

**Republic of Iraq
Ministry of Higher Education
and Scientific Research
Al-Nahrain University
College of Science**



A LAN-Based Instant Messaging System with Some Security Aspects

*A Thesis
Submitted to the College of Science, Al-Nahrain University
In Partial Fulfillment of the Requirements for
The Degree of Master of Science in Computer Science*

**By
Marwa Saad Malki AL-Kas
(B.Sc. 2005)**

Supervisor

Dr. Abeer M. Yousif

Dr. Jamal M. Kadhem

October 2008

Shawwal 1429

Dedication.....

To My
Dear Parent
Sister Dahlia
Brother Laith
Everyone who help
and support me.

Marwa



Acknowledgment

First, I would like to thank God, for all the blessings that have given us and enabled me to achieve this research work.

I would like to thank my supervisors, **Dr.Abeer M. Yousif** and **Dr.Jamal M. Kadhem**, for giving me opportunity to work on this interesting subject. Dr.Abeer's and Dr.Jamal's discussion, directions, reviews and valuable comments help direct me to the right path to accomplish this work.

Grateful thanks for the Head of Department of Computer Science **Dr.Taha S. Bashaga**, staff and my friends for continuous support and encouragement.

After all, I want to give my biggest thanks to my father and my mother for their support and patience during the period of my study.

Marwa

Abstract

Instant messaging is a form of online, real time form of communication between two or more people based on typed text. IM system has grown rapidly among network users. But most of existing instant messaging systems have severe security problems, people want to retain their privacy and communication should not copy or modified by a third party.

This thesis presents the design and implementation of a secure instant messaging system. It achieved security objectives such as data integrity and confidentiality through encryption. It ensures that the conversation is only read by intended recipient. The name of the proposed system is chosen to be *SIMSM* (the acronym for **SIM**ple **S**ecure **M**essenger).

SIMSM is designed for local networks. Based on client-server connection, it enables users to send and receive secure instant messaging between them. No internet connection is required. Easily in sending and receiving text messages. It supports standard messaging features such as private chat, group chat (conference), message notification and encryption.

The primary constituting modules are: *Registration Module* which identifies users to the system; *Login Module* which allows users to access the system; *Sign-In Problem Module* that deals with forgetting the identification (ID) and password problem; *Private Chat Module* that enable private chat between online users; and *Conference Chat Module* that enable more than one online user to chat with each others.

The proposed IM system has been evaluated according to two important factors in instant messaging: security and time consuming. Many test cases were taken to show that SIMSM is quite suitable for secure chatting service. The proposed secure instant messaging had been established using windows API functions with Java platform version 6.0.0.105 programming language.

List of Abbreviations

<u>Abbreviation</u>	<u>Meaning</u>
3DES	<i>Triple Data Encryption Standard</i>
AES	<i>Advanced Encryption Standard</i>
AIM	<i>American Instant Messaging</i>
ANSI	<i>American National Standards Institute</i>
AOL	<i>American OnLine</i>
API	<i>Application Programming Interface</i>
ATM	<i>Automated Teller Machine</i>
BOS	<i>Basic OSCAR Service</i>
DES	<i>Data Encryption Standard</i>
DH	<i>Diffie-Hellman</i>
DSA	<i>Digital Signature Algorithm</i>
DSS	<i>Digital Signature Standard</i>
EIM	<i>Enterprise Instant Messaging</i>
GPRS	<i>General Packet Radio Service</i>
GUI	<i>Graphic User Interface</i>
HTTPS	<i>Hyper Text Transfer Protocol Secure socket layer</i>
ICQ	<i>I Seek You</i>
IETF	<i>Internet Engineering Task Force</i>
IM	<i>Instant Messaging</i>
IMKE	<i>Instant Messaging Key Exchange</i>
IMPS	<i>Instant Messaging and Presence Service</i>
IMS	<i>Instant Messaging System</i>
IPSEC	<i>Internet Protocol SECURITY</i>
JDBC	<i>Java Data Base Connectivity</i>
JDK	<i>Java Development Kit</i>
JJC	<i>Java JabberC</i>

LAN	<i>Local Area Network</i>
MD	<i>Message Digest</i>
MIT	<i>Massachusetts Institute of Technology</i>
MSN	<i>MircoSoft Network</i>
NET	<i>NETwork</i>
NIST	<i>National Institute of Standards and Technology</i>
OMA	<i>Open Mobile Alliance</i>
OSCAR	<i>Open System for Communication in Realtime</i>
PEM	<i>Privacy Enhanced Mail</i>
PGP	<i>Pretty Good Privacy</i>
POS	<i>Point Of Sale</i>
RSA	<i>Rivest Shamir Adleman</i>
SHA	<i>Secure Hash Algorithm</i>
SHS	<i>Secure Hash Standard</i>
SKIP	<i>Simple Key Management for Internet Protocols</i>
SSL	<i>Secure Socket Layer</i>
TCP/IP	<i>Transmission Control Protocol/ Internet Protocol</i>
UDP	<i>User Datagram Protocol</i>
URL	<i>Uniform Resource Locator</i>
VPN	<i>Virtual Private Network</i>
XCP	<i>EXtensible Communications Platform</i>
XML	<i>Extensible Markup Language</i>
YAHOO	<i>Yet Another Hierarchical Officious Oracle</i>

List of Figures

<u>Figure name</u>	<u>Page</u>
(2.1) Symmetric key encryption	21
(2.2) Asymmetric key encryption	24
(2.3) Digital signature	27
(2.4) The Diffie-Hellman key exchange algorithm	30
(3.1) SIMSM global design	35
(3.2) SIMSM structure	39
(3.3) Diffie-Hellman man-in-the-middle attack	65
(3.4) Encrypt DH public key	65
(3.5) Generating shared key between user A and user B	66
(4.1) Server login frame	72
(4.2) Server main frame	73
(4.3) Server about frame	74
(4.4) Log of users	74
(4.5) View log by performing a specific query	75
(4.6) SIMSM main form	76
(4.7) Error login message	77
(4.8) Exit message	77
(4.9) Registration frame	78
(4.10) Password error message	79
(4.11) Successful registration message	80
(4.12) Sign-In problem frame	80
(4.13) Changing password frame	82
(4.14) SIMSM chat frame	83
(4.15) SIMSM first menu	84
(4.16) Conference chat menu	85
(4.17) Skin style menu	86
(4.18) Motif and Metal skin	86
(4.19) Help menu	87
(4.20) About flash window	87
(4.21) Invite users to conference	88

(4.22) Adding users to selected list	88
(4.23) Conference invitation message	89
(4.24) Accept the invitation	89
(4.25) Starting conference chat	90
(4.26) Deny joining the conference	90
(4.27) Private chat window	91
(4.28) Smile faces code	92
(4.29) Protecting a password in SIMSM	94

List of Tables

<u>Table name</u>	<u>Page</u>
(2.1) Advantages and disadvantages of symmetric key cryptography	22
(2.2) Advantages and disadvantages of the public key cryptography	24
(3.1) User message object fields	51
(3.2) User object fields	51
(3.3) User status	51
(3.4) Headers types in message object	52
(4.1) Time to register one user	96
(4.2) Time to register five users	96
(4.3) Time for authentication process	97
(4.4) Time to login five uses	98
(4.5) One-to-One chatting	99
(4.6) Five-to-One chatting	99
(4.7) Chatting process	100

List of Algorithms

<u>Algorithm name</u>	<u>Page</u>
(3.1) Asymmetric Key Maker	41
(3.2) Receiving Registered Information	41
(3.3) RSA Decryption	43
(3.4) Check ID Existence	43
(3.5) Store In DataBase	44
(3.6) Sign-In Problem Thread	45
(3.7) Changing Password	46
(3.8) Retrieving User ID	47
(3.9) Check Authentication Process	48
(3.10) Look Up Password	49
(3.11) Make Digest Message	50
(3.12) Process Client Message	53
(3.13) Write to Clients	54
(3.14) Invite to Conference Chat	54
(3.15) Write to Conference Chat	55
(3.16) Request Conference List to Client	55
(3.17) Write to Client	56
(3.18) Encryption	58
(3.19) Protect Login Process	59
(3.20) Message Digest	60
(3.21) Generate DH Key Pair	61
(3.22) Process Server Message	62
(3.23) Sign-In Problem	63
(3.24) Generate Secret Key	66
(3.25) Decryption Message	68

List of Contents

<i>C H A P T E R 1: Introduction to Instant messaging</i>	<i>Page</i>
1.1 Overview to Instant Messaging	1
1.2 Main Types of IM Applications	2
1.3 Instant Messaging Services	3
1.3.1 Instant Message	3
1.3.2 Chat Session	4
1.3.3 ChatRoom	4
1.4 IM Network Components	5
1.4.1 IM Servers	5
1.4.2 IM Clients	5
1.5 Server Configuration	5
1.6 Basic IM Features	6
1.7 Risks of using IM	7
1.8 Related Works	8
1.9 Aim of Thesis	11
1.10 Thesis Layout	12

C H A P T E R 2: Secure Instant Messaging System

2.1 Introduction	13
2.2 Security concepts and requirements	14
2.3 Authentication Methods	15
2.3.1 Password	15
2.3.2 Public Key Cryptography	17
2.3.3 Zero-Knowledge proof	17
2.3.4 Digital Signature	18
2.4 IM Security Solutions	18
2.4.1 Secure Protocol Solutions	18
2.4.2 Third-Party Solutions	19
2.4.3 Cryptography Solution	19
2.5 Taxonomy of Cryptography	20
2.5.1 Hash Functions	20

2.5.2 Symmetric Key Encryption	21
2.5.3 Asymmetric Key Encryption	24
2.6 Diffie-Hellman Key Exchange	27
2.6.1 Mathematical Concepts	28
2.6.2 Diffie-Hellman Key Exchange Algorithm	30
2.6.3 Simple Key Management for Internet Protocols (SKIP)	31
2.6.4 The security of Diffie-Hellman	32

CHAPTER 3: Design and Implementation of SIMSM

3.1 Introduction	34
3.2 SIMSM Global Design	34
3.3 SIMSM Requirements	36
3.4 Features of SIMSM system	37
3.5 SIMSM Structure	37
3.6 Server Program	40
3.6.1 Initialization Module	40
3.6.2 Check Authentication Module	47
3.6.3 User Information Module	50
3.6.4 Process Client Message Module	52
3.7 Client Program	57
3.7.1 Registration Module	57
3.7.2 Login Module	59
3.7.3 Sign-In Problem Module	63
3.7.4 Secure Private Chat Module	64
3.7.5 Conference Chat Module	68

CHAPTER 4: SIMSM Interface and Evaluation

4.1 SIMSM Interfaces	72
4.1.1 SIMSM Server Program Interface	72
4.1.2 SIMSM Client Program Interfaces	76
4.2 SIMSM Evaluation	92
4.2.1 Security Factor	92

4.2.2 Time Consuming Factor 95

CHAPTER 5: Conclusion and Suggestion for Future Work

5.1 Conclusion 101
5.2 Future Work 102

List of Reference

Appendix A: Database Connection

Chapter One

Introduction To Instant Messaging

1.1 Overview to Instant Messaging

Instant Messaging (IM) is a form of online, real time communication. It allows users to see whether their contacts are online, and to send them a typed message. It is similar to email as a communication tool, but, unlike traditional email, is instantaneous. Messages sent via IM appear immediately on the recipient's computer screen. In this way, IM is a truly synchronous (that is, real time) form of communication [Dav07]. Instant messaging is a combination of the telephone, which facilitates conversations with multiple people in real time, and of e-mail, which combines the speed of online communication with a written record of your conversation [Fly04].

Instant Messaging also known as a "chatting," has become very popular for both business and personal use. In business, there is variety of benefits associated with the use of IM in business, not least of which is the ability to communicate more efficiently, but also for faster problem solving and decision making [Ost06].

Popular messengers are include American Online Instant Messenger (AIM), Windows Live Messenger (formerly MicroSoft Network (MSN) Messenger), Yahoo! Messenger and I Seek You (ICQ) Messenger. These individual messengers generally don't interoperate, so that someone using Windows Live Messenger, for example, cannot chat to someone using Google Talk. However, there are a third-party client (also known as aggregator or multi-network clients) that allows you to log in to accounts on many different networks simultaneously.

Beside the ability to send typed messages backward and forwards, IM also allows the user to transfer files, white boarding, audio and video conferencing, and other IM features [Dav07].

IM provides convenient communication between people using a variety of different device types. The most familiar form today is computer-to-computer instant text messaging, but IM also can work with mobile devices, such as digital cellular phones, and can incorporate voice or video [IEC04].

1.2 Main Types of IM Applications

There are two main types of IM applications: *Public* and *Enterprise* IM applications [Sha05].

1- Public applications are free and able to be downloaded off the Internet.

They are the most widely used type of IM application in both the home and workplace. Examples of public IM are American OnLine (AOL) Instant Messenger (AIM), Yahoo! Messenger, MSN, Google Chat, and others. The proprietary services in public IM networks offered by major Internet service providers, including AOL, Yahoo! [Wil04].

2- Enterprise IM (EIM) application solutions were created to specifically target the security problems that public IM posed on organizations. Enterprise IMs are not free and must be purchased as with any proprietary software package. IBM Lotus Sametime, Microsoft Office Live Communications Server, and Jabber Extensible Communications Platform (Jabber XCP) are examples of EIM applications. Enterprise IM systems are provided by an internal software solution that is owned and controlled by the enterprise [Wha08].

1.3 Instant Messaging Services

An instant messaging system is a combination of two services: *Presence Service* and *Instant Messaging Service*. Presence service, allows its users to learn who is actually online and to whom an instant message can be sent [Wrz02]. It's serves to accept information, store it, and distribute it. Examples of this type of information would be client status (online, away, busy, etc.), user name, public profile information, and so on. Basically, this is what users are allowed to see or know about other users. Instant messaging service is a means of sending small and simple messages that are delivered immediately to online users [Rit05]. The instant messaging service makes use of the presence service to determine which users are online and which can be therefore contacted [Wrz02].

In the following sections, types of communication in Instant Messaging Service will be explained [Wrz02].

1.3.1 Instant message

An *instant message* is a **small unit of data** exchanged by the users of an instant messaging system. An instant message consists mainly of plain text. It can also contain other elements. For example, a system can support Uniform Resource Locator (URL) which can be recognized by the client-side software and marked out in the text displayed to the user. A user can launch the application by clicking on the URL [Wrz02].

1.3.2 Chat session

Instant messaging systems also support a concept of *chat session* or, to use more human-friendly terminology, a conversation. In a chat session, a **sequence of instant messages** exchanged by two users can be recognized and presented to them as a coherent whole. The client-side software of an instant messenger presents all the messages of the session as a whole by, for

example, displaying them in one window. To hold such a conversation, a user *creates* a chat session and then *invites* another user to join it. The other user can choose between *joining* the session and *rejecting* the invitation. If he/she joins, the session is established and the users can send and receive messages within that session. When a user wants to terminate a conversation, he/she *leaves* the session [Wrz02].

1.3.3 Chatroom

A *chatroom* can be seen as the persistent version of a multi-party chat session. A chatroom is a virtual place where people can meet and talk. A chatroom itself is not a group but rather a meeting point that the user to easily initiate a conversation. A chatroom is public, that is to say, directly accessible from the outside without further need of being invited by a current group member. When a user wants to take part in a chatroom conversation, he/she *enters* the chatroom. Every message sent within this chatroom is delivered to this user. Messages sent by the user within this chatroom are delivered to all other users of the chatroom [Wrz02].

1.4 IM Network Components

The basic components in most IM networks are: *Server* and *Client* [Wil04].

1.4.1 IM Servers

IM servers provide IM services such as presence notification, account registration and instant messaging. A client must connect to the server in order to be given an online presence status on the IM network. An IM network may have multiple servers working in tandem to provide scalability which is the ability to handle large numbers of users distributed over geographically large area. Transfer between servers is generally transparent, meaning principles have no knowledge as to which server they may be

connected to. The servers of the public IM networks are located on the premises of the organization providing the service. For example, the MSN servers are located in an area chosen by Microsoft.

1.4.2 IM Clients

The client software is the public face of an IM network. Once installed on a workstation, the client software can be used to connect to the IM server to access IM services. Clients are also able to store specific user information locally. This information includes passwords and account names (for automated login) and records of chat sessions. Clients for the public IM services are available as a free download on the Internet.

1.5 Server Configuration

Server configuration explains how servers, the backbone of an IM network, function in the network hierarchy.

There are two types of server configurations: *Single Server Architecture* and *Multiple Server Architecture* [Wil04].

In single server architecture, all IM tasks can be conducted on a single server; including account creation, authentication and IM services. Single server architecture allows for easier management and increased security but is not a scalable solution and may struggle with a large load of users.

While in multiple server architecture most large IM networks operate using multiple servers. The role of servers in multiple server architecture can be divided into two groups: *Replicated Servers* and *Distributed Services*. In *Replicated Servers*, individual servers capable of providing all IM functionalities can be replicated and interconnected to support a large network. While in *Distributed Services*; servers may be divided based on the role that they play within the IM network. One server type may fulfill the

function of account registration or login, while another server type may provide notification and messaging [Wil04].

1.6 Basic IM Features

The following features are existing relatively in the most popular IM software products currently in operation [Rit05].

1- Instant Messaging (*also known as IM*): The transmission of text from one user to another via an IM service. These messages generally have no security and are routed over the Internet.

2- Voice/Video Chat: A direct connection must be established between two users to enable voice/video chat. The data is typically transferred via User Datagram Protocol (UDP) connections. AIM does not support video chat, but it does support voice chat.

3- File Transfers: File transfers require a direct connection to be established between users. However, once a file transfer is completed, the direct connection is closed.

4- File Sharing: File sharing allows a user to browse a selected directory structure and download files. File sharing is an optional capability that must be enabled in AIM and ICQ before any sharing can take place. However, file sharing is enabled by default in Yahoo! The connection method for file sharing is the same as for a regular file transfer.

1.7 Risks of Using IM

Although Instant Messaging (IM) has significant advantages, including ease of use and instantaneous communication, it also provides a significant security risk [Rit05]. Many of IM protocols not design to carry sensitive data in a corporate environment, and therefore do not offer encryption or other security features [Pic02]

Because of public IM is free, easy to use, and availability, the most existing popular public instant messaging systems, such as AIM, MSN, Yahoo, and ICQ Messenger, cannot be considered secure [Wrz02]. For examples in AOL instant Messenger (AIM), there are two types of servers on the AIM network: the OSCAR (Open System for Communication in Realtime) server, and the BOS (Basic OSCAR Service) servers. While the OSCAR is responsible for authorizing clients, the BOS is responsible for handling various features of the AIM service. All the communication sent or received via AIM has no encryption. Also the Yahoo! Messenger has the weakest security features of the major messaging platforms. Its protocol does not encrypt the password, and the messaging is simply passing in clear text messages from one user to another, it means it has *Unencrypted communication* risk. All the popular IM mentioned above have another two risks: *Social Engineering*, and the *Theft of Identity* [Oll04].

Social Engineering means some malicious IM users have convinced others to divulge sensitive information, such as username, passwords, and credit card numbers, and due to social engineering the theft of identity occur. There are several utilities that enable a malicious use to impersonate another user. This can also lead to more serious social engineering issues, where a user can mistakenly trust a malicious user and provide sensitive and confidential information [Pic02].

Companies using IM experience a reduction in phone bills, a reduction in voice mail minutes, and improved worker productivity through faster problem solving and decision making. But Unmanaged or unknown use of IM can also pose many risks to organizations. These risks including for examples leakage of confidential information, potential productivity loss as employees chat with family and friends, legal and corporate exposure from inappropriate use of IM, and other security risks [Sha05].

1.8 Related Works

Various efforts in the field of providing secure instant messaging system were introduced; some of these efforts are summarized below:

- 1- **Wrzesinska, (2002), [Wrz02]**, his project was aimed to introduce an instant messaging system that is both secure and scalable. Security in his system relies mostly on using security protocol called SSL (Secure Socket Layer). Scalability in his system achieved through the distribution of its components, multiple servers, and distributed database.
- 2- **Symantec Inc., (2002), [Sym02]**, Symantec describes significant vulnerabilities that are present in common instant messaging system and the type of attacks that can exploit them. Also explores the security issues introduced with the use on instant messaging and offers best practices that can help in deploying a secure IM platform within enterprise. Examples for these practices are: *Establish a corporate IM usage policy, Properly configure perimeter firewalls, Deploy desktop antivirus software, Deploy corporate IM servers, Employ personal firewalls, and others practices.*
- 3- **Kim, and et al (2003), [Kim03]**, they introduced a way to make data, sent over various networks via instant messaging clients, encrypted and a way to authenticate users on the already established system. They decided to use Rivest Shamir Adleman (RSA) asymmetric key encryption algorithm for authentication and Advanced Encryption Standard (AES) symmetric key encryption algorithm for communication once authentication is completed.
- 4- **Almanei, (2003), [Alm03]**, his project is to present a secure working Jabber client, based on Jabber Protocol, which is actually an Extensible Markup Language (XML) messaging protocol. He used an already

available client called Java JabberC (JJC) client, and modifying the code of this client by using Diffie-Hellman key exchange and Data Encryption Standard (DES) algorithms for key exchange and encryption, decryption respectively to satisfy a secure Java JabberC. He used a cryptographic library of java, which provides a framework for key exchange and encryption.

- 5- **Kling, (2003), [Kli03]**, the aim of his project is to point out weaknesses in the ICQ instant messaging, he makes an investigate to the ICQ security and the result of his investigate showed that the ICQ had a non-secured authentication phase, non-secured messages and no protection for stored messages. From these results, the main conclusion that he was derived is the use of the ICQ resulted in non-secure instant messaging sessions. Through his investigation he used a forensic technique to try to extract digital evidence and investigate if the ICQ history files is encrypted or unencrypted.
- 6- **Fredrik, (2003), [Fre03]**, his project was aimed to create an instant messaging client that implements the specification from OMA (Open Mobile Alliance), called Instant Messaging and Presence Service (IMPS). The purpose of this specification is to create an IM protocol that should be available to the public and possible to implement on both mobile and stationary devices. The security in OMA IMPS has many different levels of cryptology. On the lowest level, encryption of the network traffic is used by GPRS (General Packet Radio Service) networks, on the system level, it uses an encrypted protocol like HTTPS (Hyper Text Transfer Protocol Secure socket layer) that used to secure the communication with the server and on the highest level in the system, and this is done by ensuring that sensitive data never transmitted from the client. Since it

consider IMPS protocol as secure protocol, but one of the major problems with OMA IMPS is that the system may be setup in a way allowing the user to authenticating in non-secure way, giving an opportunity for malicious user to gain unauthorized access to the system. In other words, OMA IMPS offers support for both secure and not secure logging into the server and sending messages.

- 7- **Williams and Ly (2004), [Wil04]**, they studied the security of IM applications within the workplace and suggested a secure method of implementing a secure IM prototype that can be used at work. Their IM application developed based on Jabber, which is an open source IM system, and used DES encryption for java based that can be implemented to a public IM application to keep the information exchanged secure.
- 8- **Abdul Mannan, (2005), [Abd05]**, he presented in his research work the most significant threats to IM systems, and decided that there is need a technique to secure IM protocol, so he used an Instant Messaging Key Exchange (IMKE) protocol as a step toward secure IM. IMKE was designed to provide secure communication and authentication, and its implementation was done using the open source Jabber server and client on Linux operation system.

1.9 Aim of Thesis

This research work aims to design and implement a secure instant messaging system for local networks. This is done by adding standard encryption methods, which provides two basic services for users: a way to protect data between client-server and client - client from unintended viewers through encryption, and a way to authenticate users. This research work

aimed also to add some features and functionalities to the secure IM system like conference chat, transfer smiles with text and etc.

1.10 Thesis Layout

Beside chapter one, the remaining part of the thesis consists of the following chapters:

- ▼ **Chapter Two:** Concerned with the definition of secure instant messaging solutions and types of cryptography algorithms. At the end of this chapter, the Diffie-Hellman key exchanged algorithm is presented with its related mathematical concepts.
- ▼ **Chapter Three:** Introduces the design and implementation requirements of the established secure instant messaging system, and then the structure of the system will be explained together with modules and algorithms that are used to implement the system.
- ▼ **Chapter Four:** Presents user interface and evaluation of the designed system.
- ▼ **Chapter Five:** Introduces the derived conclusions and suggestions for future works are given.

Chapter Two

Secure Instant Messaging System

2.1 Introduction

In recent years, instant messaging systems have gained more and more popularity. Most of the existing systems have shown to have severe security problems with respect to user privacy, message authenticity and eavesdropping. Now, as security became more important, some efforts are being made to incorporate security features into IM systems [Wrz02].

One thing that must be kept in mind when designing a secure system is the users. As the number of users that use instant messaging is quickly growing and increasing, the number of security risks increases also [Leg04]. The persons using an instant messaging system might very well be the biggest security threat against the system. A system is never be secure if the people using it don't care about protecting it. Thus, the system must not only be designed with a high level of encryption and other security measures, it must also guard against its own users.

Also, one of the major security problems within the networks is the transmission of information in plaintext over insecure networks. The sensitive data such as account names or passwords are being transferred across any number of networks to reach the IM servers [Wil04]. So we need security because people want to retain their privacy. Communications should not be overheard, copied, blocked or modified by a third party. Therefore, there is a considerable effort being made to incorporate security into the existing communication systems and to create new secure communication tools.

2.2 Security concepts and requirements

There are some specific security requirements relating to network security, which are: Authentication, Authorization, Integrity, and Confidentiality [Can01].

1- Authentication: authentication serves as proof that you are who you say you are or what you claim to be. Authentication is critical if there is to be any trust between parties. Authentication is required when communicating over a network or logging onto an instant messaging systems. When logging onto a network, one or more of three basic schemes are used for authentication: something you know, something you have, and something you are [Can01 and Pip04].

- a. ***Something you know:*** The most commonly employed scheme is "something you know" Typically, the something you know that authenticates your identity is a password, code, or sequence. The security is predicated on the idea that if you know the secret password or code then you must be who you claim to be and be authorized to access the network.
- b. ***Something you have:*** "Something you have" requires a key, badge, or token card, some device or "thing" that provides you with access. Security is predicated on the concept that only authorized individuals or entities will have access to the specific device. The drawback to this scheme is that the "thing" can be lost or stolen.
- c. ***Something you are:*** "Something you are" authentication relies upon some physical or behavioral characteristic. It is referred to as *biometric authentication*. Biometrics refers to the automated identification of a person, on the basis of physiological and behavioral characteristics. The physiological characteristics could be face, fingerprints, hand geometry, handwriting, iris, retinal etc...[Kol01].

2- Authorization (Access Control): this refers to the ability to control the level of access that individuals or entities have to a network or system and how much information they can receive. Your level of authorization basically determines what you are allowed to do once you are authenticated and allowed access to a network, system, or some other resource such as data or information [Can01 and Wil03].

3- Integrity: assuring the receiver that the received message has not been altered in any way from the original [Kes07].

4- Confidentiality/Privacy: refers to the protection of information from unauthorized disclosure. Usually achieved either by restricting access to the information or by encrypting the information [Can01].

2.3 Authentication Methods

Authentication mechanisms can be classified in many ways; the following overview will generalize several authentication mechanisms to authenticate users over network [Dun02].

2.3.1 Password

One of the most commonly used authentication schemes employs passwords. Password can be classified into two main types: fixed password and one-time password.

1- Fixed Password

Along with a user name, a password must be presented to the authentication system to gain access [Fis00]. Authentication is based on knowing the (name, password) pair. The length and format of the password also vary from one system to another. The system compares the entered password against the expected response and reacts accordingly. If the password matches that on file for the user, the user is authenticated to the system. If the password match fails, the system requests the password again [Pfl97].

A fixed password system is vulnerable to interception and replay. The extent of the risk to which the system is exposed will depend on the deployment. If passwords are being transmitted across an unprotected network the risk is greater than with a closed system where there are limited opportunities for eavesdropping. Since user-chosen passwords are memorable, they are likely to contain some inherent structure. To help provide additional protection, some proposals deploy machine-generated passwords, but these are not especially popular. In fact, such approaches can sometimes be counter-productive since a password that is difficult to remember is sometimes written down, which might degrade the overall security of the system [Pip04].

There is a reason why passwords are so popular: they are fast, they are cheap, and, in practice, people don't forget them or lose the pieces of paper all that often. Username and passwords are an authentication solution for low-value transactions and for accessing non-sensitive online information [Zwi00].

2- One-time password

There are two types of one-time passwords, a *password list* and *challenge-response password* [Dun02].

- a.** The challenge-response password responds with a challenge value after receiving a user identifier. The response is then calculated from either the response value or select from a table based on the challenge.
- b.** A one-time password list makes use of lists of passwords, which are sequentially used by the person wanting to access a system. The values are generated so that it is very hard to calculate the next value from the previously presented values. It is important to keep in mind that password systems only authenticate the connecting party. It does not provide the connecting party with any method of authenticating the

system they are accessing, so it is vulnerable to spoofing or a man-in-middle attack.

2.3.2 Public Key Cryptography

Public key cryptography is based on complex mathematical problems that require very specialized knowledge. Public key cryptography makes use of two keys, one private and the other public. The private key is used to decrypt and also to encrypt messages between the communicating machines. Both encryption and verification of signature is accomplished with the public key. Basically, the public-key authentication process includes the following: Client selects some random numbers and sends the results to the server as a message: Message 1. The server then sends different random numbers back to the client based on Message 1. The Clients then computes the new value and sends Message 2 to the server. The Server then uses the clients public key to verify that the values returned could have only been computed using the private key. This authenticates the client to the server. If the client wants to authenticate the server, the same procedure is repeated with changed roles [Dun02].

2.3.3 Zero-Knowledge Proof

A zero-knowledge proof is an interactive method for one party to prove to another that a (usually mathematical) statement is true, without revealing anything other than the veracity of the statement [Zer08].

2.3.4 Digital Signature

Digital signature is completely analogous to a handwritten signature used for centuries to authenticate documents. The reason that signatures have been used is that, everyone's signature is unique and hard to forge and a document, which has been signed, would be hard to repudiate later. These

same assurances are desirable for computer generated and transmitted documents. It's liked to be able to prove that a document sent by somebody was indeed sent by that individual [Fis00].

2.4 IM Security Solutions

In order to provide secure instant messaging system, there are many solutions to make those systems relatively secure [Man04 and Wil04]: Secure Protocol Solutions, Third-Party Solutions, and Cryptography or Encryption Solutions.

2.4.1 Secure Protocol Solutions

Security protocol (cryptographic protocol or encryption protocol) is an abstract or concrete protocol that performs a security-related function and applies cryptographic methods. Protocols describe how the algorithms should be used. A sufficiently detailed protocol includes details about data structures and representations, at which point it can be used to implement multiple, interoperable versions of a program. Cryptographic protocols are widely used for secure application-level data transport [Wik07].

The most security protocols that used extensively in the real world are: SSL (Secure Socket Layer) which is used to secure most Internet transactions today. The second protocol is IPSEC (Internet Protocol SECurity), which is a complex and over-engineered protocol with several security flaws. The final real-world protocol is Kerberos, a popular authentication protocol built on symmetric keys cryptography [Sta06].

2.4.2 Third-Party Solutions

The third solution that makes an IM relatively secure is through using and recommending the following practices for securely deploying IM system within public and enterprise IM system [Man04 and Sha05].

1. Using Desktop antivirus software to scan IM file transfers for computer viruses, worms, such as Norton Anti Virus and ZoneAlarm, and these antiviruses must keep up-to-date.
2. Controlling instant messaging by stopping any unauthorized access and establishing and enforcing rules on current usage and defining policies, also stop inappropriate conversations by using keyword and phase filter to block people from discussing certain topics.
3. Recommended instant messaging client settings.
4. Establish a corporate instant messaging usage policy, example for enterprise use, given the risks involved in using public instant messaging systems, corporations should consider prohibiting the use of public instant messaging systems entirely, or ask employees to refrain from using public instant messaging systems for business communications.

2.4.3 Cryptography Solution

Cryptography is the study of methods for sending messages in *secret* (namely, in *enciphered* or *disguised* form) so that only the intended recipient can remove the disguise and read the message (or *decipher* it). The process of transforming plaintext into ciphertext is called *encryption* or *enciphering* [Mol07], which is one of solutions to make an instant messaging system secure.

2.5 Taxonomy of Cryptography

There are three categories of ciphers: *hash functions*, *Symmetric ciphers*, and *Asymmetric cipher* [Sta06].

2.5.1 Hash Functions

Hashing functions, also called *message digests* and *one-way encryption* are algorithms that do not use a key and refer to a mathematical function that takes a variable size string as input and transforms (hashes) it into a fixed-

size string, which is called the hash value, as output. In network security applications, the mathematical function should have the essential properties that the transformation is one-way and that it is computationally infeasible for two different inputs to produce the same output. One of the most common uses of hashing in network security is to produce condensed representations of messages or “fingerprints,” often known as “message digests” by applying a hashing algorithm to an arbitrary amount of data. The two most commonly used hashing algorithms are MD5 and SHA [Fun05].

1- Message Digest (MD) algorithms: A series of byte-oriented algorithms that produce a 128-bit hash value from an arbitrary-length message [Kes07]. MD5 creates a unique 128-bit message digest value derived from the contents of a message or file. This value, which is a fingerprint of the message or file content, is used to verify the integrity of the message's or file's contents. If a message or file is modified in any way, even a single bit, the MD5 cryptographic checksum for the message or file will be different. It is considered very difficult to alter a message or file in a way that will cause MD5 to generate the same result as was obtained for the original file. While MD5 is more secure than MD4, it too has been found to have some weaknesses: Analysis has found a collision in the compression function of MD5, although not for MD5 itself. Nevertheless, this attack casts doubts on the whether MD5 is truly a collision-resistant hash algorithm. The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA.

2- Secure Hash Standard (SHS): Algorithm for National Institute of Standards and Technology NIST's Secure Hash Standard (SHS). SHA-1 produces a 160-bit hash value, five algorithms in the SHS: SHA-1 plus SHA-224, SHA-256, SHA-384, and SHA-512 which can produce hash values that

are 224, 256, 384, 512 or 1024 bits in length [Kes07]. Although SHA-1 is slower than MD5, but this large digest size makes it stronger against brute force attacks, and therefore more secure than MD5 [Sec01]. The SHA-1 may be used with the DSA (Digital Signature Algorithm) in electronic mail, electronic funds transfer, software distribution, data storage, and other applications which require data integrity assurance and data origin authentication. The SHA-1 may also be used whenever it is necessary to generate a condensed version of a message [Fip02].

2.5.2 Symmetric Key Encryption

Symmetric key also referred to as private key or secret key is based on a single key and algorithm being shared between the parties who are exchanging encrypted information. The same key both encrypts and decrypts messages [Can01]. This concept is illustrated in Figure (2.1).

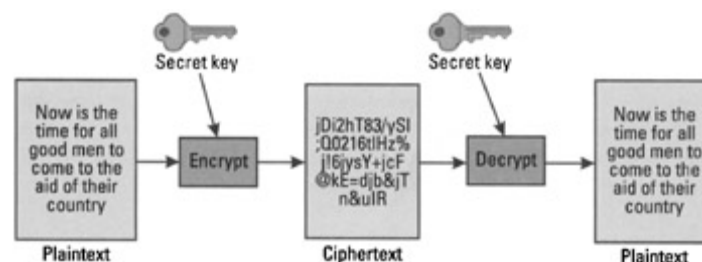


Figure (2.1) Symmetric key encryption

The strength of the above scheme is largely dependent on the size of the key and on keeping it secret. Generally, the larger the key, the more secure the scheme. In addition, symmetric key encryption is relatively fast.

The main weakness of the system is that the key or algorithm has to be shared. You can't share the key information over an unsecured network without compromising the key. As a result, private key cryptosystems are not well suited for spontaneous communication over open and unsecured networks. In addition, symmetric key provides no process for authentication.

Table (2.1) lists the advantages and disadvantages of symmetric key cryptosystems.

Table (2.1) Advantages and disadvantages of symmetric key cryptography

Advantages	Disadvantages
Fast	Requires secret sharing
Relatively secure	Complex administration
Widely understood	No authentication

The most popular symmetric-key algorithms are:

1- Data Encryption Standard (DES)

A popular symmetric-key encryption method developed in 1975 and standardized by American National Standards Institute (ANSI) in 1981 as ANSI X.3.92. DES consists of an algorithm and a key. The key is a sequence of eight bytes, each containing eight bits for a 64-bit key. Since each byte contains one parity bit, the key is actually 56 bits in length [Can01]. The DES algorithm is a careful and complex combination of two fundamental building blocks of encryption: substitution and transposition. The algorithm derives its strength from repeated application of these two techniques, one on top of the other, for a total of 16 cycles. The algorithm begins by encrypting the plaintext as blocks of 64 bits. The key is 64 bits long, but in fact it can be any 56-bit number. (The extra 8 bits are often used as check digits and do not affect encryption in normal implementations) [Pfl02].

DES is widely used in automated teller machine (ATM) and point-of-sale (POS) networks [Can01].

2- Triple Data Encryption Standard (TDES)

Also referred to as *3DES*, a mode of the DES encryption algorithm that encrypts data three times. Three 64-bit keys are used, instead of one, for an overall key length of 192 bits (the first encryption is encrypted with second key, and the resulting cipher text is again encrypted with a third key). Both

DES and 3DES are used in all common security technologies that utilize encryption and decryption with secret keys [Fun05].

3- Advanced Encryption Standard (AES)

The AES standard was developed to replace DES and 3DES. AES uses the Rijndael algorithm, a symmetric block cipher algorithm that can process blocks of 128 bits using cipher keys with lengths of 128, 192, and 256 bits. The Rijndael algorithm is a substitution–linear transformation network with ten, twelve, or fourteen rounds, depending on the key size. A data block to be encrypted by the algorithm is split into an array of bytes, and each encryption operation is byte-oriented. Just as DES and 3DES, AES is used in all common security technologies that require cryptography utilizing secret keys [Fun05].

2.5.3 Asymmetric Key Encryption

Asymmetric cryptography is also known as public key cryptography. Public key cryptography uses two keys as opposed to one key for a symmetric system. With public key cryptography there is a public key and a private key. The keys' names describe their function. One key is kept private, and the other key is made public. Knowing the public key does not reveal the private key. A message encrypted by the private key can only be decrypted by the corresponding public key. Conversely, a message encrypted by the public key can only be decrypted by the private key. This process is illustrated in Figure (2.2).

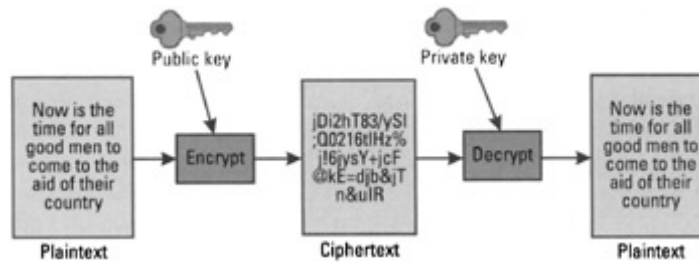


Figure (2.2): Asymmetric key encryption

With the aid of public key cryptography, it is possible to establish secure communications. Public key cryptosystems can provide a means of authentication and can support digital certificates. Unlike symmetric key cryptosystems, public key allows for secure spontaneous communication over an open network. In addition, it is more scalable for very large systems (tens of millions) than symmetric key cryptosystems. With symmetric key cryptosystems, the key administration for large networks is very complex [Can01].

Table (2.2) summarizes the advantages and disadvantages of the public key cryptosystems.

Table (2.2) Advantages and disadvantages of the public key cryptography

Advantages	Disadvantages
No secret sharing necessary	Slower or computationally intensive
Authentication supported	Certificate authority required
Scalable	

There are three public key algorithms in wide use today—Diffie-Hellman; RSA; and the Digital Signature Algorithm (DSA).

1- Diffie-Hellman (DH)

The Diffie-Hellman algorithm was developed by Whitfield Diffie and Martin Hellman at Stanford University. It was the first usable public key algorithm. Diffie-Hellman is based on the difficulty of computing discrete logarithms. It can be used to establish a shared secret key that can be used by

two parties for symmetric encryption. Diffie-Hellman is often used for Internet Protocol Security (IPSEC) key management protocols.

For spontaneous communications with Diffie-Hellman, two communicating entities would each generate a random number that is used as their private keys. They exchange public keys. They each apply their private keys to the other's public key to compute identical values (shared secret key). They then use the shared secret key to encrypt and exchange information [Can01].

2- Rivest, Shamir, Adelman (RSA)

The RSA public key algorithm was developed by Ron Rivest, Adi Shamir, and Len Adelman at Massachusetts Institute of Technology (MIT). RSA multiplies large prime numbers together to generate keys. Its strength lies in the fact that it is extremely difficult to factor the product of large prime numbers. This algorithm is the one most often associated with public key encryption. The RSA algorithm also provides digital signature capabilities. They are used in SSL protocol to set up sessions and with privacy-enhanced mail (PEM) and Pretty Good Privacy (PGP) [Can01].

3- Digital Signature Algorithm (DSA)

DSA was developed as part of the Digital Signature Standard (DSS). Unlike the Diffie-Hellman and RSA algorithms, DSA is not used for encryption but for digital signatures [Can01]. DSA is an algorithm that provides the capability to generate and verify signatures. Signature generation makes use of a private key to generate a digital signature. Signature verification makes use of a public key which corresponds to, but is not the same as, the private key. Each user possesses a private and public key pair. Public keys are assumed to be known to the public in general. Private keys are never shared. Anyone can verify the signature of a user by employing that user's public key. Signature generation can be performed only by the possessor of the user's private key.

A hash function is used in the signature generation process to obtain a condensed version of data, called a message digest see Figure (2.3). The message digest is then input to the digital signature (ds) algorithm to generate the digital signature. The digital signature is sent to the intended verifier along with the signed data (often called the message). The verifier of the message and signature verifies the signature by using the sender's public key. The same hash function must also be used in the verification process [Kam00].

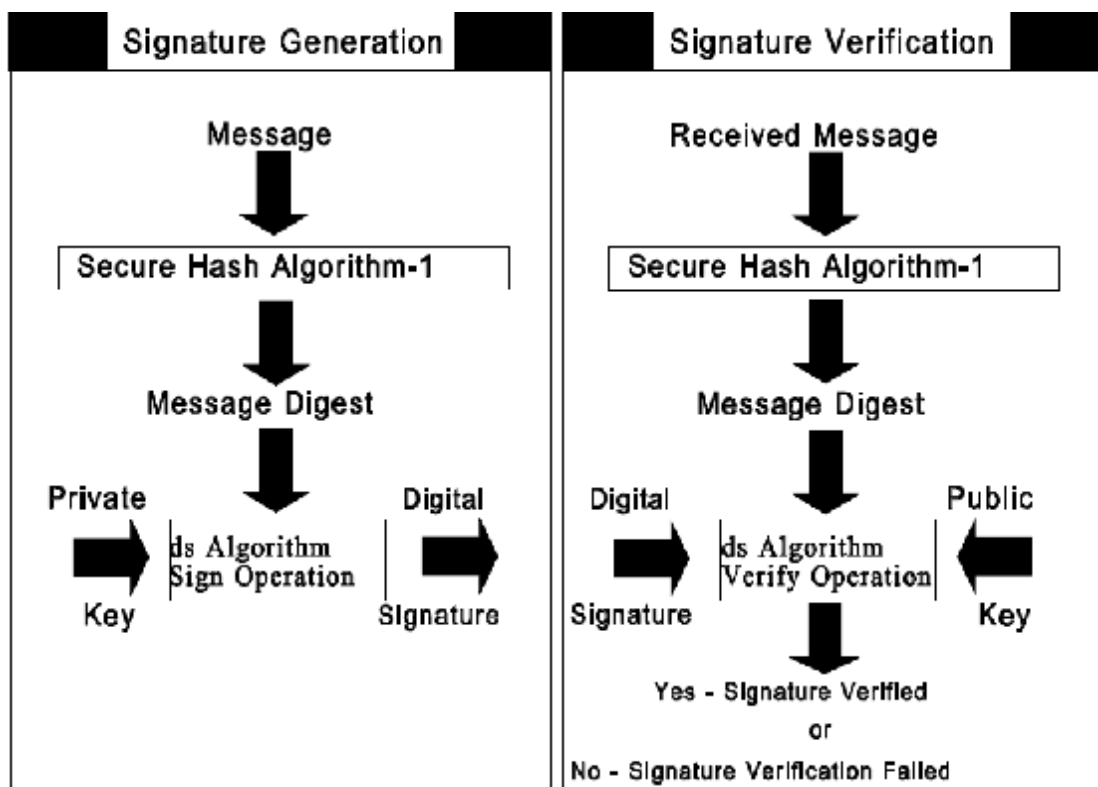


Figure (2.3) Digital signature

2.6 Diffie-Hellman Key Exchange

Whitfield Diffie and Martin Hellman discovered what is now known as the Diffie-Hellman (DH) algorithm in 1976. It is an amazing and ubiquitous algorithm found in many secure connectivity protocols on the Internet. In an era when the lifetime of “old” technology can sometimes be measured in months, this algorithm is now celebrating its 32th anniversary while it is still

playing an active role in important Internet protocols. DH is a method for securely exchanging a shared secret between two parties, in real-time, over an untrusted network.

Mathematical Concepts

To understand Diffie-Hellman key exchange algorithm, some mathematical concepts related to this algorithm must be well understood, these concepts are:

1- Prime Number [Pfl02]: a prime number is any number greater than 1 that is divisible (with remainder 0) only by itself and 1.

2- Primitive Root [Bry06]: let p be a prime. Then b is a *primitive root* for p if the powers of b , $1, b, b^2, b^3, \dots$ include all of the residue classes mod p (except 0). Since there are $p-1$ residue classes mod p (not counting 0), that means the first $p-1$ powers of b have to be different mod p . It have noticed that the powers of b form a repeating cycle, and that cycle can't be longer than $p-1$.

So, b is a primitive root if the cycle is as long as it can possibly be.

Examples: If $p=7$, then 3 is a primitive root for p because the first 6 powers of $(3 \bmod 7)$ are 1, 3, 2, 6, 4, 5---that is, every number mod 7 occurs except 0.

But 2 isn't a primitive root because the powers of 2 are 1, 2, 4, 1, 2, 4, 1, 2, 4...missing several values.

3- Discrete Logarithm [Cun06]: the ordinary logarithm problem given a base b and a number X , find Y such that

$$b^y = X \quad (2.1)$$

Examples: The logarithm to base 2 of 128 is 7.

This can be done in modular arithmetic too. Suppose for a particular choice of n, x, b , that there is a y such that

$$b^y = x \pmod{n} \quad (2.2)$$

Then finding that y is the Discrete Logarithm Problem modulo n .

The Discrete Logarithm Problem is Useful for the following reason: suppose n is large; then given n, b, y it's easy to find x , but no algorithm is known that given n, b, y , will efficiently find x .

4- Modular Arithmetic [Wik08]: modular arithmetic can be handled mathematically by introducing a congruence relation on the integers that is compatible with the operations of the ring of integers: addition, subtraction, and multiplication. For a fixed modulus n , it is defined as follows.

Two integers a and b are said to be **congruent modulo n** , if their difference $a - b$ is an integer multiple of n . If this is the case, it is expressed as:

$$a \equiv b \pmod{n} \quad (2.3)$$

The above mathematical statement is read: " a is congruent to b **modulo n** ". For example,

$$38 \equiv 14 \pmod{12}$$

because $38 - 14 = 24$, which is a multiple of 12. For positive n and non-negative a and b , congruence of a and b can also be thought of as asserting that these two numbers have the same remainder after dividing by the modulus n . So,

$$38 \equiv 2 \pmod{12}$$

because, when divided by 12, both numbers give 2 as remainder

The properties that make this relation a congruence relation (respecting addition, subtraction, and multiplication) are the following.

If $a_1 \equiv b_1 \pmod{n}$ and $a_2 \equiv b_2 \pmod{n}$, then:

- $(a_1+a_2) \equiv (b_1+b_2) \pmod{n}$
- $(a_1-a_2) \equiv (b_1-b_2) \pmod{n}$
- $(a_1a_2) \equiv (b_1b_2) \pmod{n}$.

2.6.2 Diffie-Hellman Key Exchange Algorithm

The purpose of the algorithm is to enable two users to exchange a key securely that then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of the keys [Sta03]. The discrete logarithms are defined in the Figure (2.4).

Step 1: Global Public Elements	
q	Prime number
α	$\alpha < q$ and α a primitive root of q
Step 2: User A Key Generation	
Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \pmod{q}$
Step 3: User B Key Generation	
Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \pmod{q}$
Step 4: Generation of Secret Key by User A	
$K = (Y_B)^{X_A} \pmod{q}$	
Step 5: Generation of Secret Key by User B	
$K = (Y_A)^{X_B} \pmod{q}$	

Figure (2.4) Diffie-Hellman key exchange algorithm

The two calculations in (step 4) and (step 5) produce identical results.

$$\begin{aligned}
K &= (Y_B)^{X_A} \bmod q \\
&= (a^{X_B} \bmod q)^{X_A} \bmod q \\
&= (a^{X_B})^{X_A} \bmod q && \text{(By the rules of modular arithmetic)} \\
&= a^{X_B X_A} \bmod q \\
&= (a^{X_A})^{X_B} \bmod q \\
&= (a^{X_A} \bmod q)^{X_B} \bmod q \\
&= (Y_A)^{X_B} \bmod q
\end{aligned}$$

The result is that the two sides have exchanged a secret key. Furthermore, because X_A and X_B are private, an opponent only has the following ingredients to work with: q, a, Y_A and Y_B . Thus, the opponent is forced to take a discrete logarithm to determine the key.

For any integer b and a primitive root a of prime number p , the unique exponent i can be found such that

$$b \equiv a^i \bmod p \quad \text{where } 0 \leq i \leq (p-1) \quad (2.4)$$

The exponent i is referred to as the discrete logarithm, or index, of b for the base $a \bmod p$. This value is denoted as $ind_{a,q}(b)$.

For example, attacking the secret key of user B, the opponent must compute

$$X_B = ind_{a,q}(Y_B) \quad (2.5)$$

The opponent can then calculate the key K in the same manner as user B calculates it.

2.6.3 Simple Key Management for Internet Protocols (SKIP)

The base and modules used in Diffie-Hellman may be dictated by a standard. One such standard is Simple Key Management for Internet Protocols (SKIP). **SKIP** is a protocol developed by the Internet Engineering Task Force (IETF) security working group for the sharing of encryption keys. Skip is hybrid Key distribution protocol. It is a Simple Key Management for Internet Protocols. It is similar to SSL, except that it

requires no prior communication in order to establish or exchange keys on a session-by-session basis. Therefore, no connection setup overhead exists and new keys values are not continually generated.

SKIP uses Diffie-Hellman so that hosts can agree on a session key that will be used to encrypt each data packet that is sent between them. It can be used in firewalls, to secure communications on a local network, or to create a *Virtual Private Network* (VPN). SKIP defines base and modulus values for different sizes of Diffie-Hellman keys like (256, 512, 1024, 4096 bit) [Knu98].

2.6.4 The Security of Diffie-Hellman

The security of the Diffie-Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

Here is an example, Key exchange is based on the use of the prime number $q=353$ and a primitive root of 353, in this case $\alpha = 3$.

A and B select private keys $X_A = 97$ and $X_B = 233$, respectively.

Each computes its public key:

$$\text{A computes } Y_A = 3^{97} \bmod 353 = 40.$$

$$\text{B compute } Y_B = 3^{233} \bmod 353 = 248.$$

After they exchange public key, each can compute the common secret key:

$$\text{A computes } K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160.$$

$$\text{B computes } K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160.$$

We assume an attacker would have available the following information:

$$q = 353; \alpha = 3; Y_A = 40; Y_B = 248$$

In this simple example, it would be possible by brute force to determine the secret key 160. In particular, an attacker E can determine the common key by

discovering a solution to equation $3^a \bmod 353 = 40$ or the equation $3^b \bmod 353 = 248$. The brute-force approach is to calculate power of 3 modulo 353, stopping when the result equals either 40 or 248. The desired answer is reached with the exponent value of 97, which provides $3^{97} \bmod 353 = 40$. With large numbers, the problem becomes impractical.

As an example of another use of the Diffie-Hellman algorithm, suppose that a group of users (e.g., all users on a LAN) each generate a long-lasting private value X_A and calculate a public value Y_A .

These public values, together with global public values for q and α , are stored in some central directory. At any time, user B can access user A's public value, calculate a secret key, and use that to send an encrypted message to user A. If the central directory is trusted, then this form of communication provides both confidentiality and a degree of authentication. Because only A and B can determine the key, no other user can read the message (confidentiality). Recipient A knows that only user B could have created a message using this key (authentication) [Sta03].

Chapter Three

Design and Implementation of SIMSM

3.1 Introduction

This thesis presents the design and implementation of a secure instant messaging system for LAN networks. It concentrates on the sets of security requirements that instant messaging system should comply with.

The name of the proposed system is chosen to be *SIMSM*, the acronym for (SIMple Secure Messenger). SIMSM provides two basic services for users: a way to protect messages from unintended viewers through symmetric key encryption, and a way to authenticate users through asymmetric key encryption such that each message between client and server, and between client and client is encrypted.

In this chapter the word "Client user" and "Administrator user" will be used as indication to real human being, while the words "Client" and "Server" indicate the software programs that run by User and Administrator respectively.

3.2 SIMSM Global Design

SIMSM is designed to be Client-Server model as shown in Figure (3.1). Such that all the messages are transmitted between the clients through central server. The reasons for adopting centralized server are for the sake of its simplicity and security.

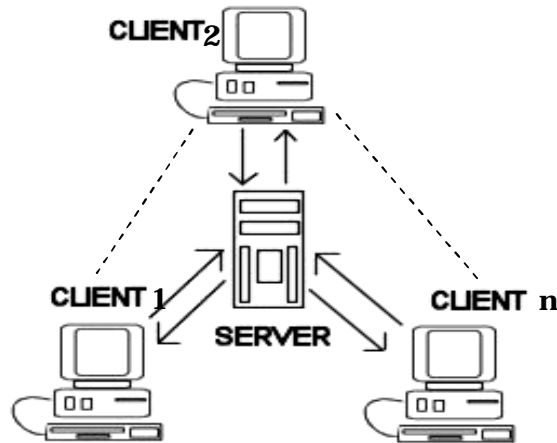


Figure (3.1) SIMSM global design

The roles of the server will be:

1. Register users.
2. Authenticate users.
3. Generate asymmetric key used to encrypt data sent between server and clients.
4. Process client messages.
5. Distribute presence notification.
6. Manage user's accounts.
7. Deliver messages to the intended reception.

On the other hand, a client is relatively simple piece of software whose most advanced part is probably the graphical user interface. A client sets up a connection with the instant messaging server through the Transmission Control Protocol (TCP) sockets.

The roles of client in the system are:

1. Communicate with the server over TCP sockets.
2. Process server message.
3. The client sends and receives presence notifications, participates in conference chat or send subscription message through the connection with the server.

3.3 SIMSM Requirements

To implement the established system, a number of software and hardware requirements are needed. Software requirements are:

1. Operating Systems to run Chat Server and Chat Client such as Windows XP.
2. Java Development Kit (JDK) essentially a Java platform, consisting of the Application Programming Interface (API) classes, a Java compiler, and the Java Virtual Machine interpreter. The JDK is used to compile Java applications and applets.
3. Programming language that support network operations like connection, receiving incoming network packets and sending messages from host to another using socket API functions. Java language is chosen to implement this thesis.
4. Application to store clients information. Microsoft Office Access software is used. And Java accesses information through the connection with JDBC driver. See appendix A for more details.

While the hardware requirement is that there are at least three computers in the network to make the system work properly, one of them is the server run by the administrator and the others are clients run by users.

Features of SIMSM System

SIMSM will be designed to have the following features:

1. Provide secure private chat.
2. Enable conference chat.
3. Transfer smiles with text.
4. Has nice look and feel, and has simple GUI that makes it easy to use by users.

5. Has relatively fast execution.
6. Does not need an internet connection.

3.5 SIMSM Structure

The structure of the SIMSM consists of two main parts; *Server* and *Client* as illustrated in Figure (3.2).

These two parts imply different modules, and each module is performed by specific units. Some modules execute units in both parts independently, while in the other modules, there is overlapping between executing jobs of units of both parts when the same shared result are needed.

Server modules are:

1. **Initialization Module:** It initializes server information. This module performs the following functions:
 - A. Define and create Server Socket.
 - B. Define and starting threads.
2. **Check Authentication Module:** It authenticates users to use the system depending on the clients information.
3. **Complete User Information Module:** In this module, the server will complete creating user's information to permit him/her to use the system.
4. **Process Client Message Module:** This module processes each message came from the client according to specific information in the message object.

On the other hand, Client consists of the following modules:

1. **Registration Module:** It identifies users to the system.
2. **Login Module:** This module logging the authenticated users into the system, and generates Diffie-Hellman key pair to them.

- 3. Sign-In Problem Module:** This module changing the forgetting passwords of the user and retrieving their forgotten IDs.
- 4. Private Chat Module:** It makes a private chat between two users only if these users are in online status.
- 5. Conference Chat Module:** The aim of this module is to enable user to make a conference chat with more than one online users.
- 6. Process Server Message Module:** This module handle and process all messages come from server.
- 7. Conference Chat Module:** The aim of this module is to enable user to make a conference chat with more than one online users.
- 8. Process Server Message Module:** This module handle and process all messages come from server.

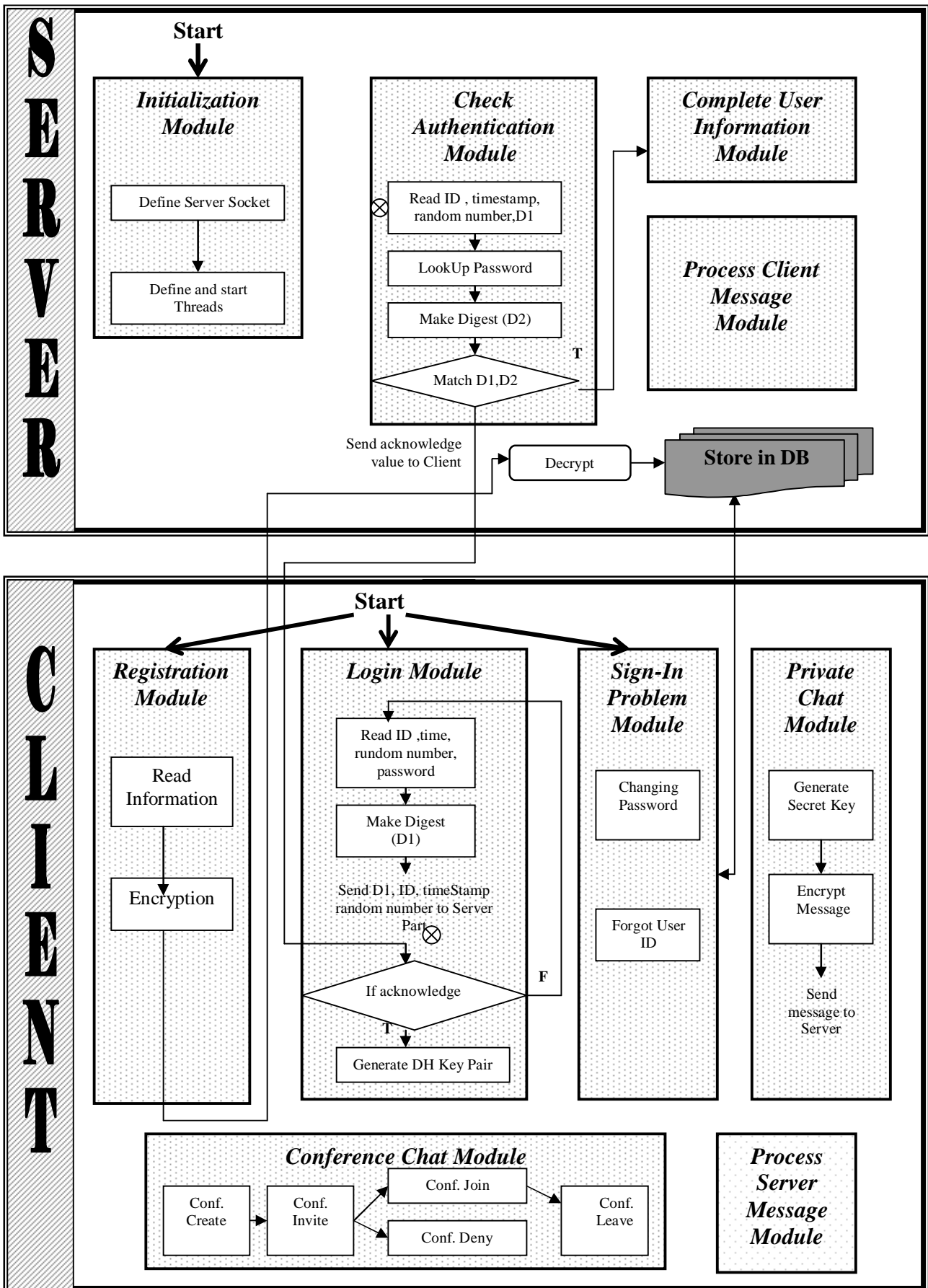


Figure (3.2) SIMSM structure

3.6 Server Program

The SIMSM server is the core of the system. The server is multi-threaded program. These threads depend on the requested function of the server.

The server program consists of four modules as follows:

3.6.1 Initialization Module

The main goal of this module is to initialize all necessary server and client information. It executes two units, *Define and Create server socket, unit* and *Define and Start Threads* unit. The first unit defines and creates a `ServerSocket` with a specific port number to accept communication from clients. The later unit defines and run a number of threads, namely as `Registration` thread and `Sign-In` thread.

A. Registration Thread: This thread is dedicated to register new users to the system. First, it defines a `ServerSocket` named "Reg_Socket" with a specific port number to enable users to connect to the server and to register into the system.

Second, the server will execute a number of functions as follows:

1. Generate RSA key pair (public and private keys) with 1024 bit key size.
2. Sends the RSA public key to the client to encrypt the registered information of the user.
3. Receives the registered encrypted information from the client.
4. Decrypt the above received information by RSA private key.
5. Store the received information in the (RegDB) database after checking the existence case of the user.

Algorithm (3.1) explains how server executes the above functions.

Algorithm (3.1) Asymmetric Key Maker

Goal: Generate 1024 RSA key pair.

Const: Algorithm = "RSA".

KeySize = 1024.

Input: None.

Output: Key pair with 1024 bit size.

Step1: Define Server Socket with register port number (Registr_Port).

Reg_Socket = new ServerSocket(Register_Port)

Step2: Define DataOutputStream (out) with (Reg_Socket).

Step3: Create a KeyPairGenerator (KeyGen) using **getInstance** API function.

KeyGen \leftarrow getInstance (Algorithm)

Step4: Initialize KeyGen with 1024 bit KeySize.

Step5: Creating key pair using **genKeyPair** API function and put result in (keyPair).

keyPair \leftarrow KeyGen.genKeyPair()

Step6: Connect to client using socket API functions, and send the public key to the client part using DataOutputStream.

Out \leftarrow keyPair.PublicKey

Step7: Store the keyPair in file named as "KeyFile".

Step8: Call **Algorithm (3.2)** to receive the registered encrypted information from the client

Step9: End.

Algorithm (3.2) Receiving Registered Information

Goal: Register new member into the SIMSM system.

Input:

Register Socket (rs).

Output: None.

Step1: Read Private Key (Prk) from a file (KeyFile).

Step2: Define a DataInputStream (Ins) to read the user's registration information from the client.

Continue

and a *DataOutputStream* (*Outs*) to write an acknowledgment to the same client.

Step3: Read the encrypted registered information through the *DataInputStream* (*Ins*), (*Fname*, *Lname*, *Uid*, *address*, *ans*, *birth*, *pass*).

Step4: Call **Algorithm (3.3)** to decrypt the above received information by (*Prv*) key.

FName ← *Fname*.

LName ← *Lname*.

UID ← *UID*.

Address ← *Decrypt* (*address*, *Prv*).

Ans ← *Decrypt* (*ans*, *Prv*).

Birth ← *Decrypt* (*Birth*, *Prv*).

Pass ← *Decrypt* (*pass*, *Prv*).

Step5: Call **Algorithm (3.4)** to check the existence of User's ID and returns a boolean value.

Step6: if (Check ID existence) then //See Algorithm (3.4)//

write a message to the user client through the *DataOutputStream* (*Outs*) to tell him/her that this ID is used before.

else

Step6.1: Call **Algorithm (3.5)** to store register user information in the database.

Step 6.2: send a client user an acknowledgment" successful registration".

end if.

Step7: Close *DataInputStream* and *DataOutputStream*.

Step8: End.

Algorithm (3.3) RSA Decryption

Goal: Decrypt data by RSA algorithm.

Input:

EData: the encrypted data.

Prv: the Private Key.

Output: *ClData*: string represents the decrypted data.

Step1: Creating a cipher object using **getInstance** API method with RSA cryptograph algorithm.

Step2: Initialize this cipher object with the **DECRYPT_MODE** and (*Prv*) key.

Step3: Decrypt *EData* using **doFinal** API function.

$ClData \leftarrow doFinal(EData).$

Step4: Return (*ClData*).

Step5: End.

Algorithm (3.4) Check ID existence

Goal: Check the existence of User's ID.

Input:

UserID: the decrypted user ID.

Output:

Flag: Boolean variable represents **true**: if the ID is exist or **false**: otherwise.

Step1: Load the JDBC driver from ("sun.jdbc.odbc.JdbcOdbcDriver") to connect java with Microsoft access.

Step2: Establish the connection to the database (RegDB).

Step3: Define a ResultSet (*Rset*) that select all Id's from the RegDB.

Step4: Set *Flag* = false.

Step5: While (*Rset.next*) and not (*Flag*)

Step 5.1: $returnId \leftarrow Rset.getString$ (Field name of User ID in the database(*UserID*)).

Step 5.2: if (*UserID*) is equal to (*returnId*) then

Continued

Step 5.2.1: Set Flag = true.

Step 5.2.2: Close connection with RegDB.

end if.

end While

Step6: Close connection with RegDB.

Step7: Return (Flag).

Step8: End.

Algorithm (3.5) Store In DataBase

Goal: Store registered information of the users in database table.

Input:

Fname: First name of the user.

Lname: Last name of the user.

Id: ID of the user.

Address: address of the user.

Pass: password of the Client's user.

Birth: Birthdate of the Client's user.

Ans: answer of the challenge question.

Output: None.

Step1: Load the JDBC driver from ("sun.jdbc.odbc.JdbcOdbcDriver") to connect java with Microsoft access.

Step2: Establish the connection to (RegDB) database.

Step3: Execute the INSERT query into the user register database (RegDB).

Step4: Close the connection with the (RegDB) database.

Step5: End.

B. Sign-In Problem Thread: This thread is concerned with *Sign-In Problem Module*. Sign-In problem occurs when the user either forget his/her password or ID. According to specific value receives from the client program; the server will execute either algorithm (3.7) in the case of forgetting password or execute Algorithm (3.8) in the other case.

Algorithm (3.6) Sign-In Problem Thread

Goal: Enable user to change the password or retrieve their forgotten ID.

Input: None.

Output: None.

Step1: Define two boolean variables (*Done*) and (*Flag*) and,

Set *Done* = false,

Set *Flag* = true.

Step2: While not (*Done*) and (*Flag*)

Step3: Define new server socket with a *SIGN_PORT* that is already define in the system.

Step4: Define a socket (*sok*) that accepts the server socket.

Step5: Define *DataInputStream* (*Ins*) with the socket (*sok*).

Step6: Define *DataOutputStream* (*Outs*) with the socket (*sok*).

Step7: Read the information came from the client (*Birth*, *City*, *EAns*, *EPass*, *Botm*).

Step8: Read the Private Key (*Prv*) from the "Keyfile".

Step9: Call **Algorithm (3.3)** to decrypt the reading information with (*Prv*).

Step10: if *Botm* = 1 then

Step10.1: Call **Algorithm (3.7)** for changing the password of the user.

Step10.2: Write a message to the client through the

DataOutputStream (*Outs*).

Outs ← "successful changing your password".

Step10.3: Set *Flag* = false.

Else

Step10.1: Call **Algorithm (3.8)** for retrieving the ID of the user and put it in (*UserId*) variable.

Step10.2: If *UserId* is Null then

1. *Outs* ← "there is no such User ID belongs to this Password".

2. *Done* = true.

Continue

else

1. "Your forgotten ID is ", UserId.

2. Flag = false.

end if.

end While.

Step11: End.

Algorithm (3.7) Changing Password

Goal: *Create a new password to the user in case that he/she forgotten the password or want to change his/her knowing password.*

Input:

Birth: birthday date of the user. //read from client's user

City: address of the user. //read from client's user

Ans: answer to the challenge question. //read from client's user

Pass: new password of the user. //read from client's user

Output: *Changing password of the user in RegDB.*

Step1: *Load the JDBC driver from ("sun.jdbc.odbc.JdbcOdbcDriver").*

Step2: *Establish the connection to the RegDB.*

Step3: *Execute an UPDATE query and change the password of the user with a new password.*

Step4: *Close connection with a database.*

Step5: End.

Algorithm (3.8) Retrieving User ID

Goal: Retrieve the forgotten ID of the user.

Input:

birth: Birthday of the user. //read from client's user

add: Address of the user. //read from client's user

ans: Answer of question. //read from client's user

EPass: Encrypted password. //read from client's user

Output: *Uid:* retrieving user ID.

Step1: Call Algorithm (3.3) to decrypt the EPass received from client program and put the result in (Pass).

Step2: Load the JDBC driver from ("sun.jdbc.odbc.JdbcOdbcDriver").

Step3: Establish the connection to the RegDB.

Step4: Execute a query that search for an ID with the following (*birth*, *add*, *ans* and *Pass*) and put the result in (*UserId*) variable.

Step5: Return (*UserId*).

Step6: Close connection with a database.

Step7: End.

3.6.2 Check Authentication Module

The goal of this module is to check the authentication status of the user. If the user is identified, then he/she will be permitted to use SIMSM, else the user will have to re-login again. Algorithm (3.9) explains how the server checks the authenticity of the user.

Algorithm (3.9) Check Authentication Process

Goal: Check if the user is authenticating user to the SIMSM or not.

Input:

_socket: socket address of the user.

Output:

Flag: Boolean variable represents **True**: if this user is valid user or **False**: otherwise.

Step1: Define a *DataInputStream* (*Ins*) within the socket (*_socket*) to read the information came from client.

Step2: Read the following information:

userID \leftarrow *Ins.read()*.

time \leftarrow *Ins.read()*.

Random_Num \leftarrow *Ins.read()*.

Digest1 \leftarrow *Ins.read()*.

Step3: Call **Algorithm (3.10)** to look up the password of the user ID and put result in (*Password*) variable.

Password \leftarrow *LookUpPassword(userID)*.

Step4: *if* (*Password*) is **null** **Then**

Return (*false*)

else

Call Algorithm (3.11) to compute the message digest of (*userID*, *time*, *Random_Num* and the *Password* it just retrieved) and put resulted digest in (*Digest2*) variable.

Step5: Matching

if (*Digest2*) is equals (*Digest1*) **then**

Return (*true*).

else

Return (*false*).

Step6: Send returned (*Flag*) to client using a **socket API** function.

Step7: End.

Algorithms (3.10) Look Up Password

Goal: Find the password of the specific user ID.

Input:

userID: ID of the user.

Soc_user: socket address of the user.

Output: *return_pass:* returned user's password.

Step1: Define *DataOutputStream* (*out*) with the (*Soc_user*) socket.

Step2: Define Boolean variables:

Flag: if there is no such user ID.

Step3: Set *Flag1 = False*.

Step4: Load the JDBC driver from ("*sun.jdbc.odbc.JdbcOdbcDriver*").

Step5: Establish the connection to the (*RegDB*) database.

Step6: Execute a *SELECT* query to select all the IDs of the users and put the result set to (*set*) variable.

Step7: **While** (*set.next*)

if there exist such (*user ID*) in the database **then**

1. *return_pass* ← get the password of this *UserID*.

2. *Flag = True*.

end if.

end While.

Step8: **if not** (*Flag*) **then**

Step8.1: *Out* ← "re-enter your ID and password.If you have forgotten your ID or password, Click Sign-In Problem bouton".

Step8.2: *return_pass* ← null.

end if.

Step9: **Return** (*return_pass*).

Step10: **End.**

Algorithm (3.11) Make Digest Message

Goal: Create a message digest.

Input:

UserID: ID of the user.

Pass: password that the server retrieved from the (RegDB) database.

TimStmp: time stamp.

RandomNum: double random number.

Output: digest message (digMess).

Step1: Prepare the SHA-1 algorithm for using to make a digest message using *getInstance* API function.

Step2: Apply the SHA-1 algorithm to the input information to compute digest message and put result in (digMess) variable.

Step3: Return (digMess).

Step4: End.

3.6.3 Complete User Information Module

In this module, the server is responsible about performing the following functions:

1. Add user's socket address to a new vector list named as "list".
2. Define a DataInputStream to read the message object from client (see Table (3.1)).
3. Add user ID name with his/her socket address into a table named "_UserTable".
4. Add user's name with his/her user object into a table named "_userList".
5. Send to the client a list of online users.
6. Broadcast to all clients that this user is online now.
7. Define a new thread named "Service Thread", which is needed for every client connected to the server for processing client message object by calling *Process Client Message Module*.

Table (3.1) User message object fields

Field name	Meaning
_header	Describe what client wants to do. Table (3.4) presents the types of the header.
_username	User ID name.
_destination	Destination Client's user.
_message	Message string.
_host	Address of the user's PC.
_user	User Object, Table (3.2) presents the fields of user object.
Userlist	A list that contains all the online users in the system.

Table (3.2) User object fields

Field name	Meaning
username	ID name of the user.
Socket_address	Socket address (IP address, Port number) of the PC.
User_Status	Integer represents the status of user (see Table 3.3).
isConference	Boolean variable has two values: True: when user want to make conference chat. False: otherwise.
DHPublic	Represent the Diffie-Hellman public key.

Table (3.3) User status

Integer Number	String Represented Status
0	ONLINE
1	BUSY
2	OFFLINE

3	AWAY
4	ON THE PHONE
5	BE RIGHT BACK
6	NOT AT MY DESK

Table (3.4) Header types in message object

Header Name
CLIENT_LOGIN
CLIENT_LOGOUT
CONFERENCE_CHAT
PRIVATE_CHAT
USERS_LIST
CHANGE_STATUS
CONFERENCE_CREATE
CONFERENCE_INVITE
CONFERENCE_JOIN
CONFERENCE_DENY
CONFERENCE_LEAVE

3.6.4 Process Client Message Module

In this module, the server will execute a dedicated function according to the value of header of the message object (see Table (3.4)). Algorithm (3.12) explains how this module works.

Algorithm (3.12) Process Client Message

Goal: Process the client's user message object.

Input:

message: Message Object of the user.

Output: None.

Step1: Read the header of the message object (*_header*).

Step2: Switch (*_header*):

Case CHANGE_STATUS:

The server called Algorithm (3.13) to notify all online users that this user change his/her status and GoTo Step3.

Case CLIENT_LOGOUT:

- 1. The server get the socket of this user from UserTable.*
- 2. Remove the user's socket address from the vector list and from UserTable.*
- 3. Call Algorithm (3.13) to notify all users that this user is logout and GoTo Step3.*

Case CONFERENCE_CREAT:

Server put the user name with the Selected_List to hash table named conf_list. and GoTo Step3.

Case CONFERENCE_INVITE:

Server called Algorithm (3.14) to invite users to a conference chat and GoTo Step3.

Case CONFERENCE_JOIN:

Server called Algorithm (3.15) to write to conference chat and GoTo Step3.

Case CONFERENCE_DENY:

Remove user from conf_list and call Algorithm (3.15) to write to the conference and GoTo Step3.

Case CONFERENCE_LEAVE:

- 1. Server retrieve from conf_list the user's selected list of users.*
- 2. Called Algorithm (3.15) to notify users in his/her Selected_List that he/she leaved the conference with a specific time then GoTo Step3.*

Continu

Case CONFERENCE_CHAT:

Server called Algorithm (3.15) then GoTo Step3.

Case CONFERENCE_LIST:

Server called Algorithm (3.16) to send a conference list to clien then

GoTo Step3.

Default Case

Server called Algorithm (3.17).

Step3: End.

Algorithm (3.13) Write To Clients

Goal: Notify the SIMSM online users.

Input:

message: user Message Object.

list: List contains sockets address of client users.

Output: None.

Step1: Define DataOutputStream (dos).

Step2: For count = 0 To list size

Step2.1: Get the socket address.

_socket ← list [count]

Step2.2: Write message object to user with at the specific socket address.

dos [_socket] ← message

Step2.3: End For.

Step3: End.

Algorithm (3.14) Invite To Conference Chat

Goal: Invite the selected users to a conference chat.

Input:

Selected_List: list of users selected to invite them to a conference chat.

message: Message Object.

Output: None.

Step1: Define DataOutputStream (dos).

Continu

Step2: For $i = 0$ To size of Selected_List

Step2.1: Get the socket address from UserTable of the Userlist [i]

socket \leftarrow Get UserTabel. (get Selected_List[i])

Step2.2: write a message object to the socket define in Step2.1

dos \leftarrow message object.

end For.

Step3: End.

Algorithm (3.15) Write To Conference Chat

Goal: Join user to a conference chat.

Input:

message: Message object.

Output: None.

Step1: Get vector list (SList) of the user from the conf_list

SList $r \leftarrow$ conf_lis.get(message._destination)

Step2: For $i=0$ To size of SList

Step2.1: Get the socket address from UserTable

socket \leftarrow UserTable.get(SList(i))

Step2.2: write the message object to the user with the socket (soc).

end For.

Step3: End.

Algorithm (3.16) Request Conference List To Client

Goal: Request a list of selected user of specific user to make a conference chat with them.

Input: message: Message Object.

Output: None.

Step1: Define new Message Object (mess) and make the header of mess as

_header = CONFERENCE_LIST

Step2: Get the vector list (SList) of selected user that the user selected before.

Continue

SList ← **Get** *conf_list* [*message._destination*]

Step3: *mess._username* ← *message._username*.

Step4: *mess._destination* ← *message._destination*.

Step5: *mess.userlist* ← *SList*.

Step6: **Get** a socket address of the user from **UserTable**

socket ← **Get Socket** (*userTable* [*message.username*])

Step7: **if** (*socket*) **not equal null then**

Step7.1: create a new *DataOutputStream* (*dos*) with a socket address
define above.

Step7.2: write a *mess* object to a specific socket address
dos ← *mess*

else

Display an Exception message named *Conference List Exception*.

end if.

Step8: **End.**

Algorithm (3.17) Write to Client

Goal: Write a message object to a specific user.

Input:

message: Message Object.

Output: None.

Step1: Get a Socket from a *UserTable* of a specific user.

socket ← **Socket** (**Get** from *UserTable* [*message._destination*])

Step2: Create a new *DataOutputStream* (*dos*) with a socket address (*socket*).

Step3: Write a message object to *dos*.

dos ← *message*.

Step4: **End.**

3.7 Client Program

The client program is the main part of SIMSM that is installed on the remote computers of a LAN. The users of these computers can not access the chatting system unless their identities are authorized by the server program.

The client program is divided into five modules, as it will be explained in the next sections.

3.7.1 Registration Module

In this module, the client has to perform a number of functions, which are:

1. Create a socket to be used for connection with server.
2. Connect with the server.
3. Read the public key which is sent from the server.
4. Fill the required registered information.
5. Check the filled information if it satisfy the conditions of registration.
6. Encrypt selected registered information by RSA algorithm, see Algorithm (3.18).
7. Send the register information to the server for decrypting and storing in (RegDB) database, see Algorithm (3.2) at server part. And the client waiting for an acknowledge message from the server.

The registered information are:

1. **First Name, Last Name, and Address** : those information should not contains any special characters *, &, !, @, #, \$, %, ^, (,), |, ", :, ?, >, <.
Note that the first and last name is for interface presentation not for authentication.
2. **Password**: length of password must be at least 8 characters and less than 17 characters. The user is requested to enter a password twice times to confirm its password.

The password must meet the following requirements:

- The length must be greater than or equal to 8 characters.
 - Must contain at least two capital letters.
 - Must contain at least two lower letters.
 - Must contain at least two numeric characters.
 - Must contain at least two special characters.
 - Should not contain the first and last name.
3. **Birthday date:** the birthday date must be valid of the form (YYYY, MM, DD).
 4. **UserID:** the user ID must be unique, i.e. never used by any other client before, if it is, then client user should choose another UserID.
 5. **Challenge information:** a string represents an answer to a question; in SIMSM client program there is specific number of questions, so that the user selects one of the questions and answer it. This information will be used when user needs to change the password or wants to retrieve his/her forgot ID.

Algorithm (3.18) Encryption

Goal: *Encrypt data.*

Input:

data: data to be encrypted.

Puk: Public key.

_Algorithm: algorithm to be encrypted data with.

Output: *EData : encrypted data.*

Step1: *Creating a cipher object using **getInstance** API function with (_Algorithm)*

Continu

cryptograph algorithm.

Step2: Initialize this cipher object with the *ENCRYPT_MODE* and (*Puk*) key.

Step3: Encrypt the data using **doFinal** API function and put result in (*EData*) variable.

$EData \leftarrow doFinal (data).$

Step4: Return (*EData*).

Step5: End.

3.7.2 Login Module

After finishing registration, login module is called when the user needs to use the system. To login, the user must presents its own correct user ID and password. After successful login the user will be considered to be in online status. If the user enters invalid login information, the Server part will prompts him/her to re-enter correct information, for three times as maximum, after three times the user must waits 3 minutes to try again.

If the user entered user ID already exist in OnlineClient table, then the server informs the client that this UserID is already logged in. Algorithm (3.19) shows how the Login module works.

Algorithm (3.19) Protect Login Process

Goal: Login system's member into the chat system.

Input:

UserID: the ID of the user.

Pass1: the password of the user.

Output:

Flag: Boolean represent the state of login process (True or False).

Step1: Read *UserID* and *Pass1*.

Step2: Create a Time Stamp (*time*) and a double random number (*Rnum*).

Step3: Call **Algorithm (3.20)** to compute the digest message to the (*UserID*, *Pass1*, *time* and *Rnum*) and put result in (**Digest1**) variable.

Continue

Step4: *Send (Digest1) to the Server, and server will compute another digest named (Digest2) and matching if they are equals, and waiting a boolean from **Algorithm (3.9)** at Server Part.*

Step5: *if server sends (true) means successful login then*

GoTo Step6.

else

*Fail to login, the user has Sign-In problem. Goto **Step13**.*

Step6: *Define a DataInputStream (dis) and DataOutputStream (dos) with a socket API function.*

Step7: *Call **Algorithm (3.21)** to generate a Diffie-Hellman key pair to the user and put result in (DHKeyPair).*

Step8: *Define a UserObject (userObj) to the Client's user with (_username, _address, ONLINE, DHPublicKey).*

Step9: *Define a MessageObject (message) to the user with a*

_header = CLIENT_LOGIN

_username = UserID.

_host = IP address of user's PC.

_user = UserObj.

Step10: *Send the (message) object to the server using a socket API function to start a Service Thread to the user.*

Step11: *Receive a message object from server.*

Step12: *Call **Algorithm (3.22)** to process server message.*

Step13: *End.*

Algorithm (3.20) Message Digest

Goal: *calculating a message digest.*

Input:

UserID: the ID of the user.

Pass1: the password of the user.

time: the TimeStamp that client create it.

Rnum: the random number client create it.

Continu

Output:

messageDigest : message digest.

Step1: Create a *MessageDigest* object with **SHA-1 hashing algorithm** using *getInstance* API function.

Step2: Take (*UserID*, *Pass1*, *time* and *Rnum*) to calculate a message digest.

Step3: Return (*messageDigest*).

Step4: End.

As mentioned in chapter two, section 2.6.3, the Diffie-Hellman key agreement algorithm needs to generate DH parameters that include a *base* and *modulus* values. So the system defined 1024 bit DH parameters *sDHParameterSpec* (*skip1024Modulus*, *skip1024Base*) using a SKIP protocol.

Algorithm (3.21) Generate DH Key Pair

Goal: Generate a DH key pair (public and private keys).

Input: None.

Output: *keyPair*: DH key pair.

Step1: Create a *keyPairGenerator* (*kpg*) using *getInstance* API function with a "DH" algorithm.

Step2: Initializing a *KeyPairGenerator* with *sDHParameterSpec*.

Step3: Generating a Key Pair using a *genKeyPair* API function.

$keyPair \leftarrow kpg.genKeyPair().$

Step4: Return (*keyPair*).

Step5: End.

Algorithm (3.22) Process Server Message

Goal: Process the server message object.

Input:

message: Message Object.

Output: None.

Step1: Read the header of message object (*_header*)

Step2: Switch (*_header*):

Case CLIENT_LOGIN:

*the client will open a main frame of chatting system with a Tree Panel of the all users that login to the system and add user to(*_userlist*) then **GoTo Step3** .*

Case CLIENT_LOGOUT:

*the client will remove the user from the(*_userlist*)and from the Tree Panel, and notify the Server that this user is logging out from the system and **GoTo Step3**.*

Case CHANGE_STATUS:

*The client part will update the user status and send the message object of that user to server and **GoTo Step3**.*

Case CONFERENCE_INVITE:

*client will show the invitation windows with available users, and the user select the users want to make a conference chat with them, and send it to server and **GoTo Step3**.*

Case PRIVATE_CHAT:

- 1. client obtains the public key (*thierPuk*) of the first user from the user's user object.*
- 2. call Algorithm(3.26) to generate a secret key with the current user private key (*Prv*) and the remote user's public key (*thierPuk*).*
- 3. call Algorithm (3.27) to Decrypt the received message with the secret key.*
- 4. display the decrypted message in the private chat windows and **GoTo Step3**.*

Step3: End.

3.7.3 Sign-In Problem Module

This module is called when client's user faced a problem in logging to the chatting system. This module offers two cases: First, if user forget his/her ID. Second, if user want to change his/her password.

The work of this module involves first confirming the identity of the user by entering the birthday, address and answer to the question provided when the user registered , then offers to user what he/she has problem with.

Algorithm (3.23) will explain how this module works.

Algorithm (3.23) Sign-In Problem

Goal: Find the forget ID and changing a password.

Input: None.

Output: None.

Step1: Read the Birthday date (birth) of the user, //
Read the address (add) of the user, confirming user identity
Select a question and the associated answer (ans) to it. //

Step2: Connect to server using socket API function.

Step3: Read the public key from the Keyfile and put it in (Puk) variable.

Step4: If user wants to retrieve his/her forgotten ID then

Step4.1: Enter his/her password then

Step4.2: Call Algorithm (3.18) with (password, Puk and "RSA") to encrypt the password before sending to the server and put the result in (EPass).

Step4.3: Send (birth, add, ans and EPass) to Server part for retriving the ID of the user. The server call **Algortihm (3.8)** and put result in UserId.

Step4.4: Return (UserId)

Continu

else

if user wants to change his/her password then

Step4.1: enter new password (pass) and confirm password.

Step4.2: Call Algorithm (3.18) to encrypt the (pass) before sending to the server and put result in (EPass) variable.

Step4.3: Send (birth, add, ans and EPass) to server to change the password of user in the (RegDB) database. And the server call Algorithm (3.7) for changing the password.

end if.

Step5: End.

3.7.4 Secure Private Chat Module

The function of this module is to establish a secure two party chat session (private chat) between two users. The user must select an online user by double clicking on the ID name of the user and open a private chat window, at this moment the two users will generate a shared secret key to be used later when they transfer an instant messaging between them.

This module is classified into three units; *Generate a secret key* unit, *Encrypt a message* unit and *Private chat* unit.

A. Generate Secret Key Unit

This unit is responsible about creating session key between any two clients (see Algorithm (3.24)). To create such a key Diffie-Hellman key exchange algorithm is used.

But Diffie-Hellman algorithm is susceptible to a man-in-the-middle attack. This attack is occurred when a third party places himself/herself between two users (A and B) and captures messages from A to B and vice versa. With the third party thusly placed, DH exchange between A and B can be subverted.

The third party establishes a shared secret, say $g^{at} \bmod p$ with A, and another shared secret $g^{bt} \bmod p$ with B, as illustrated in Figure (3.3).

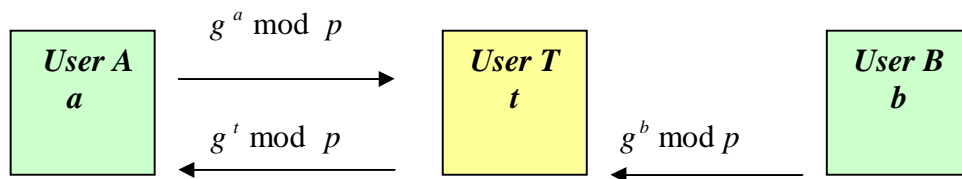


Figure (3.3) Diffie-Hellman man-in-the-middle attack

To prevent Man-In-the-Middle attack, SIMSM system encrypts the DH exchange with a shared symmetric key as shown in the Figure (3.4). It shows the Diffie-Hellman key exchange that will take place between the two users namely A and B.

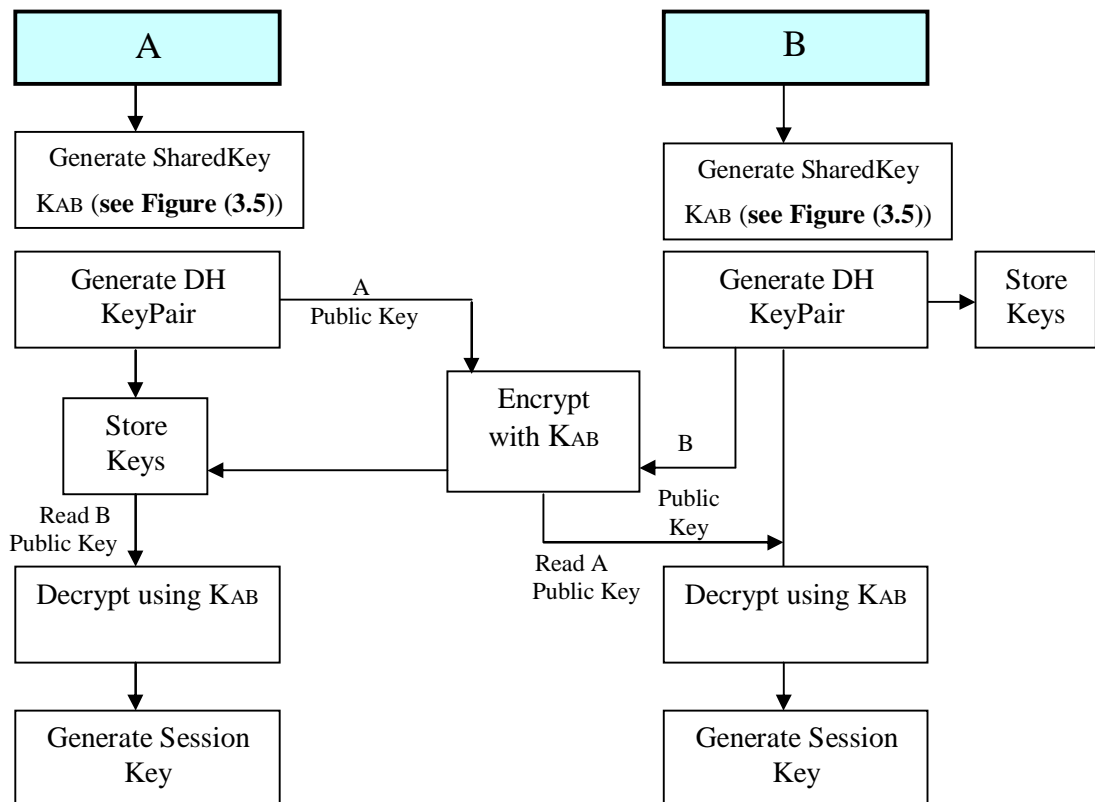


Figure (3.4) Encrypt DH public key

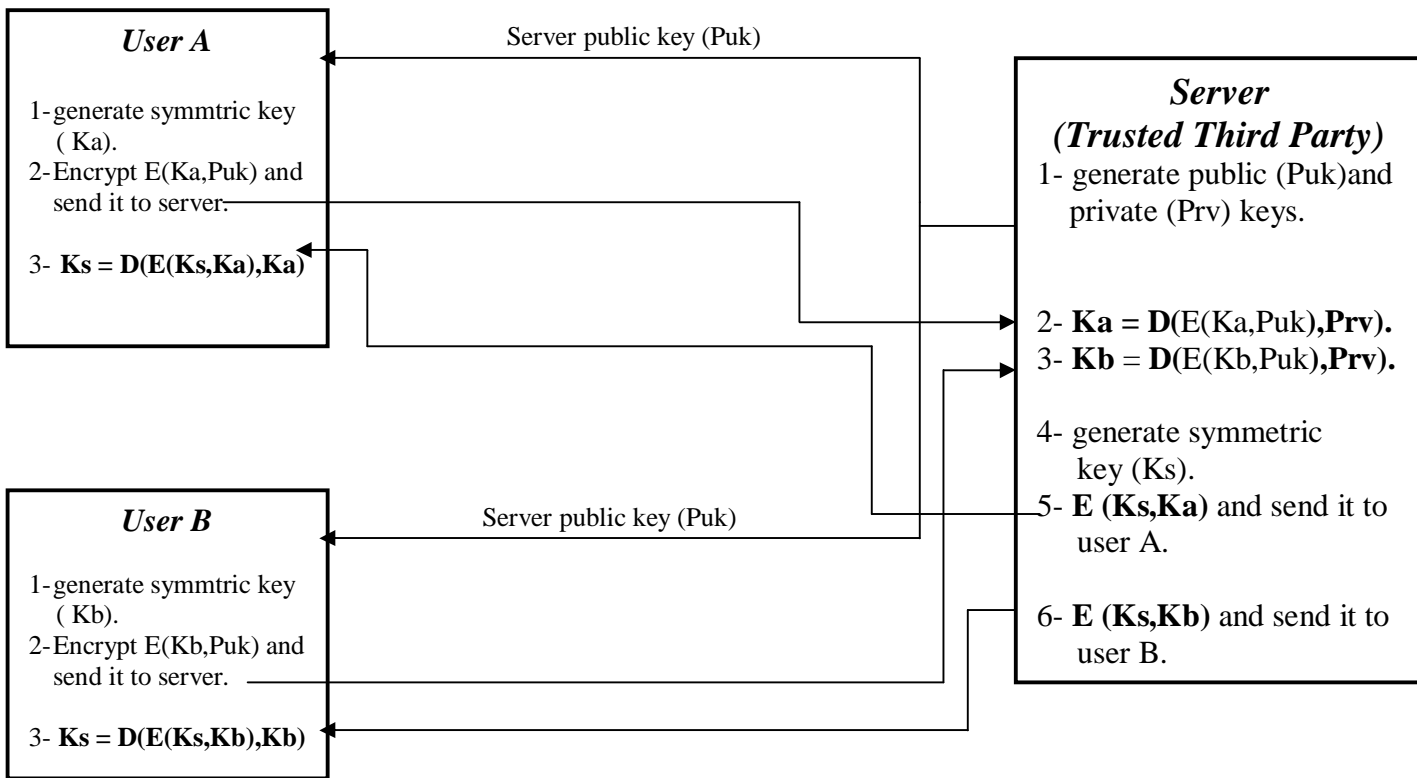


Figure (3.5): Generating shared key between user A and user B

Algorithm (3.24) Generate a Secret Key
Goal: Generate secret key.
Input: <i>their_Puk</i> : public key of the second user. <i>Prv</i> : Private key of the first user.
Output: <i>Secret_Key</i> : Secret Key.
Step1: Create a Key agreement (<i>ka</i>) using <i>getInstance</i> API function with "DH" key agreement algorithm.
Step2: Initializing a key agreement object with the private key (<i>Prv</i>).
Step3: Executing a key agreement phase using <i>dophase</i> API function. <i>ka.dophase</i> (<i>their_Puk</i> , <i>Prv</i>).
<i>Continue</i>

Step4: Generate a shared secret key using a *generateSecret* API function to a "DES" algorithm and put result in (*Secret_Key*) variable.

Step5: Return (*Secret_Key*).

Step6: End.

B. Encrypt Message unit

After the generation of a secret key by two users, the system encrypts the typed messages that are transferred between the two users using Triple DES algorithm with the shared secret key. The first user encrypts the message and sends it to the second one. When the second user receiving the message he/she will decrypt the message with the shared secret key generated before and the message will be display at the private chat window.

Client program of the sending user will call Algorithm (3.18) with (*message*, *Secret_Key*, "TripleDES") to encrypt message before send it. Client of the receiving user will call Algorithm (3.25) when the second user receiving a message to decrypt it and display on the private chat window.

C. Private Chat unit

After the message is encrypted, it will be ready to send during private chat unit. The client program changes the field of user message object to the following:

`_header = PRIVATE_CHAT.`

`_message = encrypted message.`

`_destination = second user ID.`

`_username = ID of user.`

`_user = user object of the sender.`

`_host = IP address of user.`

Then this message object will be send to the server to be sent to the destination user (see Algorithm (3.17)).

When the destination user receives the message, the Algorithm (3.25) is called with the (*_message*, *Secret_key*) to decrypt it with notification to the received user.

Algorithm (3.25) Decryption message

Goal: *Decryption the message.*

Input:

Emessage: encrypted message.

Secret_Key: Secret key.

Output: *message: clear text message.*

Step1: *Creating a cipher object using **getInstance** API method with RSA cryptograph algorithm.*

Step2: *Initialize this cipher object with the **DECRYPT_MODE** and (*Secret_Key*).*

Step3: *Decrypt the encrypted message using **doFinal** API function and put result in *Clear_Mess* variable.*

Clear_Mess ← *doFinal* (*Emessage*).

Step4: *Return* (*Clear_Mess*).

Step5: *End.*

3.7.5 Conference Chat Module

If user (A) wants to make a conference chat, the client will display to him/her an invitation window contains a list of an online users. Then the user (A) will select a number of those users who wants to chat with them. This module consists of five units: *Conference Create* unit, *Conference Invite* unit, *Conference Join* unit, *Conference Deny* unit and *Conference Leave* unit.

A. Conference Create Unit

This unit executes the following function, when users (A) want to start establishing a conference chat:

1. The client display an invitation window contains a list of online users.
2. User (A) select desirable users.
3. Client creates a list of the above users called *Selected_List*.
4. Client make a new user object named (conf-user) with the following fields:

username = user ID name.

hostaddress = socket address of the user, which contains an IP address
and port number.

User_Status = ONLINE.

isConference = true.

5. Client create a new message object with the following fields:

_header = CONFERENCE_CREAT.

_user = conf_user.

_userlist = Select_List.

6. Client sends the above message object to the server.
7. The server receives and reads the message object, and then create a table (see Algorithm (3.12)), called *conf_list* which contains user (A) ID and his/her selected list.
8. The server returns message object to the client to start invitation process unit.

B. Invite To Conference Unit

In this unit, the client will change the header of the message object to be
_header = CONFERENCE_INVITE.

And sends the message object to server, the server calls Algorithm (3.13) to send the invitation message to all users in the *Selected_List*.

C. Join To Conference Unit

1. After the invitation message arrived to all selected user in *Selected_List* through the server, the user has two options either accepts the invitation message and join to the conference or refuses the invitation.
2. If user decided to join the conference chat, the header of his/her message object will be

_header = CONFERENCE_JOIN.

3. Client sends the message object to server.
4. The server should add the joined user to a list of joined user and call Algorithm (3.15) to write to the conference chat.
5. Client reads the user object of the user, if the boolean field in his/her user object (*isConference*) is true then it will request from the server a user conference list (see Algorithm (3.16)) and make the header of user message object as

_header = CONFERENCE_LIST.

_destination = UserID.

and send it to the server.

D. Deny A Conference Unit

The second option to the user after the reception of invitation message is denying the invitation messages to join a conference chat, if the user refuse to join to conference chat, the header of message object will be

_header = CONFERENCE_DENY

and send his/her message object to the server, the server will reads the message object and finds the header is CONFERENCE_DENY, so the server removed this denying user from the *Selected_List* of conference user and from the *conf_list* and send it back to client.

E. Leave A Conference Unit

If user wants to leave the conference chat, his/her message object will updated the header to

`_header = CONFERENCE_LEAVE.`

and sends it to the server, the server will reads a message object and remove this user from the *conf_list* and call Algorithm (3.15) to notify all the users in the conference that this user has leaving the conference chat with a specific time.

Chapter Four

SIMSM Interface and Evaluation

4.1 SIMSM Interfaces

SIMSM system consists of two interfaces: Server interface, which is installed in the Administrator computer, and Client interface which is installed in the other remote computers.

The client part is executed by SIMSM_client.exe, it must be run in the remote computers, while at the Administrator side; the SIMSM_server.exe is executed to start client's serving.

4.1.1 SIMSM Server Program Interfaces

SIMSM server interface is designed to enable the Administrator to serve clients reside at the host or remote PC (s) easily.

When SIMSM_server.exe is executed, the frame shown in Figure (4.1) will appear to the administrator.



Figure (4.1) Server login frame

This frame consists of the following fields:

1. **Administrator Name** textbox: to allow Administrator to enter his/her own name for login.
2. **Password** textbox: to allow Administrator to enter his/her own password.
3. **Login** button: after the administrator enters the above information; he/she will click on login button to access the Main frame of SIMSM server part.

Then after, SIMSM server program will check if the entered name and password are both valid or not. If they are valid then the Main frame will appear as shown in Figure (4.2). Otherwise, an error message will appear for three times to re-login, and then it terminates the execution.



Figure (4.2) Server main frame

The Main frame of the server program consists of:

1. **Start Server** button: to start jobs of SIMSM server. Once the server is started it will define a Server Socket, wait an acceptance from clients and then start threads.
2. **Stop Server** button: to stop server from serving clients by closing the server socket while the administrator still logging.
3. **Exit** button: to terminate the execution of the system.

4. **About** button: A click on this option will cause the display of a frame contains short information about the SIMSM server, as shown in Figure (4.3).



Figure (4.3) Server about frame

5. **Log** button: this button is to view the log table of the users, as shown in Figure (4.4).

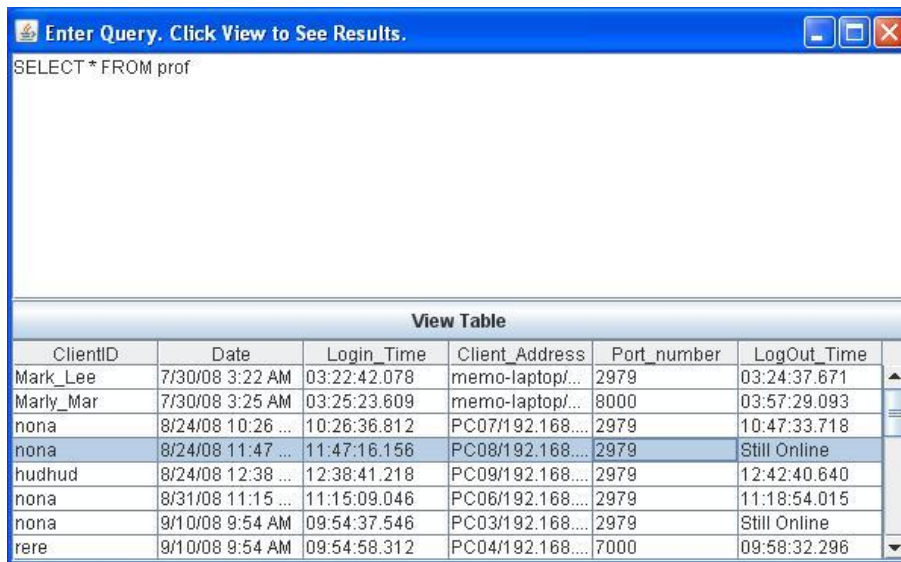
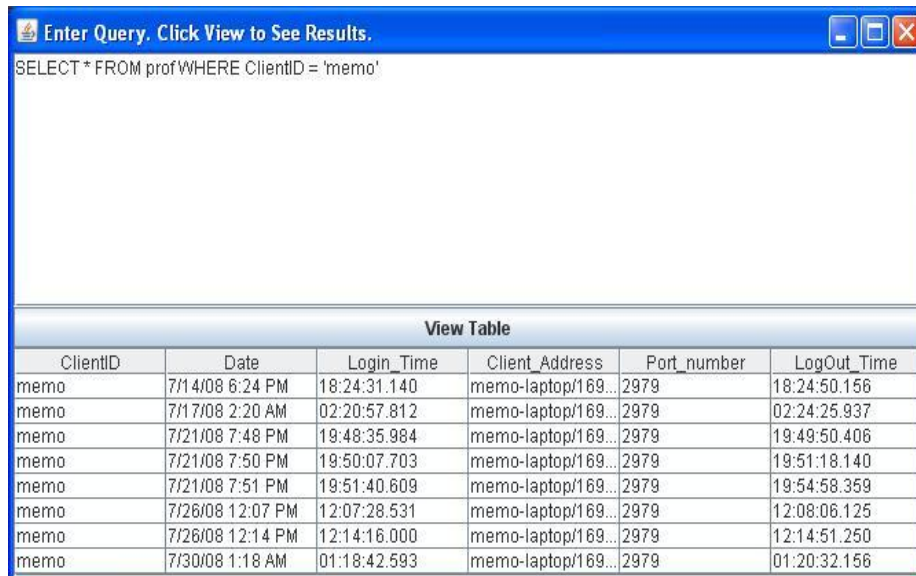


Figure (4.4) Log of users

As displayed in the Figure (4.4), this figure consists of:

1. **SQL Query** area: allow administrator to filter the table by writing an SQL query, Administrator can *Select*, *Delete* and *Update* the table of the users as shown in Figure (4.5).

2. **View Table** button: to see the result of SQL query issued by administrator, as shown in Figure (4.4).
3. **Log Table**: to view the result of the query in this table, this table contains information such as userID, date and time, login time, client address, port number and logout time of the user.



The screenshot shows a window titled "Enter Query. Click View to See Results." with a text area containing the SQL query: `SELECT * FROM prof WHERE ClientID = 'memo'`. Below the query area is a "View Table" button. The results are displayed in a table with the following columns: ClientID, Date, Login_Time, Client_Address, Port_number, and LogOut_Time. The table contains 9 rows of data.

ClientID	Date	Login_Time	Client_Address	Port_number	LogOut_Time
memo	7/14/08 6:24 PM	18:24:31.140	memo-laptop/169...	2979	18:24:50.156
memo	7/17/08 2:20 AM	02:20:57.812	memo-laptop/169...	2979	02:24:25.937
memo	7/21/08 7:48 PM	19:48:35.984	memo-laptop/169...	2979	19:49:50.406
memo	7/21/08 7:50 PM	19:50:07.703	memo-laptop/169...	2979	19:51:18.140
memo	7/21/08 7:51 PM	19:51:40.609	memo-laptop/169...	2979	19:54:58.359
memo	7/26/08 12:07 PM	12:07:28.531	memo-laptop/169...	2979	12:08:06.125
memo	7/26/08 12:14 PM	12:14:16.000	memo-laptop/169...	2979	12:14:51.250
memo	7/30/08 1:18 AM	01:18:42.593	memo-laptop/169...	2979	01:20:32.156

Figure (4.5) View log by performing specific query

4.1.2 SIMSM Client Program Interfaces

SIMSM Client is designed to allow user to chat securely. When a user executes SIMSM_client.exe, the **Login** form shown in Figure (4.6) will appear. The main function of this frame is to authenticate the user to login to SIMSM messenger.



Figure (4.6) SIMSM Login form

This frame consists of the following fields:

1. **SIMSM ID** textbox: to allow user enter its own ID for login.
2. **Password** textbox: to allow user to enter the password assigned to his/her UserID.
3. **Login** button: to login an already registered user after entering its valid ID name and password. If the user enters wrong UserID or password or

both, an error message appears as shown in Figure (4.7) for three times then the user must wait for 3 minutes and try to login again.

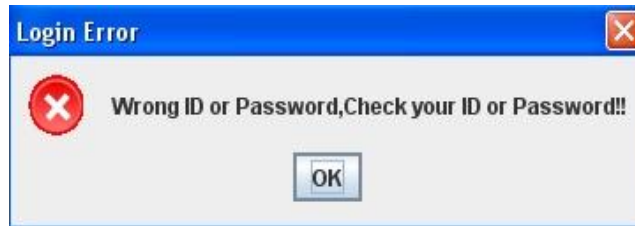


Figure (4.7) Error login message

4. **Sign-Up** button: to allow user to register as a new member to the system.
5. **Sign-In Problem** button: to allow the user to change its own password and retrieving the forgetting UserID after proving his/her identity.
6. **Quit** button: to close the SIMSM messenger login frame as shown in Figure (4.8).



Figure (4.8) Exit message

A. Registration into SIMSM

The following steps show how a new user can use SIMSM client messenger for registration process:

Step1: Click on *Sign-Up* button in Login frame, then the **Register** frame will appear as shown in Figure (4.9).

Registration

Please full the following information :

First Name : marwa

Last Name : saad

User ID : memo

Password :

Confirm your Password :

Your Birthday : January 24 , 1985 (Month,DD,YYYY)

Address : Baghdad

Select a question and answer it

Whats your favorites color?

Answer : orange

Figure (4.9) Registration frame

This frame consists of the following fields:

1. **First Name** textbox: first name of user, its length must be between 2 and 15.
2. **Last Name** textbox: last name of user its length must be between 2 and 15.
3. **UserID** textbox: user ID name to be used to login, it should not contain special characters like (!, @, #, \$, % etc).
4. **Check Availability** button: clicking on this button allows the user to check if his/her ID name is used before or not.
5. **Password** textbox: password to be used with the above UserID for login, its length must be between 8 and 15, containing at least two characters between (a-z), two characters between (A-Z), two number

value (0-9) and two special characters, if the password didn't match the required conditions, an error message will appear as shown in Figure (4.10).

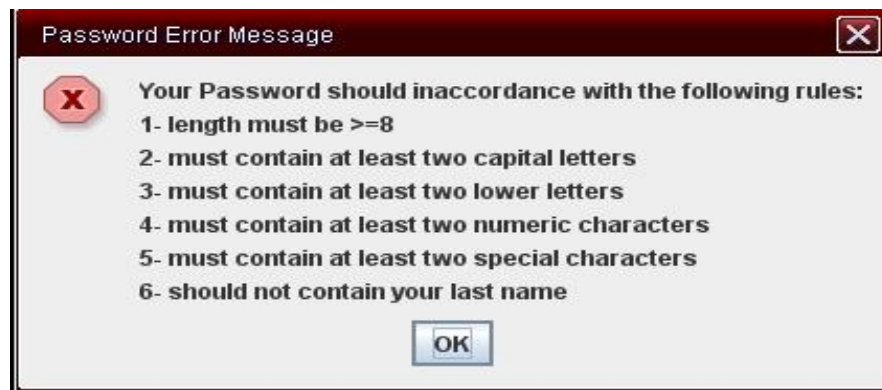


Figure (4.10) Password error message

6. **Question List:** list of questions will appear to the user to select one of them.
7. **Answer** textbox: the answer to the issued question will appear in this textbox. This answer is needed when user forget/change its password or forgot his/her ID name.

Step2: After filling the above registration information, the user must click on **Submit** button to send their information to the server. The UserID, password and answer of the question will be encrypted then they sent with the others to the server.

Step3: If the entered information accepted by server, then the frame shown in Figure (4.11) will appear, otherwise an error message corresponding to the information error will appear and another try is made by the user.

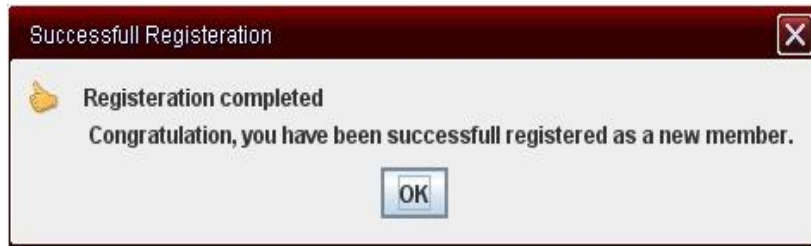


Figure (4.11) Successful registration message

B. Sign-In Problem

When the user clicks on *Sign-In Problem* button as shown in Figure (4.6), the Sign-In help frame, as shown in Figure (4.12), will appear. Through this frame, the user is allowed to change his/her password and retrieve the forgotten ID name.

Figure (4.12) Sign-In problem frame

In this frame the user can *Change password* and *Retrieve the forgotten ID name*.

In both cases, the user must first confirm his/her identity.

Confirming Identity area is done by entering the same information at registration process, which are: *Birthday Date*, *Address* and *Answer*.

A. Changing Password

The following steps illustrate the process of changing a known or forgotten password.

Step1: In order to change the password, the user must first confirm his/her identity to the SIMSM server. If the birthday date, address or answer were wrong then an error message will be appear to the user and another try will made.

Step2: After confirming the identity, the user must enter the new password with its conditions two times for confirmation as shown in Figure (4.13). Then the user should press on *Change Your Password* button to accept the change.

SIMSM Sign-in Help

SIMSM Sign-in Problems :

If you've forget the password, please Confirm your identity bellow then enter the new password.
 If you've forget your SIMSM ID, you can retrieve it by confirming your identity and providing us with your password associated with your account.

1. Confirm Your Identity :
 Please enter the Birthday, Country, answer of the question, you provided when you registered.

Your Birthday : January 24 , 1985 (Month,DD,YYYY)

Your City : Baghdad

Select the Question: whats you favorite color?

Answer : Blue

2. Choose One of These Options :

Forget Your SIMSM Password ?
 Enter a new SIMSM Password:

Re-Type your new password :

Change Your Password

Forgot Your SIMSM ID?
 Enter Your SIMSM Passwod:

Find SIMSM ID

Cancel

Figure (4.13) Changing password frame

Step3: After clicking on *Change Your Password* button, client will encrypt the password and send it to the server to change the user's password in the user registration database (RegDB). Then, a message box will appear to inform the user that password changing operation was successfully done.

B. Retrieving User ID

The following steps illustrate the process of retrieving the forgotten ID of the user.

Step1: if the user forgets his/her ID, he/she must confirm his/her identity, and enter password in the field of the second internal frame. See Figure (4.13).

Step2: Clicking on *Find SIMSM ID* button, client will encrypt the password and send it to the server to retrieve the ID name of this password.

C. Login into SIMSM

When SIMSM **Chat** frame is appeared to the user, as shown in Figure (4.6); the user should enter a valid SIMSM ID and password, and then made a click on Login button. Now connection is provided, and the login frame is displayed as shown in Figure (4.14).

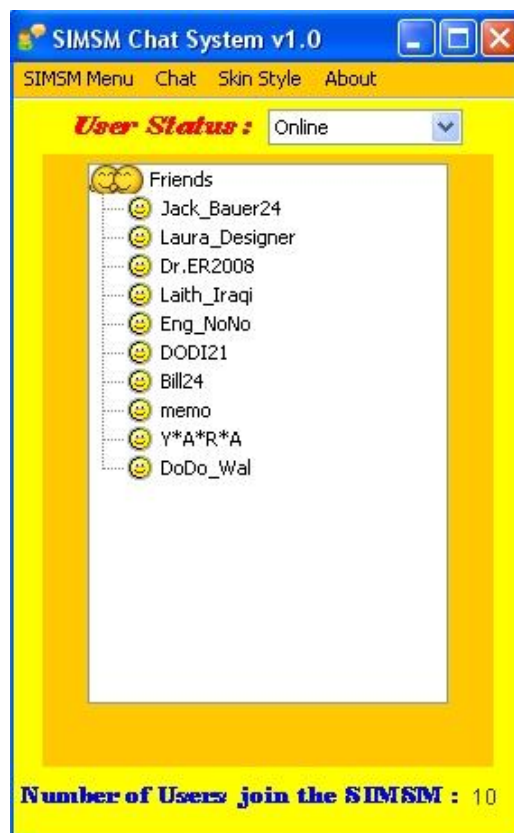


Figure (4.14) SIMSM chat frame

SIMSM chat frame has a tree whose Look & Feel (appearance and the way in which the user interacts with the program) is similar to that of YAHOO's. This tree is updated on user login, logout or change status.

SIMSM chat frame contains four menus (*SIMSM Menu*, *Chat*, *Skin Style*, and *Help* menu), and it contains user status box, list of users joined to the system, and total number of users joined to the SIMSM messenger.

1. **SIMSM Menu**: this menu contains the following options as shown in Figure (4.15).

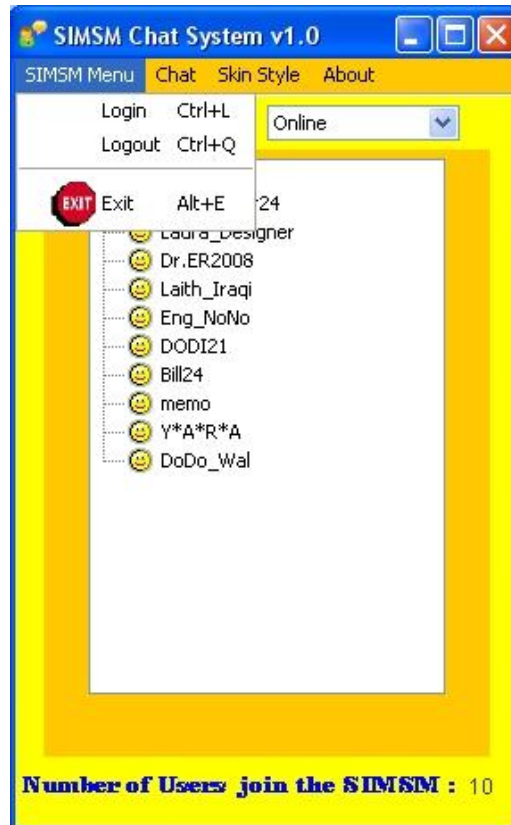


Figure (4.15) SIMSM first menu

- (a) **Login** option: if the user likes to re-login with another ID and password, he/she can perform that by pressing Login option. The same steps of Login into SIMSM will be executed again.
- (b) **Logout** option: this option is used to logout the user from the SIMSM system. Clicking on this option makes SIMSM client to send the logout user ID to the server for removing it from the Online User table.
- (c) **Exit** option: this option is used to quit the SIMSM messenger. A click on Exit option will make SIMSM ask user to confirm end of program.

If user select yes, otherwise the SIMSM will closed, SIMSM still working.

2. **Chat** menu: this menu has one option, which is of *starting a conference chat* as shown in Figure (4.16).



Figure (4.16) Conference chat menu

3. **Skin Style** menu: the SIMSM messenger contains three different skin themes as shown in Figure (4.17).

- (a) Windows skin
- (b) Metal skin
- (c) Motif skin

The default skin is the Windows skin; the other two skins are shown in Figure (4.18).

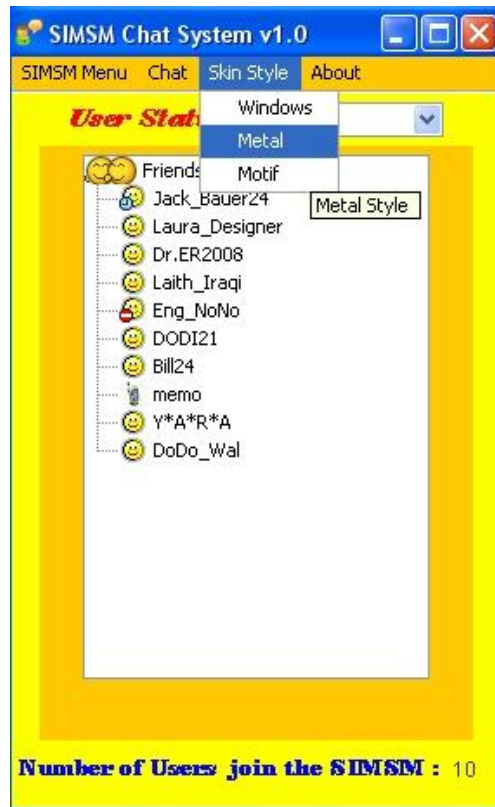


Figure (4.17) Skin style menu

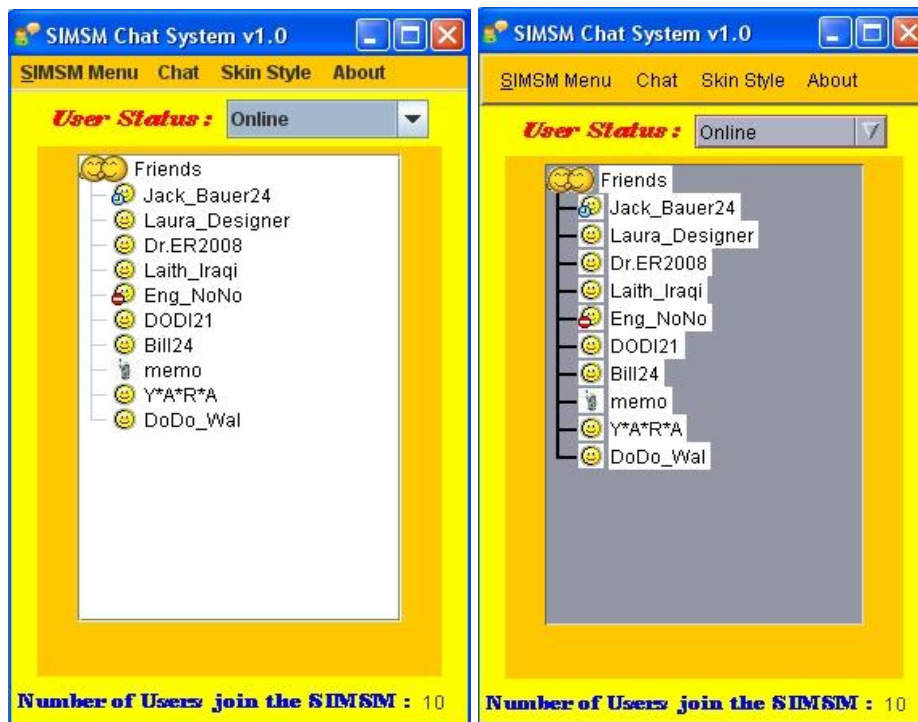


Figure (4.18) Motif and Metal skin

4. **About** menu: this menu has one option, which is *About*, as shown in Figure (4.19). Clicking on this option causes the appearance of a message demonstrates the SIMSM chat messenger, as shown in Figure (4.20).



Figure (4.19) Help Menu



Figure (4.20) About flash window

D. Starting Conference Chatting

When user starts a conference chat by clicks on the option in *chat menu* as shown in Figure (4.16), the invitation frame will appear as shown in Figure (4.21).

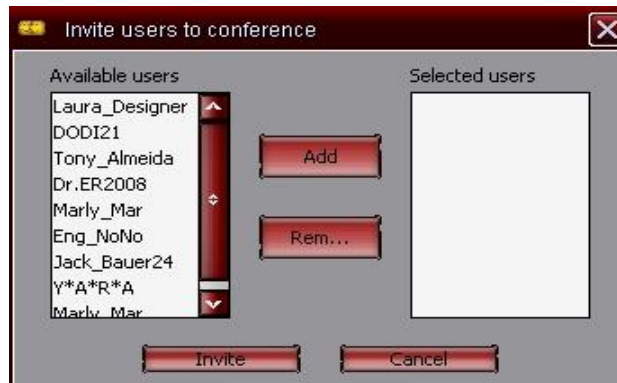


Figure (4.21) Invite users to conference

This frame contains the following:

- (a) **Available user list** area: contains the available users in the SIMSM messenger, the user can select one or more users to be invited to conference chat.
- (b) **Selected users list** area: contains the ID of the selected users for conference chat.
- (c) **Add** button: to add user (s) to the selected users list as shown in Figure (4.22).

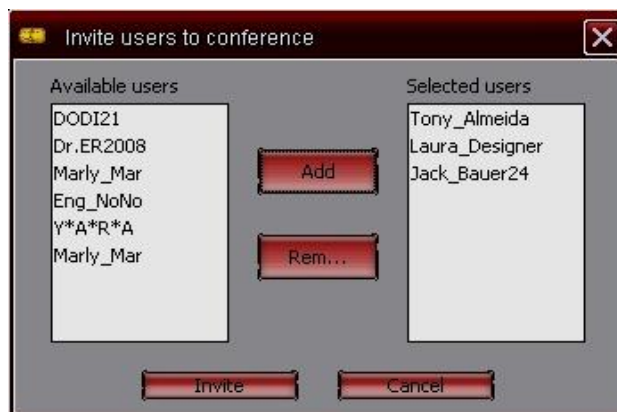


Figure (4.22) Adding users to selected list

- (d) **Remove** button: to remove user(s) from the selected users list.
- (e) **Cancel** button: to close the *Invite Users to Conference* frame and canceling the invitation to conference chat.

(f) **Invite** button: when user clicks on **Invite button** an invitation message box will be sent to all users in the **Selected users** list, as shown in Figure (4.23).



Figure (4.23) Conference invitation message

The invited user either *Accept* or *Deny* the invitation, if the user accepts the invitation, he/she will join the conference chat with the other users in conference as shown in Figure (4.24).



Figure (4.24) Accept the invitation

After joining a conference chat, the user can adding other users to joining the conference by using *Add to Conference* option in **Conference menu**.

The conference chat frame has a list component used to display the available users list as shown in Figure (4.25).



Figure (4.25) Starting conference chat

Otherwise, the user will deny the invitation, as shown in Figure (4.26).



Figure (4.26) Deny joining the conference

E. Starting Private Chat

To start secure private chat, the user must double click on the user ID name of the only online users then a private chat window will appear as shown in Figure (4.27).



Figure (4.27) Private chat window

The private chat frame is a Java frame with an area for typing a message and Java Editor Pane for displaying received messages. The Java Editor Pane has the ability to display HTML text, this helped to add smileys and other HTML tags like font, image etc. SIMSM provides smileys like :), >:), :)) etc.

Private chat frame contains one menu: Help menu.

Help menu: this menu has one option, which is **Help**. Clicking on it will display another frame contain a help with smile face symbols as shown in Figure (4.28).



Figure (4.28) Smile faces code

4.2 SIMSM Evaluation

The evaluation of SIMSM has two prospectivites; *Security* and *Time Consuming*. The security evaluation of the established instant messaging system depends on two factors; the strength of the used algorithms, and the secure way to generate the required keys. While the time consuming of SIMSM is computed according to the time required for sending the instant messaging between users including the time required to encrypt and decrypt the message.

4.2.1 Security Factor

The security of the system was evaluated regarding to the three main processes: (1) Registration process, (2) Authentication process, (3) Chatting process.

A. Registration Process Security

To protect sensitive registration information from being attacked through the communication channel, SIMSM encrypts those information using RSA algorithm to satisfy confidentiality and integrity. The selection of RSA is based on two mathematical problems: the problem of factoring large number, and the discrete logarithm problem.

From the registration information is the password which is considered the most sensitive data. If the password of the user is attacked by brute force attack then it will takes very long breaking time to know the password. In the established system, the password is recommended to be between 8-15 consisting characters from (A-Z), (a-z), (0-9) and (*,#, \$,@,% ,!) i.e. 68 characters, so the system as whole consist of $(68)^8 + (68)^9 + \dots + (68)^{15}$ possible password of length between 8 to 15. It takes long time so it is very enough to make this type of attack not attractive.

B. Authentication Process Security

At the authentication phase, a secure procedure was used in the authentication to **protect password**. A basic problem in client/server applications is that the server wants to know who its clients are. In a password-based scheme, the client prompts the user for his/her name and password. The client relays this information to the server. The server checks the name and password and either allows the user into the system or denies access. The password is a *shared secret* because both the user and the server must know it. The obvious solution is to send the user's name and password directly to the server. But most computer networks, however, are highly susceptible to eavesdropping, so this is not a very secure solution.

To avoid passing a clear text password from client to server, SIMSM send a message digest of the password instead and the server creates a message digest of its copy of the password. If the two message digests are equal, then

the client is authenticated. But this simple procedure is vulnerable to a *replay attack*. A malicious user could listen to the digested password and replay it later to gain illicit access to the server. So to avoid this problem, some session-specific information were added to the message digest. In particular, the client generates a random number and a timestamp and includes them in the digest. These values must also be sent, in the clear, to the server, so that the server can use them to calculate a matching digest value. The server receives the extra information and includes it in its message digest calculations. Figure (4.29) shows how authentication of the proposed system works at the client side.

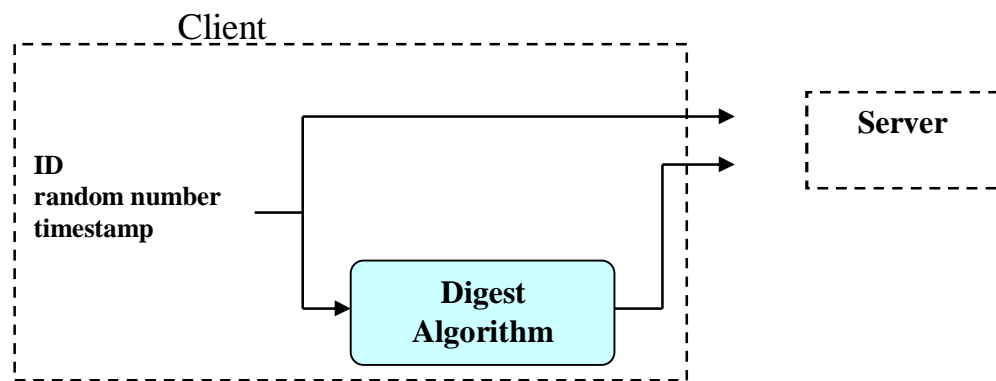


Figure (4.29) Protecting a password in SIMSM

C. Chatting Process Security

To achieve the goal of security of chatting which is the confidentiality of instant messaging three points are taken into consideration:

1. **Key distributed problem:** Any private-key system suffers from the key distributed problem. In order for secure communication to occur, the key must first be securely sent to the other party. In the proposed system, the Diffie-Hellman key exchange algorithm is used to solve this problem. Such that between any two clients, the secret key is calculated separately.
2. **Man-In-The-Middle attack problem:** This is solved by encrypting DH exchange with a shared symmetric key created by DES algorithm.

3. *Unencrypted communication problem:* The main target of the system which is the encryption of the exchange messages by using triple DES algorithm using the session key generated in the above point.

4.2.2 Time Consuming Factor

One of the important issues of any instant messaging system is depending on the consuming time to send and receives messages between clients. The time consuming was calculated regarding to the main processes in the system which are: Registration process, Authentication process, and Chatting process. The time of each process was calculated first alone for multiple cases, then the total time for receiving the message was calculated next.

A. Registration Process

The time for the following test cases including the time for computing keys in addition to the time for encrypting and decrypting the registered data using RSA.

Test Case One:

The consuming time in case one is calculated for the same user but at different times.

Table (4.1) Time to register one user

PC Name	PC Address	Time for Registration (measured in mile seconds)
PC1	192.168.0.11	625
PC1	192.168.0.11	578
PC1	192.168.0.11	641
PC1	192.168.0.11	563
PC1	192.168.0.11	609
Average time = 623.2 ms		

Test Case Two:

In this case five users make registration at the same time on different PCs.

Table (4.2) Time to register five users

PC Name	PC Address	Time for registration (measured in mile seconds)		
PC12	192.168.0.12	1047	579	872
PC10	192.168.0.10	953	625	679
PC8	192.168.0.8	875	750	969
PC3	192.168.0.3	1028	984	593
PC2	192.168.0.2	830	890	730
		Ave=946.6	Ave=756.6	Ave =768
Average Time = 832.73ms				

One can notice from the above tables, the average time to complete the registration process is less than one second, which is considered acceptable waiting time for the user. Especially it occurs for once.

B. Authentication Process

The time consuming for authentication process is the time for generating DH key pair, computing digest message plus additional time.

Test Case One:

The consuming time in case one is calculated for the same user but at different times.

Table (4.3) Time for authentication process

PC Name	PC Address	Server Address	Time to generate DH key pair (ms)	Total Time for login
PC1	192.168.0.1	192.168.0.101	172	828
PC1	192.168.0.1	192.168.0.101	172	719
PC1	192.168.0.1	192.168.0.101	172	719
PC1	192.168.0.1	192.168.0.101	188	656
PC1	192.168.0.1	192.168.0.101	172	859
Average Time (ms) =			175.2	756.2

Test Case Two:

In this case five users were logged-in to the SIMSM at the same time on different PCs.

Table (4.4) Time to login five users

PC Name	PC Address	Time to generate DH key pair				Time for Login		
PC12	192.168.0.12	187	188	171	Ave =182	2328	1125	656
PC10	192.168.0.10	187	250	203	Ave =231.3	950	1422	1906
PC8	192.168.0.8	187	172	188	Ave =182.3	2391	894	1281
PC3	192.168.0.3	204	188	188	Ave =193.3	937	1266	765
PC2	192.168.0.2	188	187	188	Ave =187.6	834	960	847
Average Time (ms) =						1488	1123.4	1091

C. Chatting Security

The time consuming for chatting process is the time for generating session key for the first send message only between two users. In addition to the consumed time for encryption of the message at client A, sending the message, and decryption of the message at client B.

Test Case One:

The consuming time in case one is calculated for the chatting process between two users (A and B) using different transmitted message length characters, see Table (4.5).

Test Case Two:

The consuming time in case one is calculated for the chatting process between six users using different transmitted message length, see Table (4.6).

Table (4.5) One-to-One chatting

Source PC Name	Destination PC Name	Message Length (char)	Time to generate Secret key (in millisecond)	Time for Encryption (in millisecond)			Time for Decryption (in millisecond)			Time to receive message from destination (ms)					
						Ave=			Ave=				Ave=		
PC1	PC2	20-40	16	15	0*	0*	Ave=15	15	0*	0*	Ave=15	243	131	147	Ave=13.6
PC1	PC2	40-60	16	16	0*	0*	Ave=16	16	0*	0*	Ave=16	147	152	147	Ave=148.6
PC1	PC2	60-80	16	16	0*	0*	Ave=16	16	0*	0*	Ave=16	153	147	147	Ave=149
PC1	PC2	80-100	16	32	0*	0*	Ave=32	32	0*	0*	Ave=32	153	153	147	Ave=151

Table (4.6) Five-to-One chatting

Source PC Name	Destination PC Name	Message Length (char)	Time to generate Secret key (in millisecond)	Time to receive message from destination (in millisecond)					
				PC2	PC3	PC4	PC5	PC6	
PC2,PC3,PC4,PC5,PC6	PC1	20-40	16	550	787	1146	816	848	Ave 829.4
PC2,PC3,PC4,PC5,PC6	PC1	40-60	16	520	692	1342	620	850	Ave 820.8
PC2,PC3,PC4,PC5,PC6	PC1	60-80	16	960	1140	750	840	670	Ave 872
PC2,PC3,PC4,PC5,PC6	PC1	80-100	16	869	1385	947	754	950	Ave 981

* Zero (0) indicate the measured time is not significant (it is less than 0.5 millisecond)

Test Case Three:

In this case seven PCs are chosen to be used for chatting. Such that the users of these PCs are chatting among themselves. For example the user of PC1 is *sending messages to* the users of PC3, PC5 and PC7, at the same time the user of PC1 is *receiving messages from* the users of PC2, PC3, PC4 and PC6.

Table (4.7) Chatting process

		PC Name (Sending Messages To) →						
PC Name (Receiving Messages From) ↓		<i>PC1</i>	<i>PC2</i>	<i>PC3</i>	<i>PC4</i>	<i>PC5</i>	<i>PC6</i>	<i>PC7</i>
	<i>PC1</i>			762		864		876
	<i>PC2</i>	673		830		764		737
	<i>PC3</i>	948	843					787
	<i>PC4</i>	790	906	864				
	<i>PC5</i>		890	853				
	<i>PC6</i>	890	925		830			
	<i>PC7</i>				744	643	984	

Once can notice from Table (4.7), when the server is busy to services a number of clients, the required time for serving and chatting between users is still less than one second.

Chapter Five

Conclusions and Suggestions for Future Work

5.1 Conclusions

During the design, implementation and evaluation phases of this research project, a lot of remarks have been issued in the following are some of them are listed:

1. The established system allows users on a LAN (Local Area Network) to communicate with each other securely, quickly and with the features: secure private chat, conference chat, and transfer smile with text.
2. The proposed protocol is estimated to have a real time performance good enough to the instant messaging from the time of generating the secret key to the time of decrypting messages, which is less than one second.
3. Applying security to conference chat with three or more parties using Diffie-Hellman key exchange techniques is timing involved because each party listens for keys coming from the left, performs a phase of the key agreement , and passes the resulting key to the right.
4. The appearance of **high values** of the time required for testing the system is due to many reasons for examples, the time needed to run the interface, the time needed to run resident programs, the time required to optimize the pc clock, etc.

5.2 Future Work

In the following some suggestions for future work are given to enhance the proposed secure instant messaging and make it more effective:

1. Provide more features to the proposed system (like File transfer, Voice chat, Application sharing, Offline messages, etc).
2. Running the proposed system on a large network area.
3. Making the established system scalable, by increasing number of servers, using distributed server instead of centralized server and therefore, the system will be scalable with respects to the number of users it can handle. Adding new user to the system without making worse its performance. Also make the system geographically scalable. It means that if its users or components lay far a part, this does not affect the overall performance of the system.
4. Future implementation can also support more types of messages; possible extensions include support for URLs.
5. Using a cluster of servers instead of using single server to achieve recovery and backup cases.

References

- [Abd05] Abdul Mannan, M.; "*Securing Public Instant Messaging*", M.Sc. thesis, School of Computer Science, Carleton University, Ottawa, Ontario, Canada, August, 2005.
- [Alm03] Almani, S.; "*Secure Instant Messaging: the Jabber Protocol*", M.Sc. thesis, Electrical Computer Engineering, Oregon State University, June 3, 2003.
- [Bry06] Bryn Mawr College Website, "*What are primitive roots?*", 2006.
<http://www.brynmawr.edu/math/people/stromquist/numbers/primitive.html>
- [Can01] Canavan, E., J.; "*Fundamentals of Network Security*", Artech House, ISBN 1-58053-176-8, 2001.
- [Cun06] Cunningham & Cunningham Inc, "*Discrete Logarithm Problem*", 2006.
- [Dav07] Davis, K.; "*AskNow Instant Messaging: innovation in virtual reference*", National Library of Australia, Staff Paper, 2007.
- [Dun02] Duncan, R.; "*An Overview of Different Authentication Methods and Protocols*", SANS (SysAdmin, Audit, Network, and Security) Institute, White Paper, 2002.

- [Fip02] Federal Information Processing Standards Publications 180-2 (FIPS), "*Secure Hash Standard*", U.S. DOC, National Institute of Standards and Technology, August 1, 2002.
- [Fis00] Fisch, A., E.; White, B., G.; "*Secure Computers and Networks: Analysis, Design and Implementation*", CRC Press, 2000.
- [Fly04] Flynn, N.; "*Instant Messaging Rules: a business guide to managing policies, security, and legal issues for safe IM communication*", AMACOM, ISBN 0-8144-7253-2, 2004.
- [Fre03] Fredrik, S.; "*Implementation of an Instant Messaging Client using the OMA IMPS Protocol*", M.Sc. thesis, Umea University, Sweden, 2003.
- [Fun05] Fung, T., K.; "*Network Security Technologies, Second Edition*", CRC press Company, 2005.
- [IEC04] International Engineering Consortium; "*Instant Messaging: Definition And Overview*", December, 2004.
http://www.iec.org/online/tutorials/instant_msg/
- [Kam00] Kammer R. G.; "*Digital Signature Standard (DSS)*", Federal Information Processing Standards Publication 186-2, U.S. Department of commerce, National Institute of Standard and Technology, Paper, January 27, 2000.

- [Kes07] Kessler, C., G.; *"An Overview of Cryptography"*, White Paper, December, 2007.
<http://www.garykessler.net/library/crypto.html>
- [Kim03] Kim H., C.; Au L.; Wang G., W.; *"Secure Instant Messaging System"*, CSE 400 Final Report, April 27, 2003.
- [Kli03] Kling, M. ; *"Unsecured Session with ICQ: applying forensic computing"*, M.Sc. thesis, Department of Software Engineering and Computer Science, Bleking Institute of Technology, Sweden, June, 2003.
- [Knu98] Knuden, B., J.; *"Java Cryptography"*, First Edition, O'Reilly, ISBN 1-56592-402-9, May, 1998.
- [Kol01] Kolodgy, C.; *"Biometrics: You Are Your Own Key"*, School of Management, the University of Texas at Dallas, Center for Information Technology and Management, 2001.
http://citm.utdallas.edu/research/Publications/white_papers_source/Biometrics.pdf
- [Leg04] Liggio, S.; Kulve, T.; Riva, O.; Saarto, J., and Kojo.; *"An Analysis of Instant Messaging and E-mail Access Behavior in Wireless Environment"*, IIP Mixture Project, University of Helsin Department of Computer Science, March, 2004.
- [Man04] Mannan, M.; Oorshot V.; *"Secure Public Instant Messaging"*, M.Sc. thesis, Carleton University, September, 2004.

- [Mol07] Mollin, A., R.; *"An introduction to cryptography, Second Edition"*, Chapman & Hall/CRC, Taylor & Francis Group, ISBN 1-58488-618-8, 2007.
- [Oll04] Ollmann, G.; *"Instant Messaging Security: Securing Against the Threat of Instant Messengers"*, White Paper, March, 2004.
<http://www.technicalinfo.net/papers/IMSecurity.html>.
- [Ost06] Osterman Research Inc.; *"Instant Messaging Tough Enough for Business: No Server Required"*, An Osterman Research, White Paper Prepared for WebEx, September 2006.
- [Pfl97] Pfleeger, P., C.; *"Security in Computing"*, Prentice Hall, 1997.
- [Pfl02] Pfleeger, P., C.; Pfleeger, L., S.; *"Security in Computing"*, Third Edition, Prentice Hall PTR, ISBN 0-13-035548-8, December 2, 2002.
- [Pic02] Piccard, P.; *"Risk Exposure Through Instant Messaging And Peer-to-Peer (P2P) Networks"*, an X-Force White Paper, Internet Security Systems, April, 2002.
- [Pip04] Piper, F., Robshaw, J., B., M., and Schwiderski-Grosche S.; *"Identities and authentication"*, Royal Holloway College, University of London, 2004.
<http://www.berr.gov.uk/files/file15274.pdf>

- [Rit05] Rittinghouse, J., W.; Ransome, J., F.; ***"IM Instant Messaging Security"***, Elsevier digital press, ISBN 1-55558-338-5, 2005.
- [Sec01] The Secure Hash Algorithm Directory, ***"The Secure Hash Algorithm Directory MD5, SHA-1 and HMAC Resources"***, 2001.
<http://www.secure-hash-algorithm-md5-sha-1.co.uk/>
- [Sha05] Shaw, P.; ***"Instant Messaging in the Workplace: Real Benefits, Real Risks"***, the source for IBM eServer Solution, Published 3 March, 2005.
<http://www.mcshowcase.com/mcpress/Showcase.nsf/Focus/8C77AE2EBA38BA0388256FB8007A5807?OpenDocument>
- [Sta06] Stamp, M.; ***"Information Security: principle and practice"***, Willy Interscience, ISBN 13 978-0-471-73848-0, 2006.
- [Sym02] Symantec Enterprise Security Inc.; ***"Securing Instant Messaging"***, White Paper, 2002.
- [Wha08] ***"Learn IT: Instant Messaging in the Workplace"***, 2008.
http://whatis.techtarget.com/definition/0,,sid9_gci934583,00.html
- [Wil03] William, S.; ***"Cryptography And Network Security, Principles and Practice"***, Prentice Hall, Upper Saddle River, New Jersey, 2003.

- [Wil04] Williams, N.; Ly, J.; "*Securing Public Instant Messaging (IM) At Work*", Technical Report 040726A, Swinburne University of Technology, Melbourne, Australia, September, 2004.
- [Wrz02] Wrzesinska, M.; "*A Secure Instant Messaging System*", M.Sc. thesis, Vrije Universiteit, June, 2002.
- [Zup06] Zupan, A.; "*Digital signature as a tool to achieve competitive advantage of organization*", M.Sc. thesis, University of Ljubljana, 2006.
- [Zwi00] Zwicky, D., E.; Cooper, S.; Chapman, B., D.; "*Building Internet Firewalls*", Second Edition, O'Reilly & Associates, ISBN 1-56592-871-7, June, 2000.

Web Sites:

- [Zer08] http://en.wikipedia.org/wiki/Zero-knowledge_proof
- [Web04] <http://www.webopedia.com/TERM/D/DES.html>
- [Wik07] http://en.wikipedia.org/wiki/Cryptographic_protocol
- [Wik08] <http://en.wikipedia.org/wiki/Modular-arithmetic>

Appendix A

Database Connection

In order to make Java access information in the database, it should establish a connection with JDBC driver and to make this connection the following steps are needed:

Step1: Creating the Microsoft Access Database

All you need to do is create the Access database and enter the data. To do this, launch Microsoft Access and create a database.

Step2: Making the DSN Connection

Now that the database is complete, there is one more step you need to take to ensure the Java application you develop can make a connection to it. This step is highly dependent on the platform that you are using. In this example, you need to make a Data Source name (DSN) connection to the database. This is the mechanism that allows the application to connect to the database itself.

To accomplish this, you first need to bring up the Control Panel as indicated below and click on the *Administrative Tools* icon.

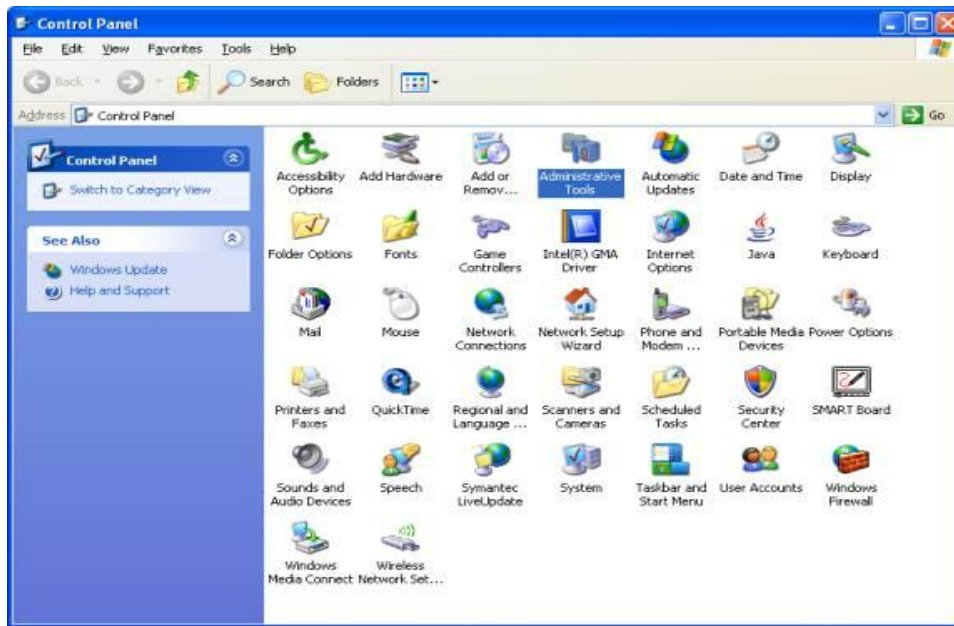


Figure (A.1) Control panel window

Once you get to the *Administrative Tools* dialog box, you then click on the *Data Sources* icon as seen in next Figure.

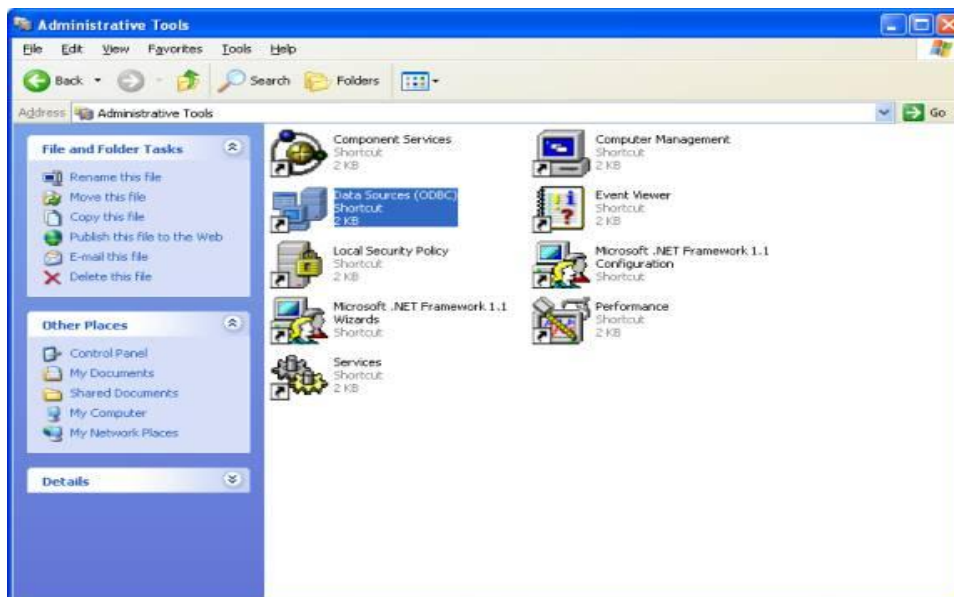


Figure (A.2) Administrative tools

Clicking on the *Data Sources* icon will bring you to the *ODBC Data Source Administrator* that will allow you to specify the name and the driver to be used by the application.

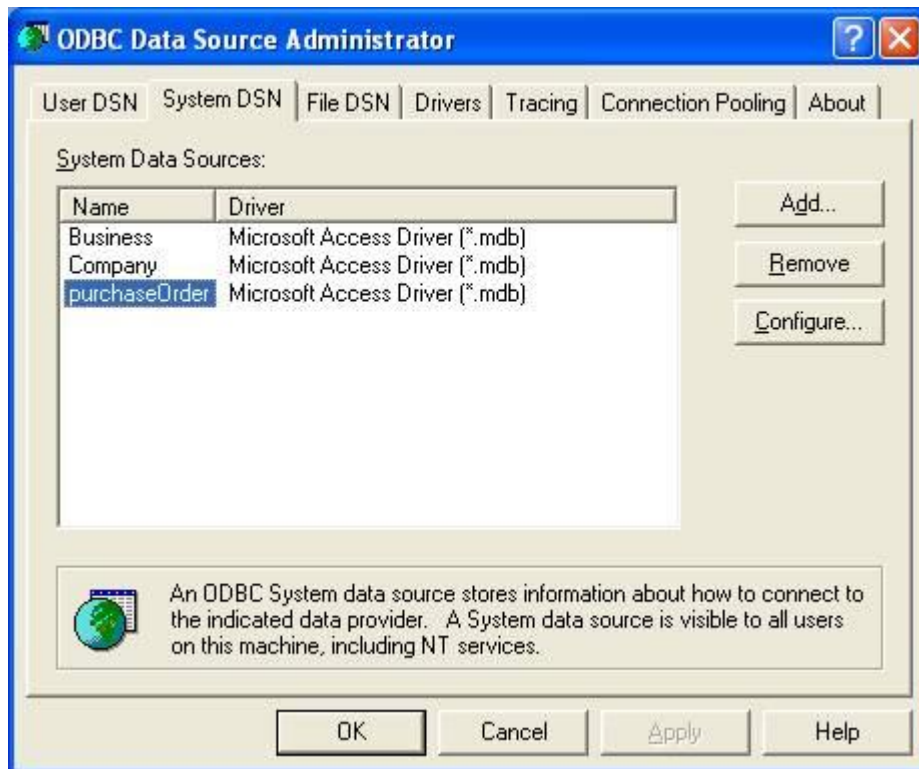


Figure (A.3) ODBC Data Source Administrator

Once you are in the *ODBC Data Source Administrator*, click on the *File DSN* tab and click on the *Add* button (note that you won't see the *purchaseOrder System Data Source Name* until you complete the add process). The *Microsoft ODBC Access Setup* dialog box appears as shown below.

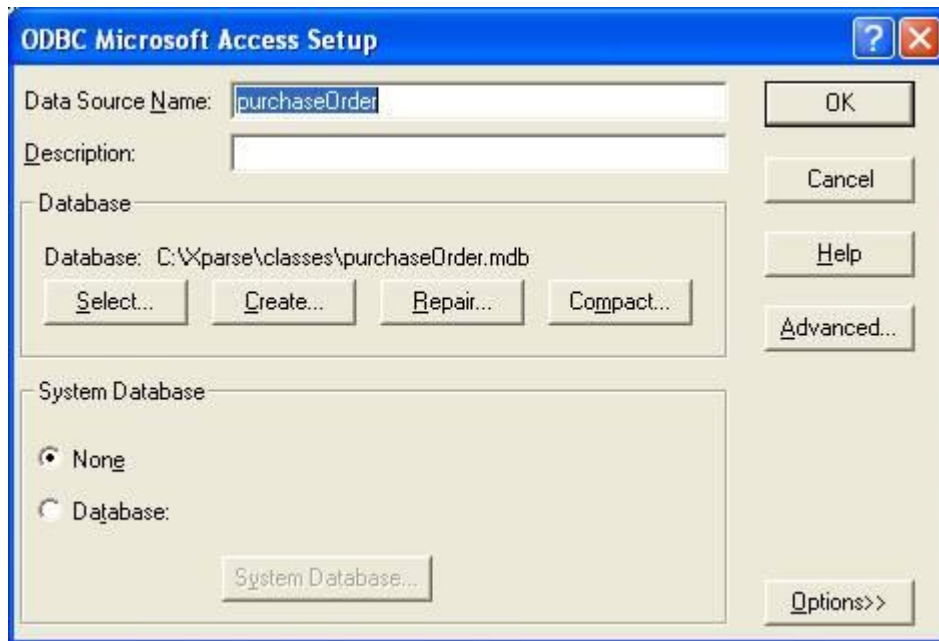


Figure (A.4) ODBC Microsoft access setup

There are two things that must be completed in this dialog box:

1. Create a Data Source Name.
2. Click on the Select button and literally find the db.mdb file that you previously created, as shown in the next figure.

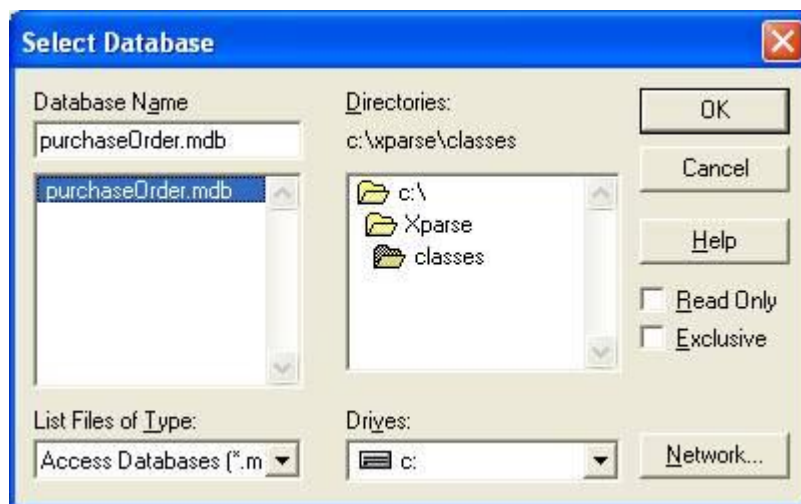


Figure (A.5) Select a database

At this point the DSN connection with database is complete.

الخلاصة

المحادثة الفوري هي شكل من أشكال الاتصال المباشر والحقيقي بين اثنين او أكثر من الناس بالاعتماد على النص المطبوع. أنظمة المحادثة الفورية نمت وانتشرت بسرعة بين مستخدمي الشبكة. لكن اغلب أنظمة المحادثة الفورية الحالية تعاني من مشاكل في الأمانة. المستخدمين يرغبون في الحفاظ على خصوصياتهم, واتصالاتهم يجب إن لا تستنسخ ولا تعدل من قبل طرف ثالث.

هذا المشروع يقدم تصميم وتنفيذ نظام محادثة فوري آمن. يحقق المشروع الأهداف الأمنية مثلاً خصوصية البيانات وسلامتها عن طريق التشفير. يضمن بان المحادثة تقرا فقط من قبل الشخص المعني. أسم النظام المقترح اختير ليكون (SIMSM) المختصر (SIMple Secure Messenger).

SIMSM صمم للشبكات المحلية بالاعتماد على تقنية الزبون-الخادم (Client-Server). SIMSM يمكن المستخدمين من إرسال واستقبال الرسائل الفورية بينهم. لا يحتاج إلى الاتصال بالانترنت. يرسل ويستقبل بسهولة الرسائل النصية القصيرة, يوفر ميزات الإرسال القياسية مثل الدردشة الفردية, دردشة المجموعة, إعلام المستخدم بالتغييرات الآنية لباقي المستخدمين و تشفير الرسائل النصية القصيرة.

هناك وحدات أساسية يتكون منها النظام المقترح وهي: وحدة التسجيل التي تسجل المستخدمين في النظام؛ وحدة الدخول التي تسمح للمستخدمين المخولين بالدخول إلى النظام؛ وحدة مشاكل الدخول الذي يتعامل مع مشاكل نسيان الهوية وكلمة السر للمستخدم؛ وحدة الدردشة الخاصة التي تمكن المستخدم من الدردشة الخاصة مع مستخدم آخر؛ ووحدة دردشة المجموعة التي تمكن أكثر من مستخدم للدردشة مع بعضهم البعض في نفس الوقت.

النظام المقترح قد فُيم طبقاً لعاملين مهمين في أنظمة الإرسال الفورية: الأمانة و الوقت المستهلك. عدة حالات تجريبية قد أخذت وبينت بان SIMSM مناسب لخدمة الدردشة

الآمنة. النظام المقترح صُمم باستخدام الدوال (Windows API Function) و لغة
البرمجة جافا _____ انسخة (6.0.0.105).



جمهورية العراق
وزارة التعليم العالي والبحث العلمي
جامعة النهريين
كلية العلوم

نظام محادثة فوري لشبكة اتصالات محلية مع بعض سمات الأمنية

رسالة

مقدمه إلى كلية العلوم في جامعة النهريين
كجزء من متطلبات نيل شهادة درجة الماجستير
في علوم الحاسوب

من قبل

مروة سعد ملكي القس

(بكالوريوس جامعة النهريين 2005)

المشرفون

د. جمال محمد كاظم

شوال 1929

د. عبير متي يوسف

تشرين الاول 2008