P-2570

SECURING MOBILE AD HOC NETWORKS USING IKM

A PROJECT REPORT

Submitted by

M.AARTHI PRIYA	71205205001
RAMYA.N.	71205205045
T.YAMINI	71205205060

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY : CHENNAI 600 025

APRIL 2009

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**SECURING MOBILE AD HOC NETWORKS USING IKM**" is the bonafide work of **M.AARTHI PRIYA, RAMYA.N AND T.YAMINI** who carried out the project work under my supervision.

[illegible]

SIGNATURE

Ms.N.Suganthi
SUPERVISOR

Department of
Information Technology,
Kumaraguru College Of Technology,
Coimbatore-641006.

[illegible]

SIGNATURE

Dr.S.Thangasamy
DEAN

Department of
Computer Science and Engineering,
Kumaraguru College Of Technology,
Coimbatore-641006.

The candidates with University Register Nos. **71205205001, 71205205045 & 71205205060** examined by us in the project viva-voce examination held on

[illegible] 27/4/09

INTERNAL EXAMINER

[illegible] 27/4/09

EXTERNAL EXAMINER

DECLARATION

We,

M.AARTHI PRIYA	**Reg.No:**	**71205205001**
RAMYA.N	**Reg.No:**	**71205205045**
T.YAMINI	**Reg.No:**	**71205205060**

hereby declare that the project entitled **"SECURING MOBILE AD HOC NETWORKS USING IKM"**, submitted in partial fulfillment to Anna University as the project work of Bachelor of Technology (Information Technology) degree, is a record of original work done by us under the supervision and guidance of Department of Information Technology, Kumaraguru College of Technology, Coimbatore.

Place: Coimbatore

Date: 27.04.2009

[illegible] [illegible] [illegible]

[M.Aarthi priya] [Ramya.N] [T.Yamini]

Project Guided by

[illegible]

[L.S Jayashree M.E., Ph.D] **[Ms.N.Suganthi M.E., Ph.D]**

ACKNOWLEDGEMENT

We express our sincere thanks to our Chairman **Padmabhushaan Arutselvar Dr.N.Mahalingam B.Sc., F.I.E.,** Vice Chairman **Dr.K. Arumugam B.E., M.S., M.I.E.,** Correspondent **Shri.M.Balasubramaniam and** Joint Correspondent **Dr.A.Selvakumar** for all their support and ray of strengthening hope extended. We are immensely grateful to our Vice-Principal, **Prof.,R.Annamalai** for his invaluable support to the outcome of his project.

We are deeply obliged to **Dr.S.Thangasamy,** BE (HONS), Ph.D., Dean, Department of Computer Science and Engineering for his valuable guidance and useful suggestions during the course of this project.

We also extend our heartfelt thanks to our project co-ordinator **Ms.L.S.Jayashree** M.E.,Ph.D, Asso. Prof., Department of Information Technology for providing us her support which really helped us.

We are indebted to our project guide **Ms.N.Suganthi**, M.E.,Ph.D, Sr.Lecturer, Department of Information Technology for her helpful guidance and valuable support given to us throughout this project.

We thank the teaching and non-teaching staffs of our Department for providing us the technical support during the course of this project. We also thank all of our friends who helped us to complete this project successfully.

TABLE OF CONTENTS

ABSTRACT

Many group-oriented and mobile ad-hoc applications require security services. We therefore need a *secure distributed group key agreement* scheme so that the group can encrypt their communication data with a common secret group key. Tree-Based Group Diffie-Hellman (TGDH), which uses a binary key tree to arrange all the keys, is used for secure group key distribution. The keys are generated from the ID's (IKM) assigned to the members. The rekeying can be implemented either for every member join/leave (Individual rekeying) or for a batch of join/leave events in a given interval (Interval-based rekeying). The Interval based rekeying (Queue Batch) is efficient than individual rekeying. In our project we compare the time taken for a set of join/leave events in these approaches. This distributed contributory group key provides backward confidentiality (i.e., joined members cannot access previous communication data) and forward confidentiality (i.e., left members cannot access future communication data).

LIST OF FIGURES:

INTRODUCTION GENERAL

1.INTRODUCTION :

1.1. GENERAL :

With the emergence of many group-oriented mobile ad-hoc applications, there is a need for security services to provide group-oriented communication privacy and data integrity. To provide this form of group communication privacy, it is important that members of the group can establish a common secret key for encrypting group communication data. We therefore will require a *secure distributed collaborative dynamic group key agreement* scheme so that the group can encrypt their communication data with a common secret group key. This scheme differs from the traditional secure key communication by:

a) distributed nature in which there is *no centralized key server*,
b) collaborative nature in which the group key is *contributory* (i.e., each group member will collaboratively contribute its part to the global group key), and
c) Dynamic nature in which existing members may leave the group while new members may join.

Here, group members are arranged in a logical key hierarchy known as a *key tree*. Using the tree topology, it is easy to distribute the group key to members whenever there is any change in the group membership (e.g., a new member joins or an existing member leaves the group).

PROBLEM DEFINITION

1.2. PROBLEM DEFINITION:

To achieve secure group communication for mobile ad-hoc networks, the key tree approach is used which associates keys in a hierarchical tree. In order to efficiently maintain the group key in a dynamic group with more than two members, we use the Tree-Based Group Diffie-Hellman (TGDH) protocol which uses a binary key tree to arrange all the keys, is used for secure group key distribution. The keys are generated from the ID's (IKM) assigned to the members.

In this approach, we establish a group key via a distributed and collaborative approach for a mobile ad-hoc network. Each member of the group contributes its part to the overall group key and re-keying is performed at each join or leave event. This is implemented either for every member join/leave (Individual rekeying) or for a batch of join/leave events in a given interval (Interval-based rekeying). In our project we compare Individual and Interval-Based rekeying approaches based on the time taken for a predefined set of join/leave requests and the number of requests handled within a specified time.

OBJECTIVE OF THE PROJECT

1.3. OBJECTIVE OF THE PROJECT:

The main intent of the project is to provide a fundamental understanding about establishing a group key via a distributed and collaborative approach for mobile ad-hoc networks and performing efficient rekeying. We show that one can use the TGDH protocol to achieve such distributive and collaborative key agreement.

To reduce the rekeying complexity, an interval-based approach is used to carry out rekeying for multiple join and leave requests at the same time. To show that the interval based algorithms significantly outperform the individual rekeying approach and show that the Queue-batch algorithm performs the best.

LITERATURE REVIEW

FEASIBILITY STUDY

CURRENT STATUS OF THE PROBLEM

2. LITERATURE REVIEW:

2.1. FEASIBILITY STUDY:

2.1.1. CURRENT STATUS OF THE PROBLEM:

With the emergence of many group-oriented mobile ad-hoc applications, there is a need for security services to provide group-oriented communication privacy and data integrity. To provide this form of group communication privacy, it is important that members of the group can establish a common secret key for encrypting group communication data.

To achieve secure group communication, Wallner *et al.* and Wong *et al.* propose the key tree approach that associates keys in a hierarchical tree and rekeys at each join or leave event. Li *et al.* and Yang *et al.* then apply the periodic rekeying concept in Kronos to the key tree setting. All the key-tree-based approaches require a centralized key server for key generation. All the above schemes are decentralized and hence avoid the single-point-of-failure problem in the centralized case, though they introduce high message traffic due to distributed communication. These approaches do not have a previously agreed upon common secret key, communication between group members is susceptible to eavesdropping.

Centralized protocols rely on a *centralized key server* to distribute the group key. But Centralized servers are susceptible to single point attack, breach of security and are not suitable for peer groups and ad hoc networks. Here, Group key is not generated in a shared and contributory fashion. So backward confidentiality (i.e., joined members cannot access previous communication data) and forward confidentiality (i.e., left members cannot access future communication data) cannot be provided.

PROPOSED SYSTEM AND ITS ADVANTAGES

2.1.2. PROPOSED SYSTEM AND ITS ADVANTAGES :

To solve the problems faced by the Centralized key server, we need a *secure distributed group key agreement* so that people can establish a common group key for secure and private communication in mobile ad-hoc networks. Note that this type of key agreement protocols is both distributed and contributory in nature: each member of the group contributes its part to the overall group key. Moreover, an advantage of distributed protocols over the centralized protocols is the increase in system reliability, because the group key is generated in a shared and contributory fashion and there is no single-point-of-failure.

This scheme differs from the traditional secure key communication by:

a) distributed nature in which there is *no centralized key server*,
b) collaborative nature in which the group key is *contributory* (i.e., each group member will collaboratively contribute its part to the global group key), and
c) Dynamic nature in which existing members may leave the group while new members may join.

Here, group members are arranged in a logical key hierarchy known as a *key tree*. Using the tree topology, it is easy to distribute the group key to members whenever there is any change in the group membership. Group key is generated in a shared and contributory fashion. So backward confidentiality and forward confidentiality are ensured.

HARDWARE AND SOFTWARE REQUIREMENTS

2.2. HARDWARE REQUIREMENTS :

Processor	: Intel Pentium III 500 MHz
Primary Memory	: 128 MB
Secondary memory(Hard disk)	: 10 GB
Other Equipment	: Keyboard, Mouse.

2.3. SOFTWARE REQUIREMENTS :

Operating System	: Windows 2000 or XP
Front End Used	: NetBeans IDE 6.0
Language Used	: Java
Software Required	: J2SDK 1.6

SOFTWARE OVERVIEW

2.4. SOFTWARE OVERVIEW :

Introduction to Java :

Java is an object-oriented programming language developed by Sun Microsystems, a company best known for its high-end UNIX/LINUX workstations. Modeled after C++, the java language is designed to be small, simple and portable across platforms and operating systems, both at source and the binary level, which means that java programs can run on any machine that has java virtual machine installed. There are two types of java programs. They are java applets and java applications.

Java is a platform independent at both the source level and the binary level; platform independence means that a program can run on any computer system. Java programs can run on any system for which a Java Virtual Machine has been installed. Unlike other programming languages when java programs are compiled byte codes are generated which is a special set of machine instructions that are not specific to any one-processor or computer system.

Unlike most object-oriented languages, Java includes a set of input and output capabilities and other utility functions. Then basic libraries are part of standard environment, which includes simple libraries from networking, common Internet protocols and user interface toolkit functions. Because the libraries are written in Java, they are portable across platforms as all Java applications are. Apart from these features, Java has the following features:

Features:

1. Simple
2. Object-oriented
3. Distributed
4. Robust

5. Secure
6. Architecture neutral
7. Portable
8. Interpreted
9. High performance
10. Multithreaded
11. Dynamic

- **Java is simple.**

What it means by simple is being small and familiar. Sun designed Java as closely to C++ as possible in order to make the system more comprehensible, but removed many rarely used, poorly understood, confusing features of C++. These primarily include operator overloading, multiple inheritance and extensive automatic coercions. The most important simplification is that Java does not use pointers and implements garbage collection so that we don't need to worry about dangling pointers, invalid pointer references and memory leaks and memory management.

- **Java is secure.**

Java is intended to be used in networked environments. Toward that end, Java implements several security mechanisms to protect us against malicious code that might try to invade your file system. Java provides a firewall between a networked application and our computer.

- **Java is object-oriented.**

This means that the programmer can focus on the data in his application and the interface to it. In Java everything must be done via Method invocation on a Java object. We must view our whole application as an object; an object of a particular class.

- **Java is distributed.**

Java is designed to support applications on networks. Java supports various levels of network connectivity through classes in java.net. For instance, the URL class provides a very simple interface to networking. If we want more control over the downloading data than is through simpler URL methods, we would use a URLConnectionObject which is returned by a URL URL.openConnection() method. Also, you can do your own networking with the Socket and ServerSocket classes.

- **Java is robust.**

Java is designed for writing highly reliable or robust software. Java puts a lot of emphasis on early chacking for possible problems, later dynamic (runtime) checking and eliminating situations that are error prone. The removal of pointers eliminates the possibility of overwriting memory and corrupting data.

- **Java is architecture-neutral.**

Java program are compiled to an architecture neutral byte-code format. The primary advantage of this approach is that it allows a Java application to run on any system that implements the JVM. This is useful not only for the networks but also for single system software distribution. With the multiple flavors of Windows 95 and Windows NT on the PC and the new PowerPC Machintosh, it is becoming increasingly difficult to produce software that runs on all platforms.

- **Java is portable.**

The portability actually comes from architecture-neutrality. But Java goes even further by explicitly specifying the size of each of the primitive data types to eliminate implementation dependence. The Java system itself is quite portable. The Java compiler is written i9n Java, while the Java run-time system is written in ANSI C with a clean portability boundary.

- **Java is interpreted.**

The Java compiler generates byte-codes. The Java interpreter executes the translated byte codes directly on system that implements the Java Virtual Machine. Java's linking phase is only a process of loading classes into the environment.

- **Java is high performance.**

Compared to those high-level, fully interpreted scripting languages, Java is high-performance. If the just-in-time compilers are used, Sun claims that the performance of byte-codes converted to machine code ae nearly as good as native C or C++. Java however, was designed to perform well on very low-power CPUs.

- **Java is multithreaded.**

Java provides support for multiple threads of execution that can handle different tasks with a Thread class in the java.lang Package. The thread class supports methods to start a thread, run a thread, stop a thread and check on the status of the thread. This makes programming in Java with threads much easier than programming in the conventional single-threaded C and C++ style.

- **Java is dynamic.**

Java language was designed to adapt to an evolving environment. It is a more dynamic language than C or C++. Java loads in classes, as they are needed, even from across the network. This makes the software much easier and effectively. With the compiler, first we translate a program into an intermediate language called byte-codes, the platform-independent codes interpreted on the Java platform.

The interpreter parses and runs each Java bytecode instruction on the computer. Compilation happens just once; interpretation occurs each time the program is being executed.

The Java Platform :

A platform is the hardware or software environment in which a program runs. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's software only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

1. The Java Virtual Machine(JVM).
2. The Java Application Programming Interface(Java API).

The JVM is the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provides many useful capabilities, such as GUI, the Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages.

NetBeans IDE:

The NetBeans IDE is an open source integrated development environment written entirely in java using the NetBeans platform.NetBeans IDE supports development of all Java application types out of the box.Among other features are an Ant-based project system,version control and refactoring.

Modularity: All the functions of the IDE are provided by modules. Each module provides a well-defined function,such as support for the Java language,editing, or support for the CVS versioning system. NetBeans contains all the modules needed for Java development in a single download, allowing the user to start working immediately. Modules also allow NetBeans to be extended. New features, such as support for other programming languages, can be added by installing additional modules. For instance, Sun Studio, Sun Java Studio Enterprise, and Sun Java Studio Creator from Sun Microsystems are all based on the NetBeans IDE.

NetBeans Profiler:

The **NetBeans Profiler** is a tool for optimization of Java applications. It helps you find memory leaks and optimize speed. Formerly downloaded separately, it is integrated into the core IDE since version 6.0.

The Profiler is based on a Sun Laborites research project that was named JFluid. That research uncovered specific techniques that can be used to lower the overhead of profiling a Java application. One of those techniques is dynamic byte code instrumentation, which is particularly useful for profiling large Java applications. Using dynamic byte code instrumentation and additional algorithms, the NetBeans Profiler is able to obtain runtime information on applications that are too large or complex for other profilers. NetBeans also support Profiling Points that let you profile precise points of execution and measure execution time.

GUI design tool:

Formerly known as project Matisse, the GUI design-tool allows to prototype and Swing GUIs by dragging and positioning GUI components. The GUI builder has also built-in support for JSR 296 and JSR 295.

NetBeans JavaScript Editor:

NetBeans JavaScript Editor provides extended support for JavaScript and CSS.

Features:

1.JavaScript editor:
  - Syntax highlighting,
  - Code completion for native objects and functions,
  - All NetBeans editor's features,
  - Generation of JavaScript class skeleton,

2. CSS editor extension:
  - Code completion for styles names,
  - Quick navigation through the navigator panel,
  - Display CSS rule declaration in a List View,
  - Display file structure in a Tree View,
  - Sort the outline view by name, type or declaration order,
  - Create rule declaration
  - Refractor a part of a rule name.

The NetBeans IDE allows developers to:

- Create J2EE applications, automatically add EJB modules and web modules, and deploy the applications.
- Create EJBs from existing source, from scratch, or from an existing database.
- Automatically generate EJB business methods, database calls, and calls to EJBs.
- Create a web module for deployment to the freely available Application Server PE 8 or Tomcat.
- Add multiple source folders to an EJB module or web module and create unit tests as part of the project.
- Edit deployment descriptors in a graphical editor that is automatically synchronized with the underlying XML.
- Create,register,and test web services.
- Import our existing J2EE projects.

NetBeans Key Features:

The NetBeans IDE features significant new development including EJB Components and web services. It has over 15 modules for developing J2EE applications and services. The 3 key features are:

Creating EJB Modules, Session Beans, Entity Beans and Message Driven Beans:

- The NetBeans IDE guides the user through the process to easily learn how to write an EJB as well as deploy, package, and test applications.
- A GUI is available to select an EJB and perform tasks such as adding business methods and editing deployment descriptors.
- All EJB infrastructure methods are generated automatically and are hiden in a power code fold.
- The resulting EJB module can be easily added to a J2EE application.
- Entity Beans can be created using an existing database schema.
- The NetBeans project structure matches J2EE Java BluePrints Standards and relies on the ANT Open Standard for the build system.

Calling EJBs

- Using the "Call EJB" feature from within a Servlet, JSP, or web service saves the user the head ache of writing the JNDI lookup code and required exception handling. It will even work with your existing ServiceLocator class.

Develop and Test WebServices:

- Developers can create, modify, package, deploy and test web services from the IDE.
- Web services can be created from scratch, existing code, and /or WSDL.
- Once created, the web service can be registered with the IDE, which creates an internal client for testing the web services.

DETAILS OF THE METHODOLOGY EMPLOYED

3. DETAILS OF THE METHODOLOGY EMPLOYED:

TREE-BASED GROUP DIFFIE-HELLMAN PROTOCOL:

To efficiently maintain the group key in a dynamic peer group with more than two members, we use the Tree-Based Group Diffie-Hellman (TGDH) protocol proposed in. Each member maintains a set of keys, which are arranged in a hierarchical *binary tree*. We assign a node ID _ to every tree node. For a given node v. We associate a *secret* (or private) key K_v and a *blinded* (or public) key BK_v . All arithmetic operations are performed in a cyclic group of prime order p with the generator α . Therefore, the blinded key of node v can be generated by

$$BK_v = \alpha^{Kv} \bmod p \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Each leaf node in the tree corresponds to the individual secret and blinded keys of a group member M_i . Every member holds all the secret keys along its *key path* starting from its associated leaf node up to the root node. Therefore, the secret
key held by the root node is shared by all the members and is regarded as the *group key*. Fig. 1 illustrates a possible key tree with six members M_1 to M_6 . For example, member M_1 holds the keys at nodes 7, 3, 1, and 0. The secret key at node
0 is the group key of this peer group.

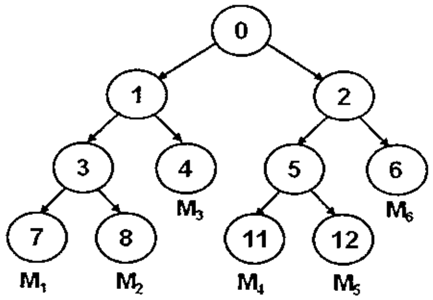


Fig. 1. A key tree used in the Tree-Based Group Diffie-Hellman protocol.

The node ID of the root node is set to 0. Each *non-leaf* node v consists of two child nodes whose node IDs are given by 2_v+ 1 and 2_v+ 2. Based on the Diffie -Hellman protocol [6], the secret key of a non-leaf node v can be generated by the secret key of one child node of v and the blinded key of another child node of v. Mathematically, we have

$$\begin{aligned} K_v &= (BK_{2v+1})^{K2v+2} \bmod p \\ &= (BK_{2v+2})^{K2v+1} \bmod p \\ &= \alpha^{K2v+1K2v+2} \bmod p \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(2) \end{aligned}$$

Unlike the keys at non-leaf nodes, the secret key at a leaf node is selected by its corresponding group member through a secure pseudo random number generator.

Since the blinded keys are publicly known, every member can compute the keys along its key path to the root node based on its individual secret key. To illustrate, consider the key tree in Fig. 1. Every member M_i generates its own secret key and all the secret keys along the path to the root node. For example, member M_1 generates the secret key K_7, and it can request the blinded key BK_8 from M_2, BK_4 from M_3 and BK_2 from either M_4, M_5 or M_6. Given M_1's secret key K_7, and the blinded key BK_8, M_1 can generate the secret key K_3 according to Eq. 2. Given the blinded key BK_4. and the newly generated secret key K_3 , M_1 can generate the secret key K_1 based on Eq. 2. Given the secret key K_1 and the blinded key BK_2, M_1 can generate the secret key K_0 at the root. From that point on, any communication in the group can be encrypted based on the secret key (or group key) K_0.

To provide both backward confidentiality (i.e., joined members cannot access previous communication data) and forward confidentiality (i.e., left members cannot access future communication data), *rekeying*, which means renewing the keys associated with the nodes of the key tree, is performed whenever there is any group membership change including any new member joining or any existing member leaving the group. Let us first consider individual rekeying, meaning that rekeying is performed after every single join or leave event. Before the group membership is changed, a special member called the *sponsor* is elected to be responsible for updating the keys held by the new member (in the join case) or departed member (in the leave case). We use the convention that the rightmost member under the subtree rooted at the sibling of the join and leave nodes will take the sponsor role. Note that the existence of a sponsor does not violate the decentralized requirement of the group key generation since the sponsor does not add extra contribution to the group key.

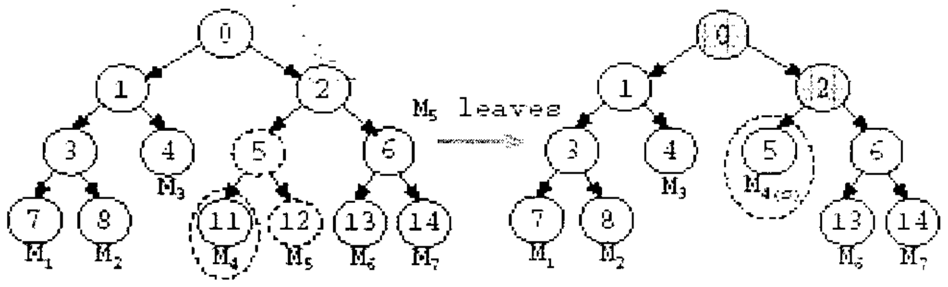


Fig. 2. Illustration of the rekeying operation after a single leave.

Fig. 2 depicts a member leave event. Suppose that member M_5 leaves the system. Node 11 is then *promoted* to node 5,and nodes 2 and 0 become *renewed nodes*, defined as the non-leaf nodes whose associated keys in the key tree are renewed.

Also, member M_4 becomes the sponsor. It renews the secret keys K_2 and K_0 and broadcasts the blinded keys BK_2 and BK_5 to all the members. Members M_1 , M_2, and M_3 , upon receiving the blinded key BK_2 , compute the new group key K_0. Similarly, members M_6 and M_7 , upon receiving BK_5 , can compute K_2 and then the new group key K_0.

PSEUDOCODE FOR INDIVIDUAL REKEYING(T)

```
{
if (a new member joins){
if (T== NULL) /* no new members in T*/
create a new tree _ with the only one new member;
Else /* there are members in T */
find the insertion node;
add the new member to T;
elect the rightmost member under the subtree rooted at the sibling of
the joining node to be the sponsor;
rekey renewed nodes;
}
If(a member wishes to leave)
Remove leaving nodes and promote their siblings;
rekey renewed nodes;
}
```

Fig. 3 illustrates a new member M_8 - that wishes to join the group. M_8- has to first determine the *insertion node* under which M_8 - can be inserted. To *add* a node, say v' to the insertion node, a new node, say n', is first created. Then the subtree rooted at the insertion node becomes the left child of the node n' , and the node v' becomes the right child of the node n'. The node n' will replace the original location of the insertion node. The insertion node is either the rightmost shallowest position such that the join does not increase the tree height, or the root node if the tree is initially well balanced (in this case, the height of the resulting tree will be increased by 1). Fig. 3 illustrates this concept.

The insertion node is node 7 and the sponsor is M_4 . M_8 - then broadcasts its blinded key BK_{12} upon insertion. Given BK_{12}, M_4 renews K_2 ,K_5 and K_0 and then broadcasts the blinded keys BK_5 and BK_2 to all members in the group. After receiving the blinded keys from M_4, all remaining members can rekey all the nodes along their key paths and compute the new group key K_0.

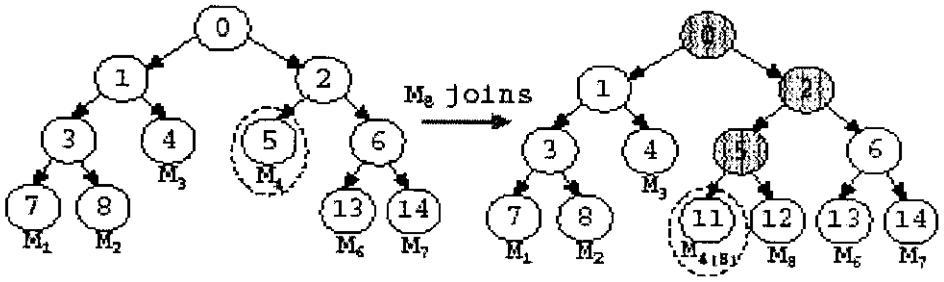


Fig. 3. Illustration of the rekeying operation after a single join.

Based on the above leave and join events in Figs. 2 and 3, we find that we can *reduce* one rekeying operation if we can simply change the association of node 12 from M_5 to M_8.

INTERVAL-BASED DISTRIBUTED REKEYING:

The Queue-batch algorithm is divided into two phases, namely the *Queue-subtree* phase and the *Queue-merge* phase. The first phase occurs whenever a new member joins the communication group during the rekeying interval. The thresholds for the completion of the queue-subtree phase are fixed number (say 15) of join requests, fixed number (say 5) of leave requests and fixed idle rekeying interval (say 2000 ms). In this case, we append this new member in a temporary key tree T'. The second phase occurs at the beginning of every rekeying interval and we merge the temporary tree T' (which contains all newly joining members) to the existing key tree T . The pseudo-codes of the Queue-subtree phase and the Queue-merge phase are illustrated below.

```
Queue-subtree (T')
1.  if (a new member joins) {
2.     if (T' == NULL) /* no new members in T' */
3.        create a new tree T' with the only one new member;
4.     else { /* there are new members in T' */
5.        find the insertion node;
6.        add the new member to T';
7.        elect the rightmost member under the subtree rooted at
          the sibling of the joining node to be the sponsor;
8.        if (sponsor) /* sponsor's responsibility */
9.           rekey renewed nodes and broadcast new blinded keys;
10.    }
11. }
```

```
Queue-merge ($T$, $T'$, $M'$, $L$)
1.  if ($L == 0$) { /* there is no leave */
2.      add $T'$ to either the shallowest node (which need not be the
        leaf node) of $T$ such that the merge will not increase the
        resulting tree height, or the root node of $T$ if the merge to
        any locations will increase the resulting tree height;
3.  } else { /* there are leaves */
4.      add $T'$ to the highest leave position of the key tree $T$;
5.      remove remaining $L - 1$ leaving leaf nodes and promote
        their siblings;
6.  }
7.  elect members to be sponsors if they are the rightmost members
    of the subtree rooted at the sibling nodes of the departed leaf
    nodes in $T$, or they are the rightmost member of $T'$;
8.  if (sponsor) /* sponsor's responsibility */
9.      rekey renewed nodes and broadcast new blinded keys;
```

The Queue-batch algorithm is illustrated in Fig. 4, where members M_8, M_9 and M_{10} wish to join the communication group, while M_2 and M_7, wish to leave. Then the rekeying process is as follows: (i) In the *Queue-subtree* phase, the three new members M_8, M_9 and M_{10} first form a tree T'. M_{10}, in this case, is elected to be the sponsor. (ii) In the *Queue-merge* phase, the tree T' is added at the highest departed position, which is at node 6. Also, the blinded key of the root node of T', which is BK_6, is broadcast by M_{10}. (iii) The sponsors M_1, M_6, and M_{10} are elected. M_1 renews the secret key K_1 and broadcasts the blinded key BK_1. M_6 renews the secret key K_2 and broadcasts the blinded key BK_2. (iv) Finally, all members can compute the group key.

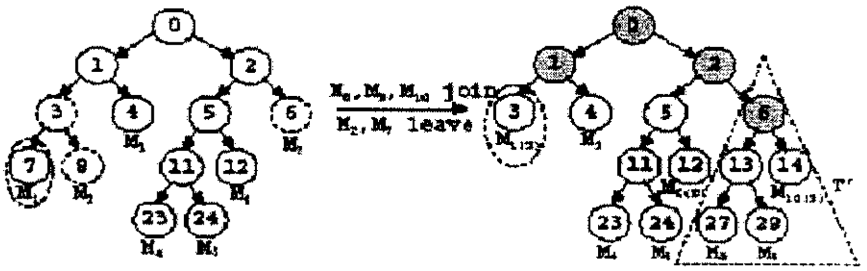


Fig. 4 Example of the Queue-merge phase.

PERFORMANCE EVALUATION

3. PERFORMANCE EVALUATION

We compare the individual rekeying's processing efficiency against that of Queue-Batch rekeying. Here a graph is plotted with the number of requests processed by individual rekeying and that of Queue-Batch rekeying for a fixed time (in milliseconds) and is compared.

Testing is a critical element of software quality and assurance and represents the ultimate review of specification design and coding. It is a vital activity that has to be enforced in the development of any system.this could be done in parallel during all phases of system development. The feedback received from these texts can be used for further enchancement of the system under consideration. The testing phase conducts test using the software requirement specification as a reference and with the goal to see whether system satisfies the specified requirements.

Standard procedure have been followed in testing our system. Test cases are generated for each screen. These test cases will cover every possibility which could result in both positive and negative results. These test plans are maintained for any further testing done on the system.

4.1 UNIT TESTING:

A series of stand-alone tests are conducted during unit testing. A unit test is also called a module test because it tests the individual units of code that comprise the application. Unit tests focus on functionality and reliability, unit testing is done in a test environment prior to system integration.

4.2 INTEGRATION TESTING:

Integration testing examines all the components and modules that are new, changed, affected by a change, or needed to form a complete system. It is the testing performed to detect errors on interconnection between modules.

4.3 SYSTEM TESTING

The system is tested against the system requirements to see if all the requirements are met and if the system performs as per the specified requirements.the system is tested as a whole for its functionality.

4.4 VALIDATION TESTING

This test is done to check for the validity of the entered input. The user inputs to the corresponding application input fields are verified before updating in the database

CONCLUSION

CONCLUSION:

Thus we have implemented distributed collaborative key agreement protocols for mobile ad-hoc networks. The key agreement setting is performed in which there is no centralized key server to maintain or distribute the group key. We show that one can use the TGDH protocol to achieve such distributive and collaborative key agreement. Individual rekeying algorithm is implemented, where for every join/leave event, rekeying is performed. A certain amount of computational overhead occurs with the Individual rekeying . To reduce the rekeying complexity, we use an interval-based approach to carry out rekeying for multiple join and leave requests at the same time. In particular, we show that the Queue-batch algorithm can significantly reduce both computation and communication costs when there exist highly frequent membership events.

FUTURE ENHANCEMENTS

5. FUTURE ENHANCEMENTS:

In this section, the *Authenticated scheme of Tree-Based Group Diffie-Hellman* (A-TGDH) protocol can be used that provides key authentication for our interval-based algorithms, with a trade-off between security and performance in mobile ad-hoc networks. The idea is to couple the session-based group key with the certified permanent private components of the group members. Each member holds two types of keys: *short-term secret* and *blinded* keys as well as *long-term private* and *public* keys. Short-term keys are randomly generated when a member joins the group and become expired when the member leaves, while long-term keys remain permanent across many sessions and are certified by a trusted CA. The protocol seeks to satisfy several requirements that are crucial for key establishment :

(i) *perfect forward secrecy* (i.e., the compromise of long-term keys does not degrade the secrecy of past short-term keys),

(ii) *known-key security* (i.e., the compromise of past short-term keys does not degrade the secrecy of future short-term keys), and

(iii) *key authentication* (i.e., all group members are assured that no outsiders can identify the group key). Also, the protocol can be implemented in a way that satisfies *key confirmation* (i.e., all group members are assured that every other member holds the same group key).

APPENDICES

5. APPENDICES:

APPENDIX 1:

SOURCE CODE:

Main class definition:

```
static class binaryNode
  {
  int memid;
  int nodeid;
  int ht;
  double Bk;
  double k;
  binaryNode left;
  binaryNode right;
  binaryNode parent;
  binaryNode sibling;


     public binaryNode(int value)
              {
        this.memid = value;
     }
  }
```

To find the node for inserting a new member:

```
public binaryNode insert(binaryNode rootnode)
  {
     binaryNode checkingNode=new binaryNode(0);
     binaryNode node=new binaryNode(0);
     binaryNode nodesib=new binaryNode(0);
     binaryNode temp=new binaryNode(0);
       binaryNode par=new binaryNode(0);
       binaryNode tempright=new binaryNode(0);
     int checkht=0,ht=0;
     temp=rootnode;
              tempright=temp.right;
```

```
      while(tempright!=null)
              {
                      if(temp.right!=null)
                              temp=temp.right;
                      else
                              break;

              }
      //setting the right most node as insertion node
      checkht=temp.ht;
      checkingNode=temp;
      node=temp;
      for(;;)
      {
                      if(node.nodeid==0)
                      {
         return checkingNode;
// the correct insertion node ie the shallowest node from right

                      }
        else
        {
                              par=temp.parent;
                              if(par.nodeid==0)
                              {
                                      return checkingNode;

                              }
                              else
                              {
                                      node=temp.parent;
                                      temp=temp.parent;
                                      nodesib=temp.sibling;
                                      if(nodesib.nodeid<node.nodeid)
                                      {
                                              temp=nodesib;
                                              while(temp.right!=null)
                                                      temp=temp.right;
                                              ht=temp.ht;
            // comparing height to get the shallowest node
```

```
                                    if(ht<checkht)
                                    {
                checkht=ht;
                checkingNode=temp;
                                    }
                               }
                               else
                   node=node.parent; // to traverse to the complete tree
                          }
                   }

              }
   }
```

Hash function:

```
public double hash(double n)
   {
     n=((3*n)+1)%127;
     return n;
   }
```

To join the new node to the tree at the insertion node:

```
  public void join(binaryNode insertionNode, binaryNode newNode)
  {
     binaryNode temp=new binaryNode(0);
     binaryNode sibling=new binaryNode(0);
     binaryNode sponsor=new binaryNode(0);

     if(insertionNode.nodeid==0 && insertionNode.memid==0)
     {
        insertionNode.memid=newNode.memid;
        insertionNode.k=hash(insertionNode.nodeid);

        insertionNode.Bk=Math.pow(3,insertionNode.k)%151;

     }
     else
     {
        //creating sibling of the newNode

        temp.nodeid=(2*insertionNode.nodeid)+1;
        temp.memid=insertionNode.memid;
        temp.ht=insertionNode.ht+1;
        temp.parent=insertionNode;
        temp.sibling=newNode;
        temp.k=hash(temp.nodeid);
        temp.Bk=Math.pow(3,temp.k)%127;

        //joining newNode to the tree
        newNode.nodeid=(2*insertionNode.nodeid)+2;
        newNode.ht=insertionNode.ht+1;
        newNode.parent=insertionNode;
        newNode.sibling=temp;

        //updating parent node

        insertionNode.memid=0;
        insertionNode.left=temp;
        insertionNode.right=newNode;

    }
    sponsor=newNode;
    rekey(sponsor);
  }
```

Rekeying operation:

```
void rekey(binaryNode node)
  {

    binaryNode temp=new binaryNode(0);
    binaryNode sib=new binaryNode(0);
    binaryNode par=new binaryNode(0);
    node.k=hash(node.nodeid);
    node.Bk=Math.pow(3,node.k)%127;
    temp=node;
    while(temp.nodeid!=0)
    {
```

```
                        sib=temp.sibling;
            par=temp.parent;
            par.k=Math.pow(sib.Bk,temp.k)%127;
            temp=par;

          }

        repaint();
      }
```

To find the member wishing to leave the group:

```
void findLeave(binaryNode root,int mem) throws IOException  // to find the
member wishing to leave the group
  {
     binaryNode temp=new binaryNode(0);
     binaryNode tempsib=new binaryNode(0);
           binaryNode tempright=new binaryNode(0);
     int a;
     temp=root;
     for(;;)
     {
                  tempright=temp.right;
                  while(tempright!=null)
                  {
                       if(temp.right!=null)
                            temp=temp.right;
                       else
                            break;
                  }

       if(temp.memid==mem)
                  {
         delNode(temp);//the node of the leaving member
                            break;
            -     }
       else
       {
           if(temp.nodeid!=0)
         temp=temp.sibling;
```

```
                              if(temp.memid==mem)
             {
                              delNode(temp);//the node of the leaving member
                                         break;
                                   }
          }
          a=0;
          do{
                              if(temp.nodeid!=0)
            temp=temp.parent;
            if(temp.nodeid==0)
            {
              System.out.println("member"+ mem +" not in group");
                   lo.println("member"+ mem +" not in group");
              return;
            }
            tempsib=temp.sibling;
            if(temp.nodeid>tempsib.nodeid)
            {
              temp=tempsib;
              a=1;
            }
          }while(a!=1);
        }
    }
```

To delete the node details:

```
void delNode(binaryNode t)throws IOException
  {
     binaryNode sponsor=new binaryNode(0);
     binaryNode leaving=new binaryNode(0);
     binaryNode leavingSib=new binaryNode(0);
     binaryNode parent=new binaryNode(0);
     binaryNode temp=new binaryNode(0);
          binaryNode sponsorright=new binaryNode(0);
     leaving=t;
     leavingSib=t.sibling;
     temp=parent=t.parent;
```

```
      leaving=null;
      parent.memid=leavingSib.memid;
      parent.left=leavingSib.left;
      parent.right=leavingSib.right;
              leavingSib=leavingSib.left;
      if(leavingSib!=null)
         update(parent);
      sponsor=parent;
                     sponsorright=sponsor.right;
                     while(sponsorright!=null)
                     {
                             //System.out.println(sponsor.memid);
                             if(sponsor.right!=null)
                                     sponsor=sponsor.right;
                             else
                                     break;
                     }
              System.out.println("member "+t.memid+" left successfully");
              lo.println("member "+t.memid+" left successfully");
      rekey(sponsor);
   }
```

To update the node details after join/leave event:

```
 void update(binaryNode sponsor)
   {
      binaryNode temp=new binaryNode(0);
              binaryNode templeft=new binaryNode(0);
      binaryNode parent=new binaryNode(0);
      binaryNode sibling=new binaryNode(0);
      int label,labela,labelp;
      temp=sponsor;
      do{
         label=0;
                     templeft=temp.left;
                     while(templeft!=null)
                     {
                             if(temp.left!=null)
                             {
                                     temp=temp.left;
```

```
                    parent=temp.parent;
                    temp.nodeid=(2*parent.nodeid)+1;
               }
               else
                    break;
          }

do{
   labela=0;
   sibling=temp.sibling;
   if(Math.abs(sibling.nodeid-temp.nodeid)==1)
   {
      if(temp.nodeid==sponsor.nodeid)
         return;
      else
         temp=sibling.parent;
      labela=1;
   }
}while(labela==1);

if(temp.nodeid%2==1)
   sibling.nodeid=temp.nodeid+1;
else
   sibling.nodeid=temp.nodeid-1;
if(sibling.left!=null)
   temp=sibling;

else
{
   do{
      labelp=0;
      if(temp.nodeid==sponsor.nodeid)
         return;
      else
         temp=sibling.parent;
      sibling=temp.sibling;
      if(Math.abs(sibling.nodeid-temp.nodeid)==1)
         labelp=1;
   }while(labelp==1);
   temp=sibling;
```

```
            parent=temp.parent;
            if(temp==parent.left)
               temp.nodeid=(2*parent.nodeid)+1;
            else
               temp.nodeid=(2*parent.nodeid)+2;
         }
         label=1;
      }while(label==1);

   }
```

To perform Individual Rekeying:

```
private void jrunActionPerformed(java.awt.event.ActionEvent evt) {

     try{
              Random randomGenerator = new Random();


      int m1 = randomGenerator.nextInt(50);
              int first=0;//for counter operation
              long startTime=0;
              long estimatedTime;
              int m,ck=0,iterator=1;
              startTime = System.currentTimeMillis();

System.out.println("Building tree with rootvalue " +
rootnode.nodeid+";"+rootnode.memid);
    jrun.setVisible(false);
    jexit.setVisible(false);

              while(iterator==1)
              {

                     int n= randomGenerator.nextInt(2);
                     switch(n)
                     {
                            case 1:
                                   m = randomGenerator.nextInt(50)+1;
                     System.out.println("member "+m+" going to join");
```

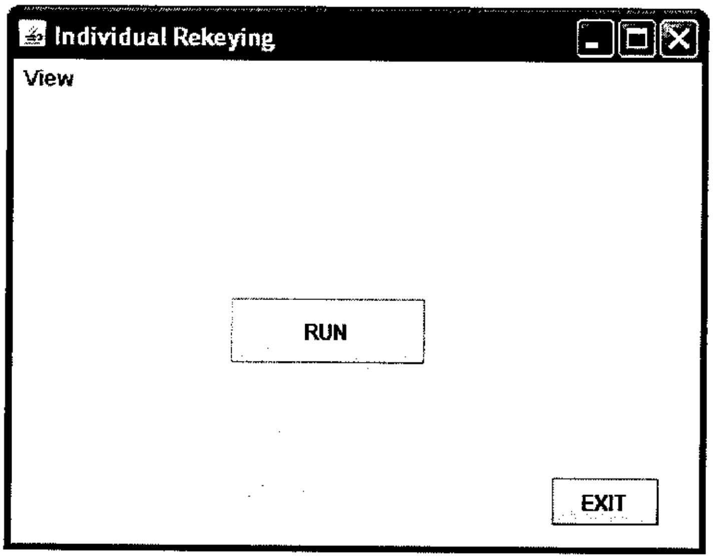
```
          lo.println("member "+m+" going to join");
         System.out.println("the value of ck is:"+ck);
                 ck=check(rootnode,m);
         System.out.println("the value of ck is:"+ck);
                 if(ck==0)
                 {
binaryNode insertion=insert(rootnode);
binaryNode newNode=new binaryNode(m);
join(insertion,newNode);
System.out.println("member "+m+" joined successfully");
lo.println("member "+m+" joined successfully");
                 printInOrder(rootnode);

               repaint();

                 }
                 else
                 {
            checker=0;
    System.out.println("member"+m+" already in group");
    lo.println("member"+m+" already in group");
                 }

                 break;
          case 0:
                 m = randomGenerator.nextInt(50)+1;
    System.out.println("member "+m+" going to leave");
                 lo.println("member "+m+" going to leave");
                 findLeave(rootnode,m);
                 printInOrder(rootnode);

         repaint();

                 break;
          default:
    System.out.println("enter the appropriate choice");
    }
    estimatedTime = System.currentTimeMillis() - startTime;
    if(estimatedTime<=1000)
          iterator=1;
```

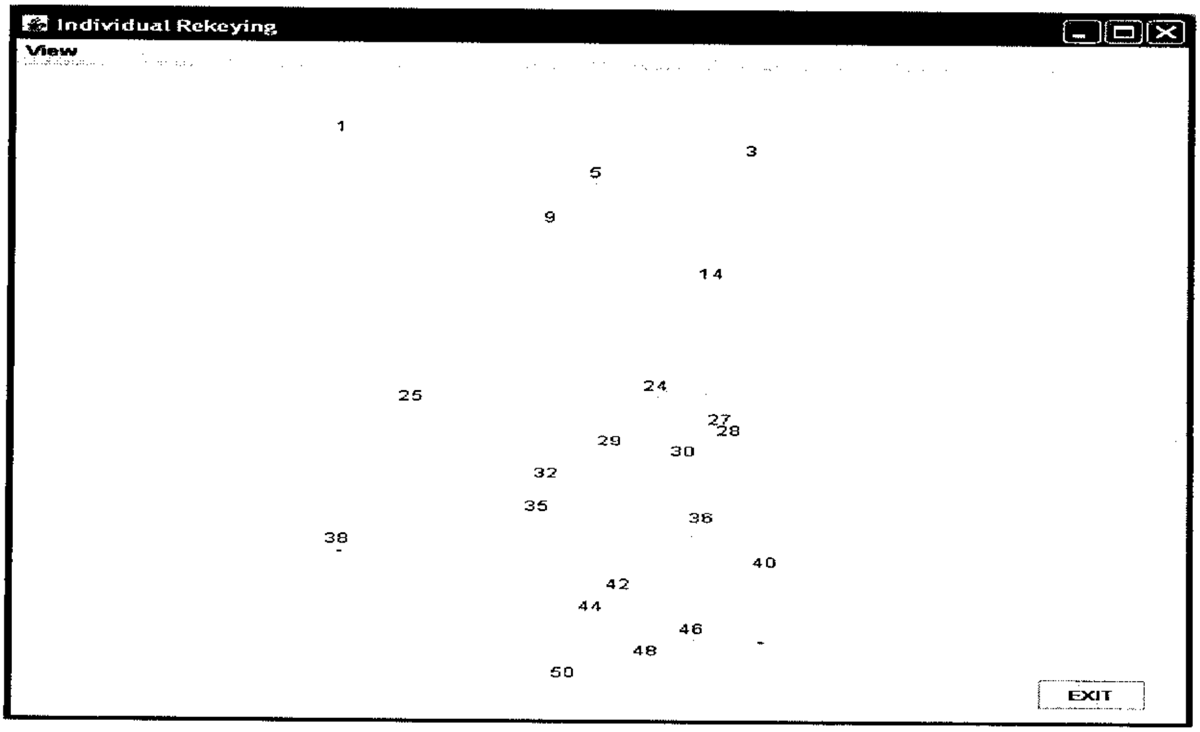
```
                    else
                 {
                   iterator=0;

                    jOptionPane1.showMessageDialog(null,"Processing
Finished");

                    repaint();
                     jrun.setVisible(true);
                    jexit.setVisible(true);
                 }
                      }

    }
    catch(Exception e){}
   }
```

To view all the members in the group:

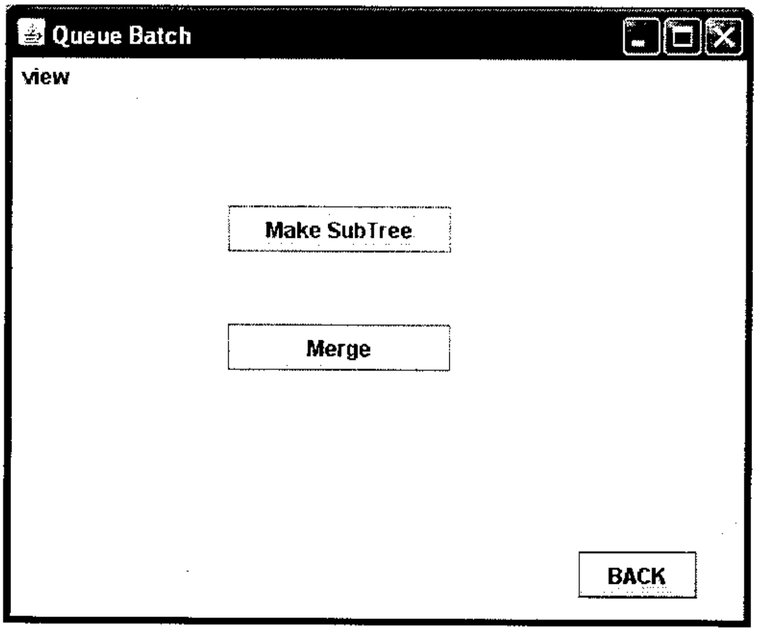
```
private void jviewallActionPerformed(java.awt.event.ActionEvent evt) {
      try{
         jrun.setVisible(true);
       jInternalFrame1.setVisible(false);
        s=" ";
       printInOrder(rootnode);
       jOptionPane2.showMessageDialog(null,"Members "+ s);

      }
      catch(Exception e){}
   }
```

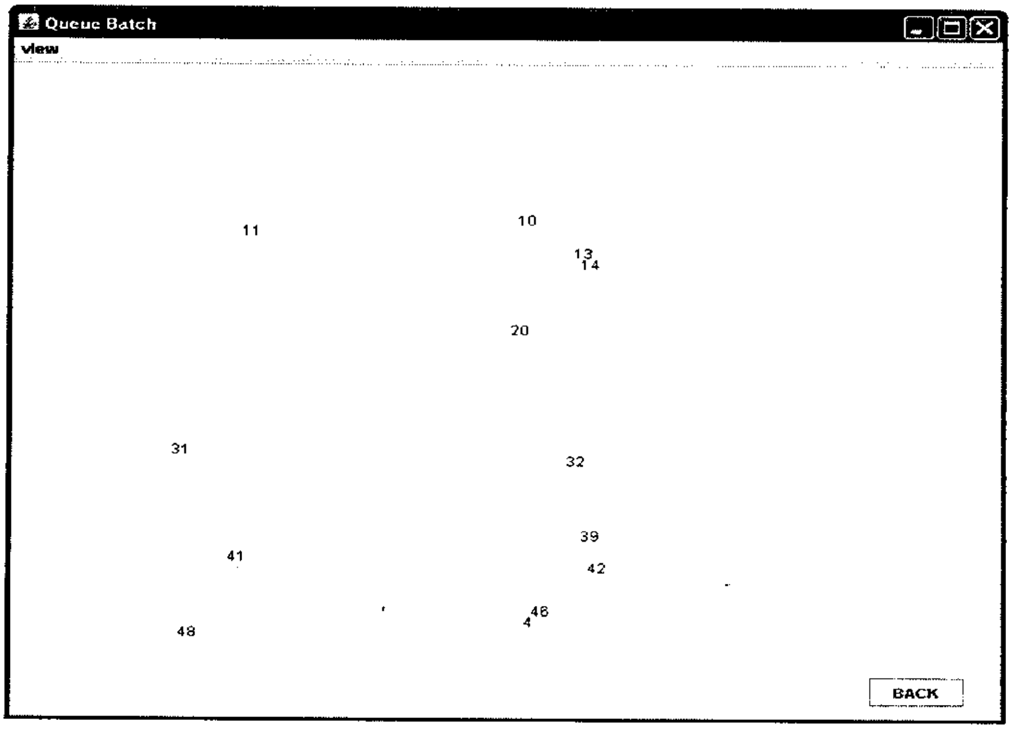
INDIVIDUAL REKEYING



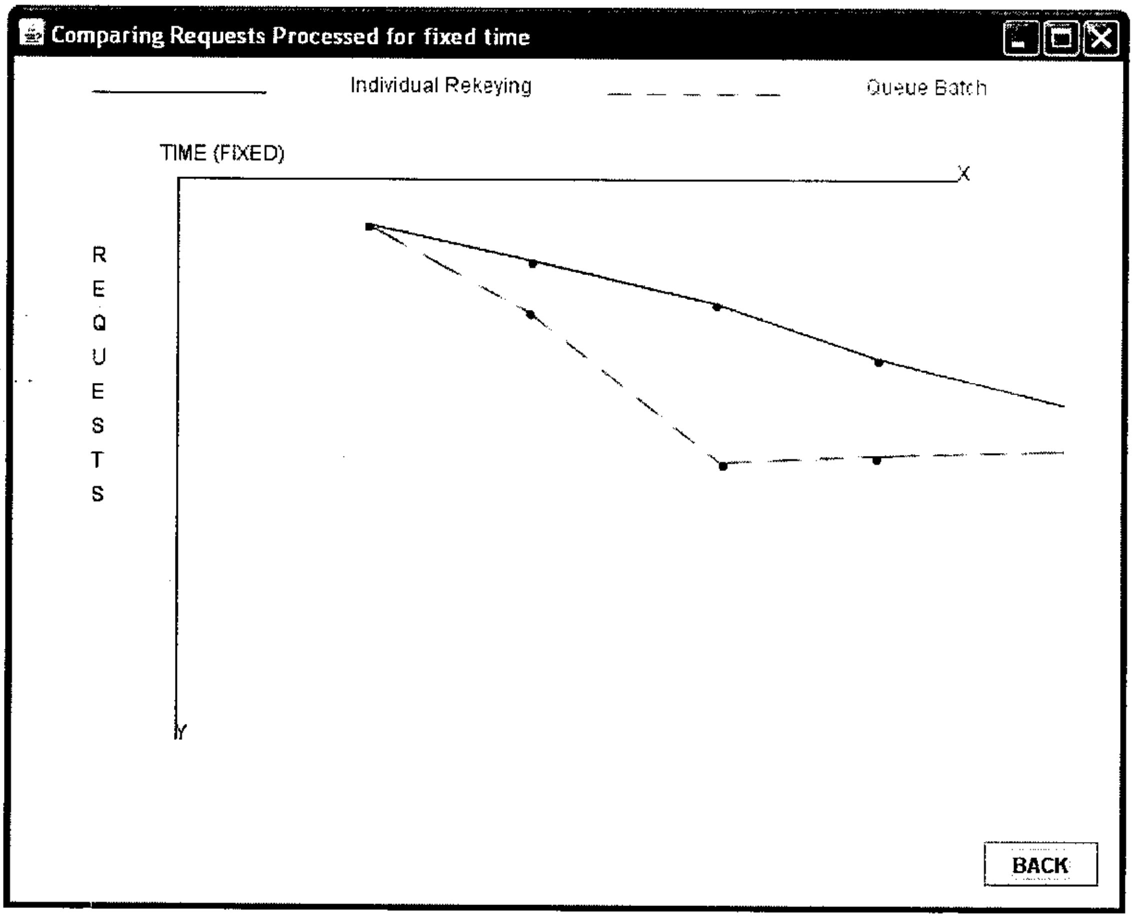


QUEUE BATCH REKEYING

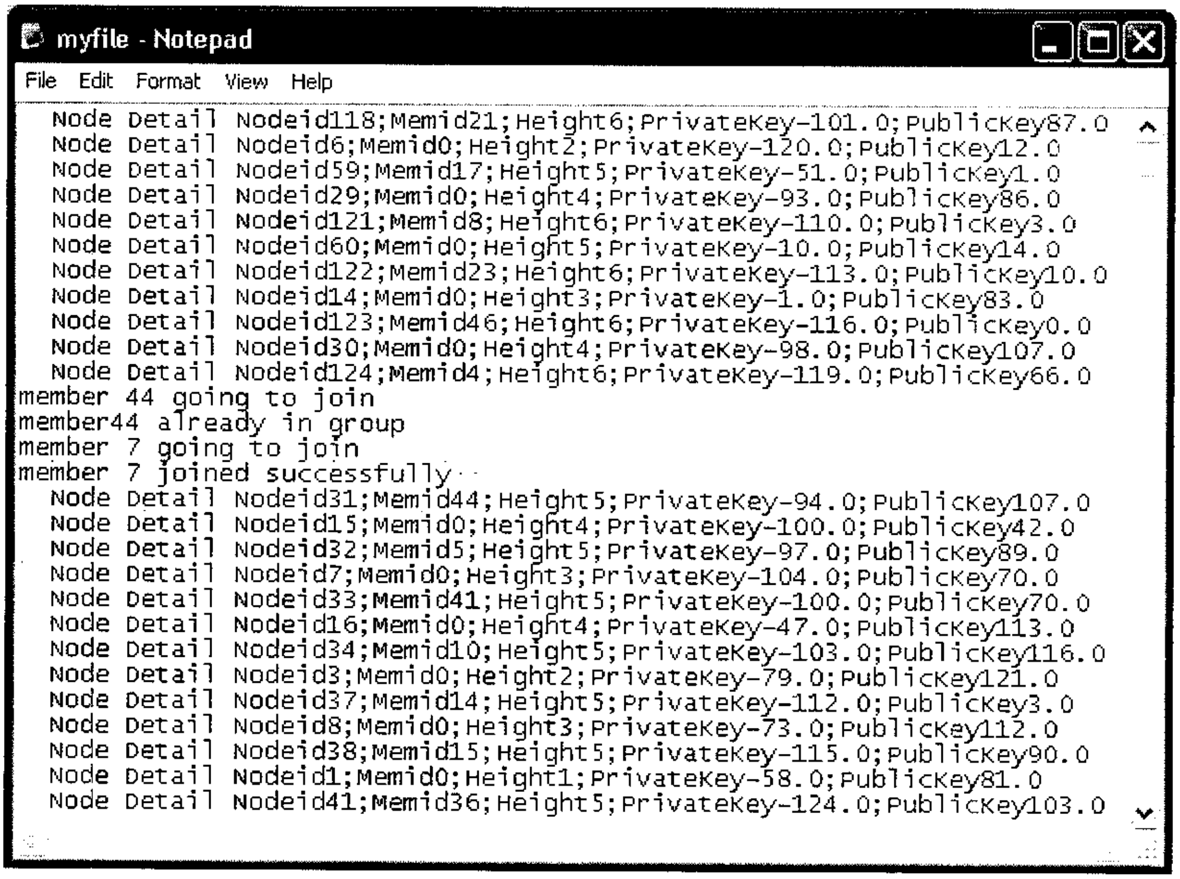




PERFORMANCE COMPARISON



LOG FILE



```
   Node Detail Nodeid118;Memid21;Height6;PrivateKey-101.0;PublicKey87.0
   Node Detail Nodeid6;Memid0;Height2;PrivateKey-120.0;PublicKey12.0
   Node Detail Nodeid59;Memid17;Height5;PrivateKey-51.0;PublicKey1.0
   Node Detail Nodeid29;Memid0;Height4;PrivateKey-93.0;PublicKey86.0
   Node Detail Nodeid121;Memid8;Height6;PrivateKey-110.0;PublicKey3.0
   Node Detail Nodeid60;Memid0;Height5;PrivateKey-10.0;PublicKey14.0
   Node Detail Nodeid122;Memid23;Height6;PrivateKey-113.0;PublicKey10.0
   Node Detail Nodeid14;Memid0;Height3;PrivateKey-1.0;PublicKey83.0
   Node Detail Nodeid123;Memid46;Height6;PrivateKey-116.0;PublicKey0.0
   Node Detail Nodeid30;Memid0;Height4;PrivateKey-98.0;PublicKey107.0
   Node Detail Nodeid124;Memid4;Height6;PrivateKey-119.0;PublicKey66.0
member 44 going to join
member44 already in group
member 7 going to join
member 7 joined successfully
   Node Detail Nodeid31;Memid44;Height5;PrivateKey-94.0;PublicKey107.0
   Node Detail Nodeid15;Memid0;Height4;PrivateKey-100.0;PublicKey42.0
   Node Detail Nodeid32;Memid5;Height5;PrivateKey-97.0;PublicKey89.0
   Node Detail Nodeid7;Memid0;Height3;PrivateKey-104.0;PublicKey70.0
   Node Detail Nodeid33;Memid41;Height5;PrivateKey-100.0;PublicKey70.0
   Node Detail Nodeid16;Memid0;Height4;PrivateKey-47.0;PublicKey113.0
   Node Detail Nodeid34;Memid10;Height5;PrivateKey-103.0;PublicKey116.0
   Node Detail Nodeid3;Memid0;Height2;PrivateKey-79.0;PublicKey121.0
   Node Detail Nodeid37;Memid14;Height5;PrivateKey-112.0;PublicKey3.0
   Node Detail Nodeid8;Memid0;Height3;PrivateKey-73.0;PublicKey112.0
   Node Detail Nodeid38;Memid15;Height5;PrivateKey-115.0;PublicKey90.0
   Node Detail Nodeid1;Memid0;Height1;PrivateKey-58.0;PublicKey81.0
   Node Detail Nodeid41;Memid36;Height5;PrivateKey-124.0;PublicKey103.0
```

REFERENCES

7. REFERENCES :

- P. P. C. Lee, J. C. S. Lui, and D. K. Y. Yau. *Distributed Collaborative Key Agreement and Authentication Protocols for Dynamic Peer Groups*. Networking, IEEE/ACM Transactions on Volume 14, Issue 2, April 2006

- Zhang,Liu,Lou and Fang. *Securing Mobile Ad Hoc Networks with Certificateless Public Keys*, Dependable and Secure Computing, IEEE Transactions Oct.-Dec. 2006. Volume: 3, Issue:4

- Y. Kim, A. Perrig, and G. Tsudik. Tree-Based Group Key Agreement. *ACM Trans. on Information and System Security*, 7(1):60–96, Feb 2004.

- P. Lee. Distributed and Collaborative Key Agreement Protocols with Authentication and Implementation for Dynamic Peer Groups. MPhil Thesis, The Chinese University of Hong Kong, June 2003.

- P. P. C. Lee, J. C. S. Lui, and D. K. Y. Yau. Distributed Collaborative Key Agreement and Authentication Protocols for Dynamic Peer Groups. CS&E Technical Report, The Chinese University of Hong Kong, Jul 2005.

- Y. Kim, A. Perrig, and G. Tsudik. Communication-Efficient Group Key Agreement. In *Proceedings of the 17th International Information Security Conference IFIP SEC'01*, Nov 2001.

- M. Steiner, G. Tsudik, and M. Waidner. Key Agreement in Dynamic Peer Groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8):769–780, Aug 2000.

- http://roseindia.com Dated Jan 2009

- http://java.sun.com Dated Jan 2009