2007

Security Integrated Messaging: a protocol for secure electronic mail

Andrew Todd Hoernecke

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Computer Sciences Commons

Recommended Citation
https://lib.dr.iastate.edu/rtd/15022

This Thesis is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
Security Integrated Messaging: A protocol for secure electronic mail

by

Andrew Todd Hoernecke

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Co-majors: Information Assurance; Computer Engineering

Program of Study Committee:
Doug Jacobson, Major Professor
Thomas Daniels,
Clifford Bergman

Iowa State University

Ames, Iowa

2007

Copyright © Andrew Todd Hoernecke, 2007. All rights reserved.
# TABLE OF CONTENTS

LIST OF FIGURES ................................................................................................................. iv
LIST OF TABLES ................................................................................................................... vi
ABSTRACT ............................................................................................................................ vii
CHAPTER 1.  INTRODUCTION ............................................................................................ 1
CHAPTER 2.  RELATED WORK ........................................................................................... 6
  2.1 Current Technologies .......................................................................................... 6
  2.2 Recent Developments ....................................................................................... 8
CHAPTER 3.  MOTIVATION ............................................................................................... 11
CHAPTER 4.  APPROACH .............................................................................................. 13
  4.1 Goals .................................................................................................................... 13
  4.2 New Messaging Protocols ................................................................................ 16
  4.3 Mail Clients ......................................................................................................... 18
  4.4 Certificate Infrastructure ................................................................................... 20
CHAPTER 5.  DESIGN ........................................................................................................ 22
  5.1 Architecture ......................................................................................................... 22
  5.2 Certificates .......................................................................................................... 29
  5.3 Mail Addresses (Aliases) .................................................................................. 34
  5.4 Encryption ............................................................................................................ 34
  5.5 Digital Signatures ............................................................................................... 38
CHAPTER 6.  USE CASES AND PROCEDURES ............................................................... 41
  6.1 Certificate Server ................................................................................................ 41
  6.2 Mail Server .......................................................................................................... 52
  6.3 Mail Client ............................................................................................................ 55
CHAPTER 7.  CONCLUSION ............................................................................................... 59
APPENDIX A.  CREATING A NEW USER CERTIFICATE .............................................. 60
APPENDIX B.  RETRIEVING A CERTIFICATE ............................................................... 63
APPENDIX C.  VERIFYING A CERTIFICATE .................................................................. 66
APPENDIX D.  SENDING/RECEIVING A MAIL MESSAGE ........................................ 69
APPENDIX E.  SAMPLE CERTIFICATE ........................................................................ 76
APPENDIX F.  CERTIFICATE EXTENSION DEFINITION ............................................ 77
APPENDIX G.  XML COMMAND SPECIFICATION ..................................................... 78
  G.1 SigningRequestAuthentication.xsd ....................................................................... 78
  G.2 SigningResponseAuthentication.xsd ................................................................. 78
  G.3 SigningRequest.xsd ............................................................................................. 79
  G.4 SigningResponse.xsd .......................................................................................... 79
LIST OF FIGURES

Figure 1. Standard E-mail Architecture ................................................................................... 3
Figure 2. Example of Digitally Signed E-mail ........................................................................ 4
Figure 3. SSL for Web Traffic ................................................................................................. 15
Figure 4. Simplified E-mail Path with Possible Protocols ..................................................... 17
Figure 5. Inbox Screenshot Mockup ...................................................................................... 19
Figure 6. Basic Architecture .................................................................................................. 23
Figure 7. Certificate Servers Can Only Create Certain Certificates ...................................... 25
Figure 8. Mail Server Functions ............................................................................................ 27
Figure 9. Client functions ...................................................................................................... 28
Figure 10. Asymmetric Encryption ......................................................................................... 36
Figure 11. Symmetric Key Encryption .................................................................................... 37
Figure 12. Encrypting a Message to be Sent ......................................................................... 38
Figure 13. Decrypting a Received Message ........................................................................... 38
Figure 14. Digital Signature Creation .................................................................................... 39
Figure 15. Checking a Digital Signature ................................................................................ 40
Figure 16. Sample PKCS#10 Certification Request ............................................................... 43
Figure 17. Path from Alias to User ........................................................................................ 45
Figure 18. Step 1 of Creating a New Certificate .................................................................... 60
Figure 19. Step 2 of Creating a New Certificate .................................................................... 60
Figure 20. Step 3 of Creating a New Certificate .................................................................... 61
Figure 21. Step 4 of Creating a New Certificate .................................................................... 61
Figure 22. Step 5 of Creating a New Certificate .................................................................... 61
Figure 23. Step 6 of Creating a New Certificate .................................................................... 62
Figure 24. Step 7 of Creating a New Certificate .................................................................... 62
Figure 25. Step 1 of Retrieving a Certificate ......................................................................... 63
Figure 26. Step 2 of Retrieving a Certificate ......................................................................... 64
Figure 27. Step 3 of Retrieving a Certificate ......................................................................... 64
Figure 28. Step 4 of Retrieving a Certificate ......................................................................... 65
Figure 29. Step 1 of Verifying a Certificate .......................................................................... 66
Figure 30. Step 2 of Verifying a Certificate .......................................................................... 67
Figure 31. Step 3 of Verifying a Certificate .......................................................................... 67
Figure 32. Step 4 of Verifying a Certificate .......................................................................... 68
Figure 33. Message Travel Flowchart ................................................................................... 69
Figure 34. Step 1 of Sending a Message ................................................................................ 70
Figure 35. Step 2 of Sending a Message ................................................................................ 70
Figure 36. Step 3 of Sending a Message ................................................................................ 71
Figure 37. Step 4 of Sending a Message ................................................................................ 71
Figure 38. Step 5 of Sending a Message ................................................................................ 72
Figure 39. Step 6 of Sending a Message ................................................................................ 72
Figure 40. Step 7 of Sending a Message ................................................................................ 73
Figure 41. Step 8 of Sending a Message ................................................................................ 73
Figure 42. Step 9 of Sending a Message ................................................................................ 74
Figure 43. Step 10 of Sending a Message................................................................. 74
Figure 44. Step 11 of Sending a Message................................................................. 75
Figure 45. Certificate Signing Request Flowchart.................................................... 92
Figure 46. Certificate Retrieval Flowchart .............................................................. 93
Figure 47. Certificate Verification Flowchart......................................................... 94
Figure 48. Send Message Flowchart....................................................................... 94
Figure 49. Mail Client to Mail Server Connection Flowchart............................... 95
LIST OF TABLES

Table 1. Certificate Types...................................................................................................... 31
Table 2. SigningAuth Values................................................................................................. 33
Table 3. Example DNs........................................................................................................... 45
ABSTRACT

Although e-mail has proven to be a powerful medium for message and information exchange, the protocols currently in use have major security shortfalls. As more and more people have turned to e-mail as a means of communication, problems experienced by e-mail users have dramatically increased. Phishing, spam, and identity theft are just three of the problems people experience because of the insecure nature of current e-mail protocols. Furthermore, as the amount of confidential information transmitted over e-mail continues to rise, these and other problems will only get worse. Although many attempts have been made to patch the issues, none have seen great amounts of success. This is most likely due to a combination of the difficulty of implementing these security improvements, the absence of a single standard for secure e-mail, and the general lack of understanding by most e-mail users. Security Integrated Messaging is a new protocol for secure message exchange, designed to solve these problems. This system uses several proven technologies, such as X.509 certificates, symmetric and asymmetric encryption, and digital signatures in order to provide confidentiality, integrity, and authentication to an e-mail-like system. This system represents a first step in revamping the way we communicate online in order to protect our digital information.
CHAPTER 1. INTRODUCTION

Over the past few decades our society has seen a massive shift in how and why people communicate. The Internet has provided us with tools that now seem indispensable parts of daily life but just a few short decades ago would have been unheard of. One of the most significant developments in this time has been the gradual shift away from traditional paper mail systems in favor of a faster, cheaper, more convenient alternative—e-mail.

The advantages of e-mail make it incredibly powerful even for inexperienced computer users: messages are delivered almost instantly; attachments can be sent in a digital form that can easily be viewed, manipulated, or shared; and software has made it much easier to keep track of, organize, and search e-mail than its paper equivalent. These abilities, among others, have led to companies, governments, and individuals relying on e-mail for more and more important business and personal functions. Common examples of this include banks sending account statements via e-mail and online retailers advertising through e-mail such that users can easily purchase products right from their inbox. Additionally, businesses such as credit card companies and utility providers are allowing customers to receive an “e-bill” through their e-mail instead of a traditional paper bill sent through the postal mail service. Unfortunately, e-mail is not well suited for these and many other tasks. Although the functionality and convenience is there, as is often the case with newer technologies, security is lagging significantly behind.

Three basic concepts of information security—confidentiality, integrity, and sender/receiver authentication—can be applied to e-mail to demonstrate that it does not live up to the expectations of a secure system. With the traditional e-mail system none of these
concerns are addressed by default. Although strides have been made to create add-ons and modifications to existing technologies and protocols, these seem to have done little to protect the average user[1]. This is demonstrated by the fact that confidential information is routinely transferred unencrypted[2]; phishing has become a rampant problem costing consumers and companies billions of dollars every year[3]; and spam and viruses sent through e-mail are also out of control[4, 5]. The unfortunate truth is that even if a company or individual wanted to be responsible by providing confidentiality, integrity, and trusted authentication with every message sent, currently it is just not possible[6].

As an example of this, imagine a user, Bob, who is concerned with the confidentiality of his messages. Bob works at a bank and sends out account statements to many of the bank’s clients. In order to be able to encrypt all of these statements, each bank customer would need to have a public/private key pair, and the public key would need to be accessible to the bank. There are two problems with this scenario. First, most users do not have a public/private key pair, so a campaign would have to be waged by the bank to get them to create such a pair. Secondly, even if a majority of the users did have a public/private key, there currently is no infrastructure channel readily available for those public keys to be easily or automatically distributed. Therefore, this best scenario solution is currently not possible.

Instead Bob wants to at least make sure that his e-mail messages are encrypted in transit from his computer to the recipients’ mail server. So he uses an e-mail client that allows him to connect to his mail server using an encrypted protocol. He also talks to his network administrator and asks her to ensure that the mail server is setup so he can connect to it securely. This will help prevent an eavesdropper from intercepting the message while it is being transmitted from his computer to the outgoing mail server. However, after this
transmission has completed, Bob has no control over how his message will be transferred. In fact, many mail servers and mail clients do not accommodate, or are not setup to use, encrypted protocols. Since Bob’s outgoing mail server will most likely have to retransmit each message at least once to the recipients’ incoming mail servers, the security of the messages will also depend on the security of the recipients’ mail servers and clients. So even if Bob has done his part, and the initial transmission from his mail client to his mail server may have been made securely, this is only a small link in a chain of often insecure communications. Figure 1 below shows that in standard e-mail, after a message leaves the message sender’s computer, it usually must travel through several more servers. In the figure, the traffic between the message sender and the first mail server is encrypted, as indicated by the lock icon; however, after this link the sender has no control over whether the message will be sent using an encrypted protocol.

![Figure 1. Standard E-mail Architecture](image)

For a second example, imagine Carol who works with an online auction company, auctions.com. Periodically auctions.com sends out e-mails to some their users asking them to update their personal information, informing them of auctions they may be interested in, or telling them about new features that have been added. Unfortunately because auctions.com is very popular, many malicious users have sent out e-mails that look very similar to actual auctions.com e-mails in order to trick users into giving out their credentials. Carol wants to
ensure that the recipients of her e-mails know that they actually came from auctions.com and that the message has not been tampered with. This is possible with some mail clients using digital signatures. Unfortunately, many mail clients do not know how to interpret these signatures, so the users are unable to verify the integrity of the message. An even bigger problem is that even though Carol may be a conscientious user who understands the security implications of using e-mail, many users do not even realize that it is possible to compromise e-mail message integrity or spoof a sender’s identity; nor do they know how to take precautions to help prevent this from occurring. Thus most users would not even know to look for a digital signature, would not know what a digital signature means, and would certainly not be able to tell the difference between a digital signature that verified the identity of the sender as opposed to one that only verified that the message had not been changed.

Figure 2 below shows how a digitally signed message appears in Gmail, a popular Internet e-mail service. The red boxes have been added to show what is added by including a digital signature. Although some e-mail clients can interpret these signatures and give users a clearer picture of what they mean, the digital signature in this message clearly would not help most users.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Hi...

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.6 (GNU/Linux)

iDQDBQfQKt+K5aGkEkd/BgwRAxSwAkJ9uFLMnQEyBT32fmoCBTTsYhmfACeCrQd
n4U2tzv+G3wR10pRbQj= =VGUL
-----END PGP SIGNATURE-----
```

Figure 2. Example of Digitally Signed E-mail
Although the root of e-mail’s security problems seems to be the ease at which a malicious user can masquerade as a trusted entity when sending e-mail, the real problem is larger than that. Because e-mail is not secure by default, it makes it very difficult to implement strong security measures, educate users about them, and bring about widespread use of security that would help prevent e-mail’s many problems and safeguard the messages that do need security. E-mail was not originally designed to be a secure system, and although add-ons and modifications have been created to address these problems, none have caught on or become easy enough to setup and use to provide strong security to the masses. It seems that until security is automatic and integrated into the system, instead of wrapped around insecure data and protocols, secure e-mail may never be widespread.

Because of the inherent weaknesses in current e-mail technologies, a new Internet messaging standard should be created by focusing on security and ease-of-use in order to enable Internet communication for the average user that integrates confidentiality, integrity, and two-way trusted authentication. This standard could use many existing technologies, such as symmetric encryption, digital signatures, and public key certificates combined in a way to provide confidentiality, integrity, and authentication without need requiring excessive setup or maintenance by the user.
CHAPTER 2. RELATED WORK

Because e-mail is so widely used and its problems so commonly experienced, many previous attempts have been made to make e-mail more secure. This section will describe some of the more popular methods currently available to secure e-mail, as well as some more recent developments in this area.

2.1 Current Technologies

There are several technologies currently available to users for securing their e-mail messages. These methods rely on the standard insecure e-mail protocols, but add a wrapper of security around the data in order to protect confidentiality or integrity.

2.1.1 OpenPGP

OpenPGP is a standard that can be used to encrypt and digitally sign messages using public-key and symmetric key cryptography methods. The OpenPGP standard itself is generally considered to be very secure and effective; however, problems do exist with OpenPGP that have led to it not being widely adopted or used.

Generally, using OpenPGP to encrypt or sign e-mails requires a plug-in to an e-mail client. Unfortunately many users are not knowledgeable enough to realize that they need a plug-in, nor skilled enough to know how to download, setup and install it. Because of this, most users of PGP are either very skilled users that understand the vulnerabilities in the e-mail system or corporate and government users who have been help by other knowledgeable users. Also, many Internet e-mail services do not support PGP at all. Figure 2 above shows an example of an e-mail service that cannot interpret PGP for the user. This means a
significant portion of users, including many small or medium-sized business users and home users, are left to fend for themselves. This is one reason it would be preferable to use e-mail protocols that integrate security from the start in an automatic way, instead of adding this functionality on as an afterthought.

Another problem with OpenPGP is you cannot necessarily definitively authenticate the sender of a message with a high level of trust[7]. This is because instead of a centralized certificate management, OpenPGP relies on a “web of trust” scheme. This works by allowing a user to have his public key signed and endorsed by other users who are willing to vouch for the former’s identity. While this scheme can work for some individuals, most users do not or cannot take the time required to build these trust relationships. Additionally, new users may not be immediately trusted until they have acquired a suitable reputation.

2.1.2 S/MIME

S/MIME is another standard that is available to add confidentiality, integrity, authentication, and non-repudiation to the current e-mail system [8, 9]. S/MIME is also considered to be secure; however, there are still obstacles preventing it from being implemented by the average user.

Unlike OpenPGP, S/MIME is integrated into many e-mail clients. The use of this functionality does require the creation of a certificate either by a public certificate authority or by an in-house authority. Because of this fact S/MIME is likely the closest method available today to truly secure e-mail. But as was discussed in the public/private key pair scenario, the creation of this certificate is not something that most users can do without some training or help from their IT support staff, and this is a major drawback of this mechanism.
Additionally, most personal certificates only bind the certificate to an e-mail address—not an actual personal or business identity[10]. The lack of authentication means S/MIME is still vulnerable to an attacker who creates an address that may appear legitimate, when in reality it is not. Because of the state of e-mail technology, it can be very difficult to tell the difference between e-mail messages that have been signed using a certificate attached to a verified identity as opposed to a certificate who owner has not verified their identity. This problem is at the heart of most e-mail security issues. If we intend to continue to use e-mail for confidential or sensitive information, it is important that we move towards a system that allows us to definitively determine the actual identity of the sender, instead of just the e-mail address and the claimed identity of the sender.

2.2 Recent Developments

Several companies and organizations have also been researching new ways to help secure the e-mail system. Two new security frameworks, DomainKeys Identified Mail and Sender-ID, will be discussed here. Both of these methods attempt to authenticate the domain from which the e-mail is being sent by using special DNS records; however, neither provides any new means for providing confidentiality or user-to-user authentication.

2.2.1 DomainKeys Identified Mail

DomainKeys Identified Mail (DKIM) is a combination of work by Yahoo! and Cisco that was created with the goal to design a server-to-server authenticated mail system[11]. DKIM's goals included not requiring new protocols or public key infrastructure (PKI). By not requiring new protocols or infrastructure, DKIM was not able to provide integrated user-
to-user authentication nor any form of confidentiality. DKIM can be used with other forms of user-to-user signatures and encryption schemes, such as S/MIME and PGP, but as mentioned before these options are generally too difficult to setup and use for many average users.

Although DKIM seems to be innovative work that would help to cut down some amounts of spam and phishing, it still relies on outdated protocols and technologies and is only a partial solution to e-mail's security problem. Problems arise because domains are allowed to sign their e-mails with their own certificates, so phishing could still occur and possibly appear *more* legitimate. For example, if a malicious user owns the ebay-security.com domain, they could create their own certificate and sign their own e-mails to make them look valid. With DKIM, the user would be assured that the e-mail did in fact come from the ebay-security.com domain so this may actual increase the likelihood that they would take this e-mail seriously. Proponents of DKIM hope that with this technique ISPs and e-mail providers will create a system to evaluate the trust level of each individual domain to help make a determination as to whether the e-mail is legitimate or not. Unfortunately, even if DKIM becomes widely used *and* if this new trust evaluation system was implemented succesfully, it would still be relatively easy for malicious users to start using random domains, such as m43dfhr4dc.com, for a period of time until it has eroded its trust level. Although a random domain would fool a smaller percentage of the population, there are certainly still users who do not pay attention to this information and would be fooled.

Further, DKIM relies on DNS for distributing public keys instead of utilizing certificate authorities (CAs). This allows DKIM to be implemented without new PKI; however, this also means that the systems still relies on the accuracy and security of DNS in
order to verify a domain's identity. This does not seem to add much security over a scheme commonly used today of using a reverse lookup on the sender's IP address. It is well known that DNS responses can be forged, and this would seem to compromise the security of either of these schemes[12].

2.2.2 Sender-ID

Microsoft has undertaken another effort to help fight spam called Sender-ID[13]. This framework is similar to DKIM in that it attempts to authenticate the domain from which an e-mail was sent, again using a special DNS record called a Sender Policy Framework (SPF) record. Using Sender-ID, upon receiving an e-mail the inbound mail server uses DNS to lookup the SPF record. If the IP address of the outbound mail server matches any of the IP addresses listed in the domain's SPF record, this verifies the identity of the outbound mail server's domain.

This scheme has problems similar to those of DKIM. The Sender-ID framework still does nothing to address the needs of confidentiality or user-to-user authentication. Also, the Sender-ID authentication is still reliant on the security of DNS and the domain itself is again responsible for creating and maintaining the SPF records. Without a having a third-party involved to help determine the legitimacy of the domain sending the e-mail, the ebay-security.com problem mentioned previously would still be possible. Finally, Microsoft's Sender-ID scheme does not use digital signatures or any other form of PKI in order to verify the integrity of the message or the IP address of the outgoing mail server.
CHAPTER 3. MOTIVATION

As previously mentioned, solutions are available (although not widely implemented) to solve many of the problems with e-mail today. However, current solutions are too difficult to implement and understand for most computer users. Further, current e-mail technologies seem to place too much burden on the users who too often are not able to setup, maintain, or understand the security mechanisms that could keep their information secure. In fact, according to one estimate, 90% of businesses have failed to enable the SSL or TLS features of simple mail transfer protocol (SMTP)[14]. Although using SSL would not provide complete security to e-mail, it would at least provide encryption when messages are transferred between mail servers.

These problems have lead to these security technologies only being adopted and used by a relatively small number of e-mail users, generally those who are disproportionately experienced and knowledgeable in the area of information security. For the most part, computer security experts are not the individuals who fall prey to phishing and other e-mail spoofing problems; instead it is the average computer users who are unaware of the security problems and ways to mitigate these risks that are unable to protect themselves.

Further complications arise because all previously developed solutions to the problems with e-mail seem to rely on modifications or add-ons to the old, insecure protocols. Although it is necessary to recognize that major changes to protocols are expensive, causing significant increases to the learning curve and labor, it seems that in order to improve the security of e-mail, we must recognize that the current e-mail technology is broken in terms of security, and it may be time to look at a new solution. Now seems to be an opportune
moment for the development of such a solution because of the international attention being
given to information security problems. E-mail has become a ubiquitous technology, and if
the general public better understood the risks of e-mail, most would likely be ready for a new
solution.

While a partial solution to this problem would be to just refrain from sending
confidential information through e-mail, this seems unlikely to happen. Many people do not
understand the security risks of using e-mail for confidential information, and even though
many organizations such have banks have made it their policy not to send confidential
information through e-mail, their users still fall prey to phishing attacks. Regardless of the
security threats, many industry experts expect e-mail to be the default method of
communication for confidential data by 2009[15]. For these reasons, it seems imperative that
a better solution be developed that allows messages to be sent in a confidential manner, while
verifying the integrity of the messages and the actual identity of the sender.
CHAPTER 4. APPROACH

This section will explain and justify the approach and goals that will be used in order to find a resolution to the problem of e-mail security. First, this new system involves a new protocol that inherently requires all communications to be secure. This was done through the addition of a specialized certificate server and certificate extension that can be used to verify the identity of senders as well as their relationship with organizations such as businesses or the government. In addition, several mail client implementation suggestions will be given in order to maximize the usefulness of the new protocols and security features without requiring more work by the user. Putting research into these areas simultaneously seemed necessary because computer security starts and ends with the users, and information security professionals must find a way to make the technology “fit” the users (because the users are rarely able to fit the technology). Also, a major inadequacy of current e-mail technologies is their lack of “in-depth” security. In order to secure the e-mail system, a system must be developed that integrates security in every step of the process without requiring additional work from the users.

4.1 Goals

There are three main goals that will be used in creating this new e-mail system: creating a system that is automatically very secure, ensuring ease of use even for non-technical users, and allowing extensibility for future development.

Because security is a main goal, it is clear that use of existing e-mail protocols would not be acceptable. These protocols are very insecure by default and this makes it difficult to
say with any certainty that a message can be transmitted in a confidential fashion. Current methods of encrypting e-mail generally include one of two approaches: using PKI to encrypt the message at the time it is sent or using a wrapper of encryption when the message is actually transmitted. While both of these methods are useful, neither are being widely implemented and therefore are ineffective.

The first method generally requires the recipient to have a pre-established public/private key pair that can be used to encrypt the message, and the public key also must be available to the sender. Although this idea is not bad, and, in fact, is part of the solution presented in this paper, most e-mail users currently do not have key pairs, so this method is currently only effective for a small number of individuals. This is because of a combination of factors including lack of knowledge, the difficulty surrounding obtaining certificates, lack of effective infrastructure for sharing public keys, lack of technical skill required to correctly configure a mail client, and the simple fact that key pairs and certificates are not required to use e-mail and thus seem unnecessary.

The second method commonly used to protect e-mail is using an encryption wrapper. While this works well for some tasks, such as transferring web pages using secure sockets layer (SSL) or transport layer security (TLS)[16] between the client and the server, e-mail is a more complicated program which usually requires transferring data through multiple servers before delivering data to the recipients. In order for SSL or TLS to start to be effective for protecting message confidentiality, all (or at least most) mail servers would need to have this functionality enabled and have keys generated. Since, as previously mentioned, most SMTP server administrators have not taken this step, and there is no way for a user to guarantee that a message would be encrypted by all servers on the way to the message’s
destination, this approach does not translate well for protecting e-mail.

Another reason SSL works well for web traffic is that when a web page is requested there are generally only two parties that are communicating directly, the web server and the web client. Both parties can negotiate encryption methods at the beginning of their connection. In this type of a connection, the client is generally the last stop for the data. With e-mail however, there are usually at least four parties: the sender’s mail client, the sender’s outgoing mail server, the receiver’s incoming mail server, and the recipient’s mail client. Since neither the sender nor the recipient can control the communication through all of these steps, they cannot even ensure that encryption will be used at each transmission. And since most mail servers do not encrypt the data they are transmitting, this method does not adequately protect message confidentiality. The new protocols designed address these problems and help ensure that every message is properly encrypted for confidentiality and digitally signed for integrity and authentication.

Figure 3 below shows that a web client can ensure that traffic is encrypted all the way from itself to the web server and back. Compare this to Figure 1 on page 3 above which shows how a standard e-mail client cannot be sure of this.

![Figure 3. SSL for Web Traffic](image)

As previously mentioned, the second goal of the new e-mail system is usability. This will be accomplished through new certificate handling procedures and mail client designs.
The new system should allow a new user to setup their secure e-mail with little or no additional effort, although it will require the additional step of requiring a new user to verify their identity before setting up a mail account. In addition, the system will provide a way for a message recipient to know not only the identity of the user that sent the message, but will also automatically show the recipient whether the sender is from a business, government entity, or just sending a personal message.

Finally, extensibility will be important to allow for new functionality and different uses. Although e-mail is currently extremely popular, recent surveys show, especially in younger generation, a trend towards more instant messaging-type technologies, both computer-based and mobile device-based. Hopefully, the new message system should translate with minor modifications into an instant messaging environment. The main limiting fact for this would be the processing and storage requirements of the messaging client. After implementation further research would be needed to gauge these requirements.

### 4.2 New Messaging Protocols

Central to the development of a new messaging system will be a new protocol for sending, receiving, and delivering messages. Currently most e-mail is sent using the Simple Mail Transfer Protocol (SMTP), which was defined in RFCs 821 in 1982[17], or Extended SMTP (ESMTP), which was later defined in RFC 1869 in 1995[18]. Although ESMTP introduced several improvements to SMTP, including allowing for sender authentication and encrypted communication using TLS, these are, in practice, not used widely. For message retrieval several different protocols are used, among the most common being POP3[19] and IMAP[20]. Again, although both protocols have been updated to support encrypted
authentication and communication, in practice these are still often not utilized.

This setup of having one (or several) protocols to send and forward e-mail messages between servers and then using several different protocols for message retrieval begs the question, “Why are so many protocols necessary?” In essence, SMTP, ESMTP, POP3 and IMAP are all ways to perform the same basic task: moving an e-mail message from one system to another. So why can’t there be one protocol that could be used as a standard for sending, receiving and delivering e-mail messages? The best answer to this question is likely that these protocols are artifacts of how the e-mail has always been setup. Although a single protocol replacing these other protocols would need to have several modes of operation—allowing users to send e-mail, servers to route this e-mail to the destination, and enabling the recipients to retrieve the e-mail—these processes all transfer the same type of data and majority of the protocol could be reused. Then, instead of having four or more different protocols, each with different security abilities and vulnerabilities, one protocol could be standardized that focuses on being secure and easy to use. Figure 4 below shows a possible path for an e-mail message, along with protocols that could be used for each step. Note how different protocols could be used at each step, limiting the control a user has over how a message is handled.

![Figure 4. Simplified E-mail Path with Possible Protocols](image)

In order to maximize the functionality and extensibility of this new protocol,
Extensible Markup Language (XML)[21] will be used for all network communications. Using XML will allow the protocols to easily share a basic syntax that can be used to communicate information between different kinds of servers and applications without needing to convert this information into different format. Also, XML can easily be modified to fit the task at hand. Finally, XML can be stored in a database format which allows the data to easily be queried and retrieved. This prevents the need for excessive conversions between different types of data structures and storage formats.

4.3 Mail Clients

Along with new e-mail protocols, another challenge will be ensuring the user interface of the mail client is extremely clear and easy to understand. In order to cater to users who are not savvy enough to understand the need for and methods of security, mail clients will need to make security functions as a completely integrated part of the system from the very beginning. First, the mail client will need to have a secure but easy way for users to setup their new e-mail accounts. This will include automatically creating a public/private key pair in order to allow for encryption and digital signatures to be handled without any additional user interaction. In addition, there needs to be an easy or automatic method to obtain the user’s certificate, signed by a certificate authority to prove identity, as well as the certificates for other mail senders and recipients.

Finally, the mail client must make the security level of message clear to the user. Any problems, such as expired or revoked certificates, must be prominently noted to the user and the implications of such problems must be explained to the user in an easy to understand way.
Hi Mark,

I wanted to let you know that I updated your itinerary as you requested. You will now be leaving on the 11 a.m. flight to Seattle instead of the 6 a.m. flight. I also attached an up-to-date version of your complete itinerary.

Let me know if there is anything else I can do.

John Marshall
Secure Mail: john@cert.johntravel.com
Travel Agent
John's Travel Agency
(555) 555-1212
Figure 5 above shows a possible inbox representation for a mail client in this system. Notice how each message in the inbox is labeled with text and an icon indicating the type of certificate used to sign the message. The messages are also color-coded according to this certificate type. Also, above the current message being displayed, the same information and color-code is used to indicate this same information to the user. This should make clear to the user where the message is coming from and help to ensure malicious users cannot misrepresent their id.

4.4 Certificate Infrastructure

In order to create a more secure e-mail system, some changes to the current certificate and public key infrastructure (PKI) should be made. These include modifications that could be used to help identify users, as well as a new way of handling certificates that would make it easier for a user to automatically obtain or validate certificates. It will also be necessary for anyone who uses the secure message system to have a certificate that can be validated through a trusted certificate authority (CA). This will allow users to easily and automatically digitally sign and encrypt messages, and users will be able to trust the claimed identity since user’s identities will be verified by a CA.

The current state of e-mail allows certificate signing to be an add-on or an option. The problem with this is that most users are not aware of this option, its implications, or even how to undertake its use. If instead there were a system where security was integrated from the start and creating a public/private key pair and obtaining a certificate was part of the process of setting up a mail account, users could know that they are sending all of their messages securely on the system.
With e-mail there are too many questions: do recipients have public/private key pairs setup so the message can be encrypted, are the public keys shared in a public location or otherwise available, does the sender have a key pair to digitally sign the message, is the sender’s mail client setup correctly to handle the key pair, is the sender’s mail server setup to handle encrypted communications, is the recipient’s mail server setup to handled encrypted communications, etc. With a system this complicated, it is unlikely that mass adoption of these security tools will ever take place.
CHAPTER 5. DESIGN

This section will describe the basic elements that will be used in order to develop the secure mail system and how these elements fit together.

5.1 Architecture

In order to provide confidentiality, integrity, and authentication to the secure mail system, three basic components will be necessary: certificate servers, mail servers, and clients. A mail server and certificate server could reside on the same machine at the same IP address; however, this will not be assumed. The certificate servers will be responsible for creating, storing, and retrieving certificates. The mail servers will be required to accept outgoing messages from clients, forward these messages to mail servers of the recipients, and deliver the messages to the recipients as requested. In the secure mail system, a mail server refers to what would be two separate services in a traditional e-mail system and acts as both an incoming and outgoing mail server. So, when a client connects to the server, it may choose to send messages, retrieve messages, or both. This removes an unnecessary layer of complexity by consolidating sending e-mail and receiving e-mail into one protocol and one server. The client application will provide users the means to interface with the mail servers and certificate service. The client will also be used to setup a mail account, as well as send and receive messages. Figure 6 below shows the basic architecture of the secure message system. In this example, one client connects to separate mail and certificate servers, while the other client connects to a combined certificate and mail server.
5.1.1 Certificate Server

Certificate servers are the core of what makes this new protocol secure. These servers will provide a network service that allows new certificates to be requested and existing certificates to be looked up over the Internet. For example, when a user wants to send a message, the recipient’s mail address will be used to determine at which server the recipient’s certificate is stored. The client will obtain the correct certificate from the appropriate server.

Two types of certificate servers will exist: subordinate certificate servers and root certificate servers. This is similar to the setup of many other certificate uses. Subordinate certificate servers will be required to have their own certificate which has been signed by a
root certificate server. This allows the clients to form a chain of trust ensuring the certificates they receive can be trusted and allowing the certificates to be bound to an actual identity. Subordinate certificate servers will be required to check the identity of any persons requesting to start a secure mail account. How this should be done is beyond the scope of this document; however, several methods could be imagined including procedures similar to those used by Verisign and other current certificate authorities before issuing certificates. It will also be necessary for the subordinate certificate server to issue certificates to the mail servers for the certificate server’s organization. This allows the mail servers to communicate in a secure manner as well. Thus, both the users and the servers can communicate securely.

Each certificate server will only be allowed to issue certificates for certain uses. For example, root certificate servers can issue certificates to subordinate certificate servers. Subordinate certificate servers cannot issue certificates to other certificate servers, but can issue certificates to mail servers and users. In addition, each server will only be allowed to issue certain types of certificates. These certificate types will be discussed in more detail later; however, their basic purpose is to allow users to easily tell what type of sender has sent each of their received messages and prevent users from spoofing their affiliation or status.

Figure 7 below shows how this system works, with one or more root certificate servers that can issue certificates to subordinate certificate servers. This figure also shows how the subordinate certificate servers can all issue certificates to mail servers. These certificates allow the mail servers to authenticate and encrypt data over a TLS connection. In the figure, the subordinate servers are also shown to be able to issue certain types of user certificates. For example, the bank’s certificate server can issue Financial Institution User Certificates. This allows users to differentiate between messages sent from different types of users. This
way it would be much more difficult for a malicious user to use a fraudulent identity, such as that of a bank employee.

Figure 7. Certificate Servers Can Only Create Certain Certificates

Clearly there would be the need for vigilance. If a certificate server was not following the correct procedures in verifying identities, issuing certificates of the wrong type, or purposefully forging certificates, that server would have its own certificate revoked, effectively making any certificate created by the server useless.

It should be noted that it will be assumed that there is a secure way to deliver the root certificates to the clients and mail servers, as these will be required in order to verify any other certificates. This assumption is made because this is currently already done in Internet browsers and other software that relies on certificates in order to allow the program to validate SSL or other types of certificates.
5.1.2 Mail Server

Mail servers will serve three basic purposes that mirror the functions of current e-mail servers. First, the mail server will accept outgoing messages from clients. Next, these servers will route the messages to the recipient’s mail server. Finally, mail servers will be responsible for delivering messages to a client upon request.

In simple terms, the mail servers will be used as an intermediate step for transferring mail messages between clients. Although it would be possible to design a protocol omitting the intermediate steps and allowing the clients to connect directly, offline users, firewalls, and network address translation would likely cause complications with this type of a scheme.

In this protocol, the mail server will actually serve mostly the same purpose as those in a traditional e-mail system with a few modifications. First, instead of requiring two different services and two different protocols for sending messages and receiving messages as current e-mail standard do (for example, SMTP for sending messages, POP3 or IMAP for receiving), clients and servers will communicate using a single protocol. This allows the system’s complexity to be reduced by only relying on one approach for transferring messages between clients and servers. Using this single protocol will also help users ensure that messages will be sent over secure connections all the way from the sender’s client to the recipient’s client. As previously mentioned, this can be very difficult or impossible to guarantee over traditional e-mail.

Another important difference between traditional e-mails systems and this system will be that the mail servers will always encrypt communications with clients and servers. This will be done using a TLS, allowing parties to mutually authenticate and provide encryption
on all communications. This means that no substantive information should be transmitted in clear text, including recipient addresses, message subjects, etc. Figure 8 below shows how a mail server accepts, routes, and delivers messages. Note that here the same protocol is used for each step. This can be contrasted against a tradition e-mail server that may use different protocols at each step as shown in Figure 4.

**Figure 8. Mail Server Functions**

### 5.1.3 Client

The client application will be similar to an e-mail client such as Microsoft Outlook. However, for this protocol the client will have more responsibilities than a standard e-mail client, including encrypting and decrypting messages, acquiring and verifying certificates and creating and validating digital signatures. In addition, the client will need to interface with both the user’s certificate server and the user’s mail server. The certificate server will be used to acquire and verify certificates in order to send messages and verify the validity of received messages. The user’s mail server will be utilized in order to send messages and also to retrieve messages sent to the user. The client will also be responsible for creating the key pair and acquiring a certificate for a user when the user’s mail account is created. This process will be described in more detail in Section 6.1 below.

Figure 9 below shows how the client connects to its certificate server for certificate
retrieval and validation and to its mail server to send and receive messages. If a message needs to be delivered to a recipient that uses another mail server, this is handled by the client’s mail server and not the client itself. Likewise, if the client tries to look up or verify a certificate for a client that resides on another certificate server, the certificate server will perform the action on the foreign certificate server, and return the result to the client.

Figure 9. Client functions

As with an e-mail client using SMTP and POP3, the client will initiate a connection to the mail server when it wants to send or receive messages; however, unlike standard e-mail protocols, this can be done in one connection. In other words, when the client connects to its mail server, it can first request to receive any new messages that the mail server may be
holding, and once this has completed, the client can elect to send any messages that the user
wants to send. Unlike current e-mail clients, before sending a message, the client will need to
make a connection to its certificate server to either acquire or verify the certificate for the
message’s recipient. In addition, when a message is received, the client will need to contact
the user’s certificate server in order to verify the certificate associated with the received
message. These processes will be discussed in more detail in Section 6.3.

### 5.2 Certificates

This protocol will make use of X.509 version 3 certificates[22] with the addition of a
specialized subject alternative name extension that will be used to help identify the client,
route messages, and provide a means of separating different types of messages. Each user
will have a certificate that will verify their ownership of a private key and contain its
associated public key. This key pair will be created by the client when setting up their
account. After the key pair is created, the certificate will be acquired automatically. This will
require that a user either verifies their identity at this time or that the user has already verified
their identity to a certificate provider. Certificates will also be required for the mail servers
and certificate servers. Appendix E below shows an example of a certificate created for a
user of this system.

#### 5.2.1 Subject Alternative Name Extension

The subject alternative name extension is a X.509 certificate extension that was
created in order to allow for additional identities to be bound to a subject of a certificate.
According to the X.509 version 3 standard, the contents of the subject alternative name
extension are considered definitively bound to the public key and must be verified by the CA. For this system, the subject alternative name extension will contain the following information: a version number, the name of the owner of the certificate, the alias of the owner of the certificate, and the certificate type. User certificates will also contain a mail server entry which will point to the fully qualified DNS address of the user’s mail server. Also for user certificates, the issuer may elect to include affiliation or title information.

Certificate server certificates must also have a signer authentication value. Appendix F below shows the ASN.1 definition of the subject alternative name extension used in this system. Additionally, a sample of this extension can be seen at the end of the certificate in Appendix E.

For certificates that use the system described in this document, the version number should be 1. The name of the owner would be the user’s full legal name in the case of a user certificate, or the name of the owning organization in the case of a server certificate. Aliases, which are the same as mail addresses, will be discussed in Section 5.3. Certificate types and signer authentication values will be discussed in 5.2.1.1. The affiliation and title values would be used, for example, by a company that issues certificates to its employees. The affiliation would allow the company to indicate that the user certificates they are created are for their employees. The title value would allow the company to give each user a specific title. This allows users to more easily identify who they are communicating with, and adds another layer of protection against identity spoofing.

5.2.1.1 Certificate Types

Certificate types will be used to tell what type of entity owns the public key
associated with a given certificate. For users, certificate types will make it easy to tell what type of sender has sent each of their received messages and will prevent users from spoofing their affiliation or associations. In other words certificate types are concerned with how entities are portraying themselves when they send mail messages, encrypt data, or sign data. Each certificate type may require a different level of identification validation depending on the security level of the certificate type. This system contains the eight certificate types listed in Table 1.

<table>
<thead>
<tr>
<th>Certificate Type</th>
<th>Intended Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal</td>
<td>Messages from an ordinary user.</td>
</tr>
<tr>
<td>Business (General)</td>
<td>Messages from a business. No financial information/transactions.</td>
</tr>
<tr>
<td>E-Business</td>
<td>Messages from a business. May be used for financial transactions.</td>
</tr>
<tr>
<td>Financial Institution</td>
<td>Messages from a financial institution (such as banks).</td>
</tr>
<tr>
<td>Government</td>
<td>Messages from a government entity (such as the IRS).</td>
</tr>
<tr>
<td>Mail Server</td>
<td>Used by a mail server for TLS communication.</td>
</tr>
<tr>
<td>Root certificate server</td>
<td>Used by root certificate server to sign subordinate certificate server certificates and TLS.</td>
</tr>
<tr>
<td>Subordinate certificate server</td>
<td>Used by subordinate certificate servers to sign user and mail server certificates and TLS.</td>
</tr>
</tbody>
</table>

Table 1. Certificate Types

Each of these eight certificate types has its own possible uses. Personal certificates could be used for sending secure personal communications between any two people (for example between family members or friends). A personal certificate would not be tied to any business or organization. A personal certificate could also be used by a customer sending a message to a business. A general business certificate could be used for a business that wants to communicate with customers or other businesses through secure messages but would not be used for anything financial (for example, soliciting account or payment information, sending e-bills, etc). An e-business certificate would have the same purposes as a general business certificate but would also allow the business to perform secure financial business. As the name suggests, financial institution certificates would be for identifying messages coming
from financial institutions such as bank, credit unions, brokerages, etc. The government certificate type could be used by a government in order to communicate official messages. For example, in the United States, the Internal Revenue Service (IRS) may want to deliver tax related information to its citizens via internet mail. This type of a message would be signed with a government certificate in order to verify that the sender was actually part of the indicated government entity. The last three types of certificates are used as described in the table above.

It is important to note that while the certificate type dictates the type of messages a user is allowed to send, it does not impact the type of messages the user can receive. So, while a personal user cannot send messages and appear to be a business user, that personal user could receive e-mail from users with any of the other types of certificates (for example, a personal user could receive a message from a user with a government certificate). Also, the type of certificate does not affect the confidentiality, integrity, or authentication aspect of the message. In other words, messages sent by someone possessing a personal certificate would be encrypted and signed using the same method and key type of someone possessing a government certificate. It is not the intent of the certificate type to be used for securing data in different ways but rather to help a user determine exactly who a message is coming from and if the information contained within a message is valid coming from a specific sender.

As mentioned before, certificate servers will only be allowed to issue certificates of certain types. This will allow users to easily ensure the message content they are receiving is valid coming from the type of user indicated by the certificate type. For example, if a user receives an e-mail requesting that he update his bank account information from a mail address linked to a personal certificate, clearly this is likely a fraudulent request. By using
Figure 5 on page 19 as an example, you can imagine what this may look like. Above the message it would clearly indicate the sender had used a personal certificate, instead of a financial institution certificate. There would also be no bank name or title shown. Similarly, if a user receives a message signed by a government certificate, but the certificate server that has signed the certificate is only authorized to sign personal certificates, the message and the certificate is likely fraudulent. In this case, the client should automatically detect this, and would not display the message. A subordinate certificate server that is signing certificates that it is not authorized to sign would need to have its certificate revoked by the root certificate server as it would not be acting in accordance to the rules of the protocol. These certificate types will be an integral part of the secure mail system. By having this information readily available, it will make it much easier for the user to determine when an e-mail is likely not from a legitimate sender.

An additional related piece of the subject alternative name extension is the SigningAuth value. This value is used to specify which types of certificates a given certificate server is allowed to create. Each of the certificate types is given a value as shown in Table 2 below. The values that a given server is allowed to sign are added up and inserted into the subject alternative extension as an integer. This allows users to verify that the signing certificate was indeed allowed to sign certificates of a given type.

<table>
<thead>
<tr>
<th>SigningAuth Value</th>
<th>Certificate Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Personal</td>
</tr>
<tr>
<td>2</td>
<td>Business (General)</td>
</tr>
<tr>
<td>4</td>
<td>E-Business</td>
</tr>
<tr>
<td>8</td>
<td>Financial Institution</td>
</tr>
<tr>
<td>16</td>
<td>Government</td>
</tr>
<tr>
<td>32</td>
<td>Mail Server</td>
</tr>
<tr>
<td>64</td>
<td>Subordinate Certificate Server</td>
</tr>
</tbody>
</table>

Table 2. SigningAuth Values
5.3 Mail Addresses (Aliases)

A client’s mail address will be similar to a standard e-mail address (for example, user@domain) except the fully qualified DNS name of the certificate server where the user’s certificate is stored will replace the domain which is used today. This allows any client that has another user’s mail address to contact the correct certificate server in order to acquire or verify the user’s certificate. The certificate will then contain the extension informing the client of the address of the mail server that should be used for the particular mail recipient. This method seems to be the easiest method of allowing client to look-up other users’ certificates even if the only information they have about their intended recipient is their mail address. In other words, the only information a user would need to send a message to a second user would be that second user’s mail address, just as in standard e-mail. For example, if I wanted to send a message to my friend, Charlie, and I knew his mail address was charlie@cert.maildomain.com, I could simply type this address in, and my client would be able to retrieve his certificate. First, my client would contact my certificate server and provide the mail address, charlie@cert.maildomain.com. The certificate server would look at Charlie’s certificate server’s DNS name, “cert.maildomain.com”, and contact the certificate server to retrieve Charlie’s certificate. After the certificate has been retrieved, my certificate server would deliver this certificate to me. The certificate would then point to the Charlie’s mail server.

5.4 Encryption

Encryption will be important in this protocol in order to provide confidential communication. The goal of the encryption methods used will be to ensure that only
information is revealed on a “need-to-know” basis. In other words, a mail server has no need to see the text within a mail message’s body or an attachment. This information should be available only to the recipients. The mail server would need other information about the message, however. For example, when a user sends a message, the mail server will need to be able to access the list of recipients of the message. Otherwise, the server would be unable to route the message on to the next mail server for the recipients. In contrast, an eavesdropper should not be allowed to gain access to any of this data. Because of this, different types of data will be encrypted in different ways and at different times. This section will attempt to summarize how and when encryption will be used.

Asymmetric encryption will mainly be used as a way of authenticating two parties to each other. Since each entity has a public/private key pair and a certificate containing their public key, these can be used as a way of setting up a secure connection. This will be done using the transport layer security (TLS) algorithm. In this protocol, the two parties can also be mutually authenticated in almost all cases since all entities should have their own key pair and certificate. The only exception to this would be when a server or user is first requesting to have their public key signed. This situation is discussed later in Section 6.1.1.

Figure 10 shows how asymmetric encryption works. Note that although the public and private keys are different, they must be part of the same key pair. The owner of the keys can publicize the public key, but must keep the private key to himself (hence the names).
Asymmetric Encryption

It is possible for a user to prove ownership of a key pair in a number of ways. First, she could be requested to encrypt a given set of data using her private key. Once this is done, anyone would be able to decrypt the data using her public key, and, if the decrypted data matches the original data, then only the owner of the private key could have done the encryption. The other way to prove ownership of a key pair is by having another user encrypt something with your public key, and then decrypting it with your private key. This works because only someone that knows the correct private key corresponding to your public key would be able to decrypt the data. This is why it is important to keep private keys secret.

The TLS protocol basically works by allowing two communicating parties to secretly create a symmetric key which is then used to encrypt data being exchanged. The parties are also authenticated through their ability to use their private key to decrypt secrets that have been encrypted with the public keys associated with their certificate. This type of an exchange will take place at the beginning of all network communications including certificate lookup between a client and a certificate server, sending a message from a client to a mail server or between two mail servers, and when a client retrieves his messages. In later sections, when information exchange between servers and clients is discussed, it can be
assumed that a TLS session had first been established in order to encrypt the data transmissions. Figure 11 below shows an example of symmetric key encryption. Notice here the two keys are identical.

![Symmetric Key Encryption Diagram](image)

**Figure 11. Symmetric Key Encryption**

While using TLS helps ensure an eavesdropper cannot garner any information exchanged over the network, it does not provide end-to-end encryption, nor does it prevent a mail server from accessing the data within a mail message. Because of this, it will be necessary for a client to encrypt messages before sending them. This will be done by generating a symmetric key and encrypting this key with the recipient’s public key. This will ensure that only the intended recipient is able to read the key and decrypt the message.

Figure 12 shows the encryption process a sender must go through before proceeding to send a message. This includes generating a symmetric key, encrypting the message using the generated key, and finally encrypting the symmetric key using the recipient’s public key. Figure 13 shows the decryption process that the recipient must go through in order to decrypt the message. First, the recipient must receive the encrypted message and encrypted key. The recipient will first decrypt the key using his private key.
Once the symmetric key has been recovered, the recipient can proceed to decrypt the encrypted message using the symmetric key. This symmetric key would then be discarded, and any future communications would take place using a newly generated symmetric key.

Using these methods in order to encrypt communications will ensure that entities are only provided with information that is required for them to fulfill their function. Further, these techniques should prevent an eavesdropper from gathering any information including the recipients, subject, or body of any messages being sent.

### 5.5 Digital Signatures

A digital signature will be sent with each message to ensure that no changes to the
message take place during transit and to authenticate the sender to the recipient. This will be done by enclosing a hashed copy of the message encrypted with the sender’s private key. Figure 14 below shows the process of creating a digital signature as previously described. In this system, the mail message, which will include the subject, timestamp, message, and the sender’s certificate will all be hashed together using the hash function. This data will then be encrypted using the sender’s private key. This signature will be sent to the recipient, along with the message encrypted with the symmetric key as described in Section 5.4.

![Figure 14. Digital Signature Creation](image_url)

Once a recipient has received a message, she will have to verify the signature to ensure that the message hasn’t been tampered with. To do this, first the recipient must recover the original message. This recovered message is then hashed in the same way as the sender previously hashed the message. The recipient then must decrypt the digital signature using the sender’s public key. This should result in the same hash value as the one she had generated.

The two hashes are compared, and, if they match, the recipient knows the message has not been changed. Also, since the public key used to verify the signature is associated
with a certificate, this allows the recipient to verify the identity of the sender. Figure 15 below shows this process.

**Figure 15. Checking a Digital Signature**
CHAPTER 6. USE CASES AND PROCEDURES

This section will describe in more detail how each of the components of the secure messaging system will work. Specifically, the workings of the certificate servers, mail servers, and client applications will be addressed. For each of the processes that requires network communication, XML schemas have been created for the socket commands. These schema files and a sample XML exchange can be seen in Appendix G. Additionally, flowchart for how and when each XML command can be sent are located in Appendix H.

6.1 Certificate Server

This section will describe how the certificate server will interface with other applications. This will include how the certificate server will be used to create, retrieve, validate, or revoke certificates. This section will also discuss how certificate servers are initially setup.

6.1.1 Certificate Creation

Certificate creation will need to take place anytime a new server or user is added to the secure mail system. This will generally be done by sending a signing request to a certificate server. Specifically, when a new user or mail server is created, it will need to request its subordinate certificate server to sign its public key. When a new subordinate certificate server is created, it will need to have a root server sign its public key. One exception to this would be root certificate servers, whose certificate would be self-signed. Because of this, care must be taken to ensure that all involved parties can keep an accurate
and unaltered list of valid root certificate servers. Requests will be sent to the certificate server through the network, and if the request is valid and authorized, a certificate will be sent back to the requestor and also stored on the certificate server for future retrieval. Figures detailing each step of this process are contained in Appendix A below.

In order for a certificate to be issued, the requestor must first be authenticated. This could either be done prior to the request (a pre-authenticated request) or possibly at the time of the request. The methods of authentication are outside the scope of this document; however, current public services such as online tax filing, credit report lookup, and certificate purchasing could be used as models to develop these procedures. This document will assume the user or server has been pre-authenticated. This could be done in person, through the postal service, or possibly through other methods.

Assuming the certificate request will be a pre-authenticated request, the requestor, whether it be a certificate server, mail server, or user, will have proven his or her identity to the owner of the certificate server which will issue the certificate. This would have been done at another time, before the creation of the request for the certificate. When the requestor authenticates to the issuer, for example, when a user signs up for a secure mail account, the requestor will be given their alias (alias generation will be discussed in Section 6.1.1.2) and an id number. The id number is a random integer that is used when requesting the certificate in order to help authenticate the user to the server. When a user actually goes to setup the account in the client program, they will be prompted for their secure mail address and this id number. These pieces of information will be hashed together and sent in the request to the appropriate certificate server. The certificate server will use this hashed value to lookup the
user’s information. If the user’s information cannot be found, this means that the user is not pre-authenticated to this certificate server or the information entered by the user was incorrect, and the request is invalid. If the server can find the user, the certificate server will respond with user’s information. Since the server not been sent this information before, this demonstrate that the server actually does a record of the user, and that the user is connected to the correct server.

This process will allow the issuing certificate server to lookup and verify the identity of the requestor, and the requestor will be able to verify the identity of the certificate server. This prevents the possibility of a malicious user from posing as legitimate user in order to falsely acquire the legitimate user’s certificate.

After the authentication takes place, the requestor must send a PKCS #10 formatted certificate request[23] to the certificate server (see Figure 16 for an example). The server will check the request to ensure that the information in the request is correct. If so, the certificate will be created and signed by the certificate server. The newly created certificate will then be stored on the certificate server and sent to the client.

-----BEGIN NEW CERTIFICATE REQUEST-----
MIIDCDBg/qIBADB3VMQswCQYDVQQGEwJUVzENMAsGA1UECBMwSW93YTENMAsGA1UEBxMEQW1iczETMBEGA1UExMKTWFBp FNc8Z2c4JTMgMBEGA1UEAxM2UU2VjdXIvTWFpbdCBznzANBgkqhkiG9w0BAQEFADBjQAwYKcCyEAcnZi8x7izlpqyD7Kr0PF02xIIv6f7MVFqD4TNFwENbp6nbhakD3Fg2hBYUpyzl8Q368iJMVIfkO2Xw8pcWa9/x8xvmhR5iyss7k6ZDQ5hakAkGcvP1jNVQl15hoG3jk3d6Q4i5pQuNvpqvo6h8MACwEAAaAAMAO6CsoqGSI1DQEBAUA4gBAF5shrS8KSDsMfsZeQHy9Y+yJqEVqEoeIrRpOr30rjbG0a00xwYrHEZcSiCMSK1mE5yLhZi93RW712sA8gWabehwLQo4rgTbJbtofMiGdnxFKLX7Cdy3XucKf1htFtiwe9towLkL5GLTNAAtJddO3Jb/sueEcN1yu5V
-----END NEW CERTIFICATE REQUEST-----

Figure 16. Sample PKCS#10 Certification Request
6.1.1.2 Aliases

In order to have a standardized method of looking up certificates, aliases will be used for each certificate. The alias generation will depend upon the type of certificate being issued. For a mail user, the alias will be the mail address of the user, for example “user@certserver.domain.com”. As mentioned previously, the mail address includes the fully qualified DNS name of the user’s certificate server after the ‘@’ symbol. This allows a user’s certificate to be looked-up using only the mail address. A certificate server will use the string “CS:” followed by its own fully qualified DNS name as its alias. This way any party that needs the certificate server’s certificate can look it up if it simply knows the DNS name of the server. Finally, mail servers will use the string “MS:” followed by the fully qualified DNS name of the server. Appended to this is an “@” symbol along with the fully qualified DNS name of the server’s certificate server, for example “MS:mailserver.domain.com@certserver.domain.com”. The strings “MS” and “CS” allow mail and certificate servers to be distinguished from each other in the case where they reside on the same host.

Figure 17 below shows how using an alias alone can allow a message to be delivered to a user. The alias points to the certificate server where the user’s certificate is stored. The certificate will indicate where the user’s messages should be sent. Once the message is delivered to this server, the user will be able to pick up the message here.
6.1.1.3 Distinguished Names

Another piece of information that is used to create certificates is the distinguished name (DN). This provides a way of uniquely identifying a certificate. DNs are made up of pairs of attributes and values. Each certificate uses two DNs: the subject DN which is used to identify the owner of the certificate, and the issuer DN which identifies the certificate server that has issued the certificate.

This system will use the following attributes: type, name, host, and domain. The type attribute will be used to identify what type of entity the certificate is for. All distinguished names will use this attribute, and it will take the appropriate value from the following values: “Root Certificate Server”, “Subordinate Certificate Server”, “Mail Server”, or “User”. The name attribute will only be used for user’s certificates and will be set to the user’s mail address. The host and domain attributes will be used to identify mail and certificate servers. These will simply be set to the host name and the domain name for the certificate owner. The table below shows examples of subject DNs for each of the different types of entities.

<table>
<thead>
<tr>
<th>Entity Type</th>
<th>Subject DN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root Certificate Server</td>
<td>type=Root Certificate Server, host=rootCS, domain=domain.com</td>
</tr>
<tr>
<td>Subordinate Certificate Server</td>
<td>type=Subordinate Certificate Server, host=subCS, domain=domain.com</td>
</tr>
<tr>
<td>Mail Server</td>
<td>type=Mail Server, host=MS, domain=domain.com</td>
</tr>
<tr>
<td>User</td>
<td>type=User, name=<a href="mailto:john@subCS.domain.com">john@subCS.domain.com</a></td>
</tr>
</tbody>
</table>

Table 3. Example DNs
6.1.2 Certificates Retrieval

When a user wants to send a message, the client needs to look up the recipient’s certificate. After a certificate is used, it will be saved in the client application. If a certificate has been previously saved, the client will use the certificate validation steps described in the next section to verify that the certificate is still valid and has not been revoked. If the certificate is not saved or is found to be invalid, the certificate will be retrieved using the user’s certificate server. Figures for each step of this process are shown in Appendix B below.

A request is created that contains the alias of the certificate needed, and this is sent to the certificate server. The certificate server will first check to see if it is the issuer of the certificate in question. If so, it will look up the certificate using the alias provided and return it to the requestor. If the certificate server is not the issuer of the requested certificate, it will check its cache for a copy of the certificate. If the certificate is in cache, it will need to ensure the certificate is valid. This is done using the certificate validation procedure described in the next section. If the certificate is valid, it will be given to the client.

If the certificate is not in cache or the cached certificate is not valid, for example because it has expired or been revoked, the certificate server will attempt to retrieve the new certificate for the recipient. This is done by sending a request to the certificate server that issued the recipient’s certificate. This is possible since, as mentioned in Section 6.1.1.2, the DNS name of the issuing certificate server is contained in the alias or mail address of the user. If this request is successful, the certificate server will perform an additional step to verify that the certificate used by the remote certificate when signing the user’s certificate has
not been revoked. This process is described further in Section 6.1.4.

If the user’s certificate server is able to find a valid certificate for the requested mail recipient, this certificate will be given to the client. If a valid certificate cannot be found, the server will alert the client of this, and the user will be unable to send a mail message. This could happen if one of the certificate is unavailable, if a certificate has been revoked without replacement, or if the secure mail user does not exist.

6.1.3 Certificate Validation

Each time another user’s certificate is used, such as when sending a message to another user or when receiving a message, the certificate must be validated. The first step in the process is ensuring the certificate is a valid certificate. This means that the certificate is the proper format, X.509 version 3, contains the correct extensions, is signed by a subordinate certificate server whose certificate was signed by a known root certificate server, and, obviously, that the certificate is in a format that the application can interpret.

Next, the application must verify that the certificate is within its validity period. Each certificate that is issued is only valid for a certain time range, referred to as the validity period. If the current time is outside of this validity period when the user needs to use the certificate, the client program will use the procedure in Section 6.1.2 to attempt to retrieve an updated certificate.

Another check that must be done is to ensure the certificate is being used correctly. This statement has several meanings. For example, a mail server’s certificate cannot be used to send mail messages—only user certificates can be used for this purpose; a mail server
cannot use a user certificate for encrypting traffic when a user tries to send a message, a user certificate cannot be used to sign another user certificate, and so on. Each of the three components of the mail system (certificate server, mail server, and mail client) has a unique type of certificate, and these certificates can only be used in the way they are intended.

Also, when validating a certificate for a sender when receiving a message, a client will need to ensure that the certificate type contained in the sender’s certificate was issued by a certificate server that has the authority to issue that type of certificate. For example, if a bank wanted to have its own subordinate certificate server to issue certificates to its employees, this certificate server would probably not be allowed to create certificates with the “government” type. Another example would be an ISP that caters primarily to home users and wants to provide secure mail to its clients. This ISP’s certificate server would most likely only be allowed to create personal certificates. The concept of certificate type is discussed in more detail in Section 5.2.1.

Another step in validating a certificate is ensuring that it has not been revoked. If a client has just retrieved a certificate from its certificate server for the first time, this process does not need to take place as it is checked by the certificate server. However, on subsequent uses of this certificate, the client will need to perform this step. This process is explained in the next section, 6.1.4.

6.1.4 Certificate Revocation

When working with certificates, it is sometimes necessary to invalidate an already issued certificate. This may occur for a variety of reasons, such as a private key being compromised or misuse of a certificate. In this system a certificate can be revoked by the
The administrator of its issuing certificate server.

In cases where a certificate is revoked, it is important that this information be communicated to any entity that may need the invalidated certificate. For example, if a user wants to send a message to a recipient whose certificate has been revoked and reissued, somehow, this sender must know that there is a new certificate available. This is generally done in one of two ways: Certificate Revocation Lists (CRLs) or Online Certificate Status Protocol (OCSP)[24]. This protocol will use an approach similar to OCSP in order to check the status of certificates instead of CRLs. OCSP has the advantage of allowing a certificate’s status to be checked at anytime. This offers a more real time solution and does not require large CRLs to be transferred on a regular basis.

Using this type of a real time certificate check also gives a higher level of security to the protocol. This is because CRLs most likely could not be distributed on a real-time basis. Instead, certificate servers would most likely store any revocations until a certain time, say once each day, and then they would be distributed. During this time where the certificate revocations are being held and until the CRLs can be distributed to all certificate servers, it would be possible that a revoked certificate would still be in use.

Checking the revocation status of a certificate will only be required if the client has already acquired the certificate since the certificate server automatically validates the certificate upon a retrieval request. The procedure for checking the status of a certificate will require two pieces of information: the certificate alias and a hash of the certificate, both of which can be obtained from the certificate in question. In order to illustrate the method of certificate validate, take the case where a user wants to send a message to a recipient. In this
case, there are really two certificates that must be validated: the recipient’s certificate and the  
certificate used to sign the recipient’s certificate. The sender will give the aforementioned  
information for these two certificates to the sender’s own subordinate certificate server. This  
certificate server will first check to see if it issued the certificate for the recipient in question.  
If so, it will check the status and reply to the requestor. If the certificate server did not issue  
the recipient’s certificate, it will first want to validate that the recipient’s subordinate  
certificate server used a valid certificate when signing the certificate in question. This  
validation will be done by sending a validation request to the recipient’s subordinate  
certificate server’s root certificate server. If this certificate proves to be valid, the sender’s  
subordinate certificate server will send a validation request to the recipient’s subordinate  
certificate server. If this certificate is also validated, the sender’s certificate server will notify  
the sender, and the process of sending a message can proceed. Appendix C below contains  
figures for each step of process.

Several different scenarios may arise here. First, and most likely, the certificate  
should be valid, in which case the sender’s certificate server will tell the client the certificate  
can be used, and the sender will proceed. Another possibility is that one of the certificates  
will have been revoked. In this case, the sender’s client will be notified, and it may request  
that the certificate server check for a new certificate. Finally, it is possible that one of the  
issuing certificate servers will be unavailable. In this case, there would be no way to ensure  
the validity of the certificate, and the sender would be unable to proceed. A similar process  
would take place by the recipient of a message, whereby the recipient would ensure that a  
message that has been received was sent using a valid certificate.
Although this process is rather complex, it was chosen for this system because it provides the highest level of security. Since a certificate would be validated definitively each time a message is sent or received, there would be virtually no ambiguity as to whether a certificate is still valid or had been revoked. Once this system is fully implemented, the performance of this verification system can be tested in order to ensure this method does not slow the system down too much. If this method does prove to be too resource intensive, a less secure alternative such as using CRLs, or allowing clients and servers to cache certificates for a certain amount of time may be required.

6.1.5 Certificate Server Setup

Certificate server setup is different for root and subordinate certificate servers. Both require a key pair to be created in order to sign certificates, encrypt data, and create digital signatures; on a root certificate, the server creates a self-signed certificate. This simply means the server creates a certificate for itself, listing itself as the issuer and using its own private key to sign the certificate. As the name implies, root servers with self-signed certificate serve as a root for the transitive chain of trust used in certificate applications. Users must implicitly trust these root certificates in order for the system to work. These certificates would most likely be distributed with the mail server, mail client, and subordinate certificate server packages so this trust would be automatic. Although this method does potentially raise security concerns, it is a requirement of the X.509 certificate model, and is generally able to provide good security.

The subordinate certificate servers must create their key pairs and then request that a root certificate server sign their public key. This is done using the process described in
Section 6.1.1, and requires that the root certificate server be able to trust that the owner of the subordinate certificate server verifies the identity of any new users that are added to the system. Of course, if a subordinate certificate server is found to be violating this requirement, the root certificate server can revoke the certificate which effectively invalidates and certificates that the subordinate certificate server has issued. Clearly, this would be a drastic step that would only need to be taken if a certificate server was found not to be fulfilling its duties.

Certificate servers in this system would not likely be as common as SMTP servers or POP3 servers are in the current e-mail system. This is because there would need to be stringent requirements in order to verify that the certificate server would only issue new certificates when the identity of a new user or mail server has been adequately verified.

6.2 Mail Server

The mail server will have several tasks that will need to be performed. These include accepting messages from clients and other mail servers, forwarding these messages to the recipient’s mail server, and delivering messages to clients. Each of these procedures will be discussed here, along with how a mail server is initially setup.

6.2.1 Accepting Outgoing Mail Messages

The first function that the mail server must perform is accepting messages from mail clients. This is the same as the role of an SMTP server in the current e-mail system. After the sender’s mail client has obtained or validated the recipient’s certificate, the client will contact the sender’s mail server and deliver the message to this server. A mail server will expect
several pieces of information. First, the id number of the message is sent to the server, followed by a list of recipient mail addresses, mail servers for delivery, and the symmetric key that can be used to decrypt the message content. A separate copy of the symmetric key must be sent for each recipient since the symmetric key itself will be encrypted with the recipients’ public keys. This is discussed in more detail in Section 6.3.1.

After the recipient list and encrypted symmetric keys have been transferred, the actual message data along with the digital signature is received by the mail server. Since this data is encrypted before it is sent to the mail server, and the server cannot access the contents of the message. Once all of this data is received, the mail server adds a tag that includes a timestamp the server’s alias, and a record of the action that was taken by the mail server (received from the sender, forwarded to another mail server, received from another mail server, delivered to a recipient). This simply indicates that the message has traveled through this server and can be used by the recipient to reconstruct the path of the message if necessary.

The next step for the message is to be transferred to the recipients’ mail servers. This step will be discussed in the next section.

6.2.2 Transferring Messages Between Mail Server

After the sender transfers a message to its mail server, the message must be transferred to the recipients’ mail server(s). This would generally be done immediately after receipt of the message, and uses the same format as was described in Section 6.2.1. Again, the first step is to transfer the message id, key and alias information for the message recipients. For this step, the sender’s mail server would only transfer symmetric keys and
aliases for the recipients for which the mail server in question is responsible. So, if a user sends the same message to three individuals, two of whom use the same mail server and one that uses a different server, the two aliases and keys would be sent to the first mail server and the other alias and key set would be sent to the last user’s mail server. This is because a mail server only needs to know about a particular recipient if it is responsible for delivering the message him or her. Once this transfer is complete, the receiving server will append the timestamp, alias and action data for message path reconstruction, the message will be stored in the mail server’s database.

6.2.3 Delivering Messages To A Mail Clients

The final step is delivering a message to the client. For simplicity sake, this system will use a method similar to POP3 where a mail client must periodically check for new messages by contacting its mail server and requesting them. When this connection is made, the client will request a list of messages stored on the server. The client may either request a list of all messages, only new messages that have not been transferred to the user’s mail client yet, or only old messages that have already been transferred to the mail client. The mail server will respond with a list of message ids indicating whether they are new or old. At this point the mail client could request an individual message or a group of messages to be retrieved. Alternatively, the mail client may simply request all new messages be retrieved.

Once the mail server knows which message or messages to transfer, the mail server will send them to the client using the same method described in Section 6.2.1. After the messages have been received, the client will acknowledge the receipt of the message, and it will append an entry to the path indicating that the message has been delivered to that
particular client. When the client acknowledges receipt of the message, it has the option to request that the server deletes the message. If this client is the only or last recipient of the message on this mail server, the message can be deleted. Otherwise, the client will be removed from the recipient list for the message and the message will be saved for the other clients that have saved or not received the message.

6.2.4 Mail Server Setup

Setting up a mail server would require steps similar to that of a certificate server. First the server must generate a key pair that would be used to encrypt data and create digital signatures. After this key pair is created, a request to have the mail server’s public key signed would be sent to a subordinate certificate server. Generally, there is a lower need for a mail server’s owner to be stringently verified as opposed to certificate servers or mail users. This is because a mail server cannot create new certificates that verify the identity of users or servers, and they cannot ever see the contents of any mail messages. Even though the need for authentication of the mail server’s owner is lower, some verification should still be done, and the steps outlined in Section 6.1.1 would be followed. Once a certificate server has signed the mail server’s certificate, the server is ready to accept and delivery mail messages.

6.3 Mail Client

The mail client application will serve as the interface to users when sending or receiving messages. This section will discuss the functionality of the mail client, including sending and receiving messages and initial mail client setup.
6.3.1 Sending messages

Just like in current e-mail clients, the secure mail client will allow users to enter a list of recipients, a subject, a message, and attachments. Once this is provided and the user elects to send the message, the mail client will contact the user’s certificate server in order to request or verify the certificates of the recipients. If none of the required certificates can be obtained or verified, the user is alerted and the message cannot be sent. If several of the required certificates cannot be obtained or verified, but some can be, the user will be alerted of this problem and prompted to either send the message to the recipients whose certificates can be obtained and validated or wait and try again later.

Once the certificates are obtained, the message must be prepared. First the client will add the sender’s certificate to the message data so the recipients automatically receive a copy of the certificate. Next, the client must digitally sign the message data using the sender’s private key. Then the client generates a symmetric key and encrypts the message and signature using this key. Finally the client must encrypt the symmetric key using each of the recipient’s public keys so they can each access the symmetric key needed to decrypt the message. The client will also generate and assign an id number to the message.

At this point, all of this information must be sent to the mail server. This could be done immediately, or the message could be queued until the user requests the client to contact the mail server. When the sender’s mail server is contacted, the client will simply request to send the message, and the information will be delivered to the server. Steps 1 through 6 in Appendix D (shown in Figure 34 through Figure 39) show the process of preparing and sending a message.
6.3.2 Receiving messages

In order to receive messages that have been sent a user, the user’s client application must initiate a connection to his or her mail server. This can either be done at the request of the user or could be setup to happen at a certain time interval, every 10 minutes for example. Once this connection is made, the client can request that all new messages be sent from the server. The server will deliver the messages, and, if they are received by the client correctly, they will be acknowledged. Depending on the setup of the client application, it may request that the server delete the message. This will be handled by the server as discussed in Section 6.2.3.

At this point the client application will need to decrypt the message’s symmetric key using the user’s private key. The symmetric key can then be used to decrypt the message. At this point, the client has access to the sender’s identity and certificate, so the certificate will be validated. Next, the message data will be hashed, the digital signature will be decrypted with the sender’s public key, and these two values will be compared to ensure no tampering has taken place. At this point, if the message is decrypted successfully, the certificate is valid, and the digital signature verifies the integrity of message, the mail client can display the message to the user. Steps 7 through 11 in Appendix D (shown in Figure 40 through Figure 44) show how a message is later retrieved, and the sender verified.

6.3.3 Mail Client Setup

As previously mentioned, when a user goes to setup a new mail account they must be able to verify their identity. Although it could be possible for someone offering a secure
messaging server to verify the identity at the time the user wants to setup the mail client, this
document will assume the user has already verified their identity before they go to setup their
mail client.

When a user wants to setup a secure mail account, the client will need to generate a
key pair. Once this key pair is created, the user will need to create a password. This password
will be used to secure the keystore where the private key and certificates will be stored. It
will be necessary that the user remember this password and enter it each time the client is
started. Loss of this password would mean a new key pair and certificate would need to be
generated.

The user will then be prompted for the mail address that they have been assigned
(their alias) and the id number they were given when they authenticated to the secure mail
provider. This information will be concatenated, hashed, and sent to the certificate server.
The certificate server will lookup the information, and, if the request is valid and the user’s
identity has been verified, the server will create and store a certificate for this user and then
send it back to the client. At this point the client can use the key pair and certificate to send
and receive mail message. Figures of the process of having a certificate created are shown in
Appendix A.
CHAPTER 7. CONCLUSION

By implementing this protocol for secure message transfer, many of the problems with current e-mail technologies could be solved. Spam would decrease because it would be much harder to constantly create new message addresses since each address must be tied to an actual identity. Phishing would be much more difficult since spoofing would no longer be a trivial task. In addition, it would be much easier for users to see when an e-mail comes from a business, financial institution, or government. This would give users an additional assurance that any messages from these entities are legitimate. Finally, confidentiality would be protected on several levels by using encryption to secure the message data and headers.

The next step for this system would be implementing the software required in order to test the system in an actual environment. Some modifications may need to be made in order to prevent the protocol from being too resource intensive. Security is a trade off after all, and although this protocol should be very secure, it may require too much network bandwidth to be feasible.

Although great strides have been made to secure existing e-mail technologies, these advances have not reached a broad market and have done little to help the average e-mail user. This protocol offers hope that it is possible to have a secure message system in place that is also easy to setup and use.
APPENDIX A. CREATING A NEW USER CERTIFICATE

This section contains diagrams showing the step-by-step process of creating a new certificate for a new secure mail user.

Figure 18. Step 1 of Creating a New Certificate

Figure 19. Step 2 of Creating a New Certificate
**Step 3:** Later, the secure mail will user the mail client to connect to the certificate server and setup the new account.

Figure 20. Step 3 of Creating a New Certificate

**Step 4:** The client will hash the alias and id number together and send the hashed value to the server.

Figure 21. Step 4 of Creating a New Certificate

**Step 5:** The certificate server will use this hashed value to key into a database, where the user information will be retrieved and sent back to the client.

Figure 22. Step 5 of Creating a New Certificate
Step 6: The secure mail user will verify this information. Assuming it is correct the user will send a certification request for his public key to be signed.

Figure 23. Step 6 of Creating a New Certificate

Step 7: If the request is valid, the certificate server will create and store the new certificate. A copy will also be sent to the user.

Figure 24. Step 7 of Creating a New Certificate
This section will describe the process used to retrieve a certificate. This will mainly be performed when a user wants to send a message to a user with whom he has never communicated before. After the first communication between two parties takes place, a user will use the procedure shown in Appendix C for any subsequent communications.

**APPENDIX B. RETRIEVING A CERTIFICATE**

- **Step 1:** If the client does not have the required certificate, it will need to request it. The client requests the certificate from its own certificate server.

*Figure 25. Step 1 of Retrieving a Certificate*
Step 2: The sender’s certificate server requests the certificate from the issuer. The issuer sends the certificate along with its own certificate.

Figure 26. Step 2 of Retrieving a Certificate

Step 3: The certificate server requests the issuer’s root certificate server to verify the issuer’s certificate.

Figure 27. Step 3 of Retrieving a Certificate
Step 4: If the issuer's certificate is valid, the certificate server will send the requested certificate to the client.
APPENDIX C. VERIFYING A CERTIFICATE

This section will describe the process used to verify a certificate. This procedure will be performed in several instances, such as when a user receives a message and when a user sends a message to a recipient whose certificate he already has.

Figure 29. Step 1 of Verifying a Certificate
Step 2: The certificate server will send a verification request to the issuing certificate server which will respond with its own certificate, and an answer to the request.

Figure 30. Step 2 of Verifying a Certificate

Step 3: The certificate server requests the issuer’s root certificate server to verify the issuer’s certificate.

Figure 31. Step 3 of Verifying a Certificate
Step 4: If all of the requests indicate the certificate is valid, the client will be notified and can use the certificate.

Figure 32. Step 4 of Verifying a Certificate
APPENDIX D. SENDING/RECEIVING A MAIL MESSAGE

This section contains diagrams showing the step-by-step communications processes required to send and receive a mail message. Steps 1 through 6 (shown in Figure 34 through Figure 39) show the process of preparing and sending a message. Steps 7 through 11 (shown in Figure 40 through Figure 44) show how a message is later retrieved, and the sender verified.

Figure 33. Message Travel Flowchart
Figure 34. Step 1 of Sending a Message

Step 1: Sender requests recipient’s certificate from its own certificate server

Figure 35. Step 2 of Sending a Message

Step 2: Sender’s certificate server sends sender’s certificate to and acquires recipient’s certificate from recipient’s certificate server.
Figure 36. Step 3 of Sending a Message

Figure 37. Step 4 of Sending a Message
Figure 38. Step 5 of Sending a Message

Step 5: Sender sends encrypted message to its mail server

Figure 39. Step 6 of Sending a Message

Step 6: Sender's mail server sends message to recipient's mail server
Figure 40. Step 7 of Sending a Message

Step 7: Recipient requests messages from mail server

Figure 41. Step 8 of Sending a Message

Step 8: Mail server delivers messages
Figure 42. Step 9 of Sending a Message

Step 9: Recipient requests certificate server to verify sender's certificate

Figure 43. Step 10 of Sending a Message

Step 10: Certificate server verifies sender's certificate
Figure 44. Step 11 of Sending a Message

Step 11: Client decrypts and displays message
APPENDIX E. SAMPLE CERTIFICATE

[0] Version: 3
SerialNumber: 11745822471258171591308404038595
IssuerDN: CN=Test Certificate
Start Date: Thu Mar 22 11:50:47 CDT 2007
Final Date: Tue Mar 06 19:08:57 CST 2007
SubjectDN: type=USER,name=john.meeker@certServ.americabank.com
Public Key: RSA Public Key
modulus:
b15947d7963c2e137cad3fb0f895b500af017e5ac59110aa29ae6c80990bcc808a5ff96b5a
1f1761e4703af4756c2278cf1885ab798f0338be8643612b8e67e3b0acda10c5ae20a3053f
4854f0505ad4b4bf5d0fe2854f2c3baf18ede3e14286981b52f019fceaba9e834e67415
785ca9c73b0cb457996f1b7f48906f11
public exponent: 10001
Signature Algorithm: SHA1WithRSAEncryption
Signature: 201dcaa8b304a4a70d2e1274105b4a2260d7b82e
593cb92d74623cbba6f5d3c8f58c6c08f41c457
add0f7cb170480cc0c6b66b76c6b61baddb583420
011b703d2114d26b7e258c8ed8184f46b14cc56
7afdeeb594881fe475706accdb24fb8ae3a1c5
94d713a626f5185cd1e7f109f0fc985d98b8f41d
61432f5182d607a2
Extensions:
critical(false) 2.5.29.35 value = DER Sequence
Tagged [0] IMPLICIT
  DER Octet String[20]
Tagged [1]
  Tagged [4]
    DER Sequence
    DER Set
      DER Sequence
        DER Identifier(2.5.4.3)
        PrintableString(Test Certificate)
  Tagged [2] IMPLICIT
    DER Octet String[6]
      critical(false) 2.5.29.14 value = DER Octet String[20]
      critical(true) BasicConstraints: isCa(true), pathLenConstraint = 1
      critical(true) KeyUsage: 0xb6
    critical(false) 2.5.29.17 value = DER Sequence
Tagged [0] IMPLICIT
  DER Sequence
    Integer(1)
    UTF8String(John Meeker)
    UTF8String(john.meeker@certServ.americabank.com)
    UTF8String(FINANCIAL)
    UTF8String(mail.americabank.com)
    UTF8String(America's Bank)
    UTF8String(Loan Officer)
APPENDIX F. CERTIFICATE EXTENSION DEFINITION

Extension ::= SEQUENCE {
  Version INTEGER,
  Name UTF8String,
  Alias UTF8String,
  Type ENUMERATED {
    BUSINESS,
    EBUSINESS,
    FINANCIAL,
    GOVERNMENT,
    PERSONAL,
    MAILSERVER,
    ROOTCERTIFICATESERVER
    SUBORDINATECERTIFICATESERVER },
  MailServer UTF8String OPTIONAL,
    -- if present must not be one of the server types
  Affiliation UTF8String OPTIONAL,
    -- if present MailServer must also be present
  Title UTF8String OPTIONAL,
    -- if present Affiliation must also be present
  SigningAuth INTEGER OPTIONAL
    -- if present must be one of the certificate
    -- server types
}
APPENDIX G. XML COMMAND SPECIFICATION

This section will give details on the commands that will be used between the communicating entities in this protocol (mail client, mail server, and certificate server) in the form of XML schema files. There are also several sample XML commands shown at the end of this section. It is important to remember that although the sample XML commands appear unencrypted, all of this data would be sent over a TLS encrypted connection. In addition, some sections (such as the MessageData section inside of the Message section) would themselves be encrypted; however, this schema files show what the decrypted data contained in these sections should look like. Sections G.1 through G.17 show actual complete commands that will be used over the network, while Sections G.18 through G.29 show pieces of commands that are separated out for clarity or reuse. G.30 will show a sample communication flow between a mail server and mail client when picking up messages. Here, some data has been replaced with a placeholder for clarity.

G.1 SigningRequestAuthentication.xsd

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SigningRequestAuthentication" type="xsd:base64Binary"/>
</xsd:schema>
```

G.2 SigningResponseAuthentication.xsd

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="Alias.xsd"/>
  <xsd:element name="SigningResponseAuthentication">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Alias"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
<xsd:element name="Name" type="xsd:string"/>
<xsd:element name="Type">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="PERSONAL"/>
            <xsd:enumeration value="BUSINESS"/>
            <xsd:enumeration value="EBUSINESS"/>
            <xsd:enumeration value="FINANCIAL"/>
            <xsd:enumeration value="GOVERNMENT"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="Provider" type="xsd:string"/>
<xsd:element name="Title" type="xsd:string"/>
<xsd:element name="Affiliation" type="xsd:string"/>
</xsd:complexType>
</xsd:schema>

G.3 SigningRequest.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="SigningRequest" type="xsd:base64Binary"/>
</xsd:schema>

G.4 SigningResponse.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:include schemaLocation="Certificate.xsd"/>
    <xsd:element name="SigningResponse">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="Certificate"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
G.5 CertificateRequest.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="Alias.xsd"/>
  <xsd:element name="CertificateRequest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Alias" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

G.6 CertificateResponses.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="Certificate.xsd"/>
  <xsd:include schemaLocation="Alias.xsd"/>
  <xsd:element name="CertificateResponses">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="CertificateResponse" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="Alias"/>
              <xsd:element ref="Certificate" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

G.7 CertificateVerificationRequest.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="Alias.xsd"/>
  <xsd:element name="CertificateVerificationRequests">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="CertificateVerificationRequest" maxOccurs="unbounded">
          <xsd:complexType>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
G.10 MessageReceived.xsd

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="MessageID.xsd"/>
  <xsd:include schemaLocation="Status.xsd"/>
  <xsd:element name="MessagesReceived">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="MessageReceived" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="MessageID"/>
              <xsd:element ref="Status" minOccurs="0"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

G.11 RetrieveMessages.xsd

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="MessageID.xsd"/>
  <xsd:element name="RetrieveMessages" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="MessageID" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

G.12 DeleteMessages.xsd

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="MessageID.xsd"/>
  <xsd:element name="DeleteMessages" minOccurs="0" maxOccurs="unbounded">
</xsd:schema>
```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element ref="MessageID" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>

G.13 DeletedMessages.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="MessageID.xsd"/>
  <xsd:include schemaLocation="Status.xsd"/>
  <xsd:element name="DeletedMessages" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="DeletedMessage" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="MessageID" maxOccurs="1"/>
              <xsd:element ref="Status" minOccurs="0"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

G.14 RetrieveMessageList.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="RetrieveMessageList">
    <xsd:complexType>
      <xsd:attribute name="type" use="required">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="OLD"/>
            <xsd:enumeration value="NEW"/>
            <xsd:enumeration value="ALL"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
**G.15 MessageList.xsd**

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="MessageID.xsd"/>
  <xsd:element name="MessageList">
    <xsd:complexType>
      <xsd:element name="NewMessages" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="MessageID" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="OldMessages" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element ref="MessageID" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

**G.16 EndCommunication.xsd**

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="Status.xsd"/>
  <xsd:element name="EndCommunication">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Status"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
G.17 Ready.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Ready"/>
</xsd:schema>

G.18 Alias.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Alias">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:pattern value="([.a-zA-Z0-9_-]+@([a-zA-Z0-9_-]+)(([a-zA-Z0-9_-]+\.|)[a-zA-Z0-9_-]+])*\([a-zA-Z0-9_-]+\)+"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
</xsd:schema>

G.19 Certificate.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Certificate" type="xsd:base64Binary"></xsd:element>
</xsd:schema>

G.20 CertificateStatus.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="CertificateStatus">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="VALID"></xsd:enumeration>
        <xsd:enumeration value="REVOKED"></xsd:enumeration>
        <xsd:enumeration value="UNKNOWN"></xsd:enumeration>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
</xsd:schema>
G.21 Code.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Code">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="SUCCESS"/>
        <xsd:enumeration value="FAILURE"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:element>
</xsd:schema>

G.22 MailServer.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="TimeStamp.xsd"/>
  <xsd:include schemaLocation="Alias.xsd"/>
  <xsd:element name="MailServer">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Alias"/>
        <xsd:element name="Action" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="TimeStamp"/>
              <xsd:element name="OldID" type="xsd:long" minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute name="type" use="required">
              <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="RECEIVED"/>
                  <xsd:enumeration value="FORWARDED"/>
                  <xsd:enumeration value="DELIVERED"/>
                  <xsd:enumeration value="CHANGEDID"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:attribute>
          </xsd:complexType>
        </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:complexType>
</xsd:schema>
G.23 Message.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="MessageID.xsd"/>
  <xsd:include schemaLocation="MessageData.xsd"/>
  <xsd:include schemaLocation="MailServer.xsd"/>
  <xsd:include schemaLocation="Recipient.xsd"/>
  <xsd:include schemaLocation="SymmetricKey.xsd"/>
  <xsd:element name="Message">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Recipients">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="Recipient" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="SymmetricKey" type="xsd:base64Binary"/>
        <xsd:element ref="MessageData"/>
        <xsd:element name="DigitalSignature" type="xsd:base64Binary"/>
        <xsd:element name="Path">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="MailServer" maxOccurs="unbounded" minOccurs="0"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

G.24 MessageData.xsd

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="Certificate.xsd"/>
  <xsd:include schemaLocation="TimeStamp.xsd"/>
  <xsd:include schemaLocation="Recipient.xsd"/>
  <xsd:element name="MessageData">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="TimeStamp"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
<xsd:element name="Subject" type="xsd:string"/>
<xsd:element name="MessageText" type="xsd:string"/>
<xsd:element ref="Certificate"/>
<xsd:element name="RecipientList" minOccurs="0">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Recipient" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="AttachmentList" minOccurs="0">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Attachment" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Data" type="xsd:base64Binary"/>
          </xsd:sequence>
          <xsd:attribute name="filename" use="required" type="xsd:string"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>

G.25 MessageID.xsd

<xsd:element name="MessageID" type="xsd:long"/>
</xsd:schema>

G.26 Recipient.xsd

<?xml version="1.0"?>
<xsd:include schemaLocation="SymmetricKey.xsd"/>
<xsd:include schemaLocation="Alias.xsd"/>
<xsd:element name="Recipient">
  <xsd:complexType>
<xsd:sequence>
  <xsd:element ref="Alias"/>
  <xsd:element ref="Alias"/>
  <xsd:element ref="SymmetricKey" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>

**G.27 Status.xsd**

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include schemaLocation="Reason.xsd"/>
  <xsd:include schemaLocation="Code.xsd"/>
  <xsd:element name="Status">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Code"/>
        <xsd:element ref="Reason" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

**G.28 SymmetricKey.xsd**

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="SymmetricKey" type="xsd:base64Binary"/>
</xsd:schema>
```

**G.29 TimeStamp.xsd**

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="TimeStamp" type="xsd:long"/>
</xsd:schema>
```

**G.30 Sample Commands to Retrieve Messages**

Mail Client to Mail Server
<RetrieveMessageList type="NEW"/>
Mail Server to Mail Client
<MessageList>
  <!--Element NewMessages is optional-->
  <NewMessages>
    <MessageID>5</MessageID>
    <MessageID>6</MessageID>
    <MessageID>7</MessageID>
  </NewMessages>
</MessageList>

Mail Client to Mail Server
<RetrieveMessages>
  <MessageID>5</MessageID>
</RetrieveMessages>

Mail Server to Mail Client
<SendMessages>
  <Message>
    <SymmetricKey>BASE64BINARY</SymmetricKey>
    <MessageData>
      <TimeStamp>1167613871</TimeStamp>
      <Subject>Update your account information</Subject>
      <MessageText>Please update your account information at your earliest convenience.</MessageText>
      <Certificate>BASE64BINARY</Certificate>
    </MessageData>
    <DigitalSignature>BASE64BINARY</DigitalSignature>
  </Message>
</SendMessages>

Mail Client to Mail Server
<MessagesReceived>
  <MessageReceived>
    <MessageID>5</MessageID>
    <Status>
      <Code>SUCCESS</Code>
    </Status>
  </MessageReceived>
</MessagesReceived>

Mail Server to Mail Client
<Ready/>

Mail Client to Mail Server
<EndCommunication>
  <Status>
    <Code>SUCCESS</Code>
  </Status>
</EndCommunication>
APPENDIX H. NETWORK COMMUNICATION

FLOWCHARTS

This section provides flow charts for the communication that takes place between two entities in this system. Each step of the flow chart will be labeled with the sending and receiving entity, as well as the XML scheme file that the command being sent is based off of. If at any time an entity receives a command that it is not expecting, it will send an end communication command and terminate the connection. Also notice that the send message flow is reused for mail server to mail server connections, mail client sending mail, and mail clients receiving mail.

![Certificate Signing Request Flowchart](image-url)

Figure 45. Certificate Signing Request Flowchart
Certificate Retrieval Request Flow

Figure 46. Certificate Retrieval Flowchart
**Certificate Verification Request Flow**

**Figure 47. Certificate Verification Flowchart**

**Send Message Flow**

**Figure 48. Send Message Flowchart**
Client to Mail Server Connection Flow

Figure 49. Mail Client to Mail Server Connection Flowchart
BIBLIOGRAPHY


Web site: http://www.ietf.org/rfc/rfc0821.txt


