DECLARATION

I certify that this dissertation titled “Notarization Authority for Emails”, does not incorporate any material previously submitted for a degree or diploma in any University and to the best of my knowledge and belief it does not contain any material previously submitted or written by another person or myself except when due reference is made in the text. This dissertation is submitted as part of the requirement for completing the B.Sc. (Special Degree) in Computer Science at University of Colombo School of Computing (UCSC), Sri Lanka.

Candidate:

H.E.M.H.B. Ekanayake (s 5934)

Signature: - . . . . . . . . . . . . . . . . ..   Date: - . . . . . . . . . . . . . . . . ..

Supervisors:

• Mr. Kasun De Zoysa

Signature: - . . . . . . . . . . . . . . . . ..   Date: - . . . . . . . . . . . . . . . . ..

• Mr. Rasika Dayarathna

Signature: - . . . . . . . . . . . . . . . . ..   Date: - . . . . . . . . . . . . . . . . ..
ACKNOWLEDGMENTS

First and foremost I would like to thank Mr. Kasun De Zoysa and Mr. Rasika Dayarathna for giving me this research idea. Their assistance and interest remained the same from beginning to the end of my project, which should be greatly appreciated.

I would like to thank Janaka, Jayantha, Indika, Thushara, Jayani and all my colleagues, for encouraging me to give a great finish to my project. My colleague Lakmal’s role in sharing a single computer with me was very grateful.

Special thanks goes to my mother and my brother, for encouraging me in nasty situations and providing me a free environment at home.
ABSTRACT

Electronic mail, or email, is now widely used by the society for distant communication. Apart from the availability and accessibility, confidentiality, message integrity, and authenticity are primary requirements expected from an email system. However, these features are now incorporated into email systems since the introduction of public key cryptography.

The world is on its way. Now, email becoming crucial for most e-commerce transactions. But, still there are some features missing from the email systems which should be incorporated in order to make this system much more perfect. Guaranteed delivery, some control over an email after send, and proofs for posting and opening for an email, are some of them.

This document describes and explains how a successful attempt has been taken to achieve this requirement, by developing a proxy for email, or as referenced in the overall context, a Notarization Authority. Also, this document describes the concept of time-stamp, and a special certificate derived from it for used as proof of time posting and opening for email. Major concerns are taken in order to preserve the efficiency and effectiveness at a higher degree.
LIST OF TABLES & FIGURES

Figure 2.1: Moving an Email from Originator to the Recipient 11
Figure 2.2: An Internet Text Message 12
Figure 2.3: Model of Communication for SMTP 13
Figure 2.4: Normal SMTP Data Flow 14
Figure 2.5: Security Threats 18
Figure 2.6: Confidentiality by Encryption 18
Figure 2.7: Authenticity by Digital Signature 18
Figure 2.8: Integrity by Fingerprint 19
Figure 2.9: Non-Repudiation by Certificates 19
Figure 2.10: Use Case Diagram for the System 21
Figure 3.1: Domain Architecture of the System 22
Figure 3.2: A High-Level Interaction Diagram of the System 23
Figure 3.3: Evolutionary Prototype Software Development Model 23
Table 3.5: Initial Work Schedule 24
Figure 4.1: High-level View of the System 24
Figure 4.2: Components of the Notary Server 25
Figure 4.3: Components of the Web Server 26
Figure 4.4: Technologies Used by the System 26
Figure 4.5: Class Diagram of the System 27
Table 4.6: Classes and their Purposes 28
Figure 4.7: Sequence Diagram of the Sign-Up Process 29
Figure 4.8: Sequence Diagram of the Mail Observing Process 30
Figure 4.9: Sequence Diagram of the Certificate Obtaining Process 31
Figure 4.10: State Diagram of the Notarization Server 32
Figure 5.1: Functional Arrangement of the Sign Up Process 44
Figure 5.2: Functional Arrangement of the Sign In Process 48
Figure 5.3: Functional Arrangement of the View Email Statistics Process 49
Figure 5.4: Functional Arrangement of the Send/Modify Email Process 51
Figure 5.5: Functional Arrangement of the Delivering Email Body Process 54
Figure 5.6: Functional Arrangement of the Certificate Issuing Process 55
Figure 5.7: User Interface of the Notarization Authority Server 56
Figure 5.8: Functional Arrangement of the Notarization Authority Server 56
Table A.1: Frequently Used Email Header Fields 74
Table A.2: Standard Time Servers 75
Table A.3: Five Standard MIME Header Fields 75
Table A.4: Standard MIME Content-Type Fields 76
Table A.5: Standard MIME Content-Transfer-Encoding Fields 76
Table A.6: Core JavaMail API Classes 77
Figure D.1: Sign Up Step 1&2 94
Figure D.2: Activation Key Received by an email 94
Figure D.3: Sign Up Step 3 95
Figure D.4: Registration Completion 96
Figure D.5: Signing In 96
Figure D.6: Outbox 97
Figure D.7: Sending an Email: Microsoft Outlook 98
Figure D.8: Sending an Email: Web mail 98
Figure D.9: Obtaining a Proof-of Certificate 99
# Table of Contents

**Declaration** 1

**Acknowledgments** 2

**Abstract** 3

**List of Tables & Figures** 4

**Table of Contents** 5

**Chapter 1** Introduction 7

1.1 Motivation 7

1.2 Taxonomy 8

1.3 Aims & Objectives 9

1.4 Report Outline 9

**Chapter 2** Background 11

2.1 Electronic Mail, or Email 11

2.1.1 How it works? 11

2.1.2 Email Message 12

2.1.3 Email & SMTP 13

2.1.4 Email & MIME 15

2.1.5 Email & POP 15

2.1.6 Web Mail 16

2.2 Digital Time 16

2.2.1 Time Server 16

2.2.2 Time Protocols 16

2.2.3 Time Stamping 17

2.3 Secure Data Transmission 17

2.3.1 Security Threats 17

2.3.2 Confidentiality by Data Encryption 18

2.3.3 Authenticity by Digital Signature 18

2.3.4 Integrity by Digital Fingerprint 18

2.3.5 Non-Repudiation by Digital Certificates 19

2.4 Digital Notarization 19

2.4.1 What is Notarization? 19

**Chapter 3** Specification 20

3.1 Problem Description 20

3.2 Features of the System 20

3.3 Problem Definition 21

3.4 The Domain Architecture of the System 22

3.5 Typical Interaction Diagram 23

3.6 Software Development 23

3.7 Initial Work Schedule 23
# Chapter 4: Design

## 4.1 Assumptions & Requirements

## 4.2 Software Design

## 4.3 Database Design

## 4.4 Technological Background

## 4.5 Overall Class Diagram

## 4.6 Sequence Diagrams

### 4.6.1 Sequence Diagram: Sign-Up (Register)

### 4.6.2 Sequence Diagram: Email Delivery

### 4.6.3 Sequence Diagram: Certificate Issuing

## 4.7 Activity Diagrams

### 4.7.1 State Diagram: Notarization Authority Server

# Chapter 5: Implementation & Testing

## 5.1 Software Development

## 5.2 Framework Implementation

## 5.3 Website Implementation

### 5.3.1 Registration (Signing Up)

### 5.3.2 Signing In

### 5.3.3 Viewing the Statistics of an Email

### 5.3.4 Sending and Modifying an Email

### 5.3.5 Delivering the Body of an Email

### 5.3.6 Obtaining a Proof-of Certificate

## 5.4 Notarization Authority Server Implementation

## 5.5 Testing

# Chapter 6: Evaluation

# Chapter 7: Future Works

# Chapter 8: References

# Chapter 9: Appendices

## 9.1 Appendix A – Additional Data

### 9.1.1 Appendix A.1: Some Frequently Used Mail Header Fields

### 9.1.2 Appendix A.2: Some Standard Time Servers

### 9.1.3 Appendix A.3: Five Standard MIME Header Fields

### 9.1.4 Appendix A.4: Standard MIME Content-Type Fields

### 9.1.5 Appendix A.5: Standard MIME Content-Transfer-Encoding Fields

### 9.1.6 Appendix A.6: Yahoo’s Filtering Mechanism

### 9.1.7 Appendix A.7: Core Classes of JavaMail API

## 9.2 Appendix B – Selected Examples

### 9.2.1 Appendix B.1: A Complex MIME Message

### 9.2.2 Appendix B.2: A Complex SMTP Header

### 9.2.3 Appendix B.3: A Real SMTP Data Flow Example

## 9.3 Appendix C – Framework Code Listing

## 9.4 Appendix D – A Short User Manual for Web System Users

### 9.4.1 Signing Up

### 9.4.2 Signing In

### 9.4.3 Sending an Email: Microsoft Outlook

### 9.4.4 Sending an Email: Web Mail

### 9.4.5 Modifying an Email

### 9.4.6 Obtaining a Proof-of Certificate
Chapter 1  INTRODUCTION

1.1 MOTIVATION

Electronic mail, or email, has been around for over two decades. With its exposition to the public, its usage grew exponentially to the point where in the year 2000, on an average day, 5.1 billion emails are sent in the US and 8.2 billion worldwide [WebStat1]. This figure is lot more than the number of snail mail letters exchanging in a day.

Email, like any other competing technology, started its “egg-age” simply by employing only file transfer protocols, with the convention that the first line of each message contained the recipient’s address. And, as the experience was gained, it has broken its eggshell and spread its wings as a more sophisticated email system. This more elaborative email system was convoyed by RFCs proposed those days, including RFC 821 and RFC 822. But still, email consisted exclusively of text messages written in English and expressed in ASCII.

MIME (Multipurpose Internet Mail Extension) has changed the situation. It strengthened the email structure by introducing some encoding mechanisms for include non-ASCII messages. As a result, in addition to text messages, images, audio, video, and various other types of messages are now easily sent by email.

As more people send confidential information by email, the potential requirement for adequate security become into spire. Because, many people would like to be able to sent email that can be read by the intended recipient and no one else. Also, they do not like their messages to be forged in transit. This desire has stimulated to apply the cryptographic principles to produce a secure email system. PGP (Pretty Good Privacy), PEM (Privacy Enhanced Mail) and S/MIME (Secure/MIME) are some widely used secure email systems. Email is now at this age and Outlook Express, Netscape Messenger and most other agent programs are compatible with these features.

However, someone can question about its marvelous ness as follows.

- “Does it posses guaranteed delivery?”
- “How do I know when my email got received?”
- “What’s happened to my email after I sent?”
- “How do I claim or do I have evidence to prove that actually I posted the email?”
- “How does the recipient prove that it has been observed or not?”
- “What happens next to the current email system; can it go beyond from its current position?”
Notarization Authority for Emails
H.E.M.H.B. Ekanayake

The next section will unroll a taxonomy listing some attempts that have put-up by some organizations, discussing strengthens and weaknesses, if possible.

1.2 TAXONOMY

Microsoft Outlook is a widely used email agent program and this discussion going to start its journey with it. Outlook supports wider range of protocols. It is MIME and S/MIME compatible. Besides from those, it supports SMTP, POP, NNTP, IMAP, and LDAP protocol functionalities. As an email agent program, it provides facilities to define an email message by filling up from, to, subject and body fields and other optional fields as wish, and then sending. Also, images, audio and video can be attached to an email without any difficulty. If security is need, there are facilities to encrypt and sign messages by using digital Ids. Moreover, a priority value can be added to an email and also a request can be made for a read receipt.

Now its time to question about strengthens and weakness in this email agent on the basis of questions identified in the previous section. “Does it supports guaranteed delivery?” the answer to this question depends on whether recipients have acknowledged to emails correctly. If acknowledged then the sender will receive a read receipt, if not then null. Also, vast majority of email agents and users in the world do not support this feature. So the answer branches out in negative direction. The same effect will be turn up for other few questions as well.

“Why do they fail to answer to those questions?” Now, this discussion had become to a critical point! As researchers have identified, there is no way other than a third-party solution to answer to these questions in a very pleasant way. Both the sender and recipient must trust these third parties in order to achieve the goals. These third parties bear up the responsibilities of guaranteed delivery, provide statistics about the delivery, and also provide evidence for proof of posting and observing an email. ReadNotify [WebReadNotify] is trademarked as a notary service provider in this manner.

It is hard to find weaknesses associated with ReadNotify. But, more essentially its service is not free. Also they have made little insight to the next generation of email and not prevailed to standardize the approach as a new protocol.

Furthermore Entrust [WebEntrust] and ValiCert [WebValiCert] are some other third-party notarize systems with similar nature. Some of these operate as proxies, while others as pipes. In proxies the email traffic must be sent through the proxy, while in pipes, plug-ins and client software plays a greater role to transfer email between two sites. In both cases, however, the third-party involvement is crucial. They operate as demons.

In digital notarization, a timestamp is used or pasted on an email to enable one to ascertain whether an email was existed at a certain time. This timestamp is not just a time value. Different techniques
are used to construct a timestamp. PGP Digital Timestamping Service [WebPGP] uses a serial number, signature and the date time it was made as a timestamp. But now as researchers and cryptography gurus all over the world are agreed, a different protocol is used to construct a timestamp [WebTimestamp].

1.3 AIMS & OBJECTIVES

It is very hard to predict how successful system can be produced at the end. This is because the inability associated with supporting the development process by existing technology and programming environment. So, some research work has to accompany the way.

The aims and objectives are identified as follows.

- Any kind of digital document, even a message, must be eligible for digital notarization.
- The system must provide legally acceptable proofs or evidence for notarized documents.
- The system must possess adequate security.
- The system must not change the world; only change itself for standstill in changing world.
- The services provided by the system must be worldwide.
- More effort has to be taken in order to standardize the methods and develop a reusable framework, for used in future, or to facilitate the extensions to the system.
- Effectiveness and efficiency are elementary disciplines that must be convoluted throughout the whole development process.

Even though the above objectives are to be met, some scope downing has to be made in order to reach the success in a limited environmental base.

1.4 REPORT OUTLINE

This section gives a brief overview of how this document is organized.

Chapter 1: Introduction plays a major role giving a brief introduction to the whole outline of the document or as short “What this all about?”

Chapter 2: Background, takes you through fundamentals or theoretical backgrounds, where this research project is standing on.

Chapter 3: Specification retake you through aims & objectives, analyzing them, and at the end states what kind of architecture has to be followed and how the work is to be organized.

Chapter 4: Design, gives the logical design of the system. Using UML it constructs a roadmap to the next stage in the development process, coding & implementation.
Chapter 5: Implementation & Testing is where the actual system is constructed. Code samples and user interface design will decorate the discussion.

Chapter 6: Evaluation, talks about itself adding boastings and feeblenesses. And discusses how successfully achieved the aims & objectives and where and why they get failed.

Chapter 7: Future Works, adds another chapter leaving out a discussion, which points how this work is to be continued into the future.

References and Appendices are additional readings in order to complete the discussions and to guide the reader to other resources used in the document.
Chapter 2  BACKGROUND

2.1  ELECTRONIC MAIL, OR EMAIL

2.1.1 How it works?

Typical email system supports five basic functions.

- **Composition** is the process of creating an email message using any email agent program, for example Microsoft Outlook.
- **Transfer** refers to moving messages from the originator to the recipient. When an email gets transferred it will pass through different servers and thus different protocols are used. Final destination to an email is always a mailbox, a folder belongs to the recipient, and it will stay there as long as the recipient collects it.

- **Reporting** tells the originator what happened to the message, is it delivered, rejected or anything else happened.
- **Displaying** incoming messages will involve special viewers to display messages.
- **Disposition** concerns what the recipient does to the message after receiving it. He will delete, forward to someone else or do anything as he wishes.

The next few sections will talk about some standard protocols used in a typical email system.
2.1.2 Email Message

A message is said to have a header (envelope) and the contents (body) as shown in the figure.

![Diagram: An Internet Text Message]

**Figure 2.2: An Internet Text Message**

**Header (Envelope):** Contains information needed to accomplish transmission and delivery of the message, and is terminated by a null line.

**Contents:** Contains objects to be delivered to the recipient.

The syntax is written in the form known as the **augmented Backus-Naur Form** (augmented BNF), which involves naming rules.

E.g.

- Elements separated by slash (/) are alternatives
- Elements enclosed in parentheses are treated as a single element
- The character *** preceding an element indicates repetition
- Square brackets enclose optional elements

The header is a list of lines (fields) of the form:

```
field-name":"" [field-body] CRLF
```

Note: CRLF = ASCII CR (Carriage Return-13) ASCII LF (Line Feed-10)

A field is said to be a **Structured field**, if it is interpreted according to an internal syntax, for examples date and address fields; and the others are called **Un-structured fields**, for examples Subject and Comment fields. (@See Appendix A.1. for a list of Header Fields)

At minimum, a header must contain DATE, FROM, BCC or TO fields, for example:

- **Date**: 26 Aug 76 1429 EDT
- **From**: Jones@Registry.Org
- **To**: Smith@Registry.Org

(@See Appendix B.2. for an example of complex header field)
2.1.3 Email & SMTP

There are three Standard Protocols, which applies to email, and SMTP is one of them. The three protocols are:

- A standard for exchange of mail between two computers (TCP/IP hosts), which is SMTP (RFC 821).
- A standard on the format of the mail message, mail header and the body, which is MAIL (RFC 822 & RFC1049).
- A standard for the routing of mail using the Domain Name System (DNS), which is DNS-MX (RFC 974).

But, the term SMTP is frequently used to refer for this combined set of protocols. SMTP can only transmit 7-bit ASCII data, with the higher order bit is 0. Even though this is adequate for English text messages, it is inadequate for non-English or non-textual data. The approaches to overcome these limitations are:

- Multipurpose Internet Mail Extensions [MIME] (RFC 1521, RFC 1522), which specifies a mechanism for encoding text and binary data as 7-bit ASCII.
- SMTP Service Extensions (RFC 1651, RFC 1652, RFC 1653), which defines a mechanism to extend capabilities of SMTP.

The SMTP mail-exchanging model is shown below:

Figure 2.3: Model of Communication for SMTP

As a result of a user mail request, the sender-SMTP establishes a two-way transmission channel to the receiver-SMTP. This receiver-SMTP may be either the ultimate destination or an intermediate (mail gateway). Then the sender-SMTP will generate commands, which are replied to by the receiver-SMTP.
TCP Connection Establishment  
(Server Port 25)

< - 220 <Server-Domain> Service Ready  
HELO <Sender-Domain> ->  
< - 250 <Server-Domain> OK  
MAIL FROM: <Reverse-Path> ->  
< - 250 <Server-Domain> OK  
RCPT TO: <Forward-Path> ->  
< - 250 <Server-Domain> OK  
DATA ->  
< - 354 Start mail input, end with <CRLF>.<CRLF>  
Line1 ->  
Line2 ->  
... ->  
Last line ->  
<CRLF>.<CRLF> ->  
< - 250 <Server-Domain> OK  
QUIT ->  
< - 221 <Server> Closing Connection

Figure 2.4: Normal SMTP Data Flow

All exchanged commands/replies/data are text lines, delimited by <CRLF>. All replies have numeric code at the beginning of the line.

- Step 1: The sender-SMTP (client) establishes a TCP connection with the receiver-SMTP (server) in the server port 25. The server then sends a 220 Service Ready message or a 421 Service Not Available message.
- Step 2: Sender-SMTP sends a HELO command (or EHLO command if it supports SMTP Service Extensions). The server then sends a 250 OK (or 500 Syntax Error, Command Unrecognized message if it does not support Service Extensions).
- Step 3: Sender-SMTP now sends a MAIL command to initiate the start of a mail transaction. This command contains a reverse-path (mailbox@host domain name and routing hosts), which can be used to report errors. If accepted, the server sends a 250 OK message.
- Step 4: Sender-SMTP sends multiple (if there are more recipients) RCPT TO: <forward-path> commands indicating destinations for the message. Each of them will receive a 250 OK command or a 550 No Such User Here command if destination is unknown to the server.
- Step 5: Sender-SMTP issues a DATA command to notify the receiver that the message contents are following. The server replies with 354 Start Mail Input, End With <CRLF>.<CRLF> command.
- Step 6: Sender-SMTP now sends the data line by line and ends with the <CRLF>.<CRLF>. The server replies with 250 OK command or an appropriate error message.
- Step 7: Now if Sender-SMTP has more messages to send, starting from the Step 3 it will sends a new MAIL message, or if ready to receive messages, by issuing a TURN command switch between their roles, or if no messages to send, terminate the connection with a QUIT command.

(@See Appendix B.3. for a real world example)
2.1.4 Email & MIME

MIME is a standard mechanism, which solves problems aroused in SMTP in a manner that is highly compatible with existing RFC 822 standards.

Some limitations in SMTP:
- SMTP cannot transmit executable files or other binary objects
- SMTP cannot transmit text data which includes national language characters
- SMTP servers may reject messages over a certain size
- SMTP gateways cause translation problems when mapping between character sets
- Some SMTP implementation and transport agents do not adhere completely to the SMTP standard

MIME assumes the worst 7-bit ASCII transport, and describes:
- Protocol for including objects other than US ASCII text within the body (RFC 1521)
- Protocol for encoding non-US ASCII text in the header fields (RFC 1522)

A MIME-compliant message must contain a header field:
MIME-Version: 1.0

(@See Appendix A.3. for a five header fields defined for MIME)

**Content-Type field**

This field describes the body of the message, and has the form:
Content-Type: type/subtype; parameter=value; parameter=value

(@See Appendix A.4. for seven standard Content-Types)

**Content-Transfer-Encoding Field**

(@See Appendix A.5. for five encoding mechanisms)
(@See Appendix B.1. for a complex multipart message example)

2.1.5 Email & POP

POP gives a protocol to copy emails from the ISP to the user. Now POP is in version 3 (POP3), which is described in RFC 1939.

When a user starts to read the emails by establishing a TCP connection with the ISP at port 110, POP3 goes through three states.
- **Authorization** state deals with having the user log in.
- **Transaction** state deals with the user collecting the emails and marking their flags
- **Update** state cause flags to be actually applied to emails. E.g. Delete
2.1.6 Web Mail

Web mail gives ability to send and display email messages throughout a website. Therefore, without
having an email agent program or separate mailbox on ISP, one can send or receive emails (but there
are other options as well) using this facility. After having registered to a web mail account, anyone can
login to his account by sending the username and the password to the server. Now, he or she is
allowed to receive the web mail services. Yahoo and Hotmail are examples for web mail service
providers.

2.2 DIGITAL TIME

The time is most crucial for modern electronic transactions. Just relying on the computer’s system
time or network time is not the solution for the problem. The time must accurate and must be obtained
from a reliable time source. The timeservers do answer to this issue in a broad manner.

2.2.1 Time Server

A Time Server is a server, which provides only the time information using a specific protocol (@See
Appendix A.2. for a list of standard Time Servers).

2.2.2 Time Protocols

The following are some standard protocols used by Time Servers:

- **DAYTIME Protocol** – Time Server is listening to port 13 and sends an ASCII string
  containing time information.

- **TIME Protocol** – Time Server is listening to port 37 and sends a 32-bit binary number
  (seconds since 1900-01-01 00:00.00 UTC)

- **Network Time Protocol (NTP)** – This is widely used for computer clock synchronization.
  NTP is capable of achieving synchronization to within microseconds. Its synchronization is
  part of a software package that includes a full suite of NTP options and algorithms, which are
  relatively complex, real-time applications. In NTP the Time Server is listening to port 123 and
  sends a 64-bit unsigned fixed-point number (seconds since 1900-01-01 00:00.00 UTC).
  Precision is about 200 Pico-seconds.

- **Simple Network Time Protocol (SNTP)** – SNTP provides simplified access strategy that
does not require the degree of accuracy that NTP provides. The main difference between the
two protocols is that SNTP does not have the error management and complex filtering
systems that NTP provides. In SNTP Time Server is listening to port 123 and the formats are
compatible to NTP protocol specification.

- **Windows Time Service** - Windows 2000 implements Kerberos V5 authentication protocol
  defined by RFC 1510. W32Time achieves the Kerberos requirements by loosely
  synchronizing clock values across the network. W32Time uses coordinated universal time
  (UTC). To start the service Windows 2000 from the services list (Administrative Tools > Services) select Windows
  Time and Start the service, or run **net start w32time** to start the service and run **net stop w32time** to stop the service.
To synchronize the computer’s clock with a Time Server run `net time /domain[:domainname] /set`, or `net time \computername /set`. (@See net time command list for more details).

### 2.2.3 Time Stamping

Time Stamping is a set of techniques using cryptography algorithms, enabling one to ascertain whether an electronic document was created or signed at (or before) a certain time. In practice, most of the time stamping systems use a trusted third party called Time Stamping Authority (TSA). A time stamping is a digital attestation of the TSA that an identified electronic document has been presented to the TSA at a certain time.

To time stamp an electronic document researchers and cryptographers agreed to use the following protocol:

- Calculate a digest of the document locally on the computer.
- Send this digest (securely) to the TSA. The TSA then appends date and time to the digest; signs this compound document with its private key, and send it back to the client.
- Now, the client verifies the signature to ensure that the TSA actually did it, and compare the digest to ensure that the TSA correctly received the digest, and check whether the correct time was included.

This seems to be a much feasible and practical mechanism, since the original document never quits and it is possible to check whether everything happened correctly. Some time stamping services are PGP [WebPGP] and CertSigner [WebCertSigner].

### 2.3 Secure Data Transmission

#### 2.3.1 Security Threats

When information flows from one computer (source) to another computer (destination) through an open network, there are several security threats or attacks to be considered.

**Normal Flow**

- **Interruption**: Attack on availability
- **Interception**: Attack on confidentiality
- **Modification**: Attack on integrity
- **Fabrication**: Attack on Authenticity
2.3.2 Confidentiality by Data Encryption

This is ensuring that the information in a computer system and transmitted information are accessible only for reading by authorized parties (protecting from eavesdropping). This threat is not easy to detect, but it is feasible to prevent by the use of cryptographic techniques, such as encryption of data. Encryption ensures that the information to be transmitted is only meaningful to only authorized parties.

![Diagram of Confidentiality by Encryption]

**Figure 2.6: Confidentiality by Encryption**

2.3.3 Authenticity by Digital Signature

This is ensuring that the end entities are authentic (each is the entity that it claims to be and protecting against information being impersonation). In here also the solution resides with the cryptographic techniques, specifically with digital signature. Digital signature ensures that two persons in the world do not have the same signature and has the following properties.

- It provides verification of the author and the date and time of the signature
- It is possible to authenticate the contents at the time of the signature
- The signature is verifiable by third parties

![Diagram of Authenticity by Digital Signature]

**Figure 2.7: Authenticity by Digital Signature**

2.3.4 Integrity by Digital Fingerprint

This is ensuring that the information in transit is not changed or replaced (protecting against tampering) and only the authorized parties have that ability. Digital fingerprint coming from the
cryptography and answers to this threat by providing unique identifier to the data, in the sense that, two different pieces of data can newer create the same digital fingerprint, and no portion of the original data can be reconstructed from the fingerprint.

2.3.5 Non-Repudiation by Digital Certificates

This deals with the issue that neither the sender nor the receiver of a message be able to deny the transmission of information. This is somewhat messy issue to be deal with, because it is totally depends on a trust mechanism and how much trust is there. In most cases a trusted third party’s involvement is great. What they do is issuing certificates indicating that the actual transaction is happened, with the evidence, and then time stamped and signed. So, no party can deny the transaction and every party gets legal coverage (this is still in doubt, because of multi-jurisdictions). There are standard certificate formats, and X.509 is one of them.

2.4 Digital Notarization

2.4.1 What is Notarization?

A complete answer to this question can be found from the next chapter (@see Specification).
Chapter 3 SPECIFICATION

3.1 PROBLEM DESCRIPTION

Email is very useful for digital communication. Public Key Cryptography solves security threats coming on to its direction. However, issues related to repudiation, such as when an email is created, who created it, when it was sent, was it delivered to the intended recipient, was it observed by the recipient and in a case do anyone has evidence to prove such things, are under questioning.

Consider the following example:

Suppose Alice discovered something new and she wants to publish her discovery. Alice has number of options to do this. Her option was to send an email to someone she believes to trust indicating her discovery. So she sends an email to some authority. Somehow Bob get to know about Alice’s discovery. Now Bob is also sending an email indicating that the discovery is belongs to him. So how does our notarization authority solve this problem?

- A Notarization Authority will Timestamp your email. Therefore no one can backdate the existence of an email. In our example if Alice’s email was time stamped, she wins the race.
- A Notarization Authority issues proof-of-posting certificate to your email. Therefore you have legally acceptable evidence to prove that you actually posted the email. In our example if some party refuses that Alice posted the email, the Alice has this certificate to prove that she actually did.
- A Notarization Authority will deliver your email to the intended recipient, and once delivered it will inform you by issuing a digital receipt. So you know exactly what is happened to your email after you send. If Alice received a receipt, then she gets to know that her email was delivered correctly.
- A Notarization Authority issues proof-of-opening certificate to your email. This certificate will prove that your email was opened by the intended recipient and when. So if Alice obtains this certificate she has a proof that her email was read by the authority.

3.2 FEATURES OF THE SYSTEM

Apart from the features discussed under the previous chapter, the following features are also incorporated into the system.

- Service interface through a website
- No plug-ins or additional software is needed by clients
- Separate system for administrative purposes
- Secure the system such a way that no unauthorized accesses are made for the administrative system and no one is able to steal the private key of the authority
Notarization Authority for Emails
H.E.M.H.B. Ekanayake

- Concern about ethical issues
- Make the service interfaces much more user friendly and attractive
- Make the system reusable and well documented

3.3 PROBLEM DEFINITION

The following Use Case diagram represents the relationships among identified actors and the use cases.

Figure 3.1: Use Case Diagram for the System
3.4 THE DOMAIN ARCHITECTURE OF THE SYSTEM

The following figure represents the domain architecture of the system.

There are at least 3 external entities incorporating to the domain of this system.

- POP Server – Holds the Inbox
- Time Server – Provides exact date and time to the system
- SMTP Server – Send emails

This domain architecture suggesting the system to be developed in subsystem basis.
3.5 **Typical Interaction Diagram**

The following high-level interaction diagram shows typical interactions between the entities (@see the example discussed at the beginning of this chapter).

![Diagram of interaction between entities](image)

*Figure 3.3: A High-Level Interaction Diagram of the System*

3.6 **Software Development**

This project belongs to research type, and thus the preferred development model for such a situation is the *Evolutionary Prototype model*. This model is used when the requirements are *unclear* and *unstable*.

![Evolutionary Prototype Development Model](image)

*Figure 3.4: Evolutionary Prototype Software Development Model*

3.7 **Initial Work Schedule**

The initial work plan was scheduled as given below:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Months</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment Analysis</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Design</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Coding</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Testing</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Implementation</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 3.5: Initial Work Schedule*
Chapter 4  DESIGN

4.1 ASSUMPTIONS & REQUIREMENTS

It is assumed that the followings requirements and assumptions are met.

- All emails are plain/text
- Client terminals are Microsoft Internet Explorer 4.0+
- Client email viewers support inline frames (layers)

In addition, specific assumptions and requirements are made during implementation as arisen.

4.2 SOFTWARE DESIGN

The above specification discussed how the system to be organized and required functionalities of the system. This description will continue the discussion with design aspects.

The following figure shows a high-level view of the system.

![High-level View of the System](image)

The above figure shows how the system is divided into two sub-systems:

- The Web Server sub-system, and
- The Notary Server (Administrative) sub-system.

The database is acting as a common gateway to those sub-systems. The notary server plays the chief role of the system, and it is suppose to do the following tasks at contrast:

- Handling Email (both sending and retrieving)
- Timestamping and issuing proof-of certificates
- Responding to member registrations
• Message handling (constructing and extraction)

In addition there are some other tasks carried out by this sub-system. (@see Implementation)

The notary server is kept isolated from the world, so there is nothing to worry about the security issues. Thus it is assumed that only one authorized person handling this system, and no logins are provided. The functionality of the server is illustrated in the use-case diagram and the state chart diagram gives how it function. The following figure shows the components of this sub-system.

![Figure 4.2: Components of the Notary Server](image)

The second sub-system is the **web server**, which is freely accessible by anyone through the Internet using a web browser. But only the members can obtain the optimum service provided by the system. Anyone can register to be a member. This web system provides following services at contrast:

• Web mail facility with email templates
• Displays the status of the sent emails
• Issue proof-of-certificate for sending and opening of emails
• Certificate validation facility

The functionality of the server is illustrated under the use-case diagram and the sequence diagrams show interactions. The following figure shows the components of this sub-system.
4.3 DATABASE DESIGN

This system has only little number of relations, only for store information about members. Therefore its design was omitted!

4.4 TECHNOLOGICAL BACKGROUND

The following figure illustrates the programming environment of the system. Note that java is used as the primary programming language.

Figure 4.3: Components of the Web Server

Figure 4.4: Technologies Used by the System
4.5 **OVERALL CLASS DIAGRAM**

The following is the class diagram of the system. **Note** that some classes are not represented in the diagram and the classes MAY sub-classed while implementation.

![Class Diagram of the System](image)

*Figure 4.5: Class Diagram of the System*
The following table listing will give the purpose of each class.

<table>
<thead>
<tr>
<th>Class</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebInterface</td>
<td>Interact with the clients by displaying messages, and extracting responses by generating web forms</td>
</tr>
<tr>
<td>Synchronization</td>
<td>Synchronizes the system clock with a standard time server, and provides information such as when the last synchronization happened.</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Timestamp a data object that is presented to it. Based on synchronization information it decides how accurate the date time and after taking all these statistics into account the final time stamped object will be issued.</td>
</tr>
<tr>
<td>Certificate</td>
<td>Generates certificates. By obtaining required information about the subject entities using other classes, it first construct the certificate, then sign it and add a time stamp.</td>
</tr>
<tr>
<td>DigitalSignature</td>
<td>Signs a data object using the private key.</td>
</tr>
<tr>
<td>Fingerprint</td>
<td>Calculate a hash for a data object.</td>
</tr>
<tr>
<td>DateTime</td>
<td>Provides date time information.</td>
</tr>
<tr>
<td>Register</td>
<td>Register clients as members.</td>
</tr>
<tr>
<td>ActivationKey</td>
<td>Random number generator.</td>
</tr>
<tr>
<td>Member</td>
<td>Contain member information.</td>
</tr>
<tr>
<td>Message</td>
<td>Contain message information.</td>
</tr>
<tr>
<td>Recipient</td>
<td>Contain mail recipients or observers information.</td>
</tr>
<tr>
<td>MailOptions</td>
<td>Contains mail options set by each member.</td>
</tr>
<tr>
<td>SendMail</td>
<td>Send messages to a SMTP server as emails.</td>
</tr>
<tr>
<td>ServerManager</td>
<td>This is the actual notary server class. Which runs and do all the server functionalities.</td>
</tr>
<tr>
<td>Receipt</td>
<td>Generates receipts (acknowledgements) on request.</td>
</tr>
<tr>
<td>RetriveMail</td>
<td>Retrieve emails from a POP server and present as messages.</td>
</tr>
<tr>
<td>DBConfig</td>
<td>Contains database configuration for share by other classes.</td>
</tr>
<tr>
<td>MailConfig</td>
<td>Contains mail server configurations for share by other classes.</td>
</tr>
</tbody>
</table>

*Table 4.6: Classes and their Purposes*
4.6 **SEQUENCE DIAGRAMS**

This section gives some important sequence diagrams of the system under several topics.

### 4.6.1 Sequence Diagram: Sign-Up (Register)

The following figure illustrates the sequence diagram of the signup process.

![Sequence Diagram of the Sign-Up Process](image-url)

*Figure 4.7: Sequence Diagram of the Sign-Up Process*

Here the applicant is registered as a member after successfully completing these steps. Note that the involvement of the notary server is not considered here. Notary server is responsible of sending the activation key to the applicant by an email. On this process the requester is asked for a user name and a password for his account, and also a primary agreement with the system is also included.
4.6.2 Sequence Diagram: Email Delivery

The following figure illustrates the sequence diagram of the email observing process.

![Sequence Diagram of the Mail Observing Process](image)

*Figure 4.8: Sequence Diagram of the Mail Observing Process*

It is assumed that the recipient had received the email earlier and the body is included or not in the email, and/or an acknowledgement is required to confirm that the email is correctly received. The first few steps of the diagram are for an email without the body, and so the recipient is requesting the body. Then the body is sent to the requestee after correct identification. The rest of the steps are for confirmation purposes. Here the recipient of the email has to information to the system that he received the email correctly and completely. It may be an optional process depending on the type of the email.
4.6.3 Sequence Diagram: Certificate Issuing

The following figure illustrates the sequence diagram of the certificate issuing process.

*Figure 4.9: Sequence Diagram of the Certificate Obtaining Process*

As the figure shows, this process is somewhat complicated process. In brief it is as follows.

First, a member (or some other party) requests a proof of certificate for a sent email message. This request straightaway goes to the Certificate object as indicated in the diagram and this object is responsible of complete all the steps required to issue the certificate. First thing it has do is collecting all necessary information including the message, the sender and recipient, which is to be included in the certificate. Then it construct the certificate, present is to the Fingerprint object to obtain a digest of the certificate. This digest is then presented to both Signature and Timestamp objects to sign and timestamp the certificate. This information also included in the certificate before issuing it to the requester.
4.7 Activity Diagrams

4.7.1 State Diagram: Notarization Authority Server

The start chart diagram of the notary server is given below.

![State Diagram of the Notarization Server](image)

*Figure 4.10: State Diagram of the Notarization Server*
Chapter 5  IMPLEMENTATION & TESTING

5.1  SOFTWARE DEVELOPMENT

This implementation process carried out in three aspects:

- Development of the Framework
- Development of the Web based System
- Development of the Notarization Authority Administrative Server

The next three sections deals with the implementation of the above three aspects. The following are tools that were used throughout the implementation process.

- JBuilder 7.0 – All jsp and java files were coded and tested using this tool.
- Macromedia Dreamweaver MX – The website was built using this tool.
- Macromedia Fireworks MX – All image files were designed and created using this tool.
- Microsoft Visual InterDev 6.0 – All client side scripts (JavaScript) were coded and tested using this tool.

The user interfaces are demonstrated under the user manual attached at the end of this document (@see Appendix D). Some codes are listed as the discussion goes on and the code of the framework is listed under separate section (@see Appendix C). Also note that some code that have minor importance have omitted from the discussion. The last section demonstrates how the testing was applied to the product.

5.2  FRAMEWORK IMPLEMENTATION

The Utility Class

This class provides methods that have very frequent use.

```java
/*
 * This method returns a randomized serial number in the form
 * of a string.
 * E.g. getSerial("ABC") ==> "ABC10531597126840.4953822Adm64"
 * @param prefix adds to the beginning of the serial number.
 * @return the serial number
 */
public String getSerial(String prefix)

/**
 * Returns the string between two strings.
 * E.g. getEnclosed("<", ",", ",Between>") ==> "Between"
 * @param pre string
 * @param post string
 * @param in input string
 * @return string between
 */
```
public String getEnclosed(String pre, String post, String in)
/**
 * Concatenates two byte arrays and returns the resulting
 * byte array.
 * @param arrayA first byte array
 * @param arrayB second byte array
 * @return concatenated byte array
 */
public byte[] concatArray(byte[] arrayA, byte[] arrayB)
/**
 * Converts a byte array to the corresponding hex
 * representation and returns the resulting buffered string
 * @param data input byte array
 * @return hex buffer
 */
public StringBuffer toHex(byte[] data)
/**
 * Converts a byte array into corresponding character
 * representation and returns the resulting buffered string
 * @param data
 * @return character buffer
 */
public StringBuffer toString(byte[] data)
/**
 * Converts a string containing a number into the
 * corresponding integer value, or returns the
 * default value if failed
 * E.g. toInt("5", 2) ==> 5
 * E.g. toInt("F", 2) ==> 2
 * @param number string number
 * @param defaultvalue default integer value
 * @return integer value
 */
public int toInt(String number, int defaultvalue)

The SecUtility Class

This class provides methods that have frequent security functionality.
/**
 * Calculate a digest for the given data.
 * Algorithm used is MD5
 * @param data input data as a byte array
 * @return the digest as a byte array
 */
public byte[] digest(byte[] data)
/**
 * Digitally signs a given data using a given private
 * key. Algorithm used is DSA.
 */
The DateUtility Class

This class provides methods that have frequent date & time functionality.

```java
/**
 * @return current date as a string
 */
public String getDate()

/**
 * @param d Date object
 * @return date as a string
 */
public String getDate(Date d)

/**
 * @param UTC time
 * @return date as a string
 */
public String getDate(long UTC)

/**
 * @param year value
 * @param month value
 * @param date value
 * @return date as a string
 */
public String getDate(int year, int month, int date)

/**
 * @return current time as a string
 */
public String getTime()

/**
 * @param d Date object
 * @return time as a string
 */
public String getTime(Date d)

/**
 * @param UTC time
 */
```
* @return time as a string
*/
public String getTime(long UTC)

/**
 * @return current data and time as a string
 */
public String getDateTime()

/**
 * @param d Date object
 * @return current date and time as string
 */
public String getDateTime(Date d)

/**
 * @param UTC time
 * @return current date and time as string
 */
public String getDateTime(long UTC)

/**
 * @return current data and time in UTC
 */
public long getUTC()

/**
 * @param year value
 * @param month value
 * @param date value
 * @return current data and time in UTC
 */
public long getUTC(int year, int month, int date)

/**
 * @return current year
 */
public int getYEAR()

/**
 * @param date Date object
 * @return year
 */
public int getYEAR(Date date)

/**
 * @return current day of the month
 */
public int getDAY_OF_MONTH()

/**
 * @param date Date object
 * @return day of the month
 */
public int getDAY_OF_MONTH(Date date)

/**
 * @return current month
 */
public int getMONTH()
The ProcessQuery Class

This class provides methods that have query processing functionality.

```java
/**
 * Execute a SQL query and returns the resulting resultset
 * E.g. getResults("SELECT * FROM MailStore")
 * @param query SQL query string
 * @return resultset or null
 */
public ResultSet getResults(String query)

/**
 * Just execute a SQL query
 * E.g. justUpdate("INSERT INTO ...")
 * E.g. justUpdate("UPDATE MailStore SET ...")
 * @param query SQL query string
 * @return 1 is successful; -1 otherwise
 */
public int justUpdate(String query)

/**
 * Check whether the SQL query executed correctly.
 * Used to check the existence of an item in a db
 * E.g. isExist("SELECT mailid FROM ... WHERE ...")
 * @param query SQL query string
 * @return true if exist; false otherwise
 */
public boolean isExist(String query)

/**
 * Execute a SQL query and returns the string contained
 * in the specified field.
 * E.g. getString("SELECT mailid FROM ...", "mailid"
 * @param query SQL query string
 * @return string
 */
public String getString(String query, String field)
```

/*
 * @param date Date object
 * @return month
 */
public int getMONTH(Date date)

/*
 * @return hour of the day
 */
public int getHOUR_OF_DAY()

/*
 * @return minute
 */
public int getMINUTE()

/*
 * @return second
 */
public int getSECOND()
public String getSchema(String query, String field)

/**
 * Execute a SQL query and returns the boolean value
 * contained in the specified field.
 * E.g. getBoolean("SELECT ismale FROM ...", "ismale")
 *
 * @param query SQL query string
 * @return true or false depending on the boolean value
 * associated with the field; default is false
 *
 */
public boolean getBoolean(String query, String field)

---

This class encapsulates a commands governed message and provides methods which extract the
parameters embedded onto a message.

/**
 * Constructor of an object
 * E.g. new ExtractFeature("X-", ",", ";")
 * will parse all commands of the form
 * "X-ParameterName=ParameterValue;"
 *
 * @param start string
 * @param delimit string
 * @param end string
 *
 */
public ExtractFeature(String start, String delimit, String end)

/**
 * Used to bind a content to an object and it will
 * automatically parse the content and loads the
 * parameter values to map
 * E.g. A (Parameter Name, Parameter Value) Map,
 * (MailTo, someone@somewhere), (Priority, 5), ...
 *
 * @param content the content
 *
 */
public void setContent(String content)

/**
 * Returns the features as requested
 * E.g. getFeature("MailTo") ==> "someone@somewhere"
 *
 * @param option parameter name
 * @return parameter value
 *
 */
public String getFeature(String option)

/**
 * Returns the rest of the body
 * E.g. if the content is "X-MTo=you;This is Body"
 * then getBody() ==> "This is Body"
 *
The TimeStamp Class

This class encapsulates a timestamp and provides methods, which act on those fields.

The timestamp encapsulates following data and these data has to be specified in the constructor.

```java
private byte[] messageDigest;
private String issuerUniqueID;
private String timeServerUniqueID;
private String PKCertificateSerialNo;
private String timePrecision;
private long UTCTime;
private byte[] timeStamp;
```

public TimeStamp(byte[] messageDigest,
                  String issuerUniqueID,
                  String timeServerUniqueID,
                  String PKCertificateSerialNo,
                  String timePrecision,
                  long UTCTime,
                  byte[] timeStamp)

The get methods add functionality to retrieve values from those fields.

E.g.

```java
/**
 * Returns the timestamp as a byte array
 * @return timestamp
 */
public byte[] getTimeStamp() {
    return timeStamp;
}
```

The following methods add additional functionality to the class.

```java
/**
 * Returns a readable string containing the timestamp
 * and other information associated with it
 * @return timestamp as a string
 */
public String toString()

/**
 * Adds together all the data contained in a timestamp
 * to form a single representative byte array
 * @return a byte array
 */
public byte[] getAll()
```

The TimeStampener Class
This class provides methods, which to define and construct a timestamp.

As the first step the following fields have to be specified using set methods provided by the class.

```java
private String issuerUniqueID;
private String timeServerUniqueID;
private String PKCertificateSerialNo;
private String timePrecision;
```

The following methods provide required functionality.

```java
/**
 * Timestamp a data object and returns a Timestamp object
 * @param data data object to be timestamped
 * @param useDigest whether a digest of the data object
 * need to be calculated or not
 * @param date date object embeded into the timestamp
 * @param prv Private key
 * @return Timestamp
 */
public TimeStamp timeStamp(byte[] data,
                           boolean useDigest,
                           Date date,
                           PrivateKey prv)

/**
 * Verify a Timestamp
 * @param timestamp Timestamp object
 * @param pub Public key
 * @return true if verified; false otherwise.
 */
public boolean verifyTimeStamp(TimeStamp timestamp,
                                PublicKey pub)
```

The DN Class

This class encapsulates X.500 distinguish names, and provide methods which act on those names.

The DN class encapsulates following data and these data has to be specified in the constructor.

```java
private String CN; // Common Name
private String OU; // Organization Unit
private String o; // Organization
private String l; // Locality
private String s; // State
private String c; // Country
private String e; // Email

public DN(String CN,
           String OU,
           String o,
           String l,
           String s,
           String c,
           String e)
```
The *get methods* add functionality to retrieve values from those fields. The following methods add additional functionalities to the class.

```java
/**
 * Returns a readable string containing distinguish names
 * @return distinguish names as a string
 */
public String toString()

/**
 * Adds together all the data contained the object
 * to form a single representative byte array
 * @return a byte array
 */
public byte[] getAll()
```

## The MessageHeader Class

This class encapsulates a message header, and provides methods, which act on those fields. The `MessageHeader` class encapsulates following data and these data has to be specified in the constructor.

```java
private String messageID;
private String from;
private String to;
private String cc;
private String bcc;
private String subject;
private String date;

public MessageHeader(String messageID, String from, String to, String cc, String bcc, String subject, String date)
```

The *get methods* add functionality to retrieve values from those fields. The following methods add additional functionalities to the class.

```java
/**
 * Returns a readable string containing message headers
 * @return the message header as a string
 */
public String toString()

/**
 * Adds together all the data contained in the object
 * to form a single representative byte array
 * @return a byte array
 */
public byte[] getAll()
```
The MessageCertificate Class

This class encapsulates a proof of time message certificate, which is based on X.509 public key certificate standard, and provides methods, which act on those data.

The MessageCertificate class encapsulates following data and these data has to be specified in the constructor.

```java
private int version;
private String serialNumber;
private String purpose;
private String signatureAlgorithm;
private DN issuer;
private DN subject;
private long notAfter;
private long notBefore;
private TimeStamp x_Timestamp;
private MessageHeader x_MessageHeader;
private byte[] x_MessageBody;
private String x_MessageStatus;
private byte[] signature;

public MessageCertificate(int version,
    String serialNumber,
    String purpose,
    String signatureAlgorithm,
    DN issuer, DN subject,
    long notAfter,
    long notBefore,
    TimeStamp timestamp,
    MessageHeader messageHeader,
    byte[] messageBody,
    String messageStatus)
```

The signature must be specified using a separate set method.

```java
/**
 * Binds the signature to the certificate
 * @param signature the signature bytes
 */
public void setSignature(byte[] signature)
```

The `get methods` add functionality to retrieve values from those fields. The following methods add additional functionalities to the class.

```java
/**
 * Returns a readable string containing information
 * associated with the certificate
 * @return certificate information as a string
 */
public String toString()

/**
 * This just adds together all the data contained in the
 * certificate, except the signature. Therefore using this
```
* verifier and the signature, one can verify the certificate
* @return a byte array
*/
public byte[] getVerifier()

/**
* Adds together all the data contained in the object
* to form a single representative byte array
* @return a byte array
*/
public byte[] getAll()

---

**The MessageCertificator Class**

This class provides methods which to define and construct a proof of time message certificate.

/**
* Construct and issues a proof of time message certificate
* @param purpose proof-of-posting or any other purpose
* @param issuer who issues the certificate
* @param subject to whom this certificate is issued to
* @param timestamp timestamp relevant to the purpose
* @param messageheader message header
* @param messagebody hash value of the message body
* @param useDigest hash value is to be calculated or not
* @param messagestatus the current status of the message
* @param prv private key
* @return the resulting message certificate
*/
public MessageCertificate issue(String purpose,
        DN issuer,
        DN subject,
        TimeStamp timestamp,
        MessageHeader messageheader,
        byte[] messagebody,
        boolean useDigest,
        String messagestatus,
        PrivateKey prv)

/**
* Verify a given message certificate
* @param ms message certificate which has to be verified
* @param pub public key
* @return true if verified; false otherwise
*/
public boolean verify(MessageCertificate ms, PublicKey pub)
5.3 **WEBSITE IMPLEMENTATION**

5.3.1 **Registration (Signing Up)**

**Functional Arrangement**

![Functional Arrangement Diagram](image)

*Figure 5.1: Functional Arrangement of the Sign Up Process*

**Signup.htm**

Contains two separate HTML forms, one for submit the email address (step-1) who will then receive an activation key via email, and the second one for submit that key (step-3) and to continue the registration process further.

**Signup1.jsp**

Feed data collected using the HTML form to the associated bean (signupb) and displays a message depending on the status output by the bean.

Data feeding script:

```jsp
<jsp:useBean id="signupbId" scope="session" class="jwebproject.signupb" />
<jsp:setProperty name="signupbId" property="*" />
```

**Signup1b Bean**
Notarization Authority for Emails  
H.E.M.H.B. Ekanayake

Provides methods for used by signup step-1.

```java
/**
 * Binds as email address to the bean
 * @param email email address
 */
public void setEmail(String email)

/**
 * Check out the bound email address is registered before;
 * if not, generate an activation key and store this
 * pair as a new pending registration request in the db
 * @return true if this process succeeded; false otherwise.
 */
public boolean doProcess()
```

### Signup2.jsp

Feeds data collected using the HTML form to the associated bean, and depending on the status returned by the bean, this page would initiate a session to continue the registration process further by redirecting the responder to signup step-3, or display a relevant message.

```java
<%=
/**
 * If this signup step gets succeeded, then initiate a session
 * and redirect the responder to the next step
 */
if (signup2bId.doProcess()) {
    HttpSession s = request.getSession(true);
    String email = signup2bId.getEmail();
    s.setAttribute("email", email);
    response.sendRedirect("signup3.jsp");
}
%
```

### Signup2b Bean

Provides methods for used by signup step-2.

```java
/**
 * Binds email address to this bean
 * @param email email
 */
public void setEmail(String email)

/**
 * Binds an activation key to this bean
 * @param activationkey activation key
 */
```
public void setActivationkey(String activationkey)
/**
 * Check that a user exist with the corresponding
 * email & activation key, If exist then
 * upgrade the user into the registered state
 * @return true if this process succeeded; false otherwise.
 */
public boolean doProcess()

---

### Signup3.jsp

This page collects information about the applicant; note that it contains an access control script to prohibit unauthorized access.

Other Script . . .<%
/**
 * Gets the existing session, donot create a new if not exist
 * Checkout that the requester came from completing signup step-2
 */
HttpSession s = request.getSession(false);
String email = (String)s.getAttribute("email");
if (email == null)
    response.sendRedirect("signup.htm");
%>
<jsp:useBean id="registrationbId" scope="session"
class="jwebproject.registrationb" />
<jsp:setProperty name="registrationbId" property="*" />
Other Script . . .

---

### Registration.jsp

This is the last page when you see from the registration process. This page also influenced by an access control script. And display a message depending on the success of the last step in signup process. If that registration process ended successfully, a script is added to terminate the session.

Other Script . . .<%
/**
 * Access control script
 */
HttpSession s = request.getSession(false);
String email = (String)s.getAttribute("email");
if (email == null)
    response.sendRedirect("signup.htm");
%>
<jsp:useBean id="registrationbId" scope="session"
class="jwebproject.registrationb" />
<jsp:setProperty name="registrationbId" property="*" />
%<
/**
 * Execute the logic
 */
registrationbId.setEmail(email);
int STATUS = registrationbId.doProcess();
if (STATUS == 0) {
    s.removeAttribute("email");
    s = null;
}
Other Script . . .

### Registrationb Bean

Provides methods and other data for used by signup step-3.

The field variables used:

```java
private String username;
private String password;
private String repassword;
private String cn;
private String ou;
private String o;
private String l;
private String s;
private String c;
private String email;
private boolean agree;
```

The set and get methods:

> Omitted

The flags used:

```java
public static final int UNSPECIFIED = 1;
public static final int AGREEE = 2;
public static final int USERNAME = 4;
public static final int PASSWORD = 8;
public static final int CN = 16;
public static final int OU = 32;
public static final int O = 64;
public static final int L = 128;
public static final int S = 256;
public static final int C = 512;
```

The methods used:

```java
/**
 * First validate all the data came up to the bean
 * If any invalid value found, then set the corresponding
 * error flags. If no such error found then do
 * the following tasks.
 * <ol>
 * <li>Insert this applicant as a new registered member
 * </li>
 * <li>Remove any temporary sessions associated for this applicant
 * </li>
 * 
 * @return 0 if ok; any flag value on field error; -1 on other error
 */
public int doProcess()
```
5.3.2 Signing In

**Functional Arrangement**

![Diagram of Functional Arrangement](image)

*Figure 5.2: Functional Arrangement of the Sign In Process*

Index.htm, sighup.htm, ...

In any of these pages there is a HTML form to submit the username and the password.

**Signin.jsp**

Do the signing in or display a relevant message.

```jsp
<jsp:useBean id="signinbId" scope="session" class="jwebproject.signinb" />
<% signinbId.doInit(); %>
<jsp:setProperty name="signinbId" property="*" />
<%
/**
 * For a valid user initiates a session and redirect
 */
if (signinbId.doProcess()) {
    HttpSession s = request.getSession(true);
    String email = signinbId.getEmail();
    s.setAttribute("email",email);
    response.sendRedirect("outbox.jsp");
}
%
```

**Signinb Bean**

Provides methods and other data for sign in process.

The field variables and set and get methods:

Omitted

The methods:

```jsp
/**
 * Initialization and Cleanup
 */
```
public void doInit()
/**
 * @return true if specified user exist; false otherwise.
 */
public boolean doProcess()

5.3.3 Viewing the Statistics of an Email

This page lists some important up to date statistics of emails sent, those including, mailid, to, date, subject and status fields.

Other Script . . .
<jsp:useBean id="outboxbId" scope="session" class="jwebproject.outboxb" />
<%
/**
 * Access Control
 */
HttpSession s = request.getSession(false);
String email = (String)s.getAttribute("email");
if (email == null)
    response.sendRedirect("signup.htm");
<%
/**
 * Initialization
 */
outboxbId.setEmail(email);
outboxbId.doInit();
<%
/**
 * Constructing the UI by displaying email data
 */
<%
while (outboxbId.getNext()) { %>
    <tr bgcolor="#FFFFEC">
        <td><a href="compose.jsp?mailid=<%=outboxbId.getMailid()%>""><%=outboxbId.getMailid(10)%></a></td>
        <td><%=outboxbId.getDate()%></td>
        <td><%=outboxbId.getMailto()%></td>
        <td><%=outboxbId.getMailsubject(25)%></td>
        <td><%=outboxbId.getStatus()%></td>
    </tr>
<% } %>

Other Script . . .

---

**Outboxb Bean**

Provides methods and other data for retrieving email information.

The field variables, and set and get methods:

Omitted

The methods:

```java
/**
 * Initialize the bean by loading
 * email information from the database
 */
public void doInit()

/**
 * Iterate through the records one by one
 * @return true if another record is
 * successfully loaded; false otherwise.
 */
public boolean getNext()
```

---

**Mailstat.jsp**

Pending!

---

**Mailstatb Bean**

Pending!
5.3.4 Sending and Modifying an Email

**Outbox.jsp**

To modify an email, from the emails listed under the outbox, the one wish to be modified is to be selected, which will then take the responder to the composing page.

**Compose.jsp**

This page contains a HTML form to modify or fill-up an email. If the email is to be modified, the user expected to came from selecting the mailid; otherwise it will be considered as a new email message. This HTML form will be automatically filled-up by the values came from the associated bean (composeb). Also, a client-side JavaScript is responsible for enabling/disabling fields according to user requirements. This page also contains an access control script.

```java
<%  
/**  
 * Access Control  
 */
HttpSession s = request.getSession(false);
String email = (String)s.getAttribute("email");
if (email == null) response.sendRedirect("index.htm");
composebId.setMailfrom(email);
%>
```
<jsp:useBean id="composebId" scope="session" class="jwebproject.composeb" />

/**
 * Modify an email or new email
 */
String mailid = request.getParameter("mailid");
if (mailid != null)
    composebId.setMailid(mailid);

composebId.doInit();
s.setAttribute("mailid",composebId.getMailid());

Composeb Bean

Provides methods and other data for used by compose process.

The flags (for email categorization) used:
public static final int SENDON = 1;
public static final int SENDBEFORE = 2;
public static final int SENDAFTER = 4;
public static final int MAXOPENINGS = 8;
public static final int PRIORITY = 16;

The field variables, and set and get methods:
Omitted !!!

The methods used:
/**
 * Initializes the bean on following sineario,
 * <ol>
 * <li> Load values from the database if the email is
 * to be modified
 * <li> Initializes all fields to null or relavent value
 * if this is going to be a new email
 * </ol>
 * @return true if initialization succeeded
 */
public boolean doInit()

Submitmail.jsp

Decide whether to update the email or consider as a new email. The criterion used is, if a mailid comes from a session then the email is to be modified, otherwise it is a new email. Associated bean does the necessary logic. This page is also access controlled. Extra initialization is there to cleanup the bean.
<% submitmailbId.doInit(); %>
<jsp:setProperty name="submitmailbId" property="*" />
<%
/**
 * Access Control
 */
HttpSession s = request.getSession(false);
String email = (String)s.getAttribute("email");
if (email == null) response.sendRedirect("index.htm");
submitmailbId.setMailfrom(email);

String mailid = (String)s.getAttribute("mailid");
submitmailbId.setMailid(mailid);

int STATUS = submitmailbId.doProcess();
%

Other Script . . .

---

## Submitmailb Bean

Provides methods and other data for use by email submission process.

The field variables, and set and get methods:

Omitted

**Flag values used.**

```java
public static final int UNSPECIFIED = 1;
public static final int INVALID = 2;
public static final int MAILTO = 4;
public static final int MAILSUBJECT = 8;
public static final int MAILBODY = 16;
public static final int SENDON = 32;
public static final int SENDBEFORE = 64;
public static final int SENDAFTER = 128;
public static final int MAXOPENINGS = 256;
public static final int PRIORITY = 512;
```

The methods used:

```java
/**
 * Initializes the bean
 */
public void doInit()

/**
 * Validate and set error flags and if no invalidity
 * found then according to category and considering
 * the mailid insert as a new email or update.
 * @return 0 if successful; flag value if invalidity found;
 * other value on error
 */
public int doProcess()
```
5.3.5 Delivering the Body of an Email

**Functional Arrangement**

![Diagram of Functional Arrangement](image)

*Figure 5.5: Functional Arrangement of the Delivering Email Body Process*

**Showmessage.jsp**

Acts as the body of an email. This page is loaded into the inline frame defined earlier on the email.

The following script constructs this page.

```html
<html>
<head>
<title></title>
</head>
<jsp:useBean id="showmessagebId" scope="session"
class="jwebproject.showmessageb" />
<jsp:setProperty name="showmessagebId" property="*" />
<body>
<jsp:getProperty name="showmessagebId" property="mailbody" />
</body>
</html>
```

**Showmessageb Bean**

Provides methods and other data for display the email body.

Methods used:

```java
/**
 * Binds a mailid
 * @param mailid mailid
 */
public void setMailid(String mailid)

/**
 * Depending on the mailid, email is loaded
 * Returns the body after checking and validating
 * Update mail information after delivering
 * @return the email body or a message
 */
public String getMailbody()
```
5.3.6 Obtaining a Proof-of Certificate

### Functional Arrangement

#### Certificate.htm

Contains a HTML form to specify the mailid of the email the proof of time certificate is required, what kind of certificate is needed, and to whom it is to be emailed to.

#### Certificatesubmit.jsp

Displays a message indicating whether the required certificate can be issued, or if not, for what reason.

```java
Other Script . . .
<jsp:useBean id="certificatesubmitbId" scope="session" class="jwebproject.certificatesubmitb" />
<% certificatesubmitbId.doInit(); %>
<jsp:setProperty name="certificatesubmitbId" property="*" />
<%
/**
 * Returns the result
 */
int STATUS = certificatesubmitbId.doProcess();
%>
Other Script . . .
```

#### Certificatesubmitb Bean

Provides methods and other data for used by certificate issuing process.

The field variables set and get methods:

The flags used:

```java
public static final int UNSPECIFIED = 1;
public static final int INCOMPLETE = 2;
```
public static final int NOTFOUND = 4;

The methods used:
/**
 * Cleanup and initializes the bean
 */
public void doInit()
/**
 * Check whether the certificate can be issued
 * if can then insert an entry in the database
 * if can't set relevant flags
 * @return 0 if ok; flag value otherwise
 */
public int doProcess()

5.4 Notarization Authority Server Implementation

![Figure 5.7: User Interface of the Notarization Authority Server](image)

This interface allows an administrator to start or stop the server and change settings.

![Figure 5.8: Functional Arrangement of the Notarization Authority Server](image)
This class defines the user interface to the system and responds to administrator requirements.

This class is the heart of this system and a complete code listing is given below:

```java
package jnaproject;

import java.io.*;
import javax.mail.*;
import javax.mail.internet.*;
import java.util.Properties;
import java.util.Date;
import nautil.*;
import java.sql.*;
import java.security.*;

public class ServiceImpl {
    // Server Processing Constants
    private static final int RUN_REGISTRATIONREQUESTS = 1;
    private static final int RUN_INCOMMINGMESSAGES = 2;
    private static final int RUN_PENDINGMESSAGES = 4;
    private static final int RUN_PENDINGCERTIFICATEREQUESTS = 8;

    // Mail Category Constants
    private static final int SENDON = 1;
    private static final int SENDBEFORE = 2;
    private static final int SENDAFTER = 4;
    private static final int MAXOPENINGS = 8;
    private static final int PRIORITY = 16;

    // Mail Processing Constants
    private static final int UNSPECIFIED = 1;
    private static final int NONTEXT = 2;

    // Mail Status Constants
    private static final int SENT = 1;
    private static final int SENDINGFAILED = 2;

    // References
    private Utility util = new Utility();
    private DateUtility dateutil = new DateUtility();
    private SecUtility secutil = new SecUtility();
    private ProcessQuery query = new ProcessQuery();
    private ExtractFeature extract = new ExtractFeature("X-","=",";";");
    private SettingDes settingDes = new SettingDes();
    private TimeStamper timeStamper = new TimeStamper();
    private MessageCertificator messageCertificator = new MessageCertificator();
    private PrivateKey pvt;

    public ServiceImpl(PrivateKey pvt) {
        this.pvt = pvt;
    }
}
```
public String Run(int WHAT) {
    StringBuffer LOG = new StringBuffer();
    if ((WHAT & RUN_REGISTRATIONREQUESTS) == RUN_REGISTRATIONREQUESTS)
        LOG.append(processRegistrationRequests());
    if ((WHAT & RUN_INCOMMINGMESSAGES) == RUN_INCOMMINGMESSAGES)
        LOG.append(processIncommingMessages());
    if ((WHAT & RUN_PENDINGMESSAGES) == RUN_PENDINGMESSAGES)
        LOG.append(processPendingMessages());
    if ((WHAT & RUN_PENDINGCERTIFICATEREQUESTS) == RUN_PENDINGCERTIFICATEREQUESTS)
        LOG.append(processPendingCertificateRequests());
    return LOG.toString();
}

/**
 * Responds to pending registration requests by sending activation key to each applicant, via an email.
 * @return string of log
 */
private String processRegistrationRequests() {
    StringBuffer LOG = new StringBuffer();
    LOG.append("[RR]Started >"); /* LOG Entry */
    try {
        /* Variables Needed */
        String email, activationkey;
        Date dateofrequest;

        /* Process Each Request */
        while (r.next()) {
            /* Initialization */
            email = r.getString("email");
            activationkey = r.getString("activationkey");
            dateofrequest = r.getDate("dateofrequest");
        }
    } catch (Exception e) {
        LOG.append(e.getMessage());
    }
}
/* Cleanup Expired Entries */
if ((dateutil.getUTC(time) - dateofrequest.getTime())
> 7*86400000) { // 7 Days Limit
LOG.append(" Exp!"); /* LOG Entry */
continue; // Continue From Next
}

/* Construct the Message to Send */
StringBuffer MSG = new StringBuffer();
MSG.append("<html><head><title>Notarization Authority for
Emails</title></head>\n\n MSG.append("<body bgcolor="#FFFFF">\n MSG.append("<h1><font
color="#0000FF">"+settingDes.WEBServerTitle+"</font></h1>\n MSG.append("<p>To continue your registration process, please go
 back to "+\nsettingDes.WEBServerDNS+" and enter this activation key
"+"</p>\n MSG.append("+activationkey+"</p>\n MSG.append("</body>:\n MSG.append("</html>\n
/* Send This Message as an Email */
if (sendThisMail(settingDes.WEBServerEmail,
 email,
 "Your Activation Key",
 MSG.toString())) { // True if Succeeded
/* Update Status */
if (query.justUpdate("UPDATE RegistrationRequest SET status=1
WHERE email="+email+""!)=1) { // Update Err!
LOG.append(" Update Err!"); /* LOG Entry */
} else {
LOG.append(" Ok!"); /* LOG Entry */
}
else {
LOG.append(" Sending Err!"); /* LOG Entry */
}
}
/* Cleanup */
r.close();
}
catch (Exception e) {
LOG.append(" Exception Err!"); /* LOG Entry */
LOG.append("[RR]Completed"); /* LOG Entry */
return LOG.toString();
}

/**
 * Obtain Emails from Server's Inbox, Check their aptness in Feature
 * Extraction and then Extract, Mask and Store Features
 * @return string of log
 */
private String processIncomingMessages() {
StringBuffer LOG = new StringBuffer();
LOG.append("[IM]Started "); /* LOG Entry */
try {
/* Load Settings */
String host = settingDes.POPHOST;

/* Load Settings */
String host = settingDes.POPHOST;
String username = settingDes.POPUserName;
String password = settingDes.POPPassword;

/* Code to Retrieve Emails From the Inbox */
// Create empty properties
Properties props = new Properties();
// Get session
Session session = Session.getDefaultInstance(props, null);
// Get the store
Store store = session.getStore("pop3");
// Connect to store
store.connect(host, username, password);
// Get folder
Folder folder = store.getFolder("INBOX");
// Open read-write
folder.open(Folder.READ_WRITE);
BufferedReader reader = new BufferedReader(
    new InputStreamReader(System.in));
// Get directory
Message message[] = folder.getMessages();

// <TODO> - Switch
/*
Message message[] = new Message[2];
FileInputStream fis = new
FileInputStream("D:/Temp/MailStore/b12.eml");
    message[0] = new MimeMessage(session, fis);
    fis.close();
    fis = new FileInputStream("D:/Temp/MailStore/12345.eml");
    message[1] = new MimeMessage(session, fis);
    fis.close();*/
// </TODO>
LOG.append(" Inbox:"+message.length); /* LOG Entry */

/* Variables Needed */
String mailid, mailfrom, mailto, mailsubject, mailbody;
String contentType, content;
boolean ack, pooself, pooreceiver, posself, posreceiver;
Date datetosend, sendbefore, sendafter;
int maxopenings, priority;
int CAT = 0, STATUS = 0;

/* Process Each Message */
for (int i=0, n=message.length; i<n; i++) {
    // Delete this Message from the Inbox
    message[i].setFlag(Flags.Flag.DELETED, true);

    /* Extract Some Header Fields */
    mailid = message[i].getHeader("Message-ID")[0];
    mailfrom = message[i].getFrom()[0].toString();
    mailsubject = message[i].getSubject();
    contentType = message[i].getContentType();
    content = message[i].getContent().toString();

    // Eliminate Name, i.e. < Name > Email ==> Email
    if (util.getEnclosed("<", ">", mailfrom)!=null)
        mailfrom = util.getEnclosed("<", ">", mailfrom);

    /* Load Default Settings of the Member */
    ResultSet r = query.getResults(  

/* SELECT * FROM MemberDefaults WHERE email='"+mailfrom+'"');
if (r.next()) {
    mailto = r.getString("mailto");
    ack = r.getBoolean("ack");
    pooself = r.getBoolean("pooself");
    pooreceiver = r.getBoolean("pooreceiver");
    posself = r.getBoolean("posself");
    posreceiver = r.getBoolean("posreceiver");
    datetosend = r.getDate("datetosend");
    sendbefore = r.getDate("sendbefore");
    sendafter = r.getDate("sendafter");
    maxopenings = r.getInt("maxopenings");
    priority = r.getInt("priority");
} else {
    /* This Sender is Not a Registered Member, So Skip */
    LOG.append(" Unregistered!"); /* LOG Entry */
    continue; // Next Message
}
r.close();

/* This System Can Process Only Text Messages */
if (!contentType.startsWith("text/plain")) {
    STATUS = STATUS | NONTEXT;
    LOG.append(" Nontext!"); /* LOG Entry */
    continue; // Next Message
}

/* Now it is Time to Extract Features From the Body
and to Apply the Default Mask (Default Settings) */
extart = extract.setContent(content);
if (extract.getFeature("MailTo")!=null)
    mailto = extract.getFeature("MailTo");
if (extract.getFeature("Ack")!=null)
    ack = extract.getFeature("Ack").equals("Y");
if (extract.getFeature("PooSelf")!=null)
    pooself = extract.getFeature("PooSelf").equals("Y");
if (extract.getFeature("PooReceiver")!=null)
    pooreceiver = extract.getFeature("PooReceiver").equals("Y");
if (extract.getFeature("PosSelf")!=null)
    posself = extract.getFeature("PosSelf").equals("Y");
if (extract.getFeature("PosReceiver")!=null)
    posreceiver = extract.getFeature("PosReceiver").equals("Y");

if (extract.getFeature("MaxOpen")!=null &&
    util.toInt(extract.getFeature("MaxOpen"),0)>0) {
    maxopenings = util.toInt(extract.getFeature("MaxOpen"),0);
    CAT = CAT | MAXOPENINGS;
}
if (extract.getFeature("Priority")!=null &&
    util.toInt(extract.getFeature("Priority"),0)>0) {
    priority = util.toInt(extract.getFeature("Priority"),0);
    CAT = CAT | PRIORITY;
}

/* Extract the Actual Body */
mailbody = extract.getBody();

/* Code to Apply Email Forwarding and Other Functionalities */
/* Store this Message and the Settings in the Database */
String queryFields = "";
String queryValues = "";
queryFields +=
"mailid,mailfrom,mailto,datearrived,mainsubject,mailbody";
queryFields += ",ack,possself,posreceiver,possself,pooreceiver";
queryFields += ",maxopenings,priority,mailcategory";
queryValues +=
"'"+mailid+"',"+mailfrom+"','"+mailto+"',"'+dateutil.getDateTime()+"',"'+mailsubject+"','"+mailbody+"'";
queryValues +=
"><"+ack+"','"+possself+"','"+posreceiver+"','"+possself+"','"+pooreceiver;
queryValues += ","+maxopenings+"','"+priority+"','"+CAT;
if (query.justUpdate("INSERT INTO MailStore("+queryFields+")
VALUES("+queryValues+"))!=1) { STATUS = STATUS | UNSPECIFIED;
LOG.append(" Insert Err!"); /* LOG Entry */
} else {
LOG.append(" Ok!"); /* LOG Entry */
}
/* Code to Send an Acknowledgment if Prompted */
// Removed due to Unexpected Error Occured

// Close connection
folder.close(true);
store.close();
} catch (Exception e) {
LOG.append(" Exception Err!"); /* LOG Entry */
}
LOG.append("< [IM]Completed"); /* LOG Entry */
return LOG.toString();

/**
 * Respond to Pending Messages by Constructing Emails and
 * Sending them to Intended Recipients
 * @return string of log
 */
private String processPendingMessages() {
StringBuffer LOG = new StringBuffer();
LOG.append(">> [FM]Started >>");
String mailLink = settingDes.WEBServerDNS+"showmessage.jsp?mailid=";
int TOTAL = 0;

try {
/* Load Pending Messages */
ResultSet r = query.getResults("SELECT * FROM MailStore WHERE status=0");

/* Variables Needed */
String mailid, mailfrom, mailto, mainsubject, mailbody;
boolean ack, possself, posreceiver, possself, pooreceiver;
Date datetosend, sendbefore, sendafter;

// Led for the Future
int category;
int STATUS = 0;

/* Process Each Message */
while (r.next()) {
    TOTAL++;

    /* Initialization */
    mailid = r.getString("mailid");
    mailfrom = r.getString("mailfrom");
    mailto = r.getString("mailto");
    mailsubject = r.getString("mailsubject");
    ack = r.getBoolean("ack");
    pooself = r.getBoolean("pooself");
    pooreceiver = r.getBoolean("pooreceiver");
    posself = r.getBoolean("posself");
    posreceiver = r.getBoolean("posreceiver");
    datetosend = r.getDate("datetosend");
    sendbefore = r.getDate("sendbefore");
    sendafter = r.getDate("sendafter");
    category = r.getInt("mailcategory");

    /* Act Upon the Category */
    if (category == 0
        // Nothing to Restrict
        || ((category & MAXOPENINGS)==MAXOPENINGS)
        // Nothing to Restrict
        || ((category & PRIORITY)==PRIORITY)
        // Sending Date has Arrived
        || ((category & SENDON)==SENDON) &&
        dateutil.getDate(datetosend).equals(dateutil.getDate()))
        // This Mail has to be Sent Before this Given Date
        || ((category & SENDBEFORE)==SENDBEFORE) &&
        (dateutil.getUTC()<=sendbefore.getTime())
        // This Mail has to be Sent After this Given Date
        || ((category & SENDAFTER)==SENDAFTER) &&
        (dateutil.getUTC()>=sendafter.getTime())) {
        /* Construct the Body of the Email */
        StringBuffer BODY = new StringBuffer();
        BODY.append("<html><head><title>Notarization Authority for
        Emails</title></head>");
        BODY.append("<body bgcolor="#FFFFFF">;
        BODY.append("<h1><font
        color="#0000FF">"+settingDes.WEBServerTitle+"</font></h1>);
        BODY.append("<p><font color="#0000FF">Message-ID:
        "+mailid+"</font></font></p>);
        BODY.append("<ilayer left="0" width="100%" height="100%"
        src=""+mailLink+mailid+"" marginwidth="0" marginheight="0"
        align="middle" scrolling="auto" frameborder="0"></ilayer>");
        BODY.append("<iframe src=""+maillink+mailid+"" width="100%"
        marginwidth="0" height="100%" marginheight="0"
        align="middle" scrolling="auto" frameborder="0"></iframe>");
        BODY.append(</body>");
}
BODY.append("</html>");
mailbody = BODY.toString();

/* Send this Email */
if (sendThisMail(mailfrom,mailto,mailsubject,mailbody)) {
    STATUS = STATUS | SENT;

    /* Perform Certificate Needs */
    // Proof of Sending Certificate to Sender
    if (posself)
        query.justUpdate("INSERT INTO CertificateRequest(email,mailid,certtype,status) VALUES('"+mailfrom+"','"+mailid+"',0,0)");
    // Proof of Sending Certificate to Recipient
    if (pooreceiver)
        query.justUpdate("INSERT INTO CertificateRequest(email,mailid,certtype,status) VALUES('"+mailto+"','"+mailid+"',0,0)");
    // Proof of Opening Certificate to Sender
    if (pooself)
        query.justUpdate("INSERT INTO CertificateRequest(email,mailid,certtype,status) VALUES('"+mailfrom+"','"+mailid+"',1,0)");
    // Proof of Opening Certificate to Recipient
    if (pooreceiver)
        query.justUpdate("INSERT INTO CertificateRequest(email,mailid,certtype,status) VALUES('"+mailto+"','"+mailid+"',1,0)");
} else {
    STATUS = STATUS | SENDINGFAILED;
}

/* Send an Acknowledgment */
// Important ! - Unexpected Errors are Reported in this Code
StringBuffer MSG = new StringBuffer();
MSG.append("<html><head><title>Notarization Authority for Emails</title></head>");
MSG.append("<body bgcolor="#FFFFFF">");
MSG.append("<h1><font color="#0000FF">"+settingDes.WEBServerTitle+"</font></h1>);
MSG.append("<p>Your Message Successfully Sent</p>");
MSG.append("</body>");
MSG.append("</html>");
if (!sendAck(mailid, mailfrom, MSG.toString())) {
    LOG.append(" Sending Err!");
}

/* Update Message Status */
if (query.justUpdate("UPDATE MailStore SET status="+STATUS+","datesent='"+dateutil.getDateTime()+"' WHERE mailid='"+mailid+"'")!=1) {
    LOG.append(" Update Err!");
}
} catch (Exception e) {
    LOG.append(" Exception Err!"); /* LOG Entry */
}
LOG.append(" Tot:"+TOTAL);
LOG.append("< [PM]Completed"); /* LOG Entry */
/**
 * Respond to Pending Certificate Requests by Extracting
 * Latest Message Information, Constructing Certificate, Attach
 * this Certificate to an Email and Sending this to Requesters
 *
 * @return string of log
 */
private String processPendingCertificateRequests() {
    StringBuffer LOG = new StringBuffer();
    LOG.append(" [CR]Started >")); /* LOG Entry */

    /* Details about the Time Server */
    timeStamper.setIssuerUniqueID(settingDes.NA_ID);
    timeStamper.setTimeServerUniqueID(settingDes.TA_ID);
    timeStamper.setPKCertificateSerialNo(settingDes.NA_PKC_Serial);
    timeStamper.setTimePrecision("20%");

    try {
        /* Load Pending Requests */
        ResultSet r = query.getResults("
            SELECT * FROM CertificateRequest WHERE status=0");

        /* Issuer Details */
        DN issuer = new DN(settingDes.NA_CN,settingDes.NA_OU,settingDes.NA_O,
            settingDes.NA_L,settingDes.NA_S,settingDes.NA_C,
            settingDes.NA_E);

        /* Process Each Request */
        while (r.next()) {
            /* Initialization and Variables */
            String mailid = r.getString("mailid");
            int certtype = r.getInt("certtype");
            String E = r.getString("email");

            String CN = ",", OU = "", O = "", L = "", S = "", C = "";

            /* Load Requester's Information if Available */
            ResultSet s = query.getResults("
                SELECT * FROM RegisteredMembers WHERE email='"+E+"'"
            );
            if (s.next()) {
                CN = s.getString("cn");
                OU = s.getString("ou");
                O = s.getString("o");
                L = s.getString("l");
                S = s.getString("s");
                C = s.getString("c");
            }
            s.close(); /* Subjecter(Requester) Details */
            DN subject = new DN(CN,OU,O,L,S,C,E);

            String mailfrom = "", mailto = "", mailcc = "", mailbcc = "",
            mailsubject = "", mailbody = "", statuslog = "";
            Date datearrived = null, datesent = null, datelastopened = null;

            /* Load Message Information as Availble */
            ResultSet t = query.getResults("
                SELECT * FROM MailStore WHERE mailid='"+mailid+"'"
            );
    }

    return LOG.toString();
}
if (t.next()) {
    mailfrom = t.getString("mailfrom");
    mailto = t.getString("mailto");
    mailsubject = t.getString("mailsubject");
    mailbody = t.getString("mailbody");
    statuslog = t.getString("statuslog");
    datearrived = t.getDate("datearrived");
    datesent = t.getDate("datesent");
    datelastopened = t.getDate("datelastopened");
}

/* Message Header Details */
MessageHeader messageHeader = new MessageHeader(mailid, mailfrom, mailto, mailcc, mailbcc, mailsubject, dateutil.getDateTime(datearrived));

String purpose = "")
    Date date = null;

    /* Timestamp to the Purpose */
    if (certtype == 0) { // Proof-of-Sending
        purpose = "PROOF-OF-SENDING";
        date = datearrived;
    } else { // Proof-of-Opening
        if (datelastopened != null)
            purpose = "PROOF-OF-OPENING (APPROX)";
            date = datelastopened;
        else if (datesent != null)
            purpose = "PROOF-OF-OPENING (PRECISE)";
            date = datesent;
        else {
            LOG.append(" Nodate Err! "); /* LOG Entry */
            continue;
        }
    }
    TimeStamp timestamp = timeStamper.timeStamp(mailbody.getBytes(), true, date, pvt);

    /* Construct the Message Certificate */
    MessageCertificate messageCertificate =
        messageCertificator.issue(purpose, issuer, subject, timestamp, messageHeader, mailbody.getBytes(),
                true,
                statuslog, pvt);

    /* Send this Certificate to Indended Recipient */
    // Code to Attach and Send - Pending
    LOG.append(" Ok!"); /* LOG Entry */
} catch (Exception e) {
    LOG.append(" Exception Err!"); /* LOG Entry */
} LOG.append("< [CR]Completed"); /* LOG Entry */
return LOG.toString();
}

/**
 * Send an Email Specified by the Parameters
 * @param from from
* @param to to
* @param subject subject
* @param content body
* @return true if sent; false otherwise
*/

private boolean sendThisMail(String from, String to, String subject, String content) {
    try {
        String host = settingDes.SMTPHOST;
        // Get system properties
        Properties props = System.getProperties();
        // Setup mail server
        props.put("mail.smtp.host", host);
        // Get session
        Session session = Session.getDefaultInstance(props, null);
        // Define message
        MimeMessage message = new MimeMessage(session);
        // Set the from address
        message.setFrom(new InternetAddress(from));
        // Set the to address
        message.addRecipient(Message.RecipientType.TO,
                              new InternetAddress(to));
        // Set the subject
        message.setSubject(subject);
        // Set the content
        message.setContent(content,"text/html");
        // Send message
        Transport.send(message);
        return true;
    } catch (Exception e) {
        return false;
    }
}
}

5.5 TESTING

The test plan was arranged as given below:

- **Unit tests** were carried out in order to confirm that the classes and modules give the exact functionality to the system. These tests were carried out at the design time, while coding the classes.

- **Integration tests** were carried out in order to confirm that modules and classes give exact interfaces to other modules and classes. These tests were carried out throughout the project to test whether the classes correctly interact with each other under the package scope and the framework gives the right functionality to other two systems. Exceptional situations occurred during this testing process and some plasters were pasted. Some of them are given below:
  - A class named SendMail was put on to the framework to provide methods to retrieve and send emails. But when these methods are called by other systems, it seen that they were not working. But this class passed the unit test. The reason for this failure
was not found. Solution was to remove that class from the framework and replicate in each system.

- The MessageCertificator class issues Message Certificates and it passed the unit test. But while this class invoked for issue more than one certificate in a while loop the system reported a memory read error.

- **System test** is carried out to prove that the software meets the agreed user requirements and works in the target environment. The user manual given at the end of the document (@see Appendix D) will prove that the functional requirements were met.

- **Acceptance tests** were carried out to persuade others that the software does indeed work. Adding to this list, some demonstration sessions were carried out to my supervisor Mr. Rasika Dayarathna and to some of my friends, by allowing them to ask questions and giving them the opportunity to use the system.
Chapter 6 EVALUATION

In this entire project what I did was I looked at the world from a different direction with the question “Can I change the way of current email exchange” in mind. So, at the beginning I firmly arranged my aims & objectives and planned the way. Even though I said like that, I was blind in this area. I knew little background theories, technologies and also I was very weak in java language and using the tools.

The first step in my plan was to educate myself and construct a foundation for the project. Following that, I surfed lot of websites and looked at the world in a researchive nature gathering information about what is happening to email right now! At the same time I learned java language and practiced to use JBuilder and other related tools. As a result I found that in java there is a package to build email programs, which is JavaMail API.

To achieve my second goal, I wanted to someway track the recipient of an email when observing it. My background materials provided me a solution. It suggested me to use the message/external-body field specifying remote location to the body of an email. If it works I could send first the header with a remote link to the body, and when the recipient wants to read the email, I could send the body via different protocol, and it allows me to track the observing. So I started experimenting. I have gone through the entire JavaMail API specification, but no information found. Then I tried by guessing codes, but no success. At last I had to throw that solution. While these things gone in this way my supervisor introduced me ReadNotify notary service. When I gone through its reading materials I found that they are also in the same eager in the same area. But they are not providing free service and no source codes. I also found some other service providers serving same services. But none of them provided me considerable information except ReadNotify. This ReadNotify gave me a hint that is to use ilayer and iframe tags. These tags specify an inline frame (layer) in an HTML page thus remote site can be loaded into this frame (layer). This solution leads be to use that tag to track the observation of an email. So I built my first prototype by use of this tag, and it worked! ! !

Not for long (oh!). While I experimenting with Yahoo! and other web mail providers I found that they filter these kinds of tags [Appendix B] form emails, because of security threats. So I wanted to find a different solution. But it seemed that there is no other possible solution except giving a link in the email to collect the body from a remote location. So assuming that this both solutions would possess some kind of a guaranteed delivery, the entire project plan was changed accordingly. But still there is a challenge to be solved, which I led for the future.

Apart from those, the other aims & objectives made a difference. This project serves its service through a website, which is worldwide. And also it did not change the world. This is because it
Notarization Authority for Emails
H.E.M.H.B. Ekanayake

operates as a proxy. What has to be done is send emails through this proxy. Therefore this proxy will change itself for standstill in this changing world.

A special provision is made to hold up the reusability of the effort. This is achieved through nucleating reusable techniques into a single framework. This framework now provides methods and other logic to build similar projects in this area. Besides from usability, another provision is made for standardization. Standardization achieved through using standard designing techniques and tools (tired architecture, UMM, certificate derived from X.509, Rational Rose), using standard software packages and tools (JavaMail API), using standard code style (well organized code blocks, consider every path in control flow statements and exception handling, correct alignment of code, use standard naming conventions, standard commenting techniques, ...) and documenting the entire work in a standard way.

This entire project is built on a well-defined security framework. A special effort has been taken to secure the private key of the authority and to separate the logic, which require a security alert. The database acts as a bridge between these two systems. It is desirable to convert this database into a SQL or Oracle database in order to secure this bridge. At the moment this system do not possesses any transport time security. Which I failed due to the limited time frame I had.

The sole purpose of this project was to develop a notarization authority and the initial plan was to notarize any kind of digital document. But for make this project feasible, the final decision was to scope down the idea only for emails. For notarization two techniques were incorporated: Timestamp and Proof-of Certificate. These techniques were omitted from the design due to misunderstanding and uncertain about the theories until implementation. Therefore a reader is encouraged to go through implementation for better understand about the theories.

This is only an attempt taken to think differently. Much effort has been spent to achieve the targets. The conclusion is up to others, think, evaluate, understand and analyze and arrive to a final decision.
Chapter 7  FUTURE WORKS

This system is expected to have a great future. Dozens of ideas came into the mind, but not all of them were implemented. So let's take a tour describing what are those remaining and not completed ideas.

- Email tracing facility is not incorporated into the system. If the tracing facility is enabled the sender could get an idea of how an email is traveling with the location and time information. So he can visualize the current residence of any email.
- One aim of the system was to notarize any type of digital document. This is not very difficult. By allowing upload and download facilities, and presenting the uploaded document to time stamping and certificate generating objects, this objective was to be easily achieved.
- Even though this system possesses standstill time security, it does not possess any transport time security. This must be filled. There are number of techniques that can be used. For example the web system could be secured using SSL.
- This system currently supports only plain text email messages. Because there is no engine to process MIME or S/MIME messages. The implementation is constructed such that allowing extensions easy. So just by incorporating engine to process other type of messages, the system can be changed to support non-text messages also.
- Some extensions to email, for instance blocking or forwarding, are very easy to implement to the system, since proxy nature of the system. New ideas were arrived in this direction. For example, the email forwarding can be done in the following manner:
  - Direct forwarding – Forwarding the email to someone else, not to the recipient given in the email
  - Cluster forwarding – Forward any single email to a group
  - Chaining – If the email can’t reach to the recipient A then forward to recipient B, if that attempt also failed then forward it to recipient C, and so on. This mode has some other options. For example, the reaching condition may be the email is expiration
- This system already introduced some extensions to email, introducing date to send, send before, send after, priority and maximum observations fields. These extensions are working well. Innovative thinker might go on this direction and pullout lots of ideas.
- As mentioned earlier, the database acts as a bridge to two subsystems. This bridge is implemented using Microsoft Access. This DBMS can’t guarantee any efficient or secure bridge. Also it is very weak in providing multi-user accesses. So the idea is to transfer this database into more efficient and secure DBMS.
- There are remaining parts of the system that has to be implemented. Also there are lot of slots and hooks to be filled. These parts have to be completed in order to make this system much more perfect.
The trustworthy of the certificates comes from the accuracy of the information contained in these certificates. Currently the only trusted information contained in a certificate is the email address (email address is validated before registering any applicant). So it is up to the future to validate other information as well.

In addition to those were pointed out, the reader may encounter some ideas or suggestions. These ideas are valuable for this system to have a great future. This system is implemented as a full commercial product. So it is assumed that this system is implemented in the future in a commercial web server and servicing to the world through its website.
Chapter 8  REFERENCES

<table>
<thead>
<tr>
<th>RFCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC 821  SMTP</td>
</tr>
<tr>
<td>RFC 822  MAIL</td>
</tr>
<tr>
<td>RFC 1049  MAIL</td>
</tr>
<tr>
<td>RFC 974  DNS-MX</td>
</tr>
<tr>
<td>RFC 1521  MIME Body</td>
</tr>
<tr>
<td>RFC 1522  MIME Header</td>
</tr>
<tr>
<td>RFC 1651  SMTP Service Extensions</td>
</tr>
<tr>
<td>RFC 1652  SMTP Service Extensions</td>
</tr>
<tr>
<td>RFC 1653  SMTP Service Extensions</td>
</tr>
<tr>
<td>RFC 867  DAYTIME Protocol</td>
</tr>
<tr>
<td>RFC 868  TIME Protocol</td>
</tr>
<tr>
<td>RFC 1305  NTP Protocol</td>
</tr>
<tr>
<td>RFC 2030  SNTP Protocol</td>
</tr>
<tr>
<td>RFC 1510  Kerberos V5 authentication protocol</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Web Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www4.ulpgc.es/tutoriales/tcpip/pru/3376fm.htm">http://www4.ulpgc.es/tutoriales/tcpip/pru/3376fm.htm</a></td>
</tr>
<tr>
<td><a href="http://java.sun.com/docs/books/tutorial/index.html">http://java.sun.com/docs/books/tutorial/index.html</a></td>
</tr>
<tr>
<td><a href="http://www.mindview.net/">http://www.mindview.net/</a></td>
</tr>
<tr>
<td>[WebDateTime]  “Date and time from a time server”  [WebDateTime]  “Date and time from a time server”</td>
</tr>
<tr>
<td><a href="http://www.kloth.net/">http://www.kloth.net/</a></td>
</tr>
<tr>
<td><a href="http://www.itconsult.co.uk/index.htm">http://www.itconsult.co.uk/index.htm</a></td>
</tr>
<tr>
<td>[WebReadNotify]  “Readnotify Email Notary and Timestamping Service”  [WebReadNotify]  “Readnotify Email Notary and Timestamping Service”</td>
</tr>
<tr>
<td><a href="http://www.readnotify.com/readnotify/">http://www.readnotify.com/readnotify/</a></td>
</tr>
<tr>
<td><a href="http://www.certco.com/">http://www.certco.com/</a></td>
</tr>
<tr>
<td><a href="http://www.msb.edu/faculty/culnanm/EC/Briefings2/Dieuj/Main.html">http://www.msb.edu/faculty/culnanm/EC/Briefings2/Dieuj/Main.html</a></td>
</tr>
<tr>
<td><a href="http://www.microsoft.com/WINDOWS2000/">http://www.microsoft.com/WINDOWS2000/</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Sources</th>
</tr>
</thead>
</table>
Chapter 9 APPENDICES

9.1 APPENDIX A – ADDITIONAL DATA

9.1.1 Appendix A.1: Some Frequently Used Mail Header Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Receiver Fields</strong></td>
<td></td>
</tr>
<tr>
<td>TO</td>
<td>Primary recipients of the message.</td>
</tr>
<tr>
<td>CC</td>
<td>Secondary recipients of the message. (Carbon-copy)</td>
</tr>
<tr>
<td>BCC</td>
<td>Additional recipients of the message. (Author's Copy)</td>
</tr>
<tr>
<td><strong>Originator Fields</strong></td>
<td></td>
</tr>
<tr>
<td>FROM</td>
<td>Identity of sender.</td>
</tr>
<tr>
<td>SENDER</td>
<td>Authenticated identity of the AGENT (person, system or process)</td>
</tr>
<tr>
<td>REPLY-TO</td>
<td>The mailbox to which responses are to be sent. Added by the originator.</td>
</tr>
<tr>
<td><strong>Trace Fields</strong></td>
<td></td>
</tr>
<tr>
<td>RETURN-PATH</td>
<td>Addresses and route back to the originator. Added by the final transport system.</td>
</tr>
<tr>
<td>RECEIVED</td>
<td>Useful for tracing transport problems. Added by each transport service.</td>
</tr>
<tr>
<td><strong>Reference Fields</strong></td>
<td></td>
</tr>
<tr>
<td>MESSAGE-ID</td>
<td>Unique identifier of the message.</td>
</tr>
<tr>
<td>IN-REPLY-TO</td>
<td>Previous correspondence, which this message answers.</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>Other correspondence, which this message references.</td>
</tr>
<tr>
<td>KEYWORDS</td>
<td>Keywords or phases, separated by commas</td>
</tr>
<tr>
<td><strong>Other Fields</strong></td>
<td></td>
</tr>
<tr>
<td>SUBJECT</td>
<td>Summary of the message.</td>
</tr>
<tr>
<td>COMMENTS</td>
<td>Text comments.</td>
</tr>
<tr>
<td>ENCRYPTED</td>
<td>Encryption details of the body of the message.</td>
</tr>
<tr>
<td>EXTENSION-FIELD</td>
<td></td>
</tr>
<tr>
<td>USER-DEFINED-FIELD</td>
<td></td>
</tr>
</tbody>
</table>

Table A.1: Frequently Used Email Header Fields

9.1.2 Appendix A.2: Some Standard Time Servers

<table>
<thead>
<tr>
<th>Server</th>
<th>Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>time.nrc.ca</td>
<td>37, NTP</td>
</tr>
<tr>
<td>ptbtime1.ptb.de</td>
<td>13, 37</td>
</tr>
<tr>
<td>ptbtime2.ptb.de</td>
<td>13, 37</td>
</tr>
<tr>
<td>ntp0.fau.de</td>
<td>37, NTP</td>
</tr>
<tr>
<td>ntp1-0.cs.tu-berlin.de</td>
<td>13, 37</td>
</tr>
<tr>
<td>ntp1-1.cs.tu-berlin.de</td>
<td>13, 37</td>
</tr>
<tr>
<td>ntp1-0.uni-erlangen.de</td>
<td>37, NTP</td>
</tr>
<tr>
<td>ntp-p1.obspm.fr</td>
<td>13, 37</td>
</tr>
<tr>
<td>time.ien.it</td>
<td>13 (local time), 37</td>
</tr>
<tr>
<td>ntp.irit.cnr.it</td>
<td>13 (local time), NTP</td>
</tr>
<tr>
<td>NIST - US National Institute of Standards and Technology - <a href="http://www.boulder.nist.gov/">www.boulder.nist.gov/</a></td>
<td>13, 37</td>
</tr>
<tr>
<td>time-a.timefreq.bldrdoc.gov</td>
<td>13, 37</td>
</tr>
<tr>
<td>time-b.timefreq.bldrdoc.gov</td>
<td>13, 37</td>
</tr>
<tr>
<td>time-c.timefreq.bldrdoc.gov</td>
<td>13, 37</td>
</tr>
<tr>
<td>time-d.timefreq.bldrdoc.gov</td>
<td>13, 37</td>
</tr>
<tr>
<td>tick.usno.navy.mil</td>
<td>13, 37, NTP</td>
</tr>
<tr>
<td>tock.usno.navy.mil</td>
<td>13, 37, NTP</td>
</tr>
</tbody>
</table>
Notarization Authority for Emails
H.E.M.H.B. Ekanayake

<table>
<thead>
<tr>
<th>ntp2.usno.navy.mil</th>
<th>ports 13, 37, NTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>gnomon.cc.columbia.edu</td>
<td>ports 13, 37, NTP</td>
</tr>
<tr>
<td>tick.gatech.edu</td>
<td>ports 13, 37, NTP</td>
</tr>
</tbody>
</table>

Table A.2: Standard Time Servers

9.1.3 Appendix A.3: Five Standard MIME Header Fields

<table>
<thead>
<tr>
<th>MIME-Version</th>
<th>Must have the value “1.0”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-Type</td>
<td>Describes how the object within the body is interpreted. Default is “text/plain; charset=us-ascii”</td>
</tr>
<tr>
<td>Content-Transfer-Encoding</td>
<td>Describes how the object within the body is encoded</td>
</tr>
<tr>
<td>Content-Description</td>
<td>Plain text description of the object within the body, which is useful when the object is not readable</td>
</tr>
<tr>
<td>Content-ID</td>
<td>Word-unique value specifying the content of this part of this message</td>
</tr>
</tbody>
</table>

Table A.3: Five Standard MIME Header Fields

9.1.4 Appendix A.4: Standard MIME Content-Type Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>SubType</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>pony</td>
</tr>
<tr>
<td>Body is unformatted text</td>
<td>plain</td>
</tr>
<tr>
<td>Character set is specified with the charset parameter</td>
<td></td>
</tr>
<tr>
<td>multipart</td>
<td>mixed</td>
</tr>
<tr>
<td>Body contains multiple objects of independent data types, separated by lines called encapsulation boundaries, for example:</td>
<td></td>
</tr>
<tr>
<td>Content-Type:multipart/mixed; boundary=&quot;1995021309105517&quot;</td>
<td></td>
</tr>
<tr>
<td>message</td>
<td>rfc822</td>
</tr>
<tr>
<td>Body is an encapsulated message, or part of one</td>
<td></td>
</tr>
<tr>
<td></td>
<td>partial</td>
</tr>
<tr>
<td>1.0 order, display all in parallel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>alternative</td>
</tr>
<tr>
<td>Different parts are alternatives of the same information, display only the best one</td>
<td></td>
</tr>
<tr>
<td></td>
<td>digest</td>
</tr>
<tr>
<td>Variant on multipart/mixed, used in the transmission of multiple messages</td>
<td></td>
</tr>
<tr>
<td></td>
<td>rfc822</td>
</tr>
<tr>
<td>The encapsulated message is of the syntax of RFC 822</td>
<td></td>
</tr>
<tr>
<td></td>
<td>partial</td>
</tr>
<tr>
<td>Allows fragmentation of large mail items, send as fragments and re-assemble them transparently to the recipient. The three parameters are id, number and total</td>
<td></td>
</tr>
<tr>
<td></td>
<td>external-body</td>
</tr>
<tr>
<td>Contains a pointer to an object, which exists elsewhere. The parameter access-type defines how the external object is to be accessed; the access types are ftp, tftp, anon-ftp, local-file, afs and mail-server</td>
<td></td>
</tr>
<tr>
<td>image</td>
<td>jpeg</td>
</tr>
<tr>
<td>Body contains image data</td>
<td></td>
</tr>
<tr>
<td>video</td>
<td>mpeg</td>
</tr>
<tr>
<td>Body contains moving image data</td>
<td></td>
</tr>
<tr>
<td>audio</td>
<td>basic</td>
</tr>
<tr>
<td>Body contains audio image data</td>
<td></td>
</tr>
<tr>
<td>application</td>
<td>PostScript</td>
</tr>
<tr>
<td>Types, which do not fit into other categories. Poses serious security risks</td>
<td></td>
</tr>
<tr>
<td>PostScript Adobe Systems Post Script</td>
<td></td>
</tr>
<tr>
<td>octet-stream</td>
<td>General binary data consisting of 8-bit bytes, parameters are type and padding</td>
</tr>
</tbody>
</table>
9.1.5 Appendix A.5: Standard MIME Content-Transfer-Encoding Fields

<table>
<thead>
<tr>
<th>Encoding Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-bit (default)</td>
<td>No encoding, mail-safe, ASCII text lines</td>
</tr>
<tr>
<td>8-bit</td>
<td>Possible only if SMTP agents support the SMTP Service Extensions for 8bit-MIMEtransport</td>
</tr>
<tr>
<td>Binary</td>
<td>Valid if MIME were used in conjunction with other mail transport mechanisms, e.g. Internet, message/external-body</td>
</tr>
<tr>
<td>Quoted-Printable</td>
<td>Represents non-mail safe chars by hexadecimal</td>
</tr>
<tr>
<td></td>
<td>Introduces short lines (75 or less)</td>
</tr>
<tr>
<td></td>
<td>Replaces each non-text character with a 3-byte sequence, which is the equals sign (“=”) as the quoted character and two uppercase hexadecimal digits, e.g. “=3D”</td>
</tr>
<tr>
<td></td>
<td>Inefficient for binary data.</td>
</tr>
<tr>
<td>Base64</td>
<td>Only 73 mail-safe characters exist, 6-bit (64) is the only usable bit length, so treat the input stream as a bit stream, regroup them into shorter bytes, pad these bytes to 8-bit, and translate them to 7-bit characters. E.g.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Base64 Value</th>
<th>ASCII Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>62</td>
<td>+</td>
</tr>
<tr>
<td>63</td>
<td>/</td>
</tr>
</tbody>
</table>

9.1.6 Appendix A.6: Yahoo's Filtering Mechanism

Yahoo's Seven Word Fragments You Can't Say In HTML Email

eval => review
mocha => espresso
expression => statement
javascript => java-script
jscript => j-script
vbscript => vb-script
livescript => live-script

Yahoo's hack doesn't respect word boundaries: so evaluate would become reviewuate, retrieval becomes retrireview.

Note that plain text email is left untouched - only HTML mail gets this treatment.

There are also a few tags that are verboten:

link => xlink
script => cursive
object => xobject
embed => xembed
body => xbody
iframe => xframe
layer => xlayer
applet => xapplet
meta => xmeta
## 9.1.7 Appendix A.7: Core Classes of JavaMail API

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.mail.Session</td>
<td>Defines a basic mail session, uses java.util.Properties object to get information like mail server, user name, password.</td>
</tr>
<tr>
<td></td>
<td>Properties props = new Properties();</td>
</tr>
<tr>
<td></td>
<td>// fill props with any information</td>
</tr>
<tr>
<td></td>
<td>Session session = Session.getDefaultInstance(props, null);</td>
</tr>
<tr>
<td></td>
<td>// or Session session = Session.getInstance(props, null);</td>
</tr>
<tr>
<td>javax.mail.Message</td>
<td>Used to create a message. The subclass javax.mail.internet.MimeMessage is used to create MIME type messages.</td>
</tr>
<tr>
<td></td>
<td>MimeMessage message = new MimeMessage(session);</td>
</tr>
<tr>
<td></td>
<td>message.setContent(&quot;Hello&quot;, &quot;text/plain&quot;);</td>
</tr>
<tr>
<td></td>
<td>// or message.setText(&quot;Hello&quot;); since “text/plain” message.setSubject(&quot;First&quot;);</td>
</tr>
<tr>
<td>javax.mail.Address</td>
<td>Used to address the letter, also uses the subclass java.mail.internet.InternetAddress</td>
</tr>
<tr>
<td></td>
<td>Address address = new InternetAddress(&quot;<a href="mailto:someone@cmb.ac.lk">someone@cmb.ac.lk</a>&quot;);</td>
</tr>
<tr>
<td></td>
<td>message.setFrom(address);</td>
</tr>
<tr>
<td></td>
<td>// Address address[ ] = …</td>
</tr>
<tr>
<td></td>
<td>// message.addFrom(address); for multiple from addresses message.addRecipient(Message.RecipientType.TO, address);</td>
</tr>
<tr>
<td></td>
<td>// other recipient types are Message.RecipientType.CC, Message.RecipientType.BCC</td>
</tr>
<tr>
<td>javax.mail.Authenticator</td>
<td>Used to access protected resources (mail server) via a username and password.</td>
</tr>
<tr>
<td></td>
<td>Properties props = new Properties();</td>
</tr>
<tr>
<td></td>
<td>// fill props with any information</td>
</tr>
<tr>
<td></td>
<td>Authenticator auth = new MyAuthenticator();</td>
</tr>
<tr>
<td></td>
<td>Session session = Session.getDefaultInstance(props, auth);</td>
</tr>
<tr>
<td>javax.mail.Transport</td>
<td>Used to send a message.</td>
</tr>
<tr>
<td></td>
<td>Transport.send(message); // default SMTP</td>
</tr>
<tr>
<td></td>
<td>// or Protocol-specific message.saveChanges();</td>
</tr>
<tr>
<td></td>
<td>Transport transport = session.getTransport(&quot;smtp&quot;);</td>
</tr>
<tr>
<td></td>
<td>transport.connect(host, username, password);</td>
</tr>
<tr>
<td></td>
<td>transport.sendMessage(message, message.getAllRecipients());</td>
</tr>
<tr>
<td></td>
<td>transport.close();</td>
</tr>
<tr>
<td>javax.mail.Store andjavax.mail.Folder</td>
<td>Used to get messages.</td>
</tr>
<tr>
<td></td>
<td>Store store = session.getStore(&quot;pop3&quot;);</td>
</tr>
<tr>
<td></td>
<td>store.connect(host, username, password);</td>
</tr>
<tr>
<td></td>
<td>Folder folder = store.getFolder(&quot;INBOX&quot;);</td>
</tr>
<tr>
<td></td>
<td>folder.open(Folder.READ_ONLY);</td>
</tr>
<tr>
<td></td>
<td>Message message[] = folder.getMessages();</td>
</tr>
<tr>
<td></td>
<td>// to get the content System.out.println(((MimeMessage)message).getContent());</td>
</tr>
<tr>
<td></td>
<td>folder.close(aBoolean); // Boolean says update or not</td>
</tr>
<tr>
<td></td>
<td>store.close();</td>
</tr>
</tbody>
</table>

*Table A.6: Core JavaMail API Classes*
9.2 **APPENDIX B – SELECTED EXAMPLES**

9.2.1 **Appendix B.1: A Complex MIME Message**

MIME-Version: 1.0
From: Steve Hayes <steve@hayessj.bedfont.uk.ibm.com>
To: Matthias Enders <enders@itso180.itso.ral.ibm.com>
Subject: Multipart message
Content-type: multipart/mixed; boundary="1995021309105517"

This section is called the preamble. It is after the header but before the first boundary. Mail readers which understand multipart messages must ignore this.
--1995021309105517

The first part. There is no header, so this is text/plain with charset=us-ascii by default. The immediately preceding <CRLF><CRLF> sequence that ends the null header. The one at the end is part of the next boundary, so this part consists of five lines of text with four <CRLF>'s.
--1995021309105517
Content-type: text/plain; charset=us-ascii
Comments: this header explicitly states the defaults

One line of text this time, but it ends in a line break.
--1995021309105517
Content-type: multipart/alternative; boundary=_
Comments: An encapsulated multipart message!

Again, this preamble is ignored. The multipart body contains a still image and a video image encoded in Base64. See Base64 Encoding

One feature is that the character "_" which is allowed in multipart boundaries never occurs in Base64 encoding so we can use a very simple boundary!
--_
Content-type: text/plain

This message contains images which cannot be displayed at your terminal. This is a shame because they're very nice.

--_
Content-type: image/jpeg
Content-transfer-encoding: base64
Comments: This photograph is to be shown if the user's system cannot display MPEG videos. Only part of the data is shown in this book because the reader is unlikely to be wearing MIME-compliant spectacles.

--_
Content-type: video/mpeg
Content-transfer-encoding: base64
Comments: This video is to be shown if the user's system cannot display images. Only part of the data is shown in this book because the reader is unlikely to be wearing MIME-compliant spectacles.

--_--
That was the end of the nested multipart message. This is the epilogue. Like the preamble it is ignored.
--1995021309105517--
And that was the end of the main multipart message. That's all folks!

9.2.2 **Appendix B.2: A Complex SMTP Header**

Date     :  27 Aug 76 0932 PDT
From     :  Ken Davis <KDavis@This-Host.This-net>
9.2.3 Appendix B.3: A Real SMTP Data Flow Example

User nelan at the host cmb.ac.lk sends a note to users faski and juski at the host yahoo.com. R is for Receiver and the S is for Sender.

R: 220 yahoo.com Simple Mail Transfer Service Ready
S: HELO cmb.ac.lk
R: 250 yahoo.com
S: MAIL FROM:<nelan@cmb.ac.lk>
R: 250 OK
S: RCPT TO: <faski@yahoo.com>
R: 250 OK
S: RCPT TO: <juski@yahoo.com>
R: 550 No Such User Here
S: DATA
R: 354 Start Mail Input, End With <CRLF>,<CRLF>
S: Date: 4 Feb 03 12:20:56
S: From: Nelan W. <nalin@cmb.ac.lk>
S: Subject: Lecture Notes
S: To: <faski@yahoo.com>
S: cc: <juski@yahoo.com>
S: Collect Your Lecture Notes
S: from the ...
S: .
R: 250 OK
S: QUIT
R: 221 yahoo.com Service Closing Transmission Channel

9.3 Appendix C – Framework Code Listing

```java
package nautil;

import java.util.Random;

public class Utility {
    private DateUtility d = new DateUtility();
```
public StringBuffer toHex(byte[] data) {
    StringBuffer hex = new StringBuffer();
    for (int i=0; i < data.length; i++) {
        hex.append(Integer.toHexString(0xFF & data[i]));
        hex.append(" ");
    }
    return hex;
}

public StringBuffer toString(byte[] data) {
    StringBuffer string = new StringBuffer();
    for (int i=0; i < data.length; i++)
        string.append(data[i]);
    return string;
}

public int toInt(String number, int defaultvalue) {
    try {
        return Integer.parseInt(number);
    } catch (Exception e) {
        return defaultvalue;
    }
}

public byte[] concatArray(byte[] arrayA, byte[] arrayB) {
    byte[] arrayC = new byte[arrayA.length+arrayB.length];
    System.arraycopy(arrayA,0,arrayC,0,arrayA.length);
    System.arraycopy(arrayB,0,arrayC,arrayA.length,arrayB.length);
    return arrayC;
}

public String getSerial(String prefix) {
    Random rnd = new Random(System.currentTimeMillis());

    long date = d.getUTC();
    float randomA = rnd.nextFloat();
    String user = System.getProperty("user.name") + "wxyz";
    int randomB = rnd.nextInt(100);

    StringBuffer sb = new StringBuffer();
    sb.append(prefix);
    sb.append(date);
    sb.append(randomA);
    sb.append(user.substring(0,3));
    sb.append(randomB);

    return sb.toString();
}

public String getEnclosed(String pre, String post, String in) {
    int preIndex = in.indexOf(pre);
    int postIndex = in.lastIndexOf(post);
    if (preIndex>=0 && postIndex>=0 && preIndex<postIndex)
        return in.substring(preIndex+1,postIndex);
    else
        return null;
}

The SecUtility Class
package nautil;

import java.security.*;

public class SecUtility {

    public byte[] digest(byte[] data) {
        try {
            MessageDigest algorithm = MessageDigest.getInstance("MD5");
            algorithm.reset();
            algorithm.update(data);
            return algorithm.digest();
        } catch (Exception e) {
            return null;
        }
    }

    public byte[] sign(byte[] data, PrivateKey prv) {
        try {
            Signature sig = Signature.getInstance("DSA");
            sig.initSign(prv);
            sig.update(data);
            return sig.sign();
        } catch (Exception e) {
            return null;
        }
    }

    public boolean verify(byte[] data, byte[] signature, PublicKey pub) {
        try {
            Signature sig = Signature.getInstance("DSA");
            sig.initVerify(pub);
            sig.update(data);
            return sig.verify(signature);
        } catch (Exception e) {
            return false;
        }
    }
}

The DateUtility Class

package nautil;

import java.util.*;
import java.text.DateFormat;

public class DateUtility {

    public String getDate() {
        Date d = new Date();
        DateFormat df = DateFormat.getDateInstance();
        return df.format(d);
    }

    public String getDate(Date d) {
        DateFormat df = DateFormat.getDateInstance();
        return df.format(d);
    }

    public String getDate(long UTC) {

Date d = new Date(UTC);
DateFormat df = DateFormat.getDateInstance();
return df.format(d);
}

public String getDate(int year, int month, int date) {
Calendar calendar = Calendar.getInstance();
DateFormat df = DateFormat.getDateTimeInstance();
calendar.set(year, month-1, date);
return df.format(calendar.getTime());
}

public String getTime() {
Date d = new Date();
DateFormat df = DateFormat.getTimeInstance();
return df.format(d);
}

public String getTime(Date d) {
DateFormat df = DateFormat.getTimeInstance();
return df.format(d);
}

public String getTime(long UTC) {
Date d = new Date(UTC);
DateFormat df = DateFormat.getTimeInstance();
return df.format(d);
}

public String getDateTime() {
Date d = new Date();
DateFormat df = DateFormat.getDateTimeInstance();
return df.format(d);
}

public String getDateTime(Date d) {
DateFormat df = DateFormat.getDateTimeInstance();
return df.format(d);
}

public String getDateTime(long UTC) {
Date d = new Date(UTC);
DateFormat df = DateFormat.getDateTimeInstance();
return df.format(d);
}

public long getUTC() {
Date d = new Date();
return d.getTime();
}

public long getUTC(int year, int month, int date) {
Calendar calendar = Calendar.getInstance();
DateFormat df = DateFormat.getDateTimeInstance();
calendar.set(year, month-1, date);
return calendar.getTime().getTime();
}

public int getYEAR() {
Calendar calendar = Calendar.getInstance();
Date date = new Date();
calendar.setTime(date);
return calendar.get(calendar.YEAR);
}

public int getYEAR(Date date) {
Calendar calendar = Calendar.getInstance();
calendar.setTime(date);
return calendar.get(calendar.YEAR);
}

public int getDAY_OF_MONTH() {
Calendar calendar = Calendar.getInstance();
Date date = new Date();
calendar.setTime(date);
return calendar.get(calendar.DAY_OF_MONTH);
}

public int getDAY_OF_MONTH(Date date) {
Calendar calendar = Calendar.getInstance();
calendar.setTime(date);
return calendar.get(calendar.DAY_OF_MONTH);
}

public int getMONTH() {
Calendar calendar = Calendar.getInstance();
Date date = new Date();
calendar.setTime(date);
return calendar.get(calendar.MONTH)+1;
}

public int getMONTH(Date date) {
Calendar calendar = Calendar.getInstance();
calendar.setTime(date);
return calendar.get(calendar.MONTH)+1;
}

public int getHOUR_OF_DAY() {
Calendar calendar = Calendar.getInstance();
Date date = new Date();
calendar.setTime(date);
return calendar.get(calendar.HOUR_OF_DAY);
}

public int getMINUTE() {
Calendar calendar = Calendar.getInstance();
Date date = new Date();
calendar.setTime(date);
return calendar.get(calendar.MINUTE);
}

public int getSECOND() {
Calendar calendar = Calendar.getInstance();
Date date = new Date();
calendar.setTime(date);
return calendar.get(calendar.SECOND);
}

}

The ProcessQuery Class

package nautil;
import java.sql.*;

public class ProcessQuery {
    private String dbURL = "jdbc:odbc:jnadb";
    private String user = "";
    private String password = "";

    public ResultSet getResults(String query) {
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection c = DriverManager.getConnection(dbURL, user, password);
            Statement s = c.createStatement();
            ResultSet r = s.executeQuery(query);
            return r;
        } catch (Exception e) {
            return null;
        }
    }

    public int justUpdate(String query) {
        int i;
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection c = DriverManager.getConnection(dbURL, user, password);
            Statement s = c.createStatement();
            i = s.executeUpdate(query);
            s.close();
            c.close();
        } catch (Exception e) {
            i = -1;
        }
        return i;
    }

    public boolean isExist(String query) {
        boolean exist = false;
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection c = DriverManager.getConnection(dbURL, user, password);
            Statement s = c.createStatement();
            ResultSet r = s.executeQuery(query);
            if (r.next()) exist = true;
            s.close();
            c.close();
            return exist;
        }
    }

    public String getString(String query, String field) {
        String result = null;
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection c = DriverManager.getConnection(dbURL, user, password);
            Statement s = c.createStatement();
            ResultSet r = s.executeQuery(query);
            if (r.next()) result = r.getString(field);
            s.close();
            c.close();
        } catch (Exception e) {
            System.out.println("Exception!");
        }
        return result;
    }
}
public boolean getBoolean(String query, String field) {
    boolean result = false;
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection c = DriverManager.getConnection(dbURL, user, password);
        Statement s = c.createStatement();
        ResultSet r = s.executeQuery(query);
        if (r.next()) result = r.getBoolean(field);
        s.close(); c.close();
    } catch (Exception e) {
        System.out.println("Exception!");
    }
    return result;
}

The ExtractFeature Class

package nautil;

import java.util.*;

public class ExtractFeature {
    private Map m = new HashMap();
    private String start, delimit, end;
    private String content;
    private String body;

    public ExtractFeature(String start, String delimit, String end) {
        this.start = start;
        this.delimit = delimit;
        this.end = end;
    }

    public void setContent(String content) {
        this.content = content;
        m.clear();

        int a=-1, b=-1, c=-1; int i=0;
        String option, value;

        while (a<content.length()) {
            if ((a=content.indexOf(start,a))==-1) { a=c+1; break; }
            if ((b=content.indexOf(delimit,a))==-1) { a=a+2; break; }
            if ((c=content.indexOf(end,b))==-1) { a=b+1; break; }

            if (b>a) {
                option = content.substring(a+2, b);
                if (c>b) {
                    value = content.substring(b+1, c);
                    m.put(option, value);
                    i++;
                }
                a=c;
            }
        }
    }
}
The TimeStamp Class

package nautil;

public class TimeStamp {
  private Utility utility = new Utility();

  private byte[] messageDigest;
  private String issuerUniqueID;
  private String timeServerUniqueID;
  private String PKCertificateSerialNo;
  private String timePrecision;
  private long UTCTime;
  private byte[] timeStamp;

  public TimeStamp(byte[] messageDigest,
                  String issuerUniqueID,
                  String timeServerUniqueID,
                  String PKCertificateSerialNo,
                  String timePrecision,
                  long UTCTime,
                  byte[] timeStamp) {
    this.messageDigest = messageDigest;
    this.issuerUniqueID = issuerUniqueID;
    this.timeServerUniqueID = timeServerUniqueID;
    this.PKCertificateSerialNo = PKCertificateSerialNo;
    this.timePrecision = timePrecision;
    this.UTCTime = UTCTime;
    this.timeStamp = timeStamp;
  }

  public long getUTCTime() {
    return UTCTime;
  }

  public byte[] getTimeStamp() {
    return timeStamp;
  }

  public String toString() {
    StringBuffer sb = new StringBuffer();
    sb.append("Message Digest = "+utility.toHex(messageDigest)+"\n");
    sb.append("Issuer Unique ID = "+issuerUniqueID+"\n");
    sb.append("Timeserver Unique ID = "+timeServerUniqueID+"\n");
    sb.append("PK Certificate Serial No = "+PKCertificateSerialNo+"\n");
    sb.append("UTC Time = "+UTCTime+"\n");
    sb.append("Time Precision = "+timePrecision+"\n");
    sb.append("Timestamp = "+utility.toHex(timeStamp));
    return sb.toString();
  }
}
public byte[] getAll() {
    StringBuffer sb = new StringBuffer();
    sb.append(issuerUniqueID);
    sb.append(timeServerUniqueID);
    sb.append(PKCertificateSerialNo);
    sb.append(UTCTime);
    sb.append(timePrecision);
    return utility.concatArray(
            messageDigest, utility.concatArray(
                    sb.toString().getBytes(), timeStamp));
}

public String getTimeServerUniqueID() {
    return timeServerUniqueID;
}

public String getTimePrecision() {
    return timePrecision;
}

public String getPKCertificateSerialNo() {
    return PKCertificateSerialNo;
}

public byte[] getMessageDigest() {
    return messageDigest;
}

public String getIssuerUniqueID() {
    return issuerUniqueID;
}

public TimeStamp timeStamp(byte[] data, boolean useDigest, Date date, PrivateKey prv) {
    byte[] digest;
    if (useDigest) digest = s.digest(data);
    else {
        digest = new byte[data.length];
        System.arraycopy(data, 0, digest, 0, data.length);
    }

    // get UTC date as byte array
    long UTC = date.getTime();
    StringBuffer sb = new StringBuffer();
    sb.append(UTC);
    return utility.concatArray(digest, sb.toString().getBytes());
}
sb.append(UTC);
byte[] dateA = sb.toString().getBytes();

// sign the compound
byte[] sig = s.sign(u.concatArray(digest, dateA), prv);

return new TimeStamp(digest, issuerUniqueID, timeServerUniqueID,
PKCertificateSerialNo,
    timePrecision, UTC, sig);
}

public boolean verifyTimeStamp(TimeStamp timestamp,
    PublicKey pub) {
    byte[] digest = timestamp.getMessageDigest();
    long UTC = timestamp.getUTCTime();
    StringBuffer sb = new StringBuffer();
    sb.append(UTC);
    byte[] date = sb.toString().getBytes();
    return s.verify(u.concatArray(digest, date), timestamp.getTimeStamp(),
    pub);
}

public void setIssuerUniqueID(String issuerUniqueID) {
    this.issuerUniqueID = issuerUniqueID;
}

public void setPKCertificateSerialNo(String PKCertificateSerialNo) {
    this.PKCertificateSerialNo = PKCertificateSerialNo;
}

public void setTimePrecision(String timePrecision) {
    this.timePrecision = timePrecision;
}

public void setTimeServerUniqueID(String timeServerUniqueID) {
    this.timeServerUniqueID = timeServerUniqueID;
}
}

---

The DN Class

package nautil;

public class DN {

    private String CN; // Common Name
    private String OU; // Organization Unit
    private String o; // Organization
    private String l; // Locality
    private String s; // State
    private String c; // Country
    private String e; // Email

    public DN(String CN,
        String OU,
        String o,
        String l,
        String s,
        String c,
        String e) {
        this.CN = CN;
        this.OU = OU;
        this.o = o;
        this.l = l;
        this.s = s;
        this.c = c;
        this.e = e;
    }
}
package nautil;

public class MessageHeader {

    private String messageID;
    private String from;

    public String toString() {
        StringBuffer sb = new StringBuffer();
        sb.append(" CN = "+CN);
        sb.append(" , OU = "+OU);
        sb.append(" , O = "+o);
        sb.append(" , L = "+l);
        sb.append(" , S = "+s);
        sb.append(" , C = "+c);
        sb.append(" , E = "+e);
        return sb.toString();
    }

    public byte[] getAll() {
        StringBuffer sb = new StringBuffer();
        sb.append(CN);
        sb.append(OU);
        sb.append(o);
        sb.append(l);
        sb.append(s);
        sb.append(c);
        sb.append(e);
        return sb.toString().getBytes();
    }

    public String getC() {
        return c;
    }

    public String getCN() {
        return CN;
    }

    public String getE() {
        return e;
    }

    public String getL() {
        return l;
    }

    public String getO() {
        return o;
    }

    public String getOU() {
        return OU;
    }

    public String getS() {
        return s;
    }

};
private String to;
private String cc;
private String bcc;
private String subject;
private String date;

public MessageHeader(String messageID,
String from,
String to,
String cc,
String bcc,
String subject,
String date) {
    this.messageID = messageID;
    this.from = from;
    this.to = to;
    this.cc = cc;
    this.bcc = bcc;
    this.subject = subject;
    this.date = date;
}

public String toString() { 
    StringBuffer sb = new StringBuffer();
    sb.append(" Message-ID = "+messageID);
    sb.append(" From = "+from);
    sb.append(" To = "+to);
    sb.append(" Cc = "+cc);
    sb.append(" Bcc = "+bcc);
    sb.append(" Date = "+date);
    sb.append(" Subject = "+subject);
    return sb.toString();
}

public byte[] getAll() {
    StringBuffer sb = new StringBuffer();
    sb.append(messageID);
    sb.append(from);
    sb.append(to);
    sb.append(cc);
    sb.append(bcc);
    sb.append(date);
    sb.append(subject);
    return sb.toString().getBytes();
}

public String getBcc() { 
    return bcc;
}

public String getCc() { 
    return cc;
}

public String getDate() { 
    return date;
}

public String getFrom() { 
    return from;
}

public String getMessageID() { 
    return messageID;
}

public String getSubject() { 
    return subject;
}
Notarization Authority for Emails
H.E.M.H.B. Ekanayake

The MessageCertificate Class

```java
package nautil;

public class MessageCertificate{
    private Utility u = new Utility();

    private int version;
    private String serialNumber;
    private String purpose;
    private String signatureAlgorithm;
    private DN issuer;
    private DN subject;
    private long notAfter;
    private long notBefore;
    private TimeStamp x_Timestamp;
    private MessageHeader x_MessageHeader;
    private byte[] x_MessageBody;
    private String x_MessageStatus;
    private byte[] signature;

    public MessageCertificate(int version,
        String serialNumber,
        String purpose,
        String signatureAlgorithm,
        DN issuer, DN subject,
        long notAfter,
        long notBefore,
        TimeStamp timestamp,
        MessageHeader messageHeader,
        byte[] messageBody,
        String messageStatus) {
        this.version = version;
        this.serialNumber = serialNumber;
        this.purpose = purpose;
        this.signatureAlgorithm = signatureAlgorithm;
        this.issuer = issuer;
        this.subject = subject;
        this.notBefore = notBefore;
        this.notAfter = notAfter;
        this.x_Timestamp = timestamp;
        this.x_MessageHeader = messageHeader;
        this.x_MessageBody = messageBody;
        this.x_MessageStatus = messageStatus;
    }

    public String toString() {
        Utility u = new Utility();
        DateUtility d = new DateUtility();
        StringBuffer sb = new StringBuffer();
        sb.append("Version\t"+version+"\n");
        sb.append("Serial number\t"+serialNumber+"\n");
```

- 91 -
package nauti;

import java.security.*;

public class MessageCertificator {
    private Utility u = new Utility();
    private SecUtility s = new SecUtility();
    private DateUtility d = new DateUtility();

    public MessageCertificate issue(String purpose,
            DN issuer, DN subject, TimeStamp timestamp, MessageHeader messageheader, byte[] messagebody,
            boolean useDigest, String messagestatus, PrivateKey prv) {

        StringBuffer sb = new StringBuffer();
        sb.append("Purpose\t"+purpose+"\n");
        sb.append("Signature algorithm\t"+signatureAlgorithm+"\n");
        sb.append("Issuer\t"+issuer+"\n");
        sb.append("Valid from\t"+d.getDateTime(notBefore)+"\n");
        sb.append("Valid to\t"+d.getDateTime(notAfter)+"\n");
        sb.append("Subject\t"+subject+"\n");
        sb.append("X-Timestamp\t"+x_Timestamp.toString()+"\n");
        sb.append("X-Message header\t"+x_MessageHeader.toString()+"\n");
        sb.append("X-Message body\t"+u.toHex(x_MessageBody)+"\n");
        sb.append("X-Message status\t"+x_MessageStatus+"\n");
        sb.append("Signature\t"+u.toHex(signature)+"\n");
        return sb.toString();
    }

    public byte[] getVerifier() {
        StringBuffer sb = new StringBuffer();
        sb.append(version);
        sb.append(serialNumber);
        sb.append(purpose);
        sb.append(signatureAlgorithm);
        sb.append(notBefore);
        sb.append(notAfter);
        sb.append(x_MessageStatus);
        return u.concatArray(sb.toString().getBytes(),u.concatArray(issuer.getAll(),
                u.concatArray(subject.getAll(),u.concatArray(x_Timestamp.getAll(),
                        u.concatArray(x_MessageHeader.getAll(),x_MessageBody)))));
    }

    public byte[] getAll() {
        return u.concatArray(getVerifier(),signature);
    }

    public byte[] getSignature() {
        return signature;
    }

    public void setSignature(byte[] signature) {
        this.signature = signature;
    }
}
String serial = u.getSerial("MS");
long notBefore = d.getUTC(); // starting from now
long notAfter = notBefore + (1000*3600*24); // valid for 1 day

byte[] digest;
if (useDigest) digest = s.digest(messagebody);
else {
    digest = new byte[messagebody.length];
    System.arraycopy(messagebody,0,digest,0,messagebody.length);
}

MessageCertificate ms = new
MessageCertificate(3,serial,purpose,"DSA",issuer,
subject,notAfter,notBefore,timestamp,messageheader,digest,messagestatus);

    ms.setSignature(s.sign(ms.getVerifier(),prv));
    return ms;
}

public boolean verify(MessageCertificate ms, PublicKey pub) {
    return s.verify(ms.getVerifier(),ms.getSignature(),pub);
}
public static void main(String[] args) {
    MessageCertificator messageCertificator1 = new MessageCertificator();
}
9.4 APPENDIX D – A SHORT USER MANUAL FOR WEB SYSTEM USERS

9.4.1 Signing Up

- Click Sign Up

**Sign Up Step 1&2**

- **Sign Up: Step-1**
  - Enter your email address and click **Submit**
  - **Note**: Now you will receive an activation key to your mailbox

**The Email You Receive**

---

**Notarization Authority for Emails**

To continue your registration process, please go back to http://10.16.32.6:8080/ and enter this activation key

ack10539254399910.8749167Aadm65

**Figure D.2: Activation Key Received by an email**
• Sign Up: Step-2
  o Enter your email address
  o Enter your activation key you received into your mailbox
  o Click Submit

Note: Now if details you provided are correct you will be transferred to the next step.

<table>
<thead>
<tr>
<th>Sign Up Step 3</th>
</tr>
</thead>
</table>

Figure D.3: Sign Up Step 3

• Sign Up: Step-3
  o Fill the form by entering your details
  o Check the box labeled “Agree” to agree to the terms and conditions
  o Click Submit

Note: Now if the registration process succeeded you will see a nice message; otherwise you will see a page indicating what have to be modified.

Registration Complete!
Registration Completion

Registration process successfully completed. Make sure the following information about you are correct. You can correct any mistake after signing in.

- Username: hiran12
- Full Name: 1
- Department/Class: 1
- Organization/School: 1
- City: 1
- State/Province: 1
- Country: 1

Figure D.4: Registration Completion

9.4.2 Signing In

Sign In

User Name:  
Password:  
Submit

Free Services

The following are additional free services provided by the Notarization Authority:
- Issue Timestamps
- Verify Timestamps
- Issue Proof of Certificates
- Verify Proof of Certificates

Visit [http://www.ucec.cmb.ac.lk/](http://www.ucec.cmb.ac.lk/)

Figure D.5: Signing In

- Enter username and password
- Click Submit
9.4.3 Sending an Email: Microsoft Outlook

- At the Outlook click New Mail
- On the main menu click Format > Plain Text
- Enter Notarization Authority’s email address as To:
- Complete the Subject field
- Specify MailTo and other options on the body and enter your actual body after completing.
- Click Send
9.4.4 Sending an Email: Web Mail

- Click **Compose** from the menu displayed on the Outbox

**Sending a Web mail**

- Complete the form

**Figure D.8: Sending an Email: Web mail**
Notarization Authority for Emails
H.E.M.H.B. Ekanayake

- Select options as you wish
- Click Send

9.4.5 Modifying an Email
- From the Outbox click on the link of the email you want to modify
- Modify the fields as you wish
- Click Send

9.4.6 Obtaining a Proof-of Certificate
- Click Issue Proof-of Certificates

Obtaining a Proof-of Certificate

- Enter message-Id of the email and select the type of the certificate you need. Also enter your email address
- Click Submit

Note: You will receive the certificate via an email