



بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

نَرْفَعُ دَرَجَاتٍ مَن نَّشَاءُ وَفَوْقَ
كُلِّ ذِي عِلْمٍ عَلِيمٌ

صَدَقَ اللَّهُ الْعَظِيمُ

سورة يوسف (٧٦)



Supervisor Certification

We certify that this thesis was prepared under our supervision at the Department of Computer Science/College of Science/Al-Nahrain University, by **Malath Sabri Kareem** as partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Signature:

Name: **Dr. Sattar B. Sadkhan**

Title : **Assist. Prof.**

Date : / / **2006**

Signature:

Name: **Dr. Abeer M. Yousif**

Title : **Lecturer**

Date : / / **2006**

In view of the available recommendations, I forward this thesis for debate by the examination committee.

Signature:

Name : **Dr. Taha S. Bashaga**

Title : **Head of the department of Computer Science,
Al-Nahrain University.**

Date : / / **2006**

Certification of the Examination Committee

We chairman and members of the examination committee certify that we have studied this thesis "**An Authentication Scheme for Instant Messaging System**" presented by the student **Malath Sabri Kareem** and examined her in its contents and that we have found it worthy to be accepted for the degree of Master of Science in Computer Science.

Signature:

Name: **Dr. Loay E. George**

Title : **Assist. Prof.**

Date : / /2007

(Chairman)

Signature:

Name: **Dr. Bara'a A. Attea**

Title : **Assist. Prof.**

Date : / /2007

(Member)

Signature:

Name: **Dr. Jamal M. Kadhum**

Title : **Lecturer**

Date : / /2007

(Member)

Signature:

Name: **Dr. Sattar B. Sadkhan**

Title : **Assist. Prof.**

Date : / /2007

(Supervisor)

Signature:

Name: **Dr. Abeer M. Yousif**

Title : **Lecturer**

Date : / /2007

(Supervisor)

Approved by the Dean of the Collage of Science, Al-Nahrain University.

Signature:

Name: **Dr. LAITH ABDUL AZIZ AL-ANI**

Title : **Assist. Prof.**

Date : / /2007

(Dean of Collage of Science)

Acknowledgment

First of all, profusely and thanks are due to Allah who enabled me to achieve this research work.

I would like to express my special gratitude to ***Dr. Sattar B. Sadkhan*** for his helpful supervision in the first stage of this project.

I'm indebted to ***Dr. Abeer M. Yousif*** for her valuable guidance, supervision and untiring efforts that enabled me to finish work in this research.

Grateful thanks to the Head of Department of Computer Science ***Dr. Taha S. Bashaga*** for confidence gave to me and for his continues support.

My deep appreciation goes to all of M.Sc. colleagues especially ***Haider M. Jaber*** for help and effort.

Finally, my special thanks to my parents, sisters and brothers for their support and encouragement during the period of my studies.

Malath

Abstract

Instant messaging system is considered to be one of the widely used application area, it is rapidly growing as a primary communications technology among corporate, educational and home users. Users of instant messaging systems need to authenticate each other before communicating, so the need for login authentication becomes an important issue.

This research deals with the design and implementation of instant messaging system (IMS) supported with a suggested method to authenticate users. The suggested method combines two different authentication mechanisms: Fixed Password and Digital Signature, where the authentication process passes through multi levels of security checks.

The proposed instant messaging system provides the users with some features like text chatting in public and private chat area, sending offline messages using private chat area, and changing the current password in the case of its stealing or forgetting. IMS uses TCP/IP protocol for all communications between hosts. All connections between server and clients are based on client/server networking model, while connections between clients is based on peer-to-peer networking model.

The proposed system has been evaluated from three points of views, which are: easy of use, fast execution and security, where the system provides the users with a common easy way for chatting in virtual real time. The security level provided for users of IMS is quite suitable for this kind of applications. IMS was implemented using Microsoft Visual Basic 6.0 programming language and tested on PCs connected to a LAN network at which all computers are running under Window XP operating system.

List of Abbreviations

| | |
|--------|--|
| AOL | American On-Line |
| ASCII | American Standard Code for Information Interchange |
| DSA | Digital Signature Algorithm |
| EBCDIC | Extended Binary Coded Decimal Interchange Code |
| HVAL | HAsh of VAriable Length |
| IM | Instant Messaging |
| ICQ | I Seek You |
| IRC | Internet Relay Chat |
| IP | Internet Protocol |
| ISO | International Standards Organization |
| LAN | Local Area Network |
| MAC | Message Authentication Code |
| MD | Message Digest |
| NIST | National Institute of Standards and Technology |
| OSI | Open Systems Interconnection |
| PC | Personal Computer |
| PKI | Public Key Infrastructure |
| RFC | Request For Comment |
| RSA | Rivest Shamir Adleman |
| SHA | Secure Hash Algorithm |
| SHS | Secure Hash Standard |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |

List of Contents

| | |
|---|-----------|
| Chapter One: Introduction | 1 |
| 1.1 Introduction to Instant Messaging System | 1 |
| 1.2 Network Fundamentals | 2 |
| 1.2.1 TCP/IP | 3 |
| 1.2.2 Ports and Sockets | 5 |
| 1.2.3 Network Structural Models | 6 |
| 1.3 Literature Survey | 8 |
| 1.4 Aim of Thesis | 10 |
| 1.5 Thesis Layout | 10 |
| Chapter Two: Message Authentication over Computer Networks | 12 |
| 2.1 Introduction | 12 |
| 2.2 Authentication Services Types | 13 |
| 2.3 Authentication Functions | 13 |
| 2.3.1 Message Encryption | 14 |
| 2.3.2 Message Authentication Code (MAC) | 16 |
| 2.3.3 Hash function | 16 |
| 2.4 Authentication Mechanisms | 18 |
| 2.4.1 Password | 18 |
| 2.4.2 Public Key Cryptography | 20 |
| 2.4.3 Zero-Knowledge Proof | 20 |
| 2.4.4 Digital Signature | 21 |
| 2.5 ElGamal Digital Signature | 23 |

| | |
|--|-----------|
| 2.5.1 Mathematical concepts | 23 |
| 2.5.2 ElGamal Algorithm | 24 |
| 2.5.3 Security of ElGamal Signature Scheme | 25 |
| Chapter Three: Design and Implementation of IMS | 27 |
| 3.1 Introduction | 27 |
| 3.2 Design Requirements and Considerations | 28 |
| 3.3 System Structure | 29 |
| 3.4 Initialization Module | 30 |
| 3.4.1 Keys Generation | 30 |
| 3.4.2 Start Serving | 41 |
| 3.5 Registration Module | 42 |
| 3.6 Login Module | 49 |
| 3.7 Chatting Module | 55 |
| 3.7.1 Chat | 56 |
| 3.7.2 Adding UserIDs to Buddy List | 57 |
| 3.8 Change Password Module | 57 |
| 3.8.1 Changing Known Password | 57 |
| 3.8.2 Changing Forgotten Password | 59 |
| Chapter Four: IMS Interface and Evaluation | 61 |
| 4.1 IMS Interface | 61 |
| 4.2 How to Use IMS Server | 61 |
| 4.3 How to Use IMS Client | 65 |
| 4.3.1 How to register in IMS | 66 |
| 4.3.2 How to Login to IMS | 68 |

| | |
|--|-----------|
| 4.3.3 Adding a UserID to Buddy List | 71 |
| 4.3.4 Changing Password | 73 |
| 4.4 IMS Evaluation | 76 |
| Chapter Five: Conclusions and Future Work | 79 |
| 5.1 Conclusion | 79 |
| 5.2 Future Work | 81 |

CHAPTER ONE

Introduction

1.1 Introduction to Instant Messaging System

Instant Messaging is the transmission of an electronic message over a computer network using software that immediately displays the message in a window on the screen of the recipient. Instant Messaging (IM) requires that both parties be logged onto their IM service at the same time. Instant Messaging also known as a "chatting," has become very popular for both business and personal use. In business, IM provides a way to contact co-workers any time of the day, providing they are at their computers.

Instant messaging requires the use of a client program that hooks up an instant messaging service. In early instant messaging programs, each letter appeared as it was typed, and when letters were deleted to correct typos this was also seen in real time. This made it more like a telephone conversation than exchanging letters. In modern instant messaging programs, the other party in the conversation generally only sees each line of text right after a new line is started. Popular instant messaging services on the public Internet include, AOL (American On-Line), Yahoo!, Skype, Google Talk, NET, Jabber, and ICQ (I Seek You) messengers.

Instant messaging typically boosts communication and allows easy collaboration. In contrast to e-mails, the parties know whether the peer is available and conversations are then able to happen in real time. On the other hand, people are not forced to reply immediately to incoming messages. This way, communication via instant messaging can be less intrusive than communication via phone, which is partly a reason why

instant messaging is becoming more and more important in corporate environments [Ans06].

Instant messaging applications are traditionally designed to require continuous access to a centralized server. Access to the centralized server is required to authenticate users. Authentication is one of the most fundamental features a centralized instant messaging server provides [Wik06]. User Authentication is the process of determining that a user is who he/she claims to be [Mar00]. Here are a few common reasons why one wants to verify an identity in the first place [Cha05]:

1. To control access to application.
2. To bind some sensitive data to an individual.
3. To establish trust between multiple parties to form some interaction with them.
4. To assure that a piece of information is genuine.

Within an application, one or all of these aspects may apply.

Since instant messaging system is considered one of the Internet services, the following section will present some related network fundamentals.

1.2 Network Fundamentals

The term "computer network" means a collection of autonomous computers interconnected by a single technology. Two computers are said to be interconnected if they are able to exchange information. Different devices on the network communicate with each other through a predefined set of rules (protocols). The device on a network can be in the same room or scattered through a building, or they can even be scattered around the world, connected by a long-distance communication medium [Cra99].

The layered concept of networking was developed to accommodate changes in technology. Each layer of a specific network model may be responsible for a different function of the network. Each layer will pass information up and down to the next subsequent layer as data is processed [Com06].

The standard model of a layered network is the 7-layer International Standards Organization (ISO) Open Systems Interconnection (OSI) Reference Model. The entire OSI model is not implemented, where the most common layered set of protocols in use is the Transmission Control Protocol/Internet Protocol (TCP/IP) set of protocols. TCP/IP works in a very similar manner to the OSI model in that it takes approach to provide network services. Each layer in the TCP/IP model communicates with the layers above and below it in the same way that the layers in the OSI model do [Eir00].

1.2.1 TCP/IP

The TCP/IP protocol allows computers of all sizes, from many different computer vendors, running totally different operating systems, to communicate with each other. TCP/IP is normally considered to be a 4-layer system as shown in figure (1.1) [Ric93]. TCP/IP is a broad set of rules and standards, these rules control how data on the network is sent on the correct path to reach its intended destination, how some communication errors are handled, and how logical connections between nodes on the network are established, maintained, and ended [Cra99].

Table (1.1): The Four Layers of the TCP/IP protocol

| | |
|-------------|----------------------------------|
| APPLICATION | TELNET, FTP, E-MAIL, ETC. |
| Transport | TCP, UDP |
| Network | IP, ICMP, IGMP |
| Link | Device driver and interface card |

The main workhorses of this protocol are IP, TCP, and UDP:

- 1. IP (Internet Protocol)** [Par99]: the Internet Protocol resides into Internet layer. Its main tasks are addressing of information datagrams (packet) between computers and managing the fragmentation process of these datagrams.
- 2.** At the transport layer, the two most common protocols are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) [Par99] [Ric93]:
 - TCP (Transmission Control Protocol):** provides a considerable number of services to the IP layer and the upper layers. Most importantly, it provides a connection-oriented protocol to upper layers that enable an application to be sure that a datagram sent out over the network was received in its entirety. In this role, TCP acts as a message-validation protocol providing reliable communications. If a datagram corrupted or lost, TCP usually handles the retransmission, rather than the applications in the higher layers. Also, TCP does not interpret the contents of the bytes at all. TCP has no idea if the data bytes being exchanged are binary data, ASCII characters, EBCDIC characters, or whatever. The interpretation of this byte stream is up to the applications on each end of the connection.

- **UDP (User Datagram Protocol):** UDP is a simple, datagram-oriented, transport layer protocol. UDP provides no reliability: it sends the datagrams that the application writes to the IP layer, but there is no guarantee that they ever reach their destination. Any desired reliability must be added by the application layer.

1.2.2 Ports and Sockets [Par99]

All upper-layer applications that use TCP (or UDP) have a **port** number that identifies the application. In theory, port numbers can be assigned on individual machines the administrator desires, but some conventions have been adopted to enable better communication between TCP implementations, which enables the port number to identify the type of services that one TCP system is requesting from another. Port numbers can be changed, although this can cause difficulties. Most systems maintain a file of port numbers and their corresponding service. Each communication circuit into and out of the TCP layer is uniquely identified by a combination of two numbers, which together are called a **socket**. The socket is composed of the IP address (32-bit) of the machine and the port number used by the TCP software. Both the sending and receiving machines have sockets. Because the IP is unique across the interwork, and the port numbers are unique to the individual machine, the socket numbers are also unique across the entire interwork. This enable a process to talk to another process across the network, based entirely on the socket number.

1.2.3 Network Structural Models

The most two common popular structural models to setting up networks are:

1. **Client/Server Networking Model** In this design, a small number of computers are designated as centralized *servers* and given the task of providing services to a larger number of user machines called *clients*, see figure (1.2) Client/server architecture is the basis for most TCP/IP protocols and services [Cha04]. TCP/IP supports two essential modes of network communication [Dal95]:

1. **Streams** are a connection-oriented form of communication, which means that there is a persistent connection between the client and the server for the duration of the communication (like a telephone conversation).
2. **Data-grams** are a connection-less form of communication, which means that there is not a persistent connection between the client and the server. Each message between the client and the server is a separate connection, containing its own addressing information.

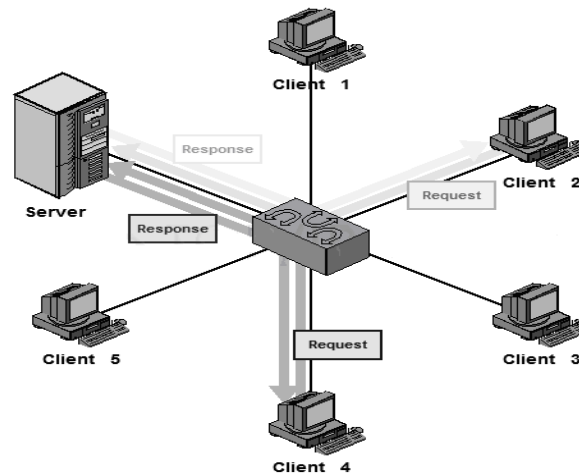


Figure (1.2) Client/Server Networking

2. Peer-to-Peer Networking Model In a strict peer-to-peer networking setup, every computer is an equal, a *peer* in the network. Each machine can have resources that are shared with any other machine. There is no assigned role for any particular device, and each of the devices usually runs similar software. Any device can and will send requests to any other, as illustrated in figure (1.3). In this model, each device on the network is treated as a peer, or equal. Each device can send requests and responses, and none are specifically designated as performing a particular role. This model is more often used in very small networks [Cha04].

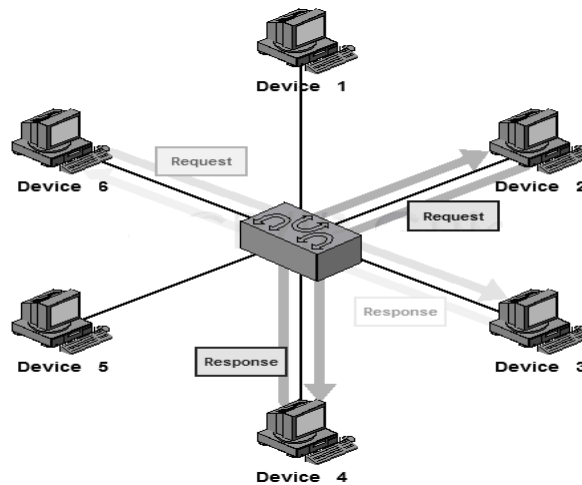


Figure (1.3): Peer-to-Peer Networking

1.3 Literature Survey

Various efforts in the field of authentication over network were introduced; the survey presented is concerned with some of the published researches related to user authentication of the most popular instant messaging software available now days

1. ICQ [Isa01]: stands for “I Seek You”. According to the ICQ documentation, ICQ operates in a server-based and peer-to-peer fashion. Client’s authentication is done using encryption, such that each packet sent from the client to the server is encrypted before being sent. Every time the client sends a packet to the server it must receive an acknowledgment from the server. ICQ is based on synchronous communication, for every ICQ message sent a reply has to be received to know if the message has been accepted, beside that many operations should be done on client-side, so ICQ might be slow.

2. MSN [Min03]: it's included in Microsoft Windows operating systems. MSN Messenger uses TCP protocol for all communication between hosts. Client can be connected to multiple servers concurrently. In this kind of architecture the server's could easily become congested. The MSN messenger is based on an asynchronous protocol, which ensures that new messages can be sent without waiting response of previous messages. The centralized server architecture also simplifies group messaging, but may cause problems if the servers become crowded. The MSN authenticates users using challenge-response password hashed using MD5 hashing algorithm.
3. The IRC [Irc03]: stands for "Internet Relay Chat", IRC was specified in 1993 by Oikarinen, it's the oldest famous "chatting" protocol still used. Using the IRC protocol clients always talks to other clients through a server using single TCP connection. The message can go through multiple servers before it reaches the other client. An IRC client connects to a server specified by the user. Every server knows every user in the network and when a client wants to talk to another user, all it needs to know is the *nickname* of the other client and the server delivers the possible message through other servers to the destination client.
4. AOL [Jer03]: stands for "American On-Line", it's based on TCP protocol for communication, the AOL architecture consists of two main components, which are the *Authorizer*, which validates username and password and the *BOS* (Basic Oscar Service). Before connections are made to any of the BOS or special-purpose servers it is necessary to be authorized by the *Authorization Server*.

5. Yahoo [Mes06]: All Yahoo communications use TCP/IP communication. To authenticate client, the client sends the username to the server. The server responds with a challenge string. The client responds to this challenge with two response strings. If authentication is successful, the connection goes into the messaging state, otherwise, an error response is sent back.

1.4 Aim of Thesis

The aim of the project is to design and implement an Instant Messaging System (IMS) provided with a proposed authentication method. The proposed authentication method adopts combination of two-authentication mechanisms: *Fixed Password* and *Digital Signature*. The designed IMS allows users to communicate with one another in virtual real time. Users communicate by typing messages, which are sent instantly to another user or group of users within the chat area. Users must be authenticated before being able to communicate with each other.

1.5 Thesis Layout

This section presents a guide for reading this thesis. The layout of the remaining individual chapters and their contents are reviewed briefly

- **Chapter Two:** This chapter concerns with the types of authentication services, types of message authentication functions and mechanisms that are used over network. ElGamal digital signature scheme algorithm is presented in this chapter with its related mathematical concepts.

- ***Chapter Three:*** This chapter presents the design requirements and considerations of the proposed 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:*** This chapter presents user interface and evaluation of the designed system.
- ***Chapter Five:*** This chapter introduces conclusions on this work, with recommendations for future work.

CHAPTER TWO

Message Authentication over Computer Networks

2.1 Introduction

Authentication is the technique by which a process verifies that its communication partner is who it is supposed to be and not imposter [Jam01]. The need for providing authenticity arises in many different situations and it especially crucial when a user operates from remote terminal. Two different cases can be distinguished [Seb89]: *user authentication* and *message authentication*. User authentication can be made either directly when a user's specific characteristics (e.g., finger prints, voice frequency spectrum, digital signature flow, etc.) are checked, or indirectly when a unique secret piece of information is proved to be in the user's possession. As a matter of fact, indirect user authentication is equivalent to the message authentication. Message authentication relies upon imposing a prearrangement structure for the message.

Authentication must be done solely on the basis of message and data exchange as part of an authentication protocol. The goal of an authentication protocol is to provide the communicating parties with some assurance that they know each other's true identities [Jam01].

Network entities do not typically have physical access to the parties they are authenticating. Malicious users or programs may attempt to obtain sensitive information, disrupt service, or forge data by impersonating valid entities. Distinguishing these malicious parties from valid entities is the role of authentication, and is essential to network security [Pat02].

2.2 Authentication Services Types

Two specific authentication services are defined in OSI (Open System Interconnection) standard [Wil03]:

- 1. Peer entity authentication:** provided for the cooperation of the identity of a peer entity in an association. It is provided for use at the establishment of, or at times during the data transfer phase, of a connection. It attempts to provide confidence that an entity is not attempting either a masquerade or an unauthorized replay of previous connection.
- 2. Data origin authentication:** provided for the cooperation of the source of a data unit. It does not provide protection against the duplication or modification of data units. This type of service supports application like electronic mail where there are no prior interactions between the communication entities.

2.3 Authentication Functions

Any message authentication mechanism can be viewed as having fundamentally two levels. At the lowest level, there must be some sort of function that produces an *authenticator*: a value to be used to authenticate a message. This lower-level function is then used as a primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message. The types of function that may be used to produce an authenticator are grouped into three classes, as follows [Wil03]:

- 1. Message encryption**
- 2. Message Authentication Code (MAC)**
- 3. Hash function**

In the following sections these classes will be described in details.

2.3.1 Message Encryption

Encryption is the process of encoding a message so that its meaning is not obvious. Decryption is the reverse process: transforming an encrypted message back into its normal form. The original form of a message is known as plaintext, and the encrypted form is called ciphertext. Denote a plaintext message P as a sequence of individual characters $P = (p_1, p_2, \dots, p_n)$; similarly ciphertext can be written as $C = (c_1, c_2, \dots, c_m)$, formally, the transformations between plaintext and ciphertext are denoted

$$C = E(P) \quad (2.1)$$

and

$$P = D(C) \quad (2.2)$$

Where C represents the ciphertext, E is the encryption algorithm, P is the plaintext, and D is the decryption algorithm. Some encryption algorithms use a key K , so that the ciphertext message depends on both the original message and the key value, denoted

$$C = E(K, P) \quad (2.3)$$

Essentially, E is a set of encryption algorithms and the K selects one specific algorithm. Sometimes the encryption and decryption keys are the same, so that

$$P = D(K, E(K, P)) \quad (2.4)$$

This style of encryption is called symmetric encryption because D and E are mirror-image processes, see figure (2.1), the sender A uses the key to encrypt the plaintext and sends the ciphertext to the receiver. The receiver B applies the same key to decrypt the message and recover the plaintext.

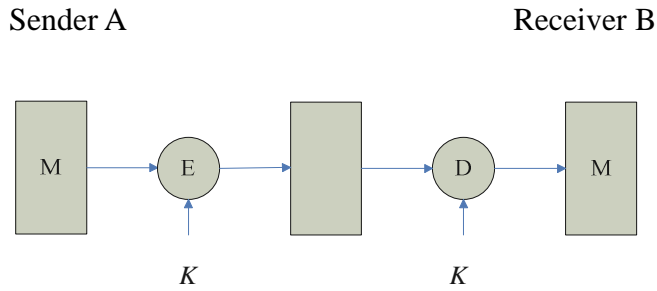


Figure (2.1): Symmetric Encryption

Other times encryption and decryption keys come in pairs. This form is called asymmetric. With a public key or asymmetric system, each user has two keys: a public key and a private key. The user may publish the public key freely. The keys operate in inverses. Let K_{PRIV} be a user's private key and let K_{PUB} be the corresponding public key [Cha97], then,

$$P = D(K_{PRIV}, E(K_{PUB}, P)) \tag{2.5}$$

Symmetric Encryption can be used for authentication, since a receiver can assume that only the sender knows the secret key. On the other hand public-key encryption provides confidentiality but not authentication. Because any opponent could also use B's public key to encrypt a message, claiming to be A. To provide both confidentiality and authentication, A can encrypt M first using its private key, and then B's public key, which provide confidentiality as shown figure (2.2), where KR_a is the sender private key, KU_a is the sender public key, KR_b is the receiver private key and KU_b is the receiver public key [Wil03].

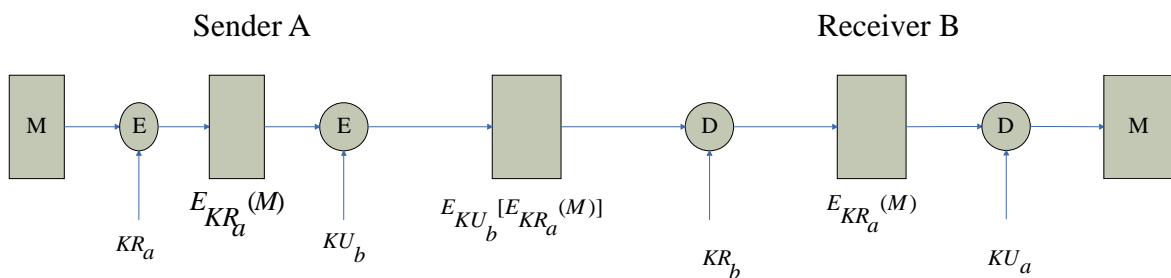


Figure (2.2) Public Key Encryption

2.3.2 Message Authentication Code (MAC)

A public function of the message and a secret key that produce a fixed-length value that serves as the authenticator. A MAC also known as a *cryptographic checksum* is generated by a function C of the form

$$MAC = C_K(M) \quad (2.6)$$

Where M is a variable-length message, K is a Shared secret key shared only by sender and receiver, and $C_K(M)$ is the fixed-length authenticator.

The MAC is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by re-computing the MAC , see figure (2.3). The MAC value protects both a message's integrity as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the message content [Wil03].

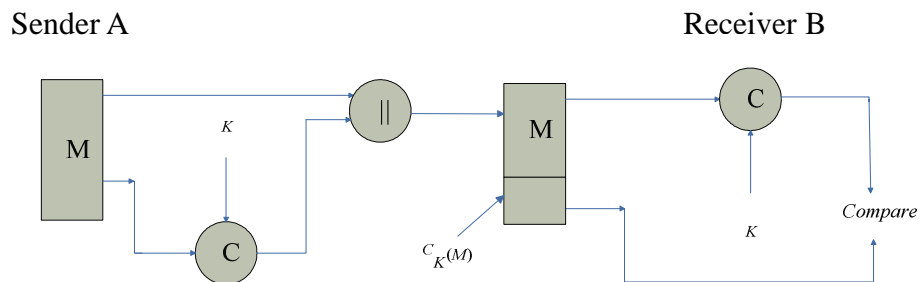


Figure (2.3) Basic Use of Message authentication Code (MAC)

2.3.3 Hash function

Hash Functions take a block of data as input, and produce a *hash* or *message digest* as output. Hash function produces a reduced form of a body of data such that most changes to the data will also change the reduced form. The usual intent is that the hash can act as a signature for the original data, without revealing its contents. Therefore, it's important that the hash function be irreversible - not only should it be nearly impossible to retrieve the original data, it must also be unfeasible to construct a data block that matches some given hash value.

Randomness, however, has no place in a hash function, which should completely deterministic. Given the exact same input twice, the hash function should always produce the same output. Even a single bit changed in the input, though, should produce a different hash value. The hash value should be small enough to be manageable in further manipulations, yet large enough to prevent an attacker from randomly finding a block of data that produces the same hash [Fre06]. Hash algorithms that are in common use include [Gar06]:

1. **Message Digest (MD) algorithms:** A series of byte-oriented algorithms that produce a 128-bit hash value from an arbitrary-length message.
 - **MD2:** Designed for systems with limited memory, such as smart cards.
 - **MD4:** Similar to MD2 but designed specifically for fast processing in software.
 - **MD5:** Developed after potential weaknesses were reported in MD4; this scheme is similar to MD4 but is slower because more manipulation is made to the original data. The MD5 algorithm is intended for digital signature applications.
2. **Secure Hash Algorithm (SHA):** Algorithm for NIST's (National Institute of Standards and Technology) 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, or 512 bits in length, respectively.
3. **RIPEMD:** A series of message digests; RIPEMD-160 was optimized for 32-bit processors to replace the then-current 128-bit

hash functions. Other versions include RIPEMD-256, RIPEMD-320, and RIPEMD-128.

4. **HVAL (HAsH of VArIable Length):** HVAL can create hash values that are 128, 160, 192, 224, or 256 bits in length.
5. **Whirlpool:** Whirlpool operates on messages less than 2^{256} bits in length, and produces a message digest of 512 bits.

The hash algorithm MD5 will be considered in designing the authentication method in chapter three, a clear description of it is stated in Appendix A.

2.4 Authentication Mechanisms

Authentication mechanisms can be classified in many ways; the following overview will generalize several authentication mechanisms to authenticate users over network [Ric01]:

2.4.1 Password

Password can be classified into two main types:

1. Fixed Password

One of the most commonly used authentication schemes employs passwords. Along with a user name, a password must be presented to the authentication system to gain access [Eri00]. 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 [Cha97].

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 [Fre04].

There is a reason why passwords are so popular: they're fast, they're 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 [Eli00].

2. One-time password [Ing06]

There are two types of one-time passwords, challenge-response password and password list.

1. 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.
2. 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.

2.4.2 Public Key Cryptography

Public Key Infrastructure (PKI) is a security architecture that was introduced to provide an increased level of security in exchanging information over an insecure Internet. Essentially, a PKI includes all the components required to establish and maintain trust relationship and binding of a public key to its owner within a system providing public key based applications [And06].

Authentication using public key cryptography was covered in section 2.3.1.

2.4.3 Zero-Knowledge Proof [Dic06]

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.

A zero-knowledge proof must satisfy three properties:

1. **Completeness:** if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover.
2. **Soundness:** if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability.
3. **Zero-knowledge:** if the statement is true, no cheating verifier learns anything other than this fact. This is formalized by showing that every cheating verifier has some *simulator* that, given only the

statement to be proven (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the cheating verifier.

2.4.4 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 [Eri00].

Digital signature powered by public key infrastructure (PKI) technology, are widely recognized as the best practice for ensuring digital accountability for electronic transactions. Digital signatures are the most effective, secure, and easy-to implement method of proving accountability while enabling electronic transactions [Ent06].

Two general classes of digital signature schemes can be briefly summarized as follows [Men96]:

1. Digital signature schemes with appendix require the original message as input to the verification algorithm. Examples of mechanisms providing digital signatures with appendix are the DSA, ElGamal and Schnorr signature schemes. ElGamal digital signature scheme is used in the proposed system in chapter three, and it's presented in this chapter.
2. Digital signature schemes with message recovery do not require the original message as input to the verification algorithm. In this case the, the original message is recovered from the signature itself. Examples

of mechanisms providing digital signatures with message recovery are RSA, Rabin and Nyberg-Rueppel public-key signature schemes.

The goal of an adversary is to *forge* signatures; that is, produce signatures, which will be accepted as those of some other entity. There are two basic attacks against public-key digital signature schemes [Joh01].

1. **Key-only attacks:** In these attacks, an adversary knows only the signer's public key.
2. **Message attacks:** Here an adversary is able to examine signatures corresponding either to known or chosen messages. Message attacks can be further subdivided into three classes:
 - **Known-message attack:** An adversary has signatures for a set of messages, which are known to the adversary but not chosen by him.
 - **Chosen-message attack:** An adversary obtains valid signatures from a chosen list of messages before attempting to break the signature scheme. This attack is non-adaptive in the sense that messages are chosen before any signatures are seen. Chosen-message attacks against signature schemes are analogous to chosen-cipher text attacks against public-key encryption schemes.
 - **Adaptive chosen-message attack:** An adversary is allowed to use the signer as an oracle; the adversary may request signatures of messages, which depend on the signer's public key, and he may request signatures of messages, which depend on previously, obtained signatures or messages.

2.5 ElGamal Digital Signature

In 1985, Tahir ElGamal [Tah85] published a public-key cryptosystem based upon the discrete logarithm problem. The security of ElGamal's signature scheme is based on solving the discrete logarithm problem.

2.5.1 Mathematical Concepts

To understand ElGamal algorithm, some mathematical concepts related to this algorithm must be well understood, these are listed below:

1. Discrete Logarithm Problem [C2c06]: The ordinary logarithm problem: given a base b and a number x , find y such that

$$b^y = x \quad (2.7)$$

E.g., 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

$$b^y = x \pmod{n} \quad (2.8)$$

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 .

2. *Primitive Root (Generator)* [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 . We 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 powers of 3 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. *Multiplicative Inverse* [Enw06]: In modular arithmetic, the multiplicative inverse of x is also defined: it is the number a such that $(a * x) \bmod n = 1$. However, this multiplicative inverse exists only if a and n are relatively prime. For example, the inverse of 3 modulo 11 is 4 because it is the solution to $(3 * x) \bmod 11 = 1$.

2.5.2 ElGamal Algorithm [Men96]

The ElGamal signature scheme generates a signature with appendix on binary messages of arbitrary length, and requires a hash function $h: \{0,1\}^* \rightarrow Z_p$, where p is a large prime number. The algorithm can mainly be divided into three algorithms: key generation, signature generation and signature verification as shown in algorithms 2.1, 2.2, 2.3 respectively.

Algorithm (2.1): ElGamal Key Generation

1. Generate a large prime p and a generator α of the multiplicative group Z_p^* .
2. Select a random integer a , $1 \leq a \leq p-1$
3. Compute $y = \alpha^a \bmod p$ (2.9)
4. A's public key is (p, α, y) ; A's private key is a .

Algorithm (2.2) ElGamal Signature Generation

1. Select a random secret integer k , $1 \leq k \leq p-2$
2. Compute $r = \alpha^k \bmod p$ (2.10)
3. Compute $k^{-1} \bmod (p-1)$ (2.11)
4. Compute $s = k^{-1} \{h(m) - a * r\} \bmod (p-1)$ (2.12)
5. A's signature for m is the pair (r, s)

Algorithm (2.3): ElGamal Signature Verification

1. Obtain A's authentic public key (p, α, y)
2. Verify that $1 \leq r \leq p-1$; if not, then reject the signature.
3. Compute $v_1 = Y^r r^s \bmod p$ (2.13)
4. Compute $h(m)$ and $v_2 = \alpha^{h(m)} \bmod p$ (2.14)
5. Accept the signature if and only if $v_1 = v_2$ (2.15)

Proof that signature verification works: If the signature was generated by A, then $s \equiv k^{-1}\{h(m) - a * r\} \pmod{p-1}$. Multiplying both sides by k gives $k * s \equiv h(m) - a * r \pmod{p-1}$ and rearranging yields $h(m) \equiv a * r + k * s \pmod{p-1}$. This implies $\alpha^{h(m)} \equiv \alpha^{a*r+k*s} \equiv (\alpha^a)^r r^s \pmod{p}$. Thus $v_1 = v_2$, as required.

2.5.3 Security of ElGamal Signature Scheme [Men96]

1. An adversary might attempt to forge A's signature on m by selecting a random integer k and computing $r = \alpha^k \bmod p$. The adversary must then determine $s = k^{-1}\{h(m) - a * r\} \pmod{p-1}$. If the discrete logarithm problem is computationally infeasible, the adversary can do no better than to choose an s at random; the success probability is only $1/p$, which is negligible for large p .

2. A different k must be selected for each message signed; otherwise, the private key can be determined with high probability as follows:

$$\text{Suppose } s_1 = k^{-1}\{h(m_1) - a * r\} \pmod{p-1}$$

$$\text{and } s_2 = k^{-1}\{h(m_2) - a * r\} \pmod{p-1}$$

$$\text{Then } (s_1 - s_2)k \equiv (h(m_1) - h(m_2)) \pmod{p-1}$$

$$\text{If } s_1 - s_2 \neq 0 \pmod{p-1}, \text{ then } k = (s_1 - s_2)^{-1}(h(m_1) - h(m_2)) \pmod{p-1}.$$

Once k is known, a is easily found.

3. If no hash function h is used, the signing equation is $s = k^{-1}\{m - ar\} \bmod(p-1)$. It is then easy for an adversary to mount an existential forgery attack as follows. Select any pair of integers (u, v) with $\gcd(v, p-1) = 1$. Compute $r = \alpha^u y^v \bmod p = \alpha^{u+av} \bmod p$ and $s = -rv^{-1} \bmod(p-1)$. The pair (r, s) is a valid signature for the message $m = su \bmod(p-1)$, since $\alpha^m \alpha^{-ar} s^{-1} = \alpha^u y^v = r$.
4. Step 2 in algorithm 2.3 requires the verifier to check that $0 < r < p$. If this check is not done, then an adversary can sign messages of its choice provided it has one valid signature created by entity A, as follows. Suppose that (r, s) is a signature for message m produced by A. the adversary selects a message m' of its choice and computes $h(m')$ and $u = h(m') \cdot [h(m)]^{-1} \bmod(p-1)$ (assuming $[h(m)]^{-1} \bmod(p-1)$ exists). It then computes $s' = su \bmod(p-1)$ and r' such that $r' \equiv ru \pmod{p-1}$ and $r' = r \pmod{p}$. The pair (r, s) is a signature for message m , which would be accepted by the verification algorithm if step 2 were ignored.

CHAPTER THREE

Design and Implementation of an Instant Messaging System

3.1 Introduction

The aim of the project is to design and implement an Instant Messaging System (IMS) provided with authentication method. This work is concerned with online/offline interactive communication method using text to send and receive instant messages among users.

The system has the ability to open or start text conversation between two or more users using two types of chat area: *public chat area* and *private chat area*. In the first type, each member of the chat area can see what every other member types, and in turn any responses sent will be sent to all other members of the conversation. In the second type, two users can exchange instant messages using private chat area, only those two users participating in this area can see what each other types. For both chatting types, the text appears as it is typed on PCs participating in the chat. Also, users can send offline messages using private chat area; each user can see his/her offline messages (if there is any) after login.

The presented authentication method is used as a control to allow only members of the system to communicate with each other. This method utilizes a combination of two authentication mechanisms: *Fixed Password* and *Digital Signature*. ElGamal algorithm is used to generate digital signature.

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

IMS was implemented using Microsoft Visual Basic 6.0 programming language and tested on LAN at which all computers are connected to network HUB, and all computers are running under Windows XP operating system.

3.2 Design Requirements and Considerations

The Proposed Instant Messaging System requires a high-speed central server and a number of clients connected through a network. At least, three computers must exist, one of them is the server run by an administrator, and the other two computers are clients run by users. A client logs onto IMS server using a UserID and Password, each client represent a unique UserID, so that each user may represent more than one client in IMS. The server authenticates the request and either allows or denies access to services. The implementation of the proposed system requires data storing, so Microsoft Office Access software was used to store the required data in files and these files can be accessed easily using Visual Basic programming language.

The design considerations of the system are:

1. Easy of use by users.
2. Fast execution: where user's registration and login time should be acceptable, and regarding to the chatting process, it should be a virtual real time chatting between users.
3. Security: where only authenticated users should be allowed to login.

In the next sections a clear description of the proposed system will be presented.

3.3 System Structure

From the functional point of view, the system consists of five modules (*Initialization* module, *Registration* module, *Login* module, *chatting* module and *change password* module) as shown in figure (3.1).

The main function of the Initialization module is to generate list of public keys using ElGamal algorithm that will be used for authentication process in the next modules. The aim of the second module is to accomplish client registration process; registration process implies creation of a secret key and distributes public keys (that generated in first module) for each client. These keys will be calculated from (UserID, Password) pair and from challenge information. In the third module a registered client login to the system after providing a valid digital signature, which is verified by server, this signature is used as a challenge proves what client claim to be. When client login then it is considered to be online. The term *Online* is used to indicate that the client is allowed to login and ready to chat with any other online clients, while *Offline* term means that the client is not yet logged in. After logging in, the client is provided with set of functions by the fourth module, which are:

1. Chatting with other users using public or private chat area.
2. Adding to a list (named Buddy List) some of IMS's UserIDs selected by user.
3. Sending offline messages to offline users.
4. Changing password.

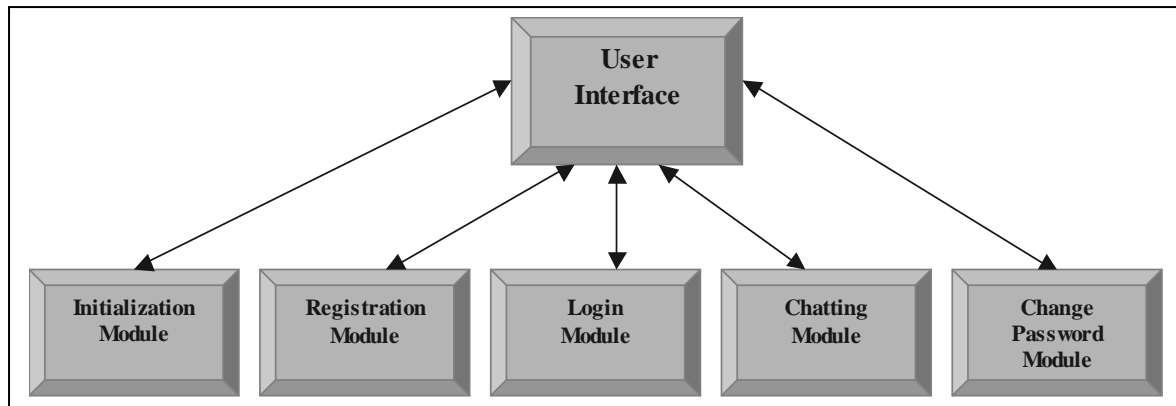


Figure (3.1): System Modules

3.4 Initialization Module

This module is executed at server side and it consists of two parts:

1. **Keys Generation:** in this part, the server creates a table of client's public keys.
2. **Starting Serving:** in this part, the server prepares to start client's serving.

3.4.1 Keys Generation

According to ElGamal algorithm (section 2.5.2) each client must generate two numbers p and α randomly. p should satisfy two conditions:

1. p should be prime number.
2. The prime p should be large enough to make discrete logarithm problem infeasible.

And α must be generator of p . Generating such large prime number p randomly by client could be time consuming because in some places along the number line, primes are closely packed, and in some other places there are large gaps. So to accommodate the designed system consideration concerning to speed of executing on client side, prime numbers generation and finding their generator are made at server side.

The server generates a table, named Primes/Generators table (table 3.1), of N numbers (prime numbers (p) and generator (α)). A third field in this table is the *flag* field, which is used as an indicator that pair (p, α) was assigned to any client in registration process or not. Initially this field is False for all N entries, this table is saved in a file named Keys. Generation process is illustrated in algorithm 3.1.

Table (3.1): Primes/Generators

| <i>P</i> | <i>α</i> | <i>Flag</i> |
|-----------------------|----------------------------|-------------|
| 590295810358705651741 | 6 | False |
| 590295810358705651817 | 3 | False |
| 590295810358705651829 | 2 | False |
| <i>etc</i> | | |

It's important to note that in computer the largest integer number that can be treated as a single unit is $2^{64} = 18, 446, 744, 073, 709, 551, 616$ (which represented with 64 bits). According to Buchmann [Joh01], p must be at least 768-bit number, so data type of p was chosen to be string in order hold large values. As a result the standard library of binary operators ($>$, $<$, $=$, $+$, $-$, $*$, $/$) can not be used with string data type, so each of these operations is replaced by a program code that do the same operation but for very large sizes of integer numbers.

According to Menezes [Men96], for each given length of p there must be security parameter t , such that if the primality test on a number p is run t times, then the probability that p is declared “prime” all t times (i.e., the probability of error) is at most $(1/2)^t$. Table 3.2, named SecurityParameter table, shows for samples of K the smallest security parameter t with error probability $(1/2)^{80}$.

Algorithm 3.1 for constructing Primes/Generators table requires passing four inputs: first input is the numbers of primes to be generated (N), minimum length (in bits) of first generated prime number (K), SecurityParameter table, and a list of small prime numbers less than 500 (SmallPrime), this list is generated in advance and then passed to the algorithm, this list is generated in advance to be used with each prime test in algorithm 3.2.

Table (3.2): SecurityParameter

| K (bits) | t |
|------------|-----|
| 100 | 27 |
| 150 | 18 |
| 200 | 15 |
| etc | |

According to the passed length (K), the security parameter t will be chosen to hold an initial value that could be changed later in the algorithm. If K length doesn't match any entry in SecurityParameter table, the initial value of t will be the closest biggest value of the passed K . Then an odd value of length equal K is generated to be test if it is prime number or not using Find Prime algorithm (algorithm 3.2).

There are two cases of the generated odd value:

- The number p is prime; then its generator α will be searched, and after finding its generator α , the pair (p, α) will be saved in Primes/Generators table (table 3.1). Then p will also be incremented by two. Incrimination will continue until N primes will be found.
- The number p is not prime; then its value will be incremented by two, in order to be tested as the next odd probable prime number.

Algorithm (3.1): Generate Primes/Generators table**Input:** Number of primes to be generated Integer numbers (N)Minimum length of first generated prime (K)

SecurityParameter table

SmallPrimes list

Output: File contain Primes/Generators table**Procedure**1. For $I \leftarrow 1$ to 2For $J \leftarrow 1$ to 40If SecurityParameter (J, I) $\geq K$ thenSet $t \leftarrow$ SecurityParameter (J, I)2. Set $P_{BIN} \leftarrow$ "1" & string of ($K-2$) zeros & "1"3. Convert P_{BIN} to decimal and put result in p 4. Call Subtract ($p, "1"$) algorithm and put result in p_1 (algorithm 3.5)5. Call Subtract ($p, "2"$) algorithm and put result in p_2 (algorithm 3.5)

6. Open Keys file

7. For $I \leftarrow 1$ to N 7.1 Call Find Prime ($p, p_1, p_2, SmallPrimes, Counter, t$) algorithm "algorithm 3.2" and put resulted prime number in p , resulted prime factors in Factors table and put size of factors table in F .7.2 Call Find Alpha ($p_1, Factors, F$) algorithm "algorithm 3.3" and put result in α 7.3 Set PrimesGenerators (I). $p \leftarrow p$, PrimesGenerators (I). $\alpha \leftarrow \alpha$ 7.4 Call Addition ($p, "2"$) algorithm "algorithm 3.6" and put result in p

8. Close Keys file

As explained above, to test p number is prime or not, Find Prime algorithm is called. There are few algorithms that are used to check if any number is prime or not. According to Menesez [Men96], the most efficient known algorithm that is used as a primality test is the Miller Rabin primality test (described in appendix A), this algorithm check an odd integer number p if it is prime or not. This algorithm is called in Find

Prime algorithm (algorithm 3.2), but before it is called, the number must pass three tests which are made to speed up deciding if the number is prime number or not, which are:

1. If the most right digit of p equals 5 then it is dividable by 5.
2. If summation of p 's digits mod 3 equal 0 then it is dividable by 3.
3. If p is dividable by any number in the SmallPrimes list, then it's not prime.

As the number is moving from test to test then its probability to be prime number will increase. After passing the last test, the Miller Rabin algorithm will make the final decision that the number is prime or not.

In algorithm 3.2, $p-1$ needs to be factorized; the prime factors are saved in *Factors* table of size F . (Example: the prime factorization of 12 is $2*2*3$, these factors are saved as follows *Factors* (1). *Base*=2, *Factors* (1). *Power*=2, *Factors* (2). *Base*=3, *Factors* (2). *Power*=1).

Algorithm (3.2): Find Prime

Input: String represents an odd decimal number (p)

String represents an even decimal number (p_1)

String represents an odd decimal number (p_2)

Security parameter (t)

SmallPrimes list

Size of the primes array (Counter)

Output: Prime number (p)

Factors table

Size of Factors table (F)

Procedure

1. Set Flag \leftarrow False

2. Do until Flag=true

 If Right ($p, 1$) =5 then

 Call addition ($p, "2"$) algorithm "algorithm 3.6" and put result in p .

 Else

```

Set Sum  $\leftarrow$  0
For I  $\leftarrow$  1 to Length (p)
    Sum=Sum+ p (I)
If Sum mod 3= 0 then
    Call addition (p,"2") algorithm "algorithm 3.6" And put result in p.
Else
    Set I  $\leftarrow$  1
    Do while I  $\leq$  Counter
        Call DivMod (p, primes (I)) algorithm "algorithm 3.9" and put result in
        Result and Reminder
        If Reminder = 0 then
            If Small Primes (I) = p then exit Do loop
        Increment I by 1
    Factorize p1 and put result in Factors table
    Set F $\leftarrow$ size of Factors table
    Do until Factors (I). Base  $\neq$  2
        Increment s by 1
        Increment I by 1
    For I $\leftarrow$  1 to F
        Call Multiplication(r, Factors (I). base) algorithm "algorithm 3.7" and put
        result in r
        Call Miller Rabin (p, r, s, t) algorithm and put result in Flag
3. Return p, Factors table and F

```

After finding a prime number p , its generator α must be searched; algorithm 3.3 illustrates finding the generator α . The search starts from value 2 and proceed until finding the first generator, which will be less than p . In this algorithm modular exponential is computed using repeated square and multiply algorithm (Appendix A). For rest of this chapter this algorithm will be called to compute different values and its named Fastexp for short, which stand for (Fast Exponential) algorithm.

Algorithm (3.3): Find Alpha**Input:** String represent an even integer number (p_1)

Factors table

Number of prime factors (F)**Output:** Generator of p_1 (α)**Procedure**1. Set $\alpha \leftarrow 2$, $I \leftarrow 1$, $Flag \leftarrow False$ 2. Do until $Flag=True$ 2.1 Do while $I \leq F$ a. Call DivMod (p_1 , Factors (I), Base) algorithm "algorithm 3.9" and put result in e b. Call FastExp (α , e , p_1) algorithm and put result in b c. If $b = "1"$ then exit loop2.2 If $I = F$ then $Flag=True$ Else Call Addition (α , " $"$ ") algorithm "algorithm 3.6" and put result in α 3. Return α

The algorithms that perform the binary operations ($<$, $=$, $>$, $+$, $-$, $*$, $/$) are listed below. Note: each of the mentioned operations requires two operands, s_1 and s_2 ; data type of each one is a string that represents an integer decimal number. Also in these algorithms some notations are used to illustrate dealing with numbers as strings, these notations are explained below with an example, let $s_1 = 549755814121$:

length (s_1): number of digits in s_1 , length (s_1)=12, s_1 (j): digit of order j in s_1 , s_1 (4)=7,left (s_1 , j), left digit(s) in s_1 of length j , left (s_1 , 5)= 54975,right (s_1 , j): right digit(s) in s_1 of length j , right (s_1 , 5)= 14121,mid (s_1 , i , j): digit(s) from s_1 of length j starting from digit of order i , mid (s_1 , 2, 6)= 497558 s_1 & s_2 : concatenation of s_1 with s_2 , suppose $s_2 = 2053$, so s_1 & $s_2 = 5497558141212053$

Algorithm (3.4): Compare**Input:** String represents integer number (s_1)String represents integer number (s_2)**Output:** Greater, Smaller, or Equal**Procedure**1. If Left ($s_1, 1$) = "-" and Left ($s_2, 1$) = "-" Then $s_1 = \text{Right}(s_1, \text{Length}(s_1) - 1)$ $s_2 = \text{Right}(s_2, \text{Length}(s_2) - 1)$ Else If Left ($s_1, 1$) = "-" Then

Return = "Smaller"

Exit

Else

If Left ($s_2, 1$) = "-" Then

Return = "Greater"

Exit

2. If Length (s_2) > Length (s_1) The Return "Smaller"3. If Length (s_2) < Length (s_1) Then Return "Greater"4. For $i \leftarrow 1$ to Length (s_1)4.1 If $s_1(i) > s_2(i)$ Then Return "Greater"4.2 If $s_1(i) < s_2(i)$ Then Return "Smaller"

5. Return "Equal"

Algorithm (3.5): Subtraction**Input:** String represents integer number (s_1)String represents integer number (s_2)**Output:** Result of $s_1 - s_2$ **Procedure**1. Call Compare (s_1, s_2) algorithm "algorithm 3.4" and put result in Answer2. If Answer = "Smaller" then Set $s \leftarrow s_1, s_1 \leftarrow s_2, s_2 \leftarrow s, \text{Sign} \leftarrow "-"$ Else set Sign $\leftarrow ""$ 3. For $I \leftarrow 1$ to length (s_2)3.1 Set $\text{Digit}_1 \leftarrow s_1(\text{Length}(s_1) - I + 1)$, Set $\text{Digit}_2 \leftarrow s_2(\text{Length}(s_2) - I + 1)$ 3.2 If $\text{Digit}_2 > \text{Digit}_1$ Thena. Set $s \leftarrow \text{Value}(\text{Digit}_1) + 10 - \text{Value}(\text{Digit}_2) + s$

b. For $J \leftarrow I + 1$ to $\text{Length}(s_1)$
 Set $\text{Digit}_1 \leftarrow s_1(\text{Length}(s_1) - J + 1)$,
 If $\text{Value}(\text{Digit}_1) > 0$ Then
 Set $\text{Digit}_1 \leftarrow \text{Value}(\text{Digit}_1) - 1$
 Set $s_1 \leftarrow \text{left}(s_1, \text{Length}(s_1) - j) + \text{Digit}_1 + \text{Mid}(s_1, \text{Length}(s_1) - j + 2, j)$
 Exit for loop
 Else
 $s_1 = \text{left}(s_1, \text{Length}(s_1) - j) + "9" + \text{Mid}(s_1, \text{Len}(s_1) - j + 2, j)$
 Else
 Set $s \leftarrow \text{Value}(\text{Digit}_1) - \text{Value}(\text{Digit}_2) + s_1$

5. Set $s \leftarrow \text{left}(s_1(\text{Length}(s_1) - \text{Length}(s_1))) + s$

6. Return Sign & s

Algorithm (3.6): Addition

Input: String represents integer number (s_1)

String represents integer number (s_2)

Output: Result of $s_1 + s_2$

Procedure

1. Call Compare (s_1, s_2) algorithm "algorithm 3.4" and put result in Answer

2. If Answer = "Smaller" then

Set $s \leftarrow s_1, s_1 \leftarrow s_2, s_2 \leftarrow s$

3 For $I \leftarrow 1$ to $\text{length}(s_2)$

3.1. Set $\text{Digit}_1 = s_1(\text{Length}(s_1) - I + 1)$, Set $\text{Digit}_2 = s_2(\text{Length}(s_2) - I + 1)$

3.2. Set $\text{Sum} \leftarrow \text{Value}(\text{Digit}_1) + \text{Value}(\text{Digit}_2)$

3.3. Set $r \leftarrow \text{Right}(\text{Sum}, I)$

3.4. Set $l \leftarrow \text{Left}(\text{Sum}, I)$

3.5. Set $s \leftarrow r \ \& \ s$

3.6. If $\text{Value}(\text{Sum}) > 9$ then

Set $s_1 \leftarrow \text{left}(s_1, \text{Length}(s_1) - I) + r + \text{Mid}(s_1, \text{Length}(s_1) - I + 1, I - 1)$

For $J = I + 1$ to $\text{Length}(s_1)$

Set $\text{Digit}_1 \leftarrow \text{Value}(s_1(\text{Length}(s_1) - J + 1)) + \text{Value}(l)$

If Value ($Digit_1$) > 9 then

Set $l \leftarrow \text{Left}(Digit_1, 1)$

$s_1 \leftarrow \text{Mid}(s_1, 1, \text{Length}(s_1) - J) + "0" + \text{Mid}(s_1, \text{Length}(s_1) - J + 2, J - 1)$

Else

$s_1 = \text{Mid}(s_1, 1, \text{Length}(s_1) - j) + Digit_1 + \text{Mid}(s_1, \text{Len}(s_1) - J + 2, J - 1)$

Set $l \leftarrow ""$

Exit for Loop

Else

$s_1 = \text{Mid}(s_1, 1, \text{Length}(s_1) - I) + r + \text{Mid}(s_1, \text{Length}(s_1) - I + 2)$

Set $l \leftarrow ""$

3.7 Set $s_1 \leftarrow l + s_1 (\text{Length}(s_1))$

3.8 Set $l \leftarrow ""$

4. Set $s \leftarrow l \& \text{Mid}(s_1, 1, \text{Length}(s_1) - \text{Length}(s_2)) \& s$

5. Return s

Algorithm (3.7): Multiplication

Input: String represents an integer number (s_1)

String represents an integer number (s_2)

Procedure

Output: Multiplication result of $s_1 * s_2$

1. Call Compare (s_1, s_2) algorithm "algorithm 3.4" and put result in Answer

2. If Answer="Smaller" then

Set $s \leftarrow s_1, s_1 \leftarrow s_2, s_2 \leftarrow s$

3. Set $Digit_1 \leftarrow s_2 (\text{Length}(s_2))$

4. Call MultiplyOne ($s_1, Digit_1$) algorithm and put result in s (algorithm 3.8)

5. For $I \leftarrow 2$ to $\text{Length}(s_2)$

5.1 Set $Digit_1 \leftarrow s_2 (\text{Length}(s_2) - I + 1)$

5.2 Call MultiplyOne ($s_1, Digit_1$) algorithm "algorithm 3.8" and put result in Temp

5.3 Set $Temp \leftarrow Temp \& "0"$

5.4 Call Add (Temp, s) "algorithm 3.6" and put result in s

6. Return s

Algorithm (3.8): MultiplyOne**Input:** String represents an integer number (s_1)Integer number of length one ($Digit_1$)**Output:** Multiplication of $s_1 * Digit_1$ **Procedure**

1. For $I \leftarrow 1$ to Length (s_1)
 - 1.1 Set $Digit_2 \leftarrow s_1$ (Length (s_1) - $I + 1$)
 - 1.2 Set $Temp \leftarrow Value (Value (Digit_1) * Value (Digit_2)) + Value (I)$
 - 1.3 If Length ($Temp$) > 1 Then Set $l \leftarrow left (Temp, 1)$
Else Set $l \leftarrow ""$
 - 1.4 Set $r \leftarrow Right (Temp, 1)$
 - 1.5 Set $s \leftarrow r \& s$
2. Set $s \leftarrow l \& s$
3. Return s

Algorithm (3.9): DivMod**Input:** String represents an integer number (s_1)String represents an integer number (s_2)**Output:** Division and reminder result of s_1 / s_2 **Procedure**

1. Call Compare (s_1, s_2) "algorithm 3.4" and put result in Answer
2. If Answer = "Smaller" Then
Set reminder $\leftarrow s_1$, result $\leftarrow "0"$
Exit
Else
If Answer = "Equal" Then
Set reminder $\leftarrow "0"$, result $\leftarrow "1"$
Exit
3. Set temp $\leftarrow ""$
4. Do until $s_1 = temp$
 - 4.1 set $s \leftarrow Mid (s_1, 1, Length (s_2))$
 - 4.2 Call Compare(s, s_2) algorithm "algorithm 3.4" and put result in Answer

4.3 If Answer = "Smaller" Then set $s \leftarrow s + \text{Mid}(s_1, \text{Length}(s_2) + 1, 1)$

4.4 For $I = 0$ To 9

Call MultiplyOne (s_2, I) algorithm "algorithm 3.8" and put result in temp

Call Subtract(s, temp) algorithm "algorithm 3.5" and put result in temp

If Left ($\text{temp}, 1$) = "-" Then

Set $I \leftarrow I + 1$

Set temp $\leftarrow \text{Right}(\text{temp}, \text{Length}(\text{temp}) - 1)$

Exit For

Else

Call compare (s_2, temp) algorithm "algorithm 3.4" and put result in Answer

If answer="Greater" or Answer="Equal" then

Exit For

4.5 set $s_1 \leftarrow \text{temp} \ \& \ \text{Right}(s_1, \text{Length}(s_1) - \text{Length}(s))$

4.6 set $r \leftarrow r \ \& \ I$

5. Set reminder $\leftarrow s_1$, result $\leftarrow r$

6. Return result and reminder

3.4.2 Starting Serving

Once the keys (p, α) were successfully generated, the server prepares to start client's serving. Starting serving requires opening two windows sockets:

1. WsTcpServerReg: this socket will be used to send and receive data between server and all the clients. These data are mainly related to the registration process. This socket will be listening (i.e. waiting for connections from clients) on port pt_1 .
2. WsTcpServerLog: this socket will have the same above socket's function except that the sent and received data is related to the login process. This socket will be listening on port pt_2 .

Once server starts listening, then it's ready to exchange data with any client for both registration and logging in operations. The exchanged

data are transmitted between clients and server in form of packets. The IMS packet consists of header and data. Figure (3.2) shows typical representation of the proposed IMS packet.

| | | | | |
|---------------|------------------|----------------|---------------|---------------------|
| Header | Packet ID | Version | Length | Service Type |
| Data | Data | | | |

Figure (3.2): IMS Packet

IMS header part consists of:

- 1. Packet ID:** The first three bytes of all packets are always “IMS” string, the messenger name.
- 2. Version:** Two bytes are for the IMS version number; the current version of IMS is 1.0.
- 3. Length:** A two-byte value stating how many bytes are in the data section of the packet.
- 4. Service Type:** Five bytes used to tell the client and server what kind of service is requested/being responded to.

IMS data part consists of sent/received data between client and server. The following packet is an example of an IMS packet

(IMS, 1.0, 5, chats, “hello”)

3.5 Registration Module

In this module, the client registers as new member in IMS; successful registration sequence is shown in figure (3.3). Client’s registration at IMS server requires first creating a windows socket (WsTcpClientReg) to be used for connecting with server on port pt_1 , if connection didn’t succeed then reconnection with server will automatically repeated until either connection accomplished or user choose to stop reconnection.

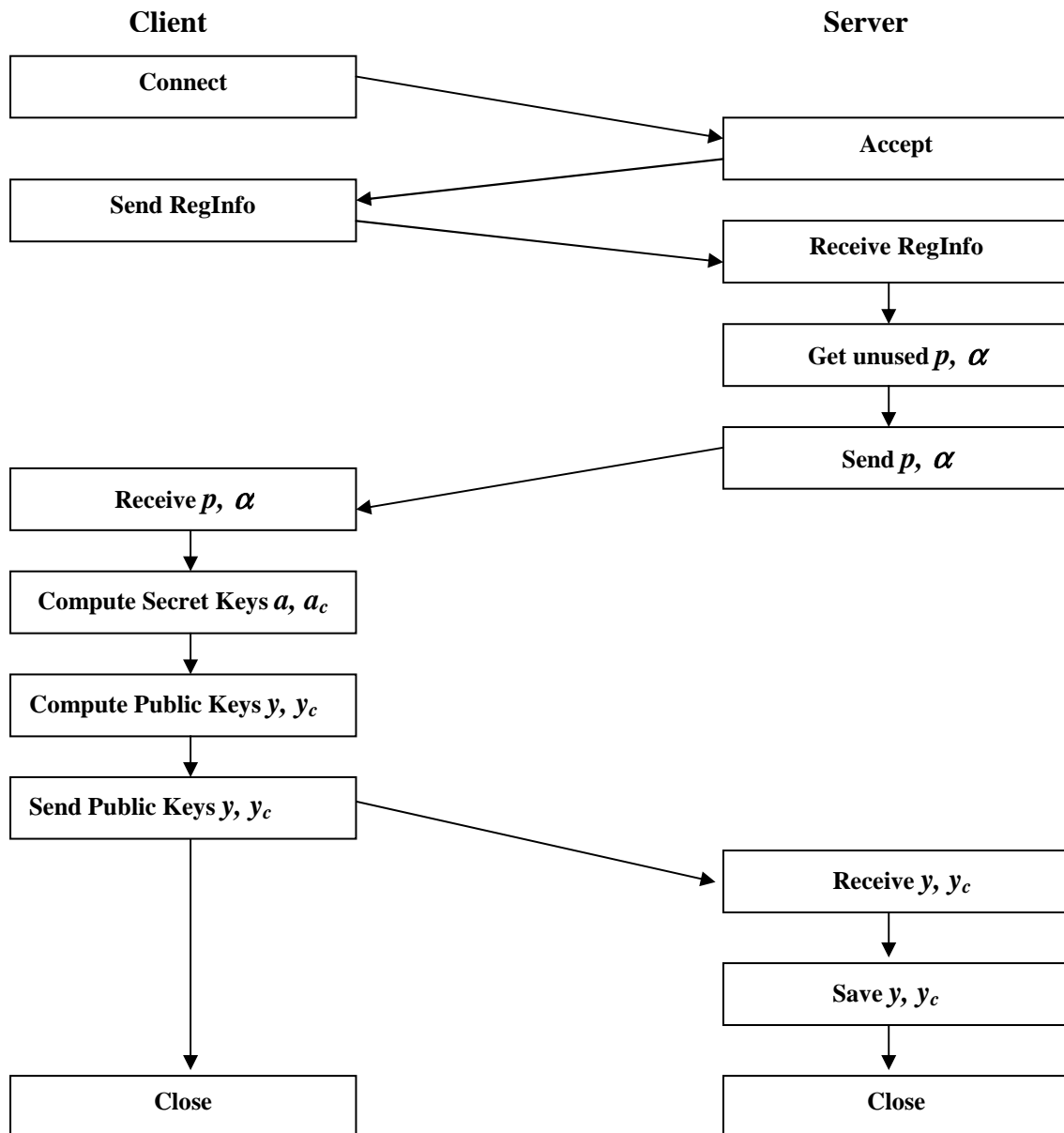


Figure (3.3): Successful Registration Sequence

After successful connection, client must present:

- The registering information (RegInfo), which include (UserID, First Name, Last Name, Birthday date).
- Password
- Challenge information: a string represents an answer to a question; in IMS client program there is specific number of questions, so that the user select one of the questions and answer it. This information

will be used when user needs to change the password because he/she forgot it.

Note that first and last name is for interface presentation not for authentication.

The password must meet two requirements:

1. Its length must be at least 8 characters long and less than 16.
2. There must be at least one alphabetic character (A through Z) or (a through z), and one numeric character (0 through 9) in the password.

The client sends to server a packet contains its RegInfo, the server accepts the Registration Info only if:

1. UserID, First Name, Last Name and challenge information do not contain any special characters *, &, ^, \$, #, @, (,) ~, >, <, ?, |, \, ', ", :, {, }, %.
2. Birthday must be valid birthday date of the form (YYYY, MM, DD).
3. The length of the UserID is greater than 3 characters and less than 16, this range is chosen to make number of possible UserID very large.
4. UserID must be unique, i.e. never used by any other client before, if it is then client should choose another UserID.

If any of the above requirements was not satisfied then the server send error message to user to inform him/her to re-choose the UserID.

After accepting the registration information by server, the server assigns unused values of p , α . The entry with false value in the *flag* field marks unused values for p , α . The index of p , α , in Primes/Generators table is saved in Pending Registering table (table 3.3) and set *flag* field of this entry to true.

Table (3.3): Pending Registering

| UserID | First Name | Last Name | Birthday | ChallengeInfoQuestion | PαIndex |
|--------------|------------|-----------|------------|--------------------------------------|---------|
| Paul_2002 | Paul | Andrew | 2000/10/10 | What's the name of your first school | 1 |
| Linda_Wilson | Linda | Wilson | 1988/2/3 | What's your childhood hero | 3 |
| Martin_Steve | Martin | Thomas | 1977/9/10 | What make was your first car or bike | 2 |
| etc | | | | | |

Now server sends p , α to client who in turn will use these two numbers for computing another numbers, which are: the secret keys a , a_c and public keys y , y_c . The secret keys a and a_c are computed using MD5 hash algorithm (Appendix A), such that $a = \text{MD5}(\text{UserID} \ \& \ \text{Password})$, $a_c = \text{MD5}(\text{UserID} \ \& \ \text{Challenge Information})$. Now, client computes public keys y and y_c as follows [Men96]:

$$y = \alpha^a \text{ mod } p \quad (3.1)$$

$$y_c = \alpha^{a_c} \text{ mod } p \quad (3.2)$$

y and y_c are computed using FastExp algorithm (Appendix A), Client sends its public keys y , y_c to server. After receiving these keys from client, server saves these two numbers in Registered Clients table (table 3.4), server also save all information collected to this client which are: First Name, Last Name, Birthday, ChallengeInfoQuestion and PαIndex from Pending Registering client table.

Table (3.4): Registered Clients

| UserID | First Name | Last Name | Birthday | y | y_c | ChallengeInfoQuestion | PαIndex |
|--------------|------------|-----------|-----------|---------------------------|---------------------------|--------------------------------------|---------|
| Linda_Wilson | Linda | Wilson | 1981/9/10 | 53180632110 6783356666 | 29478167448 3622736176 | What's the name of your first school | 3 |

The UserID of this client is saved in another table called Offline table (table 3.5). This table consists of four fields, first one is the UserID

of a client, which its registration process is in final stage, and rest fields are initially empty. Offline Messages field is used to save offline messages sent by other clients to this client in time this client was offline. The Buddy List field consists some UserIDs that was added by this client. The last field, Add Request, contains requests from other clients to add this client to their Buddy List field. Finally server informs this client that registration process was completed successfully and disconnects connection with client.

The mentioned numbers p, α, y will be used in the client's verification process at login. The above steps are presented in algorithm 3.10. The input to this algorithm is server's IP (SerIP), and server listening port pt_1 , registration information (RegInfo) and (UserID, Password) pair entered by user.

Table (3.5): Offline

| UserID | Offline Message | Buddy List | Add Request |
|-----------|--|--------------|--------------------------------------|
| Paul_2002 | "Hello, how are you?", 2006/11/14, 12:16 pm | Martin_Steve | Linda_Wilson, 2006/11/14, 12:16pm |

Algorithm (3.10): Client Registration

Input: Server IP (SerIP)

Server local port (pt_1)

Registration information (RegInfo)

Client's UserID (UserID)

Client's password (Password)

Output: Success or Fail

Procedure

1. Client connects to server using *WsTcpClientReg* on server's listening port pt_1 .

2. If Server is ready then it will:

2.1 Increment number of client's attempting to register by one,

Set $Index \leftarrow index + 1$

2.2 Accept connection request

2.3 Send to client the packet (IMS, 10, 5, Regis, Ready)

Else

Client get error message telling that server is not available at current time
Repeat step 1.

3. Client sends the server the packet (IMS, 1.0, Length, Regis, RegInfo)
4. Server receives RegInfo and check RegInfo.
5. If RegInfo doesn't satisfy its condition then
 - Set Answer to contain name of field(s) in RegInfo table that are not valid with reason
 - Send to client packet (IMS, 1.0, 5, RegRe, Answer)
 - Disconnect with client
- Else
 - Open clients file
6. Set $I \leftarrow 1$, Flag \leftarrow True
7. Do while not end of file
 - 7.1 If RegisteredClients (I). UserID = UserID then
 - Set Flag \leftarrow False
 - Exit loop
 - 7.2 Increment I by 1
8. If Flag = False then
 - Server Send to client packet (IMS, 1.0, 5, RegRe, Exist)
 - Disconnect with client
- Else
 - Open Keys file
9. assignSearch for unused p, α
10. Set $p\alpha$ Index \leftarrow index of p and α
11. Server saves client information in Pending Registering table.
 - Set Pending Registering (index). UserID \leftarrow UserID
 - Pending Registering (index). First Name \leftarrow First Name
 - Pending Registering (index). Last Name \leftarrow Last Name
 - Pending Registering (index). Birthday \leftarrow Birthday
 - Pending Registering (index). $p\alpha$ Index \leftarrow $p\alpha$ Index
 - Pending Registering (index). ChallengeInfoQuestion \leftarrow ChallengeInfoQuestion
 - Pu = p &";" & α
12. Server send client packet (IMS, 1.0, Length, RegPu, Pu)
13. Client receive packet and do the following:
 - 13.1 Set a \leftarrow UserID & Password
 - 13.2 Set $a_c \leftarrow$ UserID & Challenge Information
 - 13.3 Call MD5 (a) algorithm and put hexadecimal result in a
 - 13.4 Call MD5 (a_c) algorithm and put hexadecimal result in a_c

- 13.5 Convert a to decimal representation and put result in a .
- 13.6 Convert a_c to decimal representation and put result in a_c .
- 13.7 Call Compare (a, p) algorithm “algorithm 4.3” and put result in flag
- 13.8 If flag= “Greater” then
 Call DivMod (a, p_1) algorithm “algorithm 3.9” and put result in a
- 13.10 Call FastExp (α, a, p) and put result in y .
- 13.11 Repeat step 13.9 for a_c instead of a
- 13.12 Call FastExp (α, a_c, p) and put result in y_c .
14. Send to server the packet (IMS, 1.0, Length, RegPu, UserID; $y; y_c$)
- 15 Server receive packet and do the following:
 If UserID \neq Pending Registering (index).UserID then
 Delete client from pending registering table
 Send error packet to client (IMS, 1.0, 4, RegRe, Fail)
 Return Fail
 Else
 Open Clients file
 Add new entry to RegisteredClients table
 Registeredclients.UserID \leftarrow UserID
 RegisteredClients.FirstName \leftarrow pending registering (index).First Name
 RegisteredClients.LastName \leftarrow pending registering (index).Last Name
 RegisteredClients.Birthday \leftarrow pending registering (index).Birthday
 RegisteredClients.ChallengeInfoQuestion \leftarrow ChallengeInfoQuestion
 RegisteredClients.p α index \leftarrow pending registering(index).p α index
 RegisteredClients.y \leftarrow y
 RegisteredClients.y $_c$ \leftarrow y $_c$
16. Server Send client packet (IMS, 1.0, 4, RegRe, Done)
17. Delete client from Pending Registering table
18. Disconnect with client
19. Return Success

To check if a UserID is already exists in Registered client table, client must send to server the packet (IMS, 1.0, RegCh, UserID) to server using WsTcpClientReg, the server reply with Packet (IMS, 1.0, Length, Answer), and then server disconnects with client. If the UserID exists the Answer will be “Exists” else it will be “Not Exists”.

3.6 Login Module

The client must present its own correct UserID and Password to login, figure (3.4) illustrates successful login sequence. After successful login the client will be considered to be Online. In this module the server will make use of the windows sockets (WsTcpServerLog), which created in Starting Serving part (3.4.2).

Successful login requires client to pass the authentication test. The authentication test involves the client to send its UserID to server, server checks if UserID exists in OnlineClients table (table 3.6), if UserID was exists then server inform client that this UserID is already logged in.

Table (3.6): OnlineClients

| |
|---------------|
| <i>UserID</i> |
| Paul_2002 |
| Linda_Wilson |
| Martin_Steve |
| etc |

If UserID was not exists in OnlineClients table, server search for that client p , α and y in Registered Clients table, add then to Pending Login table (table 3.7) and send them to it.

Table (3.7): Pending Login

| UserID | p | α | y |
|---------------|-----------------------|----------------------------|-----------------------|
| Linda_Wilson | 590295810358705651829 | 2 | 531806321106783356666 |

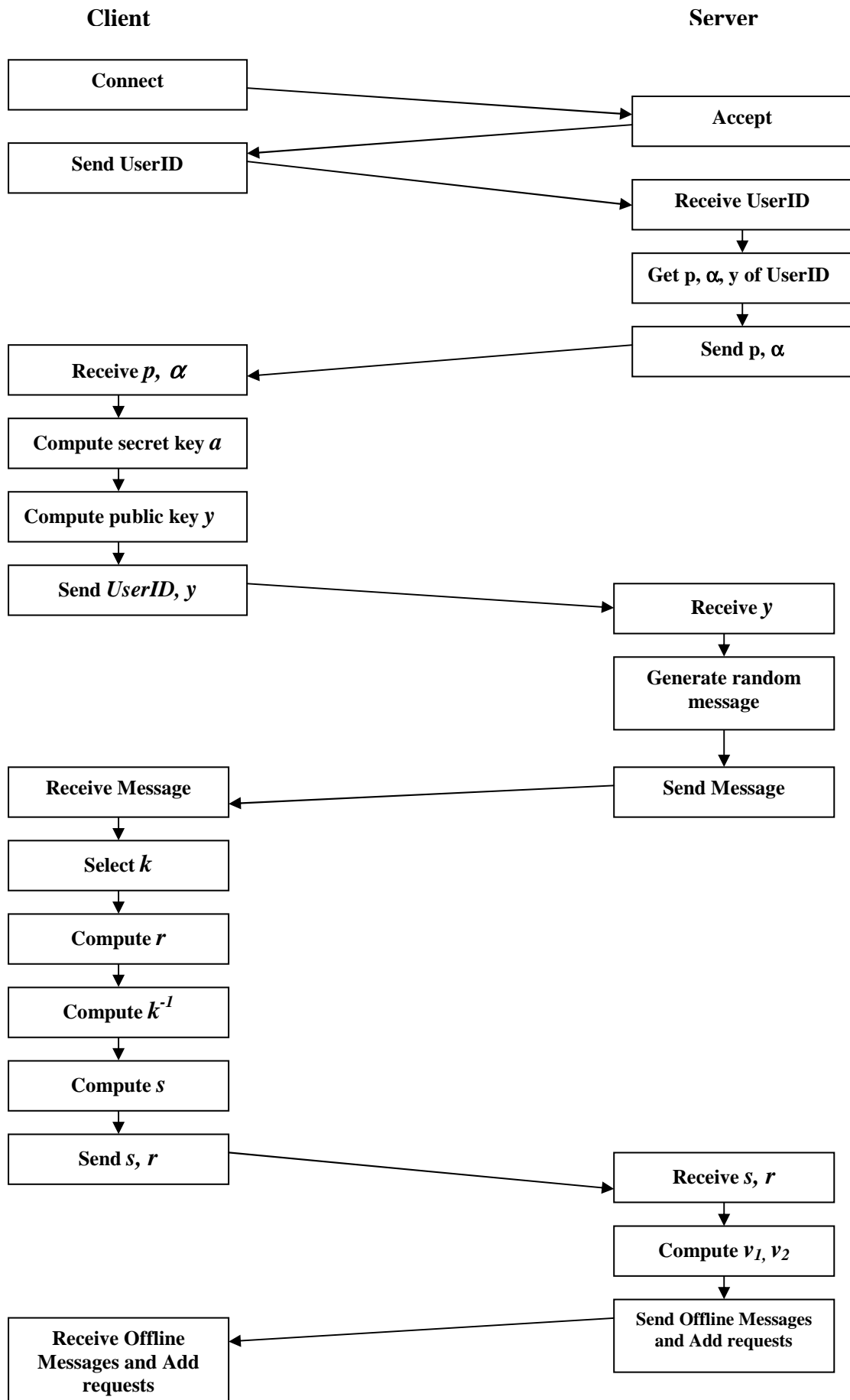


Figure (3.4): Successful Login Sequence

Client uses its UserID and password as input to MD5 hash algorithm to compute its secret key a , then using FastExp algorithm (Appendix A) to compute its public key y [Men96]:

$$y = \alpha^a \text{ mod } p \quad (3.3)$$

Then, client sends y with UserID in a packet to server, server checks if the received pair (UserID, y) matches any entry in PendingLogin table. If no match happened then server sends error message to client and then disconnect with it. But if there was matching then server sends client a message to sign it, the message is hashed using MD5 hash algorithm.

The client receives the hashed message and starts signing process. First client compute a random secret number k , such that $\text{gcd}(k, p-1)=1$, generating the random k will make use of the function Rnd, this function is a defined function in Visual Basic programming language, that returns a value less than 1 but greater than or equal to zero. Then client compute r using as follows [Men96]:

$$r = \alpha^k \text{ mod } p \quad (3.4)$$

To compute signature s , client must compute inverse of k , which is computed using extended Euclidian algorithm (Appendix A). Now client computes signature s as follows [Men96]:

$$s = k^{-1} \{h(m) - ar\} \text{ mod } (p-1) \quad (3.5)$$

Client sends the received message, r , and s to server. Upon receiving the signature, server verify the received signature as follows [Men96]:

$$v_1 = y^r r^s \text{ mod } p \quad (3.6)$$

$$v_2 = \alpha^{h(m)} \text{ mod } p \quad (3.7)$$

If $v_1 \neq v_2$ then the client failed to prove its identity (i.e. verification process failed) so server send error message to client and then disconnect

with client. But If $v_1 = v_2$ then verification process succeeds and client allowed being online.

The server adds this client's UserID to OnlineClients table (table 3.6).

The server also sends the UserID and IP of this client to all clients in Online Clients table, finally server checks Offline table and sends to this client the UserIDs (if there is any) in Buddy List field, also server sends any offline messages or add requests that may be sent by other clients. Algorithm 3.11 illustrates login process.

Algorithm (3.11): Login Process

Input: Server IP (SerIP)

Server local port (pt_2)

UserID of a client aims to login (UserID)

Password of UserID (Password)

Output: Success or Fail

Procedure

1. Using *WsTcpClientLog* client A connects to server's IP (SerIP) on its listening port pt_2 .

2. If Server is available then server do the following:

2.1 Increment number of client's attempting to login by one,

Set $index \leftarrow index + 1$

2.2 Accept connection request

2.3 Send to client the packet (IMS, 1.0, 5, Login, Ready)

Else

Client get error message telling that server is not available at current time

Repeat step 1.

3. Client sends Server Packet (IMS, 1.0, Length, Login, UserID)

5. Server receive packet and set $I \leftarrow 1$

6. Do until $I \geq Index - 1$

If *OnlineClients* (i). UserID = UserID then

a. Send client Packet (IMS, 1.0, 5, LogRe, Loged)

b. Disconnect with client
 c. Return Fail
 Else
 Increment I by 1
 7. Set $I \leftarrow 1$, $P\alpha\text{Index} \leftarrow 0$
 8. Open Clients file
 9. Do while not end of file
 If RegisteredClients (I). UserID=UserID then
 Set $P\alpha\text{Index} \leftarrow$ RegisteredClients (I). $P\alpha\text{Index}$
 Set Pending Login(index).UserID=UserID
 Set Pending Login(index). $y \leftarrow$ RegisteredClients (I). y
 Exit loop
 10. If $P\alpha\text{Index} \neq 0$ then
 10.1 Open keys file
 10.2 Set Pending Login (index) . $p \leftarrow$ PrimesGenerators ($P\alpha\text{Index}$). p
 Set Pending Login (index). $\alpha \leftarrow$ PrimesGenerators ($P\alpha\text{Index}$). α
 10.3 Sends client Packet (IMS, 1.0, Length, Log $p\alpha$, p , α)
 Else
 Sends client Packet (IMS, 1.0, 4, LogRe, Fail)
 Disconnect with client
 Return Fail
 11 Client receive packet and do the following:
 11.1 Set $a \leftarrow$ UserID & Password
 11.2 Call MD5 (a) algorithm and put hexadecimal result in a
 11.3 Convert a to decimal representation and put result in a .
 11.4 Call Compare (a , p) algorithm “algorithm 4.3” and put result in flag
 11.5 If flag= “Greater” then
 Temp=Divmod (a , p)
 11.6 Call FastExp(α , a , p) and put result in y .
 11.7 Send server the packet (IMS, 1.0, Length, Log y , UserID; y)
 12. Server receive packet and do the following:
 12.1 If received $y \neq$ Pending Login (index). y then
 Send client packet (IMS, 1.0, 4, LogRe, Fail)

Disconnect with client.

Return Fail

Else

Server create random message (M) of length 128 bit

12.5 *Call MD5 algorithm to compute message digest of (M) and put result in M.*

12.7 *Send the hashed message to client (IMS, 1.0, Length, HsMsg, M)*

13. *Client receive packet and do the following:*

13.1 *set $G \leftarrow "0"$*

13.2 *Do until $G=1$*

a. *$Rd \leftarrow Rnd$*

b. *Set $k \leftarrow \text{right}(Rd, \text{length}(Rd)-2)$*

c. *Compute $GCD(k, p_1)$ and put result in G*

13.3 *Call $FastExp(\alpha, k, p)$ and put result in r .*

13.4 *Call $Inverse(k, p-1)$ algorithm and put result in $k1$*

13.5 *call $Multiplication(a, r)$ algorithm "algorithm 3.7" and put result in s .*

13.6 *call $Subtraction(M, s)$ algorithm "algorithm 3.5" and put result in s .*

13.7 *Call $Multiplication(k1, s)$ algorithm "3.7" and put result in s .*

13.8 *cal $DivMod(s, p1)$ and put reminder result in $Temp$.*

13.9 *Call $Addition(s, p1)$ algorithm "algorithm 3.8" and put result in s .*

13.10 *Client send signature to server (IMS, 1.0, Length, SgMsg, M;s;r)*

14. *Server receive packet and start verification process as follows:*

14.1 *If $r \leq 1$ or $r \geq p-1$ then*

Send error Packet (IMS, 1.0, 4, LogRe, Fail)

Return Fail

Else

Call $FastExp(\text{Pending Login (index).y}, r, \text{Pending Login (index).p})$ algorithm and put result in $Temp1$

Call $FastExp(r, s, \text{Pending Login (index).p})$ algorithm and put result in $Temp2$

Call $Multiplication(Temp1, Temp2)$ algorithm "algorithm 3.7" and put result in $Temp2$

Call $DivMod(Temp2, \text{Pending Login (index).p})$ algorithm "algorithm 3.9" and put reminder result in v_1 .

```

14.2 Call FastExp (Pending Login (index).α, m, Pending Login (index).p)
      algorithm and put result in v2.
14.3 Call Compare (v1, v2) algorithm “algorithm 3.4” and put result in Answer
14.4 If Answer= ”Equal” then
      Send to each client in OnlineClients table the UserIDs in this table with IP
      of each client
      Else
      Send Packet= (IMS, 1.0, 4, LogRe, Fail)
      Return Fail
14.5 Set I←-1, off←””, request←””
14.6 open Offline table
      Do while not end of file
      If Offline (I). UserID=UserID then
      If Offline (I). Offline≠”” then
      Set off←Offline (I). Offline
      If Offline (I).AddRequest ≠ “” then
      Set request←Offline (I).AddRequest
14.7 Set Message←off &”;” & request
14.8 server send to client (IMS, 1.0, Length, LogOf, Message)
15. Client receive Client create windows socket WsChat1 to use it for chatting with
      other clients in private area chat
16. Return Success

```

3.7 Chatting Module

In this module the client is allowed to:

1. Chat with other clients.
2. Adding UserID to a list named Buddy List.
3. Sending offline messages to offline clients.
4. Changing known password.

The following sections provide clear description of these functions.

3.7.1 Chat

Online clients can send instant messages to each other using:

1. **Public chat area:** where all clients chat in one area such that the text in this area can be seen by all clients, this chat area contains UserIDs of all online clients. The instant message is first sent to server using the windows socket `WsTcpClientLog`, and upon receiving it; the server sends it to all clients except sender. The clients receive the instant message and print it in public chat area.
2. **Private chat area:** this chat area is specified for two users only. Also online client can use both chat areas at the same time. The client can use this chat area for chatting with online clients or for sending offline messages to offline clients. As mentioned before each online client creates windows socket `WsChat1` and start listening on port pt_3 . Suppose client A want to chat with client B, so there is two cases:
 - If client B is online then client A create windows socket `WsChat2`, and save the instant message typed by user in buffer, then client A connects to client B on the listening port pt_3 . Client B accept connection request, and inform A that its ready to chat. Client A sends buffered instant message to B, client B receive that instant message and now both clients, A and B, can exchange instant messages.
 - If client B is offline then client A save instant message typed by user in buffer and then using the windows socket `WsTcpClientLog`, client A send the typed message to server, server will update the client B entry in Offline table such that Offline field in this table will contain: UserID of client A, A's instant message and current time.

3.7.2 Adding UserIDs to Buddy List

The client is allowed to modify Buddy List field in Offline table. Suppose client A wants to add client B to its Buddy List, so using `WsTcpClientLog`, client A sends to server client's B UserID. Server checks if this UserID exists in `RegisteredClients` table, if it does not exist then server sends error message to client A to inform it that this UserID is not exists. Else if UserID exists then server check if B's UserID already exists in A's Buddy List, if its exists then server sends to client A a message telling that client B is already in A's Buddy List, else server checks if client B is online, if B is offline then server updates client B entry in Offline table so that Add request field will contain UserID of client A and current time, but if B was online then server send to it a message telling that client A wants to add it to its Buddy List table. If Client B refused add request then server send to client A a message telling that client B refused adding request, else server update client A entry in Offline table so that So that clients A and B UserID will be added to Buddy List field in A and B entries in offline table..

The function of changing password is same as described in section 3.8.1.

3.8 Change Password Module

The password can be changed by user in two cases: first when user wants to change the password which he/she knows, second when user forgot the password correspond to a UserID.

3.8.1 Changing Known Password

To change a password correspond to a specific UserID, the client must first connect with server, connection will be done using

WsTcpClientLog socket (pt_2). After successful connection, the client must presents its UserID, current password and the new password, the client send the UserID to server, server receives UserID and searches for it in RegisteredClients table, if the UserID does not exist in this table then the server sends error message to client.

But if the UserID exists in RegisteredClients table so server gets p , α correspond to this UserID from this table and sends them to it, the client receives the pair (p, α) and computes the secret key a , $a = \text{MD5}(\text{UserID} \& \text{current password})$, then uses a to generate the public key y as follows [Men96]:

$$y = \alpha^a \bmod p \quad (3.8)$$

Client sends to server its UserID and public key y , then the server verifies if the received y match an entry in Registered client table correspond to the received UserID. If the match happened then server sends client a random hashed message to sign, the client receive message and select a random value k such that $\text{gcd}(k, p-1)=1$ and then compute $k^{-1} \bmod p - 1$, to compute r as follows [Men96]:

$$r = \alpha^k \bmod p \quad (3.9)$$

and then compute s as follows [Men96]:

$$s = k^{-1} \{h(m) - a * r\} \bmod (p - 1) \quad (3.10)$$

Client send the signature pair (s, r) to server, server verify signature as follows [Men96]:

$$v_1 = y^r r^s \bmod p \quad (3.11)$$

$$v_2 = \alpha^{h(m)} \bmod p \quad (3.12)$$

If $v_1=v_2$ then sever sends to client a message that verification process succeed, now client computes a new secret key a and computes public key y and sends it to server, server replaces old value of y with the received y .

The client can make use of the new password next time client login.

3.8.2 Changing Forgotten Password

Changing forgotten password is another function of the proposed IMS. To make use of a UserID which its password was forgotten, first the client connects to server using window socket WsTcpClientLog, server accept connection request and asks client to sent its UserID and birthday date, if UserID is not exist in RegisteredClients table then server send to client error message, else server checks if the entered birthday is correct, only if birthday date is correct then server get the question of challenge information of this client from RegisteredClients table and sends it to client. Client will answer the question and then the answer will be hashed with UserID using MD5 hash algorithm to compute a_c , $a_c = \text{MD5}(\text{UserID} \ \& \ \text{Challenge Information})$, then compute public key y_c as follows [Men96]:

$$y_c = \alpha^{a_c} \text{ mod } p \quad (3.13)$$

Client sends y_c to server, server receives y_c and checks if the received y_c matches the y_c correspond to that UserID in RegisteredClients table. If it doesn't match then server sends error message to client, else server sends to client a random hashed message to sign. Client sign the message (as with login) by first selecting a random number k , such that $\text{gcd}(k, p-1)=1$ and computing r as follows [Men96]:

$$r = \alpha^k \text{ mod } p \quad (3.14)$$

Client selects a random secret number k , such that $\text{gcd}(k, p-1)=1$, and then computes $k^{-1} \text{ mod } p-1$ to compute s as follows [Men96]:

$$s = k^{-1} \{h(m) - a_c * r\} \text{ mod } (p-1) \quad (3.15)$$

Client sends signature pair (r, s) to server. Server receives signature pair and verifies signature, as follows [Men96]:

$$v_1 = y_c^r r^s \bmod p \quad (3.16)$$

$$v_2 = \alpha^{h(m)} \bmod p \quad (3.17)$$

If $v_1 = v_2$ then verification process succeeds. The new secret key will be computed using the UserID and the new password, $a = \text{MD5}(\text{UserID} \ \& \ \text{password})$, then compute public key y [Men96]:

$$y = \alpha^a \bmod p \quad (3.18)$$

and send public key to server. Server replace the old y with the new received one.

CHAPTER FOUR

IMS Interface and Evaluation

4.1 IMS Interface

IMS system consists of two interfaces, Server (IMS Server), which is installed in the Administrator's computer, and the Client (IMS Client), which is installed in the remote computers.

The IMS Client.exe must run in the remote PC in the network by adding it to the start up menu of the remote PC to be used by any user who aims to be member of IMS's clients. At the Administrator side, the IMS Server.exe is executed in the Administrators computer to start client's serving.

4.2 How to Use IMS Server

IMS server interface designed to enable the Administrator to serve clients reside at the host or remote PC(s) easily. When IMS executed, the main form, shown in figure (4.1), will appear. This form consists of three menus.

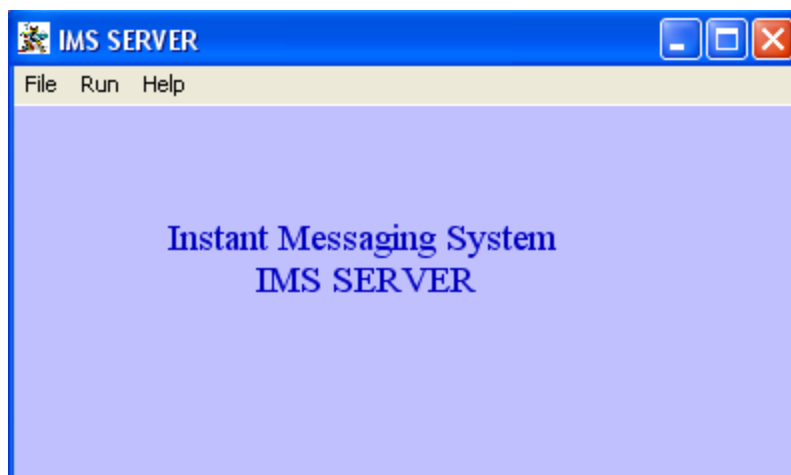


Figure (4.1): IMS Server Main form

Menus in the main form are:

1. **File:** allows administrator to generate new primes/generators table, which will contain values for (p, α) in new file or in an existing file as shown in figure (4.2).

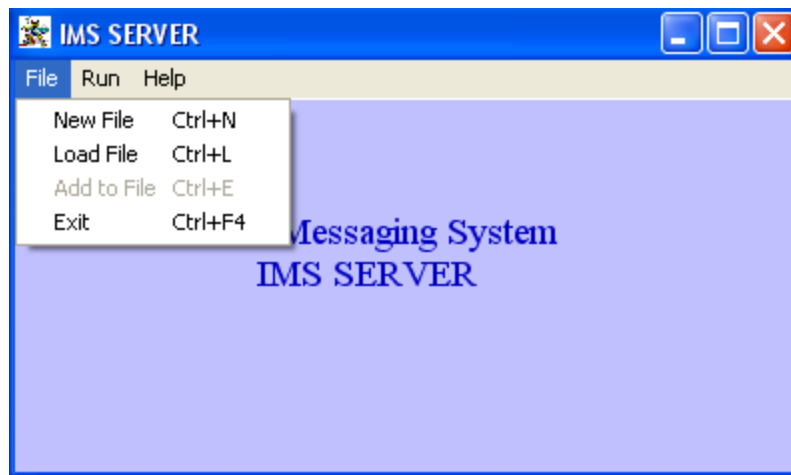


Figure (4.2): File Menu

2. **Run:** allow administrator to start or stop IMS Server's running as shown in figure (4.3).



Figure (4.3): Run Menu

3. **Help:** This option contains one function, this function is **About**, as shown in figure (4.4). Clicking on this About option will displays a message box contain short information about the IMS, as shown in figure (4.5).



Figure (4.4): Help Menu



Figure (4.5): About Option

The following steps show how administrator should use IMS Server to be able to serve IMS users properly:

Step1: when administrator first runs the IMS Server, then the main form shown in figure (4.1) will appear.

Step2: the administrator must first fill the Primes/Generators table to generate (p, α) pairs by clicking on "New File" option in File menu. When clicking on this option, the form shown in figure

(4.6) will appear, it allows administrator to choose where to save the file which Primes/Generators table will be saved in.



Figure (4.6): save form

After that, the form shown in figure (4.7) will automatically appear, the administrator enters number of primes to be generated and length (in bits) of minimum prime to be generated, then click OK button. When (p, α) generation complete message box will appear to inform administrator that generation was complete. Then administrator can click on Close button to exit.

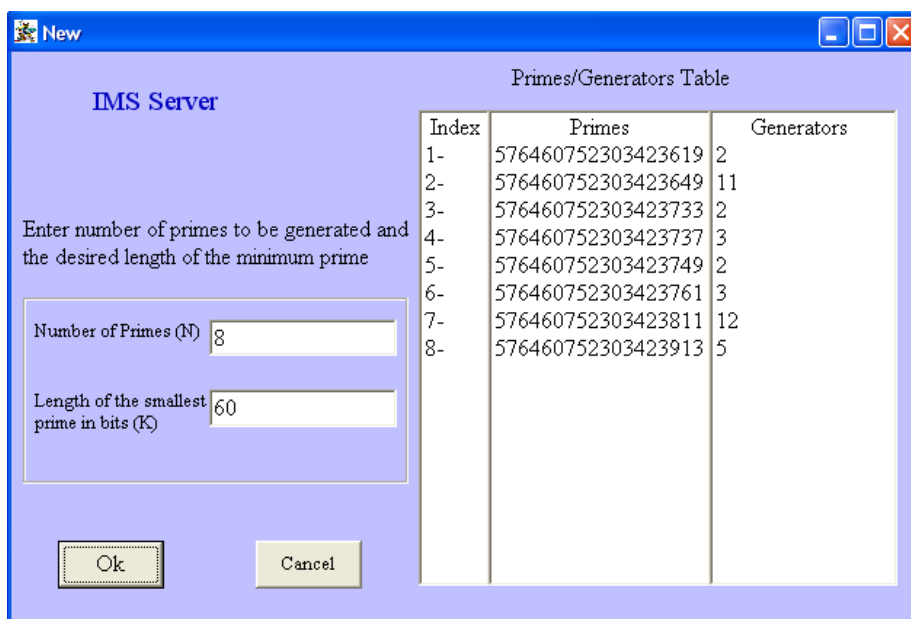


Figure (4.7): New form

Step3: once generation was completed successfully so server program is ready to start serving. To start serving administrator click on “Start” option in Run menu. Since server is running now so “Start” option will be inactive to prevent administrator clicking on this option by mistake.

The administrator can add new (p, α) values to an existing file by clicking on “Add to File” option in File menu, when clicking on this option then the form shown in figure (4.8) will appear. The administrator should enter the number of extra primes to be generated and then click on “Add”. When generation complete, a message box will appear to inform administrator that adding new primes completed successfully.

To stop server running, administrator can click on “Stop” option in Run menu.

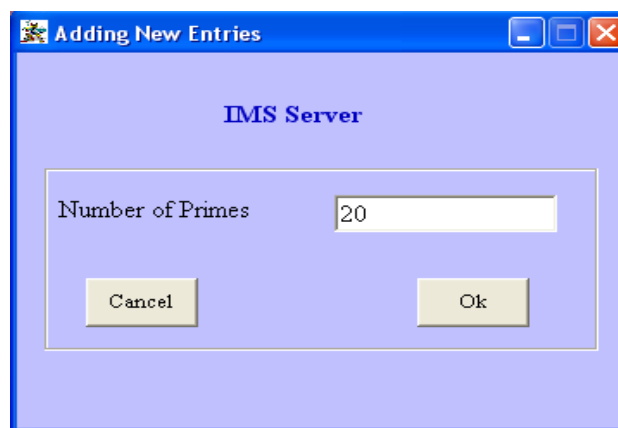


Figure (4.8): Add form

4.3 How to Use IMS Client

IMS Client is designed to allow user to chat with other users already logged in IMS. When a user executes IMS Client, the main form shown in figure (4.9) will appear. This form consists of:

1. UserID textbox: to allow user to enter its own UserID for login.
2. Password textbox: to allow user to enter the password corresponding to a specific UserID.

3. Server IP textbox: allow user to enter IP address of server to provide the ability to connect to that IP, this textbox contain a default IP, so that user will not have to enter the IP only if server IP changed.
4. Change Your Password Button: this button allows the user to change a password that he/she knows or forgot.
5. Register Button: a user who aims to register with new UserID in IMS can use the register button.
6. Login Button: an already registered user can login after entering its UserID in UserID text box and its password in Password textbox.

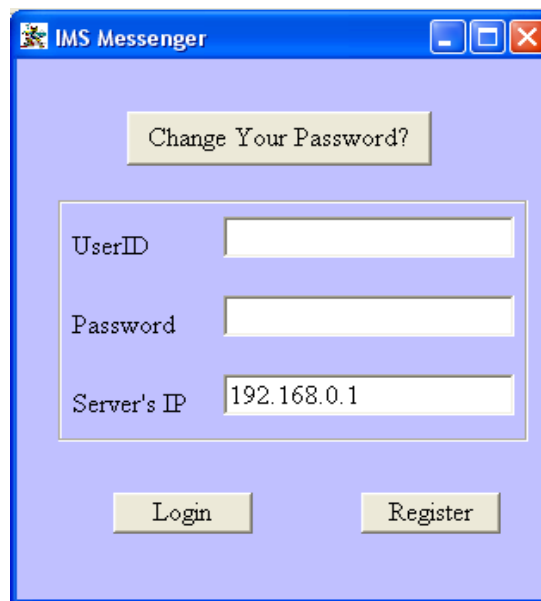


Figure (4.9): IMS Messenger Main form

4.3.1 How to register in IMS

The following steps show how a new user can use IMS Client program (IMS Messenger) for registration.

Step1: User enter server IP in “Server IP” textbox, then click on “Register” button in figure shown above, the form shown in figure (4.10) will appear. The user must enter the following:

1. First Name: represents first name of user, its length must be between 4 and 15.
2. Last Name: represents last name of user, its length must be between 4 and 15.
3. UserID: username 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 ask server if the entered UserID is used before by another user or not.
5. Password: password to be used with the above UserID for login, its length must be between 8 and 15, containing at least one character between (a-z) or one character between (A-Z) and at least one number value (0-9).
6. Questions List: number of questions that user must answer of them.
7. Challenge information: represent an answer to a question chosen by user; this information is needed when user forgot its password.

The screenshot shows a web browser window titled "Registration" with a light blue background. The text "Welcome, please fill the following information" is displayed at the top. The form contains several input fields and a button:

- First name:** Text input field containing "Linda".
- Last name:** Text input field containing "Wilson".
- Birthdate:** Three dropdown menus for year (1981), month (September), and day (10).
- UserID:** Text input field containing "Linda_Wilson" and a "Check Availability" button to its right.
- Password:** Text input field containing "*****".
- Re- enter password:** Text input field containing "*****".
- Select a question and answer it:** A dropdown menu with the question "Whats the name of your first school?" and a text input field containing the answer "Al Firdouse".
- Submit:** A button at the bottom center of the form.

Figure (4.10): Registration form

Step2: After filling the above information, the user should click on Submit button.

Step3: If the entered information accepted by server then the form shown in figure (4.11) will appear, otherwise the form (4.10) will appear again and the wrong entered information will be marked by “*” to inform user that these information are wrong.



Figure (4.11): Successful Registration form

4.3.2 How to Login to IMS

For login, the user must enter a UserID, password, and Server IP in the main form shown in figure (4.9), and then click on “Login” option. If the user didn’t enter a UserID or password or both, then a warning message box will appear as shown in figure (4.12).



Figure (4.12): Wrong Information form

But if the user enters the UserID and password, the form in figure (4.13) will appear; this is the public room where all users of the room can see what every other member types.

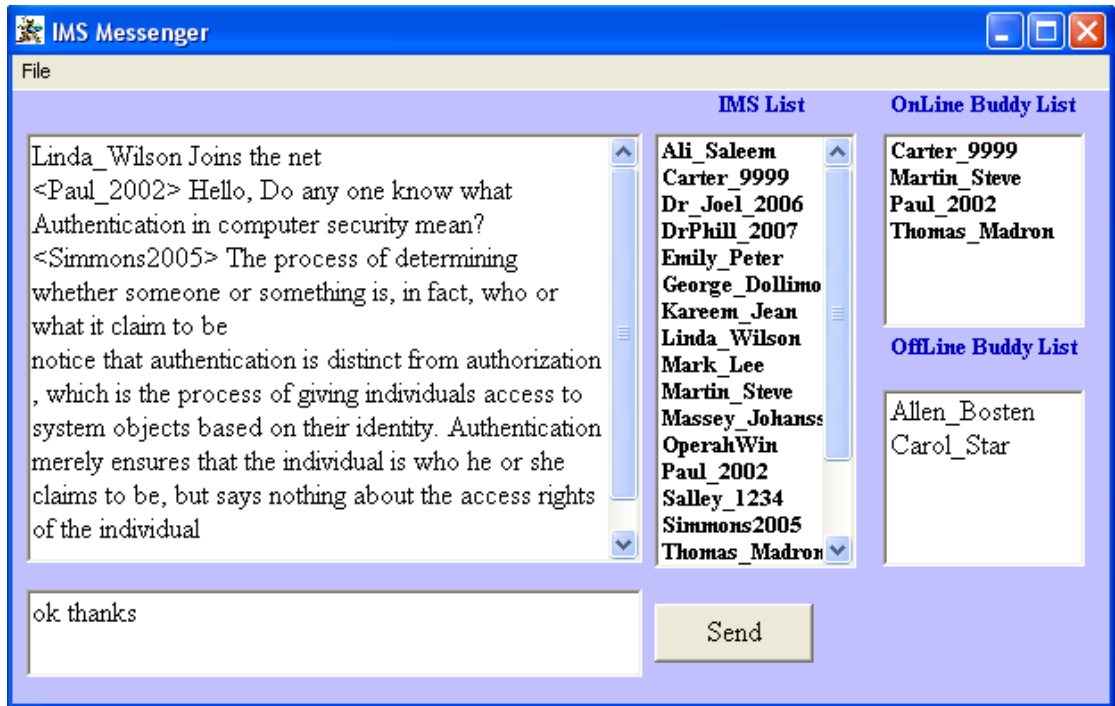


Figure (4.13): Public Area Chat

This form contains:

1. File menu: it contains four options:
 - Add user: allows user to add a UserID to his/her buddy list.
 - Change password: helps user to change the password after login, the new password can be used when the user tries to login in the future.
 - Create new UserID: allows user to create a new UserID after login.
 - About: displays a message box contains short information about the IMS, as shown in figure (4.14).
 - Logout: to exit from IMS Messenger



Figure (4.14): About form

2. In textbox: this text box contains the text typed by this user and other users logged in from another's PCs.
3. Out textbox: allows user to type text to be sent to all other users.
4. Send button: this button can be used to send any text typed in Out textbox.
5. IMS List: this list contains the UserID's of all IMS member that are logged in.
6. Online Buddy List: this list contains UserIDs of Online's users in user buddy list.
7. Offline Buddy List: this list contains UserIDs of Offline's users in user buddy list.

Both lists (Online Buddy List and Offline Buddy list) appear every time user login.

In order for a user to chat with any user listed in this IMS list or Buddy List, he/she must click on its UserID, then the form shown in figure (4.15) will appear, the title bar of this form will contains the UserIDs of users participating in this chat area, this form also contains two textbox: In textbox and Out textbox for incoming and outgoing text as in public room.

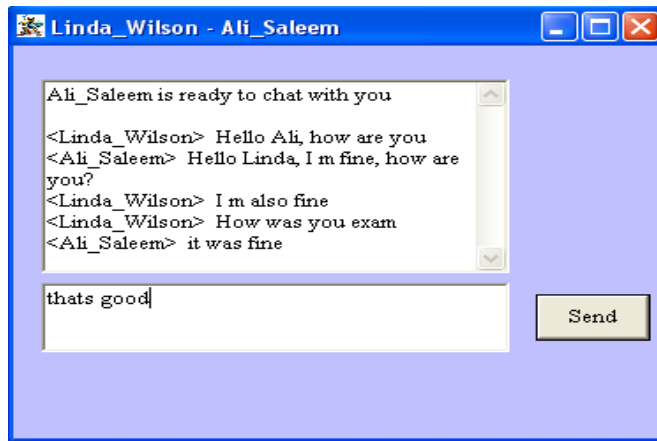


Figure (4.15): Private Area Chat

4.3.3 Adding a UserID to Buddy List

The user can add UserID(s) to his/her Buddy List by clicking on Add User option in file menu of public chat area form shown in figure (4.13), after clicking on this option the Add User form will appear as shown in figure (4.16).



Figure (4.16): Add User form

Suppose the user “Linda” aims to add the UserID of “Martin” to her Buddy List, Linda must enter the UserID of “Martin” in the text box shown in figure (4.16) and then click Add. The server send this add request to “Martin”, when “Martin” is online he will receive a message shown in figure (4.17).

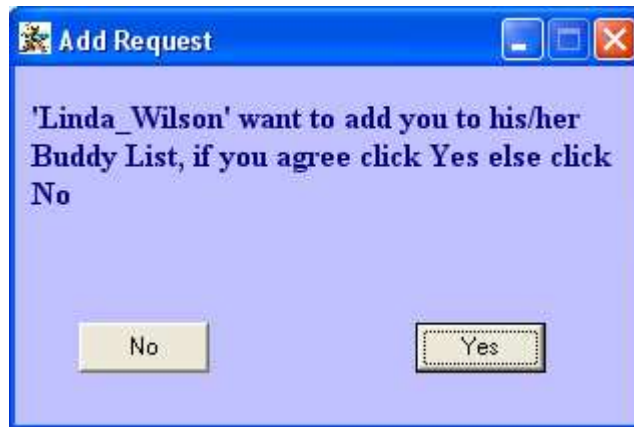


Figure (4.17): Add Request form

“Martin” must reply the add request in the above form, if “Martin” replied No then the form shown in figure (4.18) will appear on Linda’s screen, which inform “Linda” that “Martin” refused to add her to his buddy list, else the form shown in figure (4.19) will appear. The server then will add Martin’s UserID to Linda’s buddy list; also Linda’s UserID will be added to Martin’s buddy list.

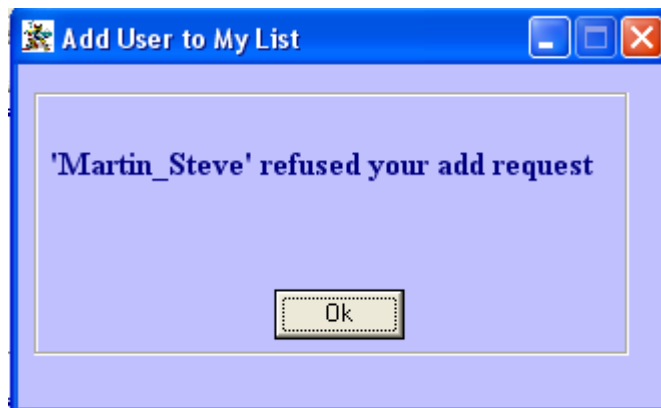


Figure (4.18): Reject Add Reply



Figure (4.19): Accept Add Reply

4.3.4 Changing Password

The user is allowed to change the password which he/she knows or forgot, the following steps illustrate the process of changing a known password:

Step1: the user click on “Change password” button in figure (4.9), then the form shown in figure (4.20) will be appear.



Figure (4.20): Change password form

Step2: in this form there are two options

1. Changing known password.
2. Changing forgotten password.

The user must select the first option and then click “next” button, then the form shown in figure (4.21) will appear.



Figure (4.21): Changing known Password form

Step3: in this form, the user must enter its UserID, the password, which he/she wants to change, and the new password. Then he/she click “Edit” button. If the (UserID, password) pair is correct and the new password satisfy the conditions then a new form will appear to inform user that changing password was completed successful, else the same above form will appear with “*” on right side of the wrong field.

To change a password that was forgotten by user:

Step 1: the user must select the second option in the form shown in figure 4.20 and then click on “Next” button, then the form in figure (4.22) will appear.



Figure (4.22): Change Forgotten Password First form

Step2: in this form the user must enter

1. UserID which user forgot this UserID ‘s password.
2. Birthday date, this date must match the date entered at registration.

Step 3: then the user must click on “Next” button, if the UserID and/or birthday date was wrong then the same form will appear with “*” mark on the wrong field, otherwise the form in figure (4.23) will appear.

Step 4: This form contains the question that was chosen and answered by user at registration. The user must answer this question in the first text box in this form, and enter the new password in the second and the third text box, then click “Send”.

Step5: Only if the answer of the question was correct and the new password satisfy its conditions then a message box will appear to inform user that this UserID can be used again with the new password.

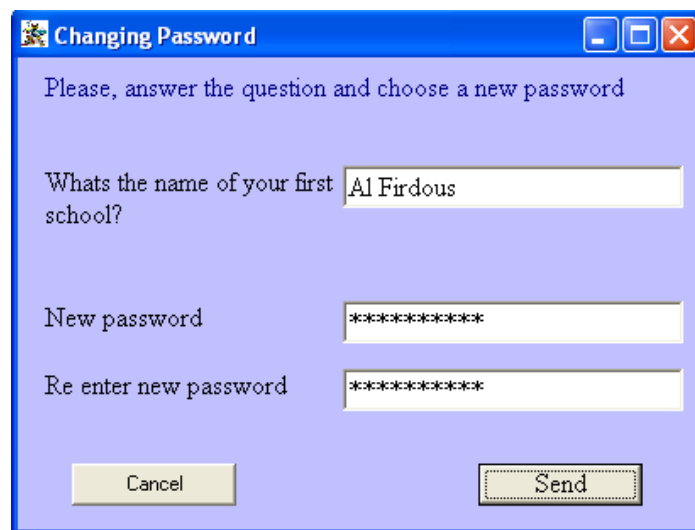


Figure (4.23): Changing Forgotten Password Second form

4.2 IMS Evaluation

In this section, the requirements that are addressed in section 3.2 are discussed to evaluate the performance of the designed system.

1. **Easy of use:** From the user point of view, IMS client interface provides an easy and simple way to do the following activities: registration, login, chatting, sending offline messages, and changing password.
2. **Fast Execution:** one of the important issues of any instant messaging system is speed. IMS passes through many execution phases, these are evaluated as follows:
 - Initialization phase: the server prepares a table of N public key parameters pairs (p, α) , where the first p value of this list is of length equal K bits. This table is generated just for once (in offline mode) on the server side. Table (4.1) shows examples of the time that spent to generate lists of 20 pairs of (p, α) for different K length.

Table (4.1): Examples of Initialization Time Consuming

| <i>K (no. of bits)</i> | <i>Time (in seconds)</i> |
|------------------------|--------------------------|
| 10 | 8.125 |
| 20 | 63.171 |
| 30 | 193.4531 |
| 40 | 428.8594 |
| 50 | 972.4219 |

- Registration phase: this phase is affected mainly by the parameter p (it's length), as shown in table (4.2).

Table (4.2): Examples of Registration Time Consuming

| <i>p</i> length in bits | <i>p</i> value | Time (in seconds) |
|-------------------------|--------------------|-------------------|
| 10 | 521 | 0.8125 |
| 20 | 524309 | 0.90625 |
| 30 | 536870923 | 1.4375 |
| 40 | 549755813911 | 2.328125 |
| 50 | 562949953421381 | 3.953125 |
| 60 | 576460752303423619 | 6.0625 |

- Login phase: as with registration, the most effected parameter by mean of time in login phase is p , table (4.3) shows the approximately time needed by client to login for different p values.

Table (4.3): Login Time Consuming

| <i>p</i> length in bits | <i>p</i> value | Time (in seconds) |
|-------------------------|--------------------|-------------------|
| 10 | 521 | 0.96875 |
| 20 | 524309 | 1.34375 |
| 30 | 536870923 | 2.046875 |
| 40 | 549755813911 | 3.609375 |
| 50 | 562949953421381 | 5.671875 |
| 60 | 576460752303423619 | 9.640625 |

From the above tables, we can notice that the longer part of execution time of the whole system is initiation phase execution time, this time does not affect the client's job, i.e. this time will be invisible to the end user, while the other phases execution's time could be considered to be unnoticeable to the end user. It is also noticed when the number of pairs (p, α) is greater than 10 pairs for K length greater than 60 bits, then the administrator will have to wait long time to finish initialization phase (for example 3 hrs), although this time will be noticed by administrator, it isn't by client.

3. Security: The security of the system is evaluated as follows:

- The password and secret key a are not sent over network neither as plaintext nor encrypted, instead the public key y will be sent, so password is protected against eavesdropping and the secret key a is protected against key only attack since solving discrete logarithm problem in equation 2.1 is very difficult.
- If the password is attacked by brute force attack then this will take very long breaking time to know the password. In this system the password must be between 8-15 length consisting characters from (A-Z or a-z) and (0-9) i.e. 36 characters, so the system as whole consist of $(36)^8 + (36)^9 + \dots + (36)^{15} \approx 2.273903 \cdot 10^{23}$ possible password of length between 8-15. At a rate of 10^9 passwords per second, it would take $7.210499 \cdot 10^6$ years to test all passwords; this time is very enough to make this type of attack not possible.
- The user can change the password when he/she suspects that the password has been compromised. Only the real owner of a UserID can change the corresponding password after presenting the correct challenge information, this challenge information is (as with password) not sent over network, but instead its used to generate secret and public keys and then a digital signature which will be verified by server, when verification process succeed the user is allowed to change the password.
- The messages to be signed are chosen by server, so any adversary will not be able to choose a message and obtains valid signature for it, because the adversary will not guess which message have to be signed.

CHAPTER FIVE

Conclusions and Future Work

5.1 Conclusions

As a result of system implementation, the following conclusions can be noticed:

1. The combination of password and digital mechanisms offer more strength to the authentication process since the drawbacks of password mechanism were improved by this combination.
2. Number of IMS's users can be increased by maximizing the number of primes p at IMS server, this may cause that the new users will have larger prime p and as a result they will get more security.
3. Using windows sockets when designing IMS provides easy of implementation as it is implemented with few statements and need few variables.

The problems of the system were:

1. Knowing the secret key (a) of a UserID by an adversary will let him/her to impersonate the real owner of this UserID.
2. The repeated use of k , when the same k is used, for example by client A, to sign two different messages then there is probability that A's secret key will be known (section 2.4.3), suppose client A logged at time t_1 with signature signed with some k
 $s_1 = k^{-1}\{h(m_1) - a * r\} \bmod (p - 1)$ and then client A logged in at time t_2 with $s_2 = k^{-1}\{h(m_2) - a * r\} \bmod (p - 1)$ and s_2 signed using the same mentioned k , so to determine secret key, adversary must compute $(s_1 - s_2)k \equiv (h(m_1) - h(m_2)) \bmod (p - 1)$, so if $s_1 - s_2 \neq 0 \bmod (p - 1)$ then k

can be determined $k = (s_1 - s_2)^{-1}(h(m_1) - h(m_2)) \bmod (p-1)$, so now secret key can be easily determined. Although this attack can happen but it may take lots of time because the adversary must be presented each time A try to login in order to get all A's signature (s), and then keep computing $s_i - s_j$ until finding that $s_i - s_j \neq 0 \bmod (p-1)$.

3. The instant messages between clients are not encrypted. This may allow someone to intercept the packet and modify the exchanged messages.
4. When number of clients becomes very large then the server's response time will be effected because multithreaded technique was not used to deal with clients.

Suggested solutions for the above last three problems are illustrated in the next section.

5.2 Future Work

The followings are some recommendations for future work in IMS:

1. To provide more security to the system, the signature can be encrypted using an algorithm based on public key infrastructure and then sent over network.
2. Encrypt the exchanged instance messages between clients to prevent eavesdropping.
3. Other functions can be added to the system to provide more reliable interaction between users, like E-mail service, exchanging files between users and using voice for conversation beside text.
4. Using multithreaded technique at server side when exchanging data with clients to ensure high response time from server when number of clients becomes very large.

Reference

- [Ans06] Answers corporation website, "Instant Messaging", 2006.
<http://www.answers.com/topic/instant-messaging.htm>.
- [And06] Andraz Zupan, "Digital signature as a tool to achieve competitive advantage of organization", M.Sc. Degree Thesis, University of Ljubijana, 2006.
<http://www.cek.ef.uni-lj.si/magister/zupan2848.pdf>
- [Bry06] Bryn Mawr College Website, "What are primitive roots?", 2006.
<http://www.brynmawr.edu/math/people/stromquist/numbers/primitive.html>.
- [Cha97] Charles P. Pfleeger, "Security in Computing", Prentice Hall, 1997.
- [Cha04] Charies M. Kozierok, "The TCP/IP Guide", No Strach Press, 2004.
- [Cha05] Chad Cook, "Authentication in Applications", 2006.
<http://www.developer.com/security/article.php/3600351>.
- [Com06] The Computer Technology Documentation Project Web Site Organization, "Network Layers", 2006.
<http://www.comptechdoc.org/independent/networking/protocol/protlayers.html>
- [Cra99] Craige Z. and Paul D., Bookshelf, "Upgrading and Repairing Networks Internet", Macmillan Computer Publishing, 1999.
- [C2c06] Cunningham & Cunningham, Inc, "Discrete Logarithm Problem", 2006.
<http://c2.com/cgi/fullSearch>
- [Dal95] Dale Jonathan, "The communication routines- A Network Layer Communication Model", Multimedia Research Group, Department of Electronic and computer Science, University of Southamptop, UK, Technical Report No.95-2, 1995.
- [Dic06] Dictionary website, "Zero-knowledge proof", 2006.
http://dictionary.laborlawtalk.com/Zero-knowledge_proof=Zero-knowledge proof - definition of Zero-knowledge proof - Labor Law Talk Dictionary.
- [Eli00] Elizabeth D. Zwicky, Simon Cooper and D. Brent Chapman, "Building Internet Firewalls", 2nd Edition, O'Reilly & Associates, 2000.
http://www.unixmexico.org/files/html/kore.hack.se/oreilly-networking/fire/ch21_01.htm.

- [Ent06] Entrust Website, "Digital Signature", 2006.
<http://www.entrust.com/digitalsig/index.htm>
- [Enw06] "Multiplicative inverse", 2006.
http://en.wikipedia.org/wiki/Multiplicative_inverse.
- [Eri00] Eric A. Fisch and Gregory B. White, "Secure Computers and Networks: Analysis, Design and Implementation", CRC Press, 2000.
- [Fre04] Fred Piper, Matt J.B. Robshaw and Scarlet Schwiderski-Grosche, "Identities and authentication", 2004.
http://217.33.105.254/Previous_Projects/Cyber_Trust_and_Crime_Prevention/Reports_and_Publications.
- [Fre06] FreeSoft website, "Hash Function", 2006.
<http://freesoft.org/CIE/Topics/142.htm>.
- [Gar06] Gary C. Kessler, "An Overview of Cryptography", 2006.
<http://www.garykessler.net/library/crypto.html>.
- [Ing06] Ing Frommholz..etc, "State of the Art and Technology Watching Plan, Version 1.0", 2006.
www.brickscommunity.org/discussion_area/reports/D311-StateOfTheArt.pdf
- [Isa01] H. Isaksoon, "Version 5 of the ICQ Protocol", ICQ protocol specification document, 2001.
<http://www.algonet.se/~henisak/icq/icqv5.html>.
- [Irc03] "IRC network statistics", 2003.
<http://irc.netsplit.de/networks/>, 04.06.
- [Jam01] James F. Kurose and Keith W. Ross. "Computer Networking: A top-down Approach Featuring the Internet". Addison-Wesley Longman, Inc, 2001.
- [Jer03] Jeremy Hamman, AIM/Oscar Protocol, 2003.
<http://www.geocities.com/smokeyjoe12345/OscarProtocol.htm>.
- [Joh01] Johannes A. Buchmann, "Introduction to Cryptography", Springer, 2001.
- [Mar00] Mark Curphey, "A guide to building secure web application", Owasp publishing, 2000.

- [Men96] Handbook of Applied Cryptography, by A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996.*
- [Mes06] The yahoo messenger official website, 2006.
<http://messenger.yahoo.com>.
- [Min03] Mintz Mike, "MSN Messaging Protocol description", 2003.
<http://mono.es.gnome.org/imsharp/tutoriales/msn/book1.html>.
- [Par99] Parker T., Bookshelf, "Teach Yourself TCP/IP in 14 Days", Sams Publishing, Second Addition, 1999.
- [Pat02] Patrick McDaniel, "Authentication", The Internet Encyclopedia, John Wiley and Sons, Inc. 2002.
<http://www.patrickmcdaniel.org/papers.html>.
- [Ric01] Richard Duncan, "An Overview of Different Authentication Methods and Protocols", 2001.
http://www.sans.org/reading_room/whitepapers/authentication/118.php
- [Ric93] W. Richard Stevens, "TCPIP Illustrated, Volume 1: The protocols", Addison Wesley, 1993.
www.goldfish.org/books/TCPIP%20Illustrated%20Vol%201/index.htm
- [Riv92] R. Rivest RFC 1321 (Request For Comment), "The MD5 Message-Digest Algorithm", 1992.
- [Seb89] Seberry J. and Pieprzyk J., "Cryptography: an introduction to computer security", prentice Hall, 1989.
- [Wil03] William Stallings, "Cryptography and Network Security: principles and practice", Third Edition, prentice Hall, 2003.
- [Wik06] OLPCWiki website, "Instant messaging challenges", 2006.
http://wiki.laptop.org/go/Discussion_of_Instant_Messaging_Challenges.

Appendix A

Algorithms

This appendix contains three algorithms (MD5, Miller-Rabin algorithm, Extended Euclidean algorithm), which are used in the implementation of the proposed system in chapter three.

A-1 The MD5 Message-Digest Algorithm [Riv92]

The algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest.

Terminologies and Notation

A "word" is a 32-bits quantity and a "byte" is an eight quantity. A sequence of bits can be interpreted in a natural manner as a sequence of bytes, where each consecutive group is interpreted as a byte with the higher-order (most significant) bit of each byte listed first. Similarly, a sequence of bytes can be interpreted as a sequence of 32-bit words, where each consecutive group of four bytes is interpreted as a word with the low-order (last significant) byte given first.

Let the symbol "+" denote the addition of words (i.e., module 2^{32} addition). Let $X \lll s$ denote the 32-bit value obtained by circularly shifting (rotating) X left by s bit position. Let $not(X)$ denote the bit-wise complement

of X , and let $X \vee Y$ denote the bit wise *OR* of X and Y . Let $X \oplus Y$ denote the bit-wise *XOR* of X and Y , and let XY denote the bit-wise *AND* of X and Y .

Description of MD5 Algorithm

We begin by supposing that we have a b -bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of eight, and it may be arbitrarily large. We imagine the bits of the message written down as follows:

$m_0 m_1 \dots m_{b-1}$

The following five steps are performed to compute the message digest of the message.

Step 1. Append padding bits

The message is “padded” (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512. Padding is performed as follows: a single “1” bit is appended to the message, and then “0” bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

Step 2. Append length

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^{64} , then only the low-order 64 bits

of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.)

At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let $M [0 \dots N-1]$ denote the words of the resulting message, where N is a multiple of 16.

Step 3. Initialize MD buffer and constants

A four-word buffer (A, B, C, D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These are initialized to the following values in hexadecimal, low-order bytes first:

Word A: 0x01 23 45 67

Word B: 0x89 ab cd ef

Word C: 0xfe dc ab 98

Word D: 0x76 54 32 10

Step 4. Process message in 16-word blocks

Define four auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

$F(X, Y, Z) = XY \vee \text{NOT}(X) Z$

$G(X, Y, Z) = XZ \vee Y \text{Not}(Z)$

$H(X, Y, Z) = X \text{ XOR } Y \text{ XOR } Z$

$I(X, Y, Z) = X \text{ XOR } (X \vee \text{NOT}(Z))$

In each bit position F acts as a conditional: if X then Y else Z . The function F could have been defined using $+$ instead of \vee since XY and $\text{not}(X) Z$ will never have 1's in the same bit position.) It is interesting to note

that if the bits of X, Y, and Z are independent and unbiased, the each bit of F (X, Y, Z) will be independent and unbiased.

The functions G, H, and I are similar to the function F, in that they act in "bitwise parallel" to produce their output from the bits of X, Y, and Z, in such a manner that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of G (X, Y, Z), H (X, Y, Z), and I (X, Y, Z) will be independent and unbiased. Note that the function H is the bit-wise "xor" or "parity" function of its inputs.

This step uses a 64-element table T [1 ... 64] constructed from the sine function. Let T [i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times abs (sin (i)), where i is in radians. The elements of the table are given in the appendix.

Do the following:

```
/* Process each 16-word block. */
```

```
For i = 0 to N/16-1 do
```

```
  /* Copy block i into X. */
```

```
  For j = 0 to 15 do
```

```
    Set X[j] to M[i*16+j].
```

```
  end /* of loop on j */
```

```
  /* Save A as AA, B as BB, C as CC, and D as DD. */
```

```
  AA = A
```

```
  BB = B
```

```
  CC = C
```

```
  DD = D
```

```
  /* Round 1. */
```

```
  /* Let [abcd k s i] denote the operation
```

```
    a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
```

```

/* Do the following 16 operations. */
[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]
[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]
[ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
[ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22
16]

/* Round 2. */

/* Let [abcd k s i] denote the operation
   a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */

/* Do the following 16 operations. */
[ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]
[ABCD 5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20 24]
[ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]
[ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]

/* Round 3. */

/* Let [abcd k s t] denote the operation
   a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */

/* Do the following 16 operations. */
[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]
[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]
[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]

/* Round 4. */

/* Let [abcd k s t] denote the operation
   a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */

/* Do the following 16 operations. */
[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]

```

```

[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]
[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]
[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]
/* Then perform the following additions. (That is increment each
   of the four registers by the value it had before this block
   was started.) */
A = A + AA
B = B + BB
C = C + CC
D = D + DD
end /* of loop on i */

```

Step 5. Output

The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

A-2 Miller Rabin Algorithm [Men96]

The probabilistic primality test used most in practice is the Miller-Rabin test, also known as the *strong pseudoprime test*.

Algorithm Miller-Rabin probabilistic primality test

Input: an odd integer $n \geq 3$ and security parameter $t \geq 1$.

Output: an answer “prime” or “composite” to the question: “Is n prime?”

1. Write $n - 1 = 2^s r$ such that r is odd.

2. For i from 1 to t do the following:

2.1 Choose a random integer a , $2 \leq a \leq n - 2$.

2.2 Compute $y = a^r \bmod n$

2.3 If $y \neq 1$ and $y \neq n - 1$ then do the following:

$j \leftarrow 1$.

While $j \leq s - 1$ and $y \leq n - 1$ do the following:

 Compute $y \leftarrow y^2 \bmod n$.

 If $y = 1$ then return (“composite”).

$j \leftarrow j + 1$.

 If $y \leq n - 1$ then return (“composite”).

3. Return (“prime”)

A-3 Extended Euclidean Algorithm [Men96]

Inverses can be computed using the extended Euclidean algorithm as next described.

Algorithm Computing multiplicative inverses in Z_n

Input: $a \in Z_n$.

Output: $a^{-1} \bmod n$, provided that it exists.

1. Use the extended Euclidean algorithm to find integers x and y such that $ax + ny = d$, where $d = \gcd(a, n)$.
2. If $d > 1$, then $a^{-1} \bmod n$ does not exist. Otherwise, return (x) .

Algorithm Extended Euclidean algorithm

INPUT: two non-negative integers a and b with $a \geq b$.

OUTPUT: $d = \gcd(a, b)$ and integers x, y satisfying $ax + by = d$.

1. If $b = 0$ then set $d \leftarrow a$, $x \leftarrow 1$, $y \leftarrow 0$, and return (d, x, y) .
2. Set $x_2 \leftarrow 1$, $x_1 \leftarrow 0$, $y_2 \leftarrow 0$, $y_1 \leftarrow 1$.
3. While $b > 0$ do the following:
 - 3.1 $q \leftarrow \lfloor b/a \rfloor$, $r \leftarrow a - qb$, $x \leftarrow x_2 - qx_1$, $y \leftarrow y_2 - qy_1$.
 - 3.2 $a \leftarrow b$, $b \leftarrow r$, $x_2 \leftarrow x_1$, $x_1 \leftarrow x$, $y_2 \leftarrow y_1$, and $y_1 \leftarrow y$.
4. Set $d \leftarrow a$, $x \leftarrow x_2$, $y \leftarrow y_2$, and return (d, x, y) .

A-4 Repeated Square and Multiply Algorithm [Men96]

Modular exponentiation can be performed efficiently with the repeated square and multiply algorithm, which is crucial for many cryptographic protocols.

Algorithm Repeated square-and-multiply algorithm for exponentiation in Z_n

INPUT: $a \in Z_n$, and integer $0 \leq k < n$ whose binary representation is $k = \sum_{i=0}^t k_i 2^i$.

OUTPUT: $a^k \bmod n$

1. Set $b \leftarrow 1$. If $k = 0$ then return (b) .
2. Set $A \leftarrow a$.
3. If $k_0 = 1$ then set $b \leftarrow a$.
4. For i from 1 to t do the following:
 - 4.1 Set $A \leftarrow A^2 \bmod n$.
 - 4.2 If $k_i = 1$ then set $b \leftarrow A \cdot b \bmod n$.
5. Return (b) .

الخلاصة

يعتبر نظام المحادثة الفوري من التطبيقات الواسعة الاستخدام، حيث انه ينمو بسرعة ليكون تقنية اتصال اساسية بين المستخدمين في الشركات والمؤسسات التعليمية والمنزل. مستخدمى نظام المحادثة الفورية بحاجة للتحقق من هويتهم قبل بدء المحادثة لذلك اصبح التحقق من امنية الدخول لهذه الانظمة امراً ضرورياً.

يهدف المشروع الى تصميم وتنفيذ نظام محادثة فورية (IMS) مدعم بطريقة مقترحة للتحقق من وثوقية المستخدم. الطريقة المقترحة تتبنى تقنيتين مختلفتين من تقنيات الوثوقية وهما: كلمة السر الثابتة و التوقيع الالكتروني، ان عملية اختبار الوثوقية تمر بعدة مستويات من اختبارات الامنية والسرية.

نظام المحادثة الفورية المقترح يوفر لمستخدميه الخصائص التالية: المحادثة الكتابية بين المستخدمين باستخدام غرف الحوار العامة والخاصة، ارسال `offline messages`، قابلية تغيير كلمة السر الحالية في حالة سرقتها او نسيانها. نظام المحادثة الفورية المُصمم يستخدم البروتوكول TCP/IP لكل الاتصالات بين `hosts`. كل انواع الربط بين الخادم والمستخدمين تعتمد على موديل الزبون والخادم (`client/server`)، بينما الربط بين الزبائن يعتمد على موديل النظير الى نظيره (`peer-to-peer`).

النظام المقترح قد فُيم من ثلاثة اوجه: سهولة الاستخدام، سرعة التنفيذ و الامنية، حيث ان النظام يوفر للمستخدمين طريقة شائعة وسهلة للتحدث في وقت مقارب للحقيقي، اما مستوى الامنية الموفر لمستخدمى النظام فهو مناسب لهذا النوع من التطبيقات. النظام المصمم نُفذ باستخدام لغة Visual Basic 6.0 وقد أُختبر على حواسيب مربوطة بشبكة محلية LAN وكل الحواسيب تعمل تحت نظام التشغيل Windows XP.



جمهورية العراق
وزارة التعليم العالي والبحث العلمي
جامعة النهرين
كلية العلوم

طريقة وثوقية لنظام محادثة فورية

رسالة مقدمة الى كلية العلوم، جامعة النهرين كجزء من متطلبات نيل شهادة
الماجستير في علوم الحاسوب

من قبل

ملاذ صبري كريم

(بكالوريوس ٢٠٠٣)

المشرفون

د. عبير متي يوسف

د. ستار بدر سدخان