this message.
- (3)  $u3$  shares a key with  $u4$  by Diffie-Hellman key agreement.

This key is  $g^{x3x4} \pmod{p}$ .

- (4)  $u3$  downgrades it's position from  $00$  to  $001$ , updates the member discovery key by replacing the new parent binary code and new member's public key (Table 5.1).
- (5)  $u3$  calculates the new intermediate node code for it's Parent as shown.

$Code_{K3,4} = (04 \parallel 6) = 046$ .

- (6)  $u3$  generates new group key .

$$K'g = f(Kg)$$

(7)  $u_3$  sends  $K'g$ , and the new node code to  $u_4$  being encrypted by the shared key between them.

$$U_3 \rightarrow U_4: (K'g, 046)_{g^{x_3 x_4}}$$

(8) Existing members,  $\{u_1, u_2, u_3, u_5, u_6, u_7, u_8\}$ , renew the group key as describe .

(9) Then, the members in the affected path set calculate the intermediate node keys by applying one-way hash function to bitwise XOR of intermediate node codes and the new group key.

$$U_3, U_4: K_{3,4} = f(K'g (+) 046).$$

$$U_3 \dots U_4: K_{3,4} = (K'g (+) 04).$$

## 7.2 Leave operation to logout the system

Following are the steps to leave a group:

(1) The user logs out the system through servlet.  $u_7$  is upgraded to its parent position.

(2)  $u_7$  updates the member list by deleting the leaving node's public key, and changes its parent binary code.  $u_7$  also broadcast message to the other nodes about the updated information.

(3)  $u_7$  generates new group key  $K''g$  by using symmetric algorithm.

(4)  $u_7$  looks up the list and use Diffie-Hellman key agreement to share a key with one of the member in each branch. Then, it unicast new group key to each member.

$$U_7 \rightarrow U_1: (K''g)_{g^{x_1 x_7}}$$

$$U_7 \rightarrow U_5: (K''g)_{g^{x_5 x_7}}$$

(5) now  $u_1$  and  $u_5$  multicast the received new group key  $G, K''$ , to members of their branch as follow:

$$U_7 \rightarrow U_2 \dots U_4: (K''g)_{K_{1,4}}$$

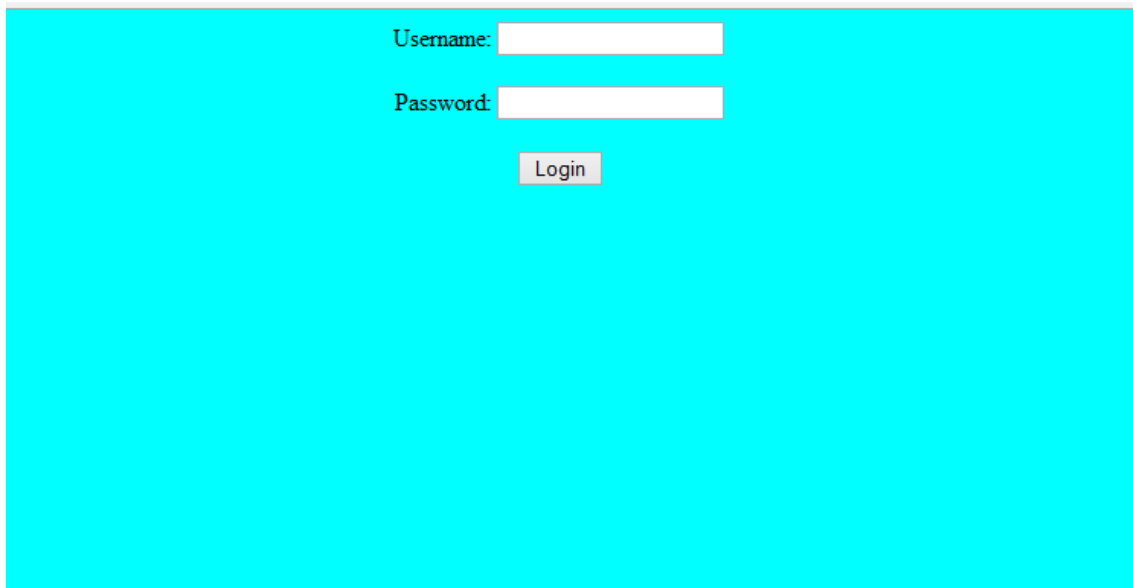
$$U_7 \rightarrow U_6: (K''g)_{K_{5,6}}$$

(6) Finally the members in affected path calculate the code of the affected intermediate node by the formula below.

$$U_5, U_6, U_7: K_{5,7} = f(K'g (+) 08).$$

## 7.3 SIMULATION:

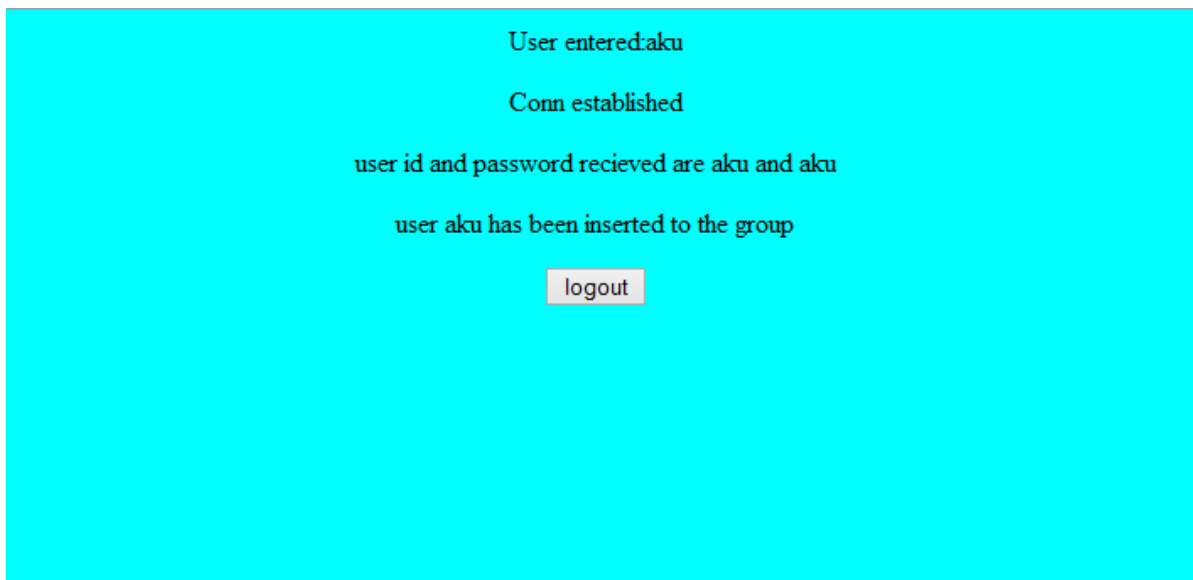
### 7.3.1 Html page to log into the system



Username:

Password:

### 7.3.2 First user logged into the system



User entered:aku

Conn established

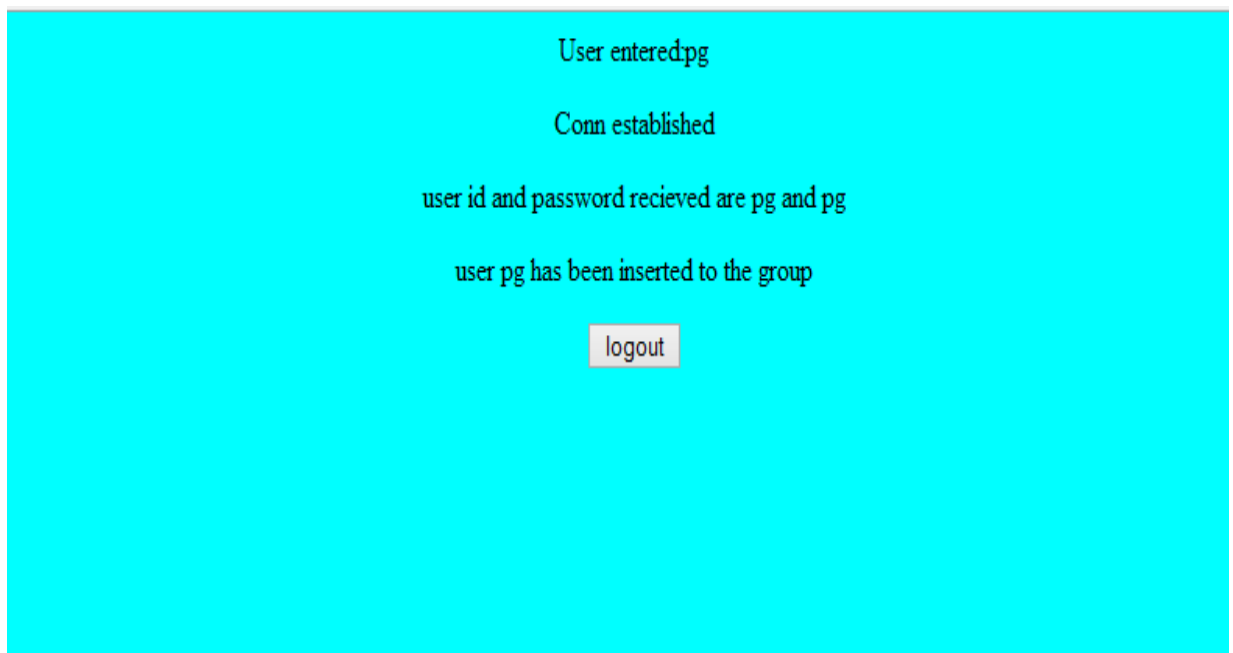
user id and password recieved are aku and aku

user aku has been inserted to the group

### 7.3.3 First user has been inserted into the tree as the user logs into the system



### 7.3.4 Second user logged into the system



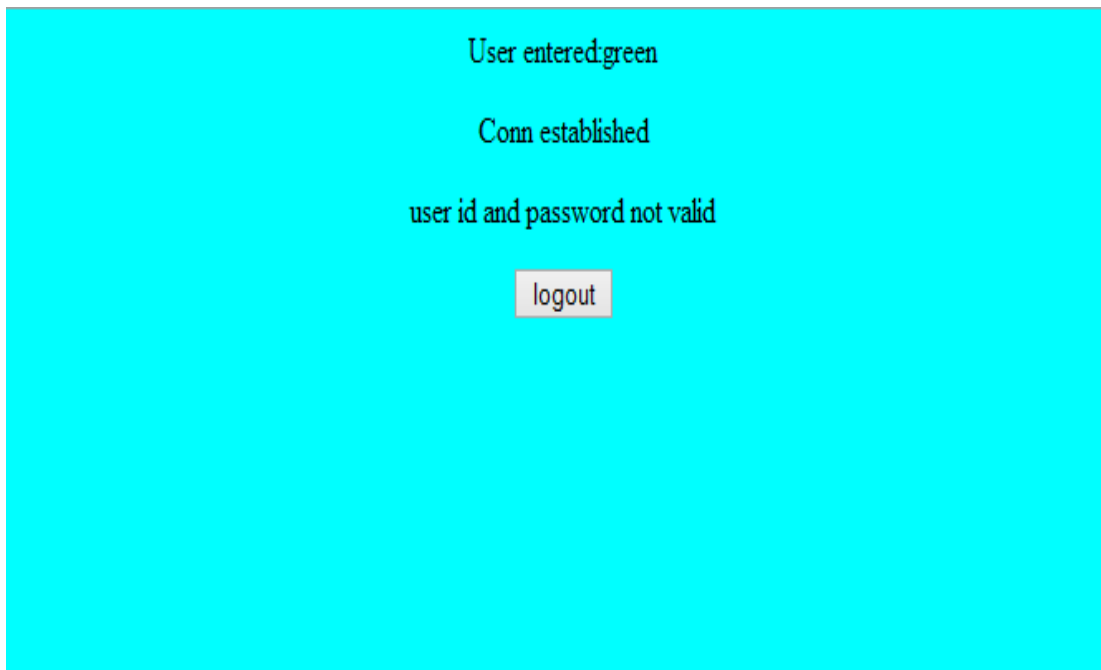
### 7.3.5 Second user has been inserted into the tree as the user logs into the system

```
user77sends hello message
user75replies message
prime number5711794904636656079
user75has key57373213
user77has key1898757720
sharedkey1898757720
old group key-1
new group key637789631
```

```
-----
0
75
77
-----
```

---

### 7.3.6 If user enters the wrong username or password





### 7.3.7 User logs out the system

Servlet logout at /WebApplication1

the user has been deleted

## 8.0 Security Requirement

Some security requirements which must be fulfilled by key management protocol in order to ensure secure group communication in dynamic systems.

1. Group key secrecy: Any non-group member is unable in any way to compute any previously existing or existing group key. This also implies that non group member is also unable to find any changed node key in the group. Mathematical operations and random numbers involved in rekeying must be cryptographically strong.
2. Forward key secrecy: Former members of the group, who may know any subset of older group keys, should not be able to find any new group key.
3. Backward key secrecy: Present members of the group, who may know any subset of group keys, should not be able to discover any previously used group key.
4. Key independence: Passive members or former members and present members of the group, who may know an group keys, are unable to determine any other key.
5. Reuse of known node keys: Evicted members must not discover any new information that is flowing within the group. Sometimes evicted users can use their prior knowledge of node keys to decrypt any future transmission. All node keys known to a leaving member must be changed during rekeying process.

## **9.0 Applications of Group Key Management**

The increased popularity of group oriented applications and protocols ,group communication becomes essential in many situations , from network layer multicasting to application layer teleconferencing . Regardless to the environment and situation , security protocol and their services are necessary to provide communication privacy and confidentiality.

### **9.1 Mobile ad-hoc network**

A mobile ad-hoc network (MANET) is a self-configuring network of mobile nodes connected by wireless links, to form an arbitrary topology. The nodes moves randomly and freely i.e the network's wireless topology may be unpredictable and may change rapidly. Minimal configuration, quick deployment and absence of a central governing authority are the basic characteristics of mobile ad-hoc network. Secure communication an important aspect of any networking environment, is an especially significant challenge in ad hoc networks. The unreliable wireless medium in MANET is a threat for Secure Data Transmission. The communication in mobile ad hoc networks comprises two phases, the route discovery and the data transmission. In an adverse environment, both Phases are vulnerable to a variety of attacks, one way to counter security attacks would be to cryptographically protect and authenticate all control and data traffic. Key management is a basic part of any secure communication structure. Most secure communication protocols rely on a secure, robust, and efficient key management system. The key is a piece of input information for cryptography algorithms. Different cryptographic keys are used for encryption like symmetric key, public key, group key and hybrid key (symmetric key + asymmetric key). In symmetric key management same keys are used by sender and receiver. This key is used for encryption the data as well as for decryption the data. In public key cryptography, two keys are used one private key and another public key.

Different keys are used for encryption and decryption. The private key is available only for individual and kept by source node and it is used for decryption. In MANET there are various Key Management Schemes proposed. To secure communications in Mobile Ad Hoc Networks (MANETs), messages are often protected by encryption using a chosen

cryptographic key, which, in the scenario of group communication is called the group key proposed in [1]. Multicast is a communication service that provides data delivery from a source to a set of recipients, also known as multicast group.

Secure group communication systems typically rely on a group key, a secret shared by all members of the group. Privacy is provided by encrypting all data with the group key. The key management system controls access to the group key, ensuring that only authenticated members receive the key.

## **9.2 Multimedia Security**

The availability of digital technologies and increasing Internet bandwidth in recent years have raised the demand for new multimedia services. The Internet service providers are now deploying the new technologies for group communications that allow the participation of many members. Service types include tele-conference, video-on-demand, interactive simulation, software updates and real-time delivery of stock market information. Multimedia security is an important requirement for the distribution networks when the delivery includes either confidential or patent data. With the deployment of digital technologies for the reproduction, storage and distribution of content, there is an increasing need for the protection of intellectual property. Content providers (movie studios and recording studios, in particular) have been evaluating the technologies that prevent unauthorized access to services.

Secure multicast communications in a computer network basically involves efficient packet delivery from one or more sources. This can be done using:

- Multicast data confidentiality: As the data traverses the public Internet, Encryption is commonly used for data confidentiality so to avoid unauthorized access to data.
- Multicast group key management: The security of the data is done using a group key being shared by the members that belong to the group. This key changes every time as a member joins or leaves the group for backward access control or forward access control. In some applications, there is a need to change the group key periodically. Encryption is commonly used to control access to the group key.
- Sessions in Multicast : In multicast communications, a session is defined as the time period in which data is exchanged among the group members. The type of member participation characterizes the nature of a session. In a one-to many application, data is

multicast from a single source to multiple receivers such as Pay-per-view, news feeds and real-time delivery of stock market information . A many-to-many application involves multiple senders and multiple receivers such as teleconferencing, white boarding and interactive simulation allows each member of the multicast group to send data as part of group communications.

### **9.3 Wired Networks**

In wired network, the root node is a key generator, which helps in generation and renewing of the common group key. Key generators can act as the multicast group creators or the group members or a trusted third party. Intermediate group members referred as key distributors can be the network devices or group members have the capability of assisting the group key management operations like join or leave. Each leaf node represents a user that attaches to the single key distributor. In the key transporting network, each member or intermediate node is associated with some parameters. The key generator maintains data of all the other nodes and holds secret group info, so to generate the common group key. Key generator establishes a globally shared common group key so to group communication only between the authenticated members. The group key generator transports only the parameters for deriving the common group key are delivered. Along the path from key generator to legitimate group members, each key distributor performs a transformation on the received data and forwards the result to a next key distributors and sub group members

## 11.0 CONCLUSION

Various classifications of group key management techniques are discussed in this report. We concentrated more on four different distributed key management techniques such as EDKAS, TGDH, DGKD and DHSA. From the analysis of the four methods, it became clear that for new member join case, DHSA has the least rekeying overheads and is a constant indicating that DHSA is more scalable than other methods. This protocol is based on logical key hierarchy tree. We have used symmetric cryptosystem along with asymmetric cryptosystem.

For asymmetric key, Diffie-Hellman key agreement is introduced. At the end, we conclude our proposal with two of its major functionalities.

- In DHSA, intermediate node keys are calculated by group members rather than distributed by a sponsor member.
- The features of this protocol are that, at join no keys are needed to be exchanged between existing members, at leave only one key, the group key, is delivered to remaining members.

## REFERENCES

- [1] S. Anahita Mortazavi, Alireza Nemaney Pour “ Distributed group key management using hierarchical approach with diffie hellman and symmetric algorithm “ 2011 CNDS,International symposium on computer networks and distributed systems,Feb 2011.
- [2] B. Jiang and X. Hu, “A Survey of Group Key Management,” 2008 IEEE, 2008 International Conference on Computer Science and software Engineering, Vol. 3, pp. 994-1002, Dec. 2008, doi:10.1109/CSSE.2008.1282.
- [3] Uday Pratap Singh,Rajkumar Singh Rathore, Computer Engineering and Intelligent Systems ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)Vol 3, No.7, 2012.
- [4] Preeti ,Banadana Sharma, International Journal For Advance Research in Engineering and Technology, Vol. 2 , Issue VI, June, 2014 ISSN 2320-6802.
- [ 5] Pratima Adusumilli, Xukai Zou, Byrav Ramamurthy,” Distributed Group Key Distribution with Authentication Capability”,2005 IEEE, Workshop on Information Assurance and Security,15-17 june 2005
- [ 6] Sandro Rafaeli and David Hutchison,” A Survey of Key Management for Secure Group Communication”ACM Computing Surveys,Vol 35,No 3,September 2003
- [7]By Mridula R. D.& Sreeja Rajesh ,Global Journal of Computer Science and Technology,Network, Web & Security,Volume 13 Issue 11 Version 1.0 Year 2013.
- [8]Uday Pratap Singh,Rajkumar Singh Rathore,International journal of computer applications,Volume 61 No 19,January 2013
- [9] Alan T. Sherman and David A. McGrew, Member,” Key Establishment in Large Dynamic GroupsUsing One-Way Function Trees”, IEEE, IEEE Transactions on Software Engineering, VOL. 29, NO. 5, MAY 2003.
- [10] [www.math.brown.edu/jhs/MathCrypto/SampleSections.html](http://www.math.brown.edu/jhs/MathCrypto/SampleSections.html)
- [11] William Stallings,Cryptography and network security Principle and Practice,Fourth edition.

[12] Bibo Jiang, Xiulin Hu, “A Survey of Group Key Management, International Conference on Computer Science and Software Engineering”, IEEE,2008, DOI 10.1109/CSSE.2008.1282



# APPENDIX

## INSERT FUNCTION

```
public void insert( int dd) throws Exception
{
int xb,xa = 0;
int lkey,rkey,temp;
int skey;
BigInteger pr;

Node newNode = new Node(); // make new node
// newNode.iData = id;      // insert data
newNode.iData = dd;//node id
newNode.leftChild=null;
newNode.rightChild=null;
newNode.pkey=-1;
// qu.add(newNode);
Iterator<Node>i=qu.iterator();
while(i.hasNext())
{
// System.out.print(i.next().iData );
qu.remove();
}
int h=height(root);
for(int k=1;k<=h;k++)
printGivenLevel(root,k);
if(root==null)
{// no node in root
// root.dData=0;
root = newNode;
```

```

qu.add(newNode);
//return;
}
else
{
Node current=qu.peek();
Node parent=new Node();
Node pre=new Node();
while(true)          // (exits internally)
{
System.out.println();
parent = current;
current = current.leftChild;
if(current == null) // if end of the line,
{          // insert on left
parent.leftChild=new Node();
parent.leftChild.iData=parent.iData;//node id
parent.leftChild.dData=2*parent.dData;//binary code
parent.leftChild.leftChild=null;
parent.leftChild.rightChild=null;
//p=getNextPrime(Math.abs(sr.nextLong()));
p=getNextPrime(Math.abs(1));
if(parent==root)
{
xa=sr.nextInt();
lkey=
Math.abs(BigInteger.valueOf((long)
g).modPow(BigInteger.valueOf((long)xa),p).intValue());
//parent.pkey=lkey;
//System.out.print("lkek"+lkey);
parent.leftChild.pkey=lkey;
parent.iData=0;

```

```

}
else
{
search(root,parent);
double code;
lkey=parent.pkey;
parent.leftChild.pkey=lkey;
}
parent.rightChild=newNode;
parent.rightChild.dData=(2*parent.dData)+1;
System.out.println("user"+parent.rightChild.iData+"sends hello message");
System.out.println("user"+parent.leftChild.iData+"replies message");
System.out.println("prime number"+p);
xb=sr.nextInt();
rkey=
Math.abs(BigInteger.valueOf((long)
g).modPow(BigInteger.valueOf((long)xb),p).intValue());
parent.rightChild.pkey=rkey;
System.out.println("user"+parent.leftChild.iData+"has key"+lkey);
System.out.println("user"+parent.rightChild.iData+"has key"+rkey);
temp=(int) Math.pow(g,xa);
skey=Math.abs(
BigInteger.valueOf((long)
g).modPow(BigInteger.valueOf((long)xb),p).intValue());
System.out.println("sharedkey"+skey);
//System.out.println("intermediate code for node "+ parent.iData);
Tree t=new Tree();
byte[] bytesOfMessage =t.generateRandomString().getBytes();

MessageDigest md = MessageDigest.getInstance("MD5");
byte[] thedigest = md.digest(bytesOfMessage);
// System.out.println(thedigest);
System.out.println("old group key"+root.pkey);

```

```

root.pkey= Math.abs(byteArrayToInt(thedigest));
System.out.println("new group key"+root.pkey);

qu.add(parent.leftChild);
qu.add(parent.rightChild);
long groupkey=root.pkey;
//      System.out.println(parent.rightChild.iData);
checkAffected(root,parent.rightChild,groupkey);
return;
} // nd if go left
/*else          //i or go right?
{
current = parent.rightChild;
if(current == null) // if end of the line
{          // insert on right
parent.rightChild = newNode;
parent.rightChild.dData=(2*parent.dData)+1;i
qu.add(parent.rightChild);
return;
}
}*/else
{
qu.remove();
//System.out.print("hey");
}

} // end else go right\
}
} // end insert()

```

## **DELETE FUNCTION**

```
public int delete(long key,Node p,Node parent)
```

```

{
int ans=0;
if(p==null)
return 0;
if(p.iData==parent.iData && p.iData==key && p==root && parent==root)
{
root=null;
/*Iterator<Node>i=qu.iterator();
while(i.hasNext())
{
Node n=i.next();
qu.remove(n);
}*/
return 1;
}
if(p.iData==key)
{
if(parent.leftChild==p)
{
parent.iData=parent.leftChild.iData;
parent.leftChild=null;
return 1;
}
else if(parent.rightChild==p)
{
Iterator <Node>i=qu.iterator();
/* while(i.hasNext())
{
Node n = i.next();
System.out.print( "data "+ n.iData );
if(n.iData==p.iData)

```

```

{
System.out.println("hey");
qu.remove();
}
if(n.iData==parent.leftChild.iData)
{
System.out.println("hey");
qu.remove();
}

}*/
parent.iData=parent.leftChild.iData;
qu.add(parent);
parent.leftChild=null;
parent.rightChild=null;
return 1;
}
}
ans=ans|delete(key,p.leftChild,p);
ans=ans|delete(key,p.rightChild,p);

return ans;
}

```

## **DISPLAY FUNCTION**

```

public void displayTree()
{
int count =0,i;
Stack globalStack = new Stack();
globalStack.push(root);
int nBlanks = 32;
boolean isRowEmpty = false;

```

```

System.out.println(
".....");
while(isRowEmpty==false)
{
Stack localStack = new Stack();
isRowEmpty = true;

for(int j=0; j<nBlanks; j++)
System.out.print(' ');

while(globalStack.isEmpty()==false)
{
Node temp = (Node)globalStack.pop();
if(temp != null )
{
// System.out.print(temp.iData);
localStack.push(temp.leftChild);
localStack.push(temp.rightChild);

if(temp.leftChild != null ||
temp.rightChild != null)
{
for(i=0;i<count;i++)
System.out.print('0');
System.out.print(Integer.toBinaryString(temp.dData));
// System.out.print(temp.iData);
isRowEmpty = false;
}
else
{
System.out.print(temp.iData);

```

```

}
}
else
{
System.out.print("--");
localStack.push(null);
localStack.push(null);
}
for(int j=0; j<nBlanks*2-2; j++)
System.out.print(' ');
} // end while globalStack not empty
System.out.println();
count++;
nBlanks /= 2;
while(localStack.isEmpty()==false)
globalStack.push( localStack.pop() );
} // end while isRowEmpty is false
System.out.println(
".....");
} // end displayTree()
// -----
}

```

## **SERVLET CODE FOR LOGIN PAGE**

```

public class login extends HttpServlet{
//static int count=0;
private outer.Tree t= new outer.Tree();
// private int i;
public void init()
{
//i=0;
// outer.Tree t = new outer.Tree();

```



```

}
public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String uid = request.getParameter("user");
String pass =request.getParameter("pwd");
//request.getSession().setAttribute("user", uid);
out.println("<head>");
out.println("<body bgcolor=\"cyan\">");
out.println("<center>");
out.println("<pfontsize=\"20\">User entered:"+uid +"</p>");
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con = (Connection)
DriverManager.getConnection("jdbc:mysql://localhost:3306/user","root","");
if(con!=null)
{
// i++;
out.println("<p fontsize=\"20\">Conn established</p>");
}
Statement stmt = (Statement) con.createStatement();
String qry = "select * from user_detail";
ResultSet rs = (ResultSet) stmt.executeQuery(qry);
while(rs.next())
{
//System.out.println("entered1");
String user = rs.getString(1);
String pwd = rs.getString(2);
// out.println("<p>User retrieved:"+user+"</p>");

```

```

// out.println("<p>Password retrieved:"+pwd+"</p>");
if(uid.equals(user) && pass.equals(pwd))
{
// out.println("entered");
HttpSession session = request.getSession(false);
session = request.getSession();
session.setAttribute("UserLogged",user );
out.println("<p fontsize=\"20\">user id and password recieved are "+uid+" and
"+pwd+"</p>");
out.println("<p fontsize=\"20\">user " +uid+" has been inserted to the group</p>");
//RequestDispatcher rd = request.getRequestDispatcher("/UserInput.jsp");
//rd.forward(request, response);
//    outer.Tree t= new outer.Tree();
//    for(int j=0;j<i;j++)
//    {
t.callinsert();
t.displayTree();
//    }
break;
}
//else
//{{
//    out.println("<p>user id and password not valid</p>");
//}}
}
}
catch(Exception ex)
{
ex.printStackTrace();
}
out.println("<form action =\"logout\" method=\"post\">");

```

```
out.println("<input type=\"submit\" value=\"logout\">");
out.println("</form>");
out.println("</center>");
out.println("</head>");
out.println("</body>");
}
}
```