3-1-2005

# Design and Evaluation of a New Authentication Mechanism for Validating the Sender of an Email

Sai Sakamuri
*University of South Florida*

# Design and Evaluation of a New Authentication Mechanism for Validating the Sender of an Email

by

Sai B. Sakamuri

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Kenneth J. Christensen, Ph.D.
Miguel Labrador, Ph.D.
Nagarajan Ranganathan, Ph.D.

Date of Approval:
March 1, 2005

Keywords: spam, SMTP, sender payment, networks, spoofing

**DEDICATION**

This thesis is dedicated to my parents and my loving husband

## ACKNOWLEDGMENTS

I would like to take this opportunity to thank everyone who enabled my research. I would like to express my sincere gratitude and appreciation to my advisor, Professor Kenneth J. Christensen, for providing me with the opportunity to work in the research area of controlling spam, for his expert guidance and mentorship, and for his encouragement and support at all levels.

I would also like to thank Professors Miguel Labrador and Nagarajan Ranganathan for serving on my committee and for offering constructive comments. . I would like to thank Dr. David Rabson , Department of Physics for offering his comments and support. I would like to thank all staff members of USF Tech Support for their timely help in supporting my test environment.

I would like to thank my family and friends for their life-long love and support. Finally, special thanks go to my husband Kamal, for his support, encouragement and patience.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

**DESIGN AND EVALUATION OF A NEW AUTHENTICATION MECHANISM FOR VALIDATING THE SENDER OF AN EMAIL**

Sai B Sakamuri

## ABSTRACT

A new authentication mechanism for validating the source of messages over the Internet is designed and evaluated. This mechanism is applied to email and is called Email++. Email++ prevents identity forging (spoofing) and tampering of email contents. By preventing identity forging, Email++ can reduce the amount of spam received and limit the spread of viruses like Melissa, Love Bug, Bagle Worm, and Killer Resume. Email++ validates both the sender and the receiver of an email by confirming the sender's identity with the domain mail server that delivered the email for the sender, and authenticates the receiver with hash value comparisons. Email++ enables payment mechanisms, including micro-cash, and challenge response schemes that use puzzle solving.

MD5 hash signatures generated both at the sender and the receiver locations are used for validating the sender's identity and for making email tamper resistant in the network. An out-of-band TCP connection established between the sender and the receiver is used as a communication channel for validating the sender as well as the sender's email server. The information needed for establishing an out-of-band TCP connection is obtained by querying the DNS (Domain Naming System), instead of using email headers from the received mail, which are susceptible to spoofing.

The Email++ technique is compared with existing anti spam and anti-spoof techniques like SPF, Yahoo Domain Keys, Microsoft Sender ID, TEOS and PGP. The Email++ specification is evaluated by developing both Email++ client and Email++ server programs in C language and using Sendmail 8.12 as the mail server. The performance of Email++ is compared with standard SMTP protocol implementation of Sendmail 8.12. Several factors are considered in evaluating the performance. CPU demand, memory demand, bandwidth demand, email latency, and extra DNS load are measured for both email sender and the receiver. The performance evaluation results show that Email++ adds an extra CPU demand of about 11%. The extra memory required by Email++ is nearly 3%. The bandwidth demand of Email++ is around 15% greater than the standard SMTP for sending 500 emails of 3.5KB each. Extra load on DNS increases by one connection for every incoming mail at the receiver.

# CHAPTER 1

# INTRODUCTION

## 1.1 Background

Spam is becoming a major security threat to email systems. According to Verisign Inc. 80% of messages handled by its clients are classified as spam [34]. To date there is no perfect solution to stop spam completely, but groups of techniques can reduce the amount of spam. This thesis investigates a new method to reduce spam by preventing spoofing of email addresses.

## 1.2 Motivation

Network protocols and applications are designed based on the concept of trusting the network and its users. Security was not a vital concern when the original Internet protocols were being designed. Hence, protocols such as Simple Mail Transfer Protocol (SMTP), Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and Simple Network Management Protocol (SNMP) are all susceptible to spoofing. Vulnerabilities present in SMTP have made possible problems like spamming, phishing, email generated viruses and zombie SMTP servers. All the above techniques use email address spoofing; According to Federal Trade Commission's report [15] 33% of the spammed emails are spoofed; Reference [23] shows 50% of email is spoofed. A study by Verisign Inc. shows that 93% of the phished emails are spoofed [36]; 100% of email generated viruses are spoofed. The underlying security hazard of these security threats is spoofing. Consequently, there is a need to incorporate sender authentication in SMTP, which is the main contribution of this thesis.

## 1.3 Contribution of this Thesis

This thesis investigates the problem of spoofing, in particular in email systems, and proposes a new authentication mechanism for avoiding email forgery. Contributions include:

- Review of existing proposals to control spam and email spoofing
- Design and implementation of Email++, a new sender authentication mechanism that will prevent spoofing and in turn help reduce spam.
- Comparison of Email++ to existing anti-spoof and anti spam techniques.
- Performance evaluation of Email++ and description of the adaptation of this authentication mechanism to other systems.

## 1.4 Organization of this Thesis

This section describes the organization of the remainder of this thesis

- Chapter 2 describes the background of SMTP and spam, and reviews existing methods used to control spam
- Chapter 3 describes the new email sender authentication mechanism called Email++, and describes the design, implementation, operation and validation of Email++

- Chapter 4 compares Email++ with other popular existing sender authentication mechanisms like SPF, Yahoo Domain Keys, sender ID, Microsoft ticket server, PGP, and TEOS
- Chapter 5 evaluates the performance of Email++ with respect to CPU utilization, memory demand, bandwidth demand, email latency, and DNS load
- Chapter 6 presents conclusions and discusses future research directions.

# CHAPTER 2

# BACKGROUND ON EMAIL AND SPAM

## 2.1 Overview of the Simple Mail Transfer Protocol (SMTP)

This chapter describes SMTP, SMTP forging, the problem of spam and the existing techniques used to control spam.

### 2.1.1 History of Email and Origin of SMTP

Originally, email was designed for the communication of information between groups of researches in the ARPANET network [4]. Even though the operation of email may appear simple, there are many conventions, standards and protocols behind its operation.

ARPANET, the research project of DARPA (Defense Advanced Research Projects Agency) was the predecessor of the Internet. In order for the small circle of ARPANET users to be able to communicate, SNDMSG [19], the first email program, was developed and deployed. The earliest protocol used for delivering the mail was FTP. Some additional functionality was added to FTP to make it efficient for delivering mail. A separate protocol for email delivery was proposed and submitted to the IETF as RFC 821 [27] in 1982. This RFC described Simple Mail Transfer Protocol (SMTP) for handling mail. Later in 1989, RFC1123 [8] was developed to clarify and improve some of the specifications of RFC821. RFC 1425 [20] describes a framework over which all the future extensions of SMTP can be built in a consistent way. RFC821, RFC1123 and RFC1425, collectively have been the core specification of SMTP for years.

### 2.1.2 Overview of SMTP

Internet email systems are client/server, store-and-forward systems that primarily uses SMTP and POP3 or IMAP for communication. The main components involved in the system are the sender User Agent (UA), sender Message Transfer Agent (MTA), receiver UA, receiver MTA and communication protocol. The UA provides the interface between the user and his/her email server. The UA can be described as a program the email user uses to compose or read messages. Sender UA interacts with the sender, accepts the message from the sender, and transfers the mail to the sender’s mail server by establishing a network connection to the sender’s mail server. Receiver UA helps the receiver in retrieving the messages from his/her mail server by establishing a network connection to their mail server. The MTAs also referred to as email servers are responsible for collecting emails from the senders and delivering them to the receiver’s MTA. The message can be delivered directly to the receiver MTA, or may pass through several intermediate MTAs before reaching the destination. To ensure reliability and assure delivery of emails TCP/IP is used as the transport protocol. The communication protocols used are SMTP and IMAP or POP3. SMTP is the protocol used to exchange messages between MTAs, and also by a UA to send messages to its MTA. POP3 or IMAP are used by receivers to retrieve messages from their MTA.

The sender UA initiates the communication with the sender’s mail server when a request to send mail is received from the sender. It establishes a two-way TCP communication channel to the sender's MTA on port 25, and the mail is delivered to the sender MTA through the established channel. The SMTP protocol is used for the communication between UA and the MTA.

The SMTP protocol defines a set of commands, which include information about the sender, receiver, and contents of the mail. The sender MTA accordingly receives the commands, processes them and replies with success or failure status codes. Figure 2.1 shows the flow of email using SMTP.

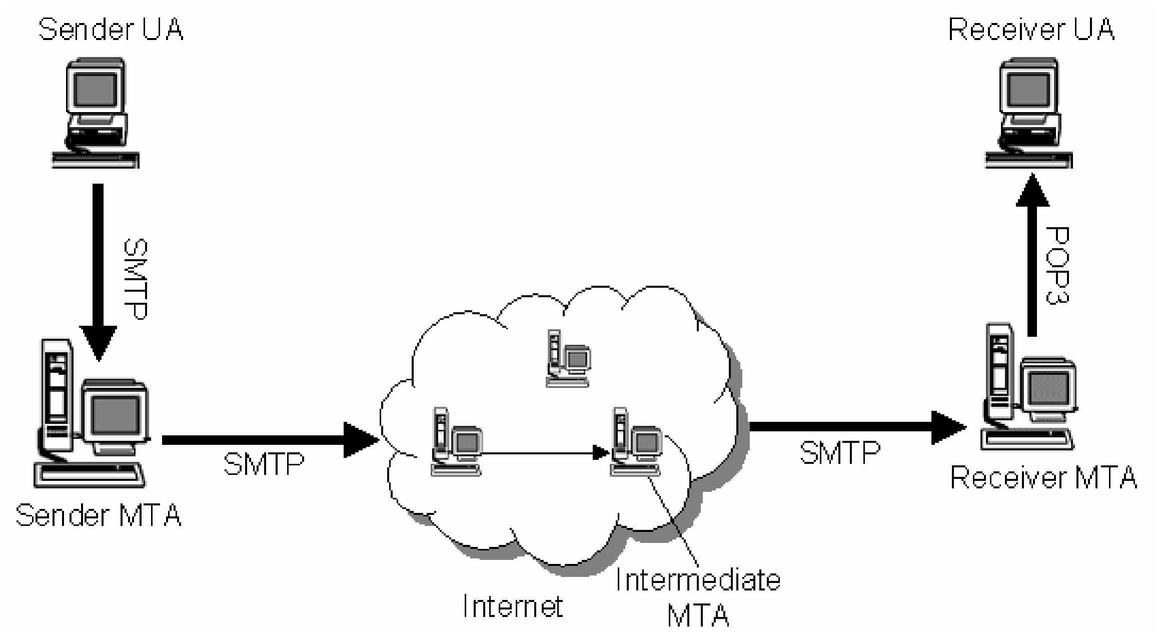


Figure 2.1: Flow of Email Using SMTP

The MTA processes the mail and places it in an outbound queue for delivery. The queue in the MTA is processed based on First Come First Serve (FCFS) or some other priority basis depending on the MTA configuration. Once the message is ready to be delivered, there are several things a sender MTA should do to transfer the mail:

- Determine the destination of the mail
- Establish a TCP connection with the next MTA
- Transfer the contents of the mail using SMTP as communication protocol
- Shut down the connection.

To determine the destination of the mail, MTA uses the receiver's email address, which contains the FQDN (Fully Qualified Domain Name). MTA uses the Domain Name System (DNS) to obtain the IP address of the receiver's MTA. To do this the MTA performs a DNS query requesting the receiver's MTA IP address. The MTA information is stored in mail exchange record (MX record) of the DNS entry for the domain. The MX record of the domain contains the host name of the mail exchanger of the receiver domain and the preference number of that exchanger. The MTA picks one of the hosts and delivers the mail using SMTP.

The next step involved in sending a mail is establishing an SMTP connection. This involves making a TCP connection to port 25 of the MTA or mail server. Sending the contents of the mail includes SMTP commands/replies between a client and server. There are several standard SMTP commands. Most prominent among them are HELO, MAIL FROM, RCPT TO, DATA and QUIT. Section 2.1.3 describes these commands in detail. Once the mail is delivered, the session is disconnected via the QUIT command.

### 2.1.3 Steps Involved in Sending and Receiving an Email

The Figure 2.2 shows the simple commands and replies involved in sending a mail. The numbers in front of the replies sent from SMTP server to SMTP client represent the reply codes. The client uses HELO command to introduce itself to the

SMTP server. The domain argument that is provided with HELO is used to populate the "`received:`" header and "`from:`" header in the email headers. Once the HELO command is received the server responds with a 250 reply code containing its domain name. After this command is successful, both client and server are ready for further SMTP commands. The MAIL FROM command is the first command in the mail transaction. The address that is provided with this command is used to populate the "`reverse-path:`" header in the email headers. The server validates the domain name of the sender by checking whether the domain exists. To issue a RCPT TO command, the MAIL FROM command must be successful. The argument with this command provides the recipient's email address. The address that is provided with this command is used to populate the "`forward-path:`" header. There can be several RCPT TO commands for one mail transaction.

The client uses a DATA command to send the actual mail content. The message provided with this data is used by the MTA to populate the mail-data buffer. Before using the DATA command, the MAIL FROM and RCPT TO commands should be issued. The end of the message is indicated via a single dot on a line by itself. Thus care should be taken to avoid a single dot on a line by itself in the actual content. If a single dot on a line is required in the data, dot stuffing is done by the server, which adds an extra dot beside the single dot. The QUIT command is issued by the sender after delivering the data. This command terminates the established connection.

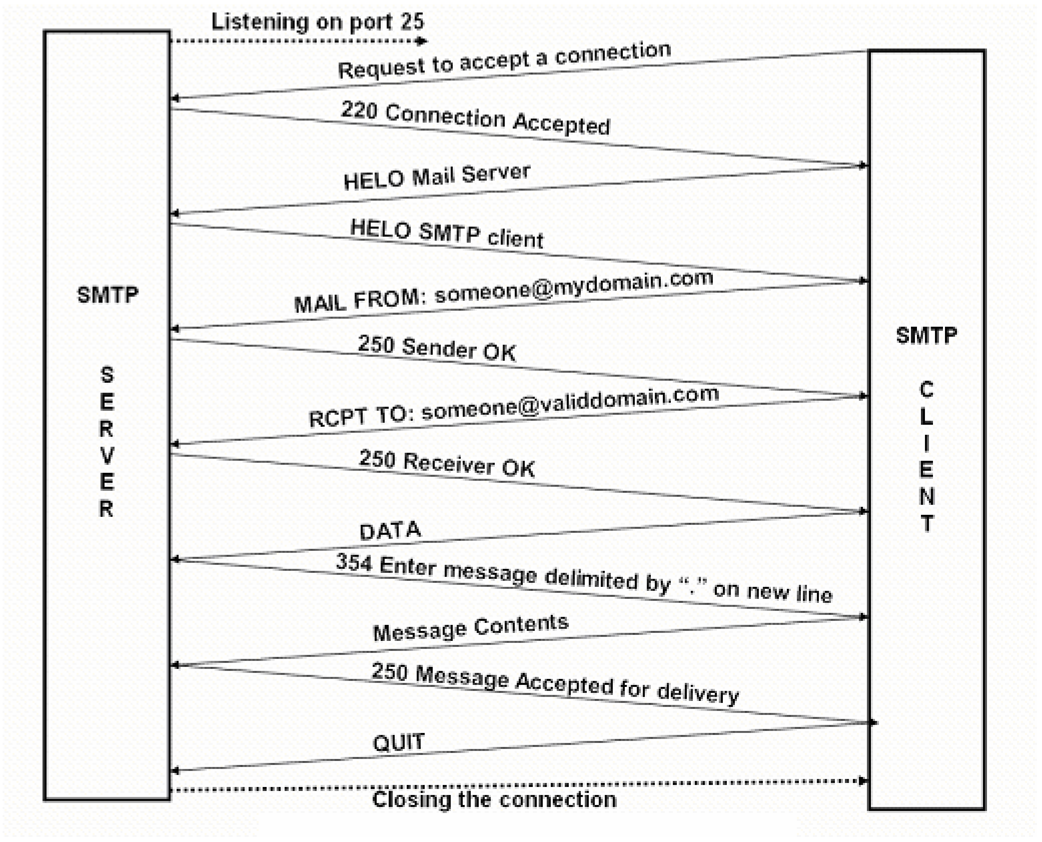

Listening on port 25
Request to accept a connection
220 Connection Accepted
HELO Mail Server
HELO SMTP client
MAIL FROM: someone@mydomain.com
250 Sender OK
RCPT TO: someone@validdomain.com
250 Receiver OK
DATA
354 Enter message delimited by "." on new line
Message Contents
250 Message Accepted for delivery
QUIT
Closing the connection
SMTP SERVER
SMTP CLIENT

Figure 2.2: Steps Involved in SMTP Handshake

## 2.2 Spoofing of SMTP Headers

Email spoofing means forging of SMTP header information in order to hide the actual origin of email. The spoofed email looks like it originated from somewhere else other than the actual source. To send spoofed email, a sender typically forges someone's email address to send the email.

Spoofing is possible because of vulnerability in SMTP protocol, which doesn't validate the sender of the email. This section describes the SMTP headers that are susceptible to spoofing. The "`from`" or "`Reply-to`" header is easily susceptible to spoofing. While doing an SMTP handshake, the MAIL FROM command can take any mail address; nowhere in the complete transaction is this address checked. The sender may provide any existing domain name and an existent or non-existent user in the domain. The mail is accepted and delivered to the destination. Another header that reveals the information is the "`Received:`" header. This header consists of the path the mail traveled before reaching the destination. The source domain and all the intermediate domains add a new "`Received:`" header to the email. Even this header is spoofable, however, through the introduction of dummy mail servers prior to the actual relaying of the mail through the network.

Spoofing is possible at two levels, the domain and the user levels. Domain level spoofing means forging a valid domain name in the SMTP header while sending the mail. This technique is used by the spammers to hide their original domain name and prevent their domain from being black listed. Black listing is a technique in which a domain that sends spam is added to the list of blocked domain databases. In domain level spoofing a valid domain name and non-existent user name is provided in the forged address. In user level spoofing a valid domain name and user of that domain is forged. User-level authentication helps in preventing users from spoofing other users within the domain.

This SMTP vulnerability has resulted in other problems like spamming, phishing and email-generated viruses. Spammers and virus generators can send email to anyone easily by spoofing email headers to hide their identity. Recent statistics on spam by Verisign Inc. show that over 80% of the email traffic was recognized as illegitimate spam. The same statistics also show an increase in email generated viruses and worms such as Melissa, Love Bug, Bagle Worm, and Killer Resume.

## 2.3 The Problem of Spam

Spam has become a major security issue in the email infrastructure. Its several disadvantages include the waste of resources like network bandwidth and it adds extra traffic in the network, which results in additional delay in the network. This affects other application in the network. Spam wastes time for workers in the organization in the form of categorizing and removing emails. It adds extra load on system administrators and mail servers who have to handle more traffic. Also, children receiving spam are exposed to adult content received through emails.

### 2.3.1 Techniques Used for Spamming Emails

This section discusses the techniques spammers use. These techniques include email harvesting, blasting through open relays, and using zombie servers and temporary servers.

Email harvesting is a technique spammers use to acquire email addresses. Harvesting is done by programs that look for email addresses from websites, chat rooms, newsgroups, and online directories for web pages and domains. Reference [14] describes the techniques spam receivers can follow to circumvent the harvesting software. Email masking is used to alter published email addresses so that harvesters do not recognize them as valid email addresses. In email masking the address is masked by adding some logical text that human beings can interpret to extract the proper email address and trick the harvesting programs.

Open relays are used by spammers for sending millions of emails. Open relays refer to a mail server that allows anyone to connect to it and thus send email anywhere.

This relaying has its own advantages, like support for mail forwarding, and the fact that users of that domain can send from anywhere on the Internet [18]. But this feature is abused by the spammers to send illegitimate emails. RFC 2505 [22] provides several recommendations for SMTP MTAs to reduce spam. It describes that closing of open relay is one of the practices that helps in reducing spam. Nowadays an open relay is blacklisted. Reference [26] shows a sample black listed database.

A zombie server refers to a computer that is infected by a Trojan horse virus that changes the settings of the infected computers and makes them work like an open relay for sending emails. According to Sandvine [32], a network management firm, 80% of spam was generated by zombie servers. Spammers host temporary mail servers to blast out mail, and shut down a server to divert the in-bound SMTP traffic coming from undelivered mail.

## 2.4 Existing Methods to Control Spam

Several methods have been proposed and implemented to reduce the amount of spam. The proposed techniques vary widely depending on the vulnerability they address, the technique they use, and the location where they are implemented. The existing methods can be broadly classified as filtering, sender authentication mechanisms, and payment-based methods.

### 2.4.1 Filtering

Filtering is a technique through which incoming mail is verified, suspicious mail is classified as spam, and only mail that passes verification is delivered to the receiver. This filtering is usually done by the filter software that executes either at the receiver before delivering the mail, or in the mail server that receives mail for that domain. Filtering is easy to implement over existing email systems.

#### 2.4.1.1 Filtering at the Receiver

Filtering of email at the receiver or at the receiver's SMTP server has become commonplace. Prominent filtering methods include keyword filters, scoring filters, and Bayesian filters. In keyword filtering, the contents of the email are searched for keywords that are most likely to appear in spam. A database for these keywords is maintained and constantly updated. Bypassing these filters is a very trivial task for spammers. Scoring filters are more efficient than keyword filters; these filters establish some rules and, based on them, assign a score to the keywords. The higher the score, the greater possibility of spam being present. The rules need to be constantly updated to maintain the efficiency of the filter. Bayesian filtering is the most accurate way to control spam [17]. This method uses the mail previously received by the user to form rules and a keyword database that dynamically adapts itself.

The solution offered by the filtering techniques is temporary, however. They need constant updating and sometimes are error prone due to false positives. (Some legitimate mails are classified as spam.) False positives resulting from filtering may be a more serious threat to users than spam [35].

#### 2.4.1.2 Filtering in the Internet

The mail server that receives mail for the domain can also employ filtering techniques such as black listing and white listing. Black listing is a technique by which the known domains that relay spam are listed in the database. The receiver's mail server checks the source of the mail it receives. If the domain is listed in the black list database the mail is classified as spam. White listing is a technique by which the mail received from addresses that are present in the address book of the receiver is assumed to be legitimate and is delivered to the receiver without any further filtering.

Distributed Checksum Clearinghouse (DCC) [12] and Vipul's Razor [37] are the two filtering techniques that rely on the fact that exactly the same spam mail is sent to several recipients within the domain. In this technique, when a user receives spam, the message is hashed and the checksum value is placed in the Vipul's server or DCC server. The recipient, after receiving an email, can hash and check if the hash is listed in the Vipul's or DCC server. If it is listed, the email is assumed to be spam and can be discarded.

### 2.4.2 Sender Authentication Techniques

The sender authentication techniques do not rely on the contents of email to decide whether the mail is genuine; instead they try to check the source of the mail, authenticating the sender of the mail and accepting any mail from a valid sender. The motivation behind such techniques is the fact that most spammers forge their identity and use unreliable sources for spamming. These techniques can be classified as anti-spoof techniques and central certifying authority techniques.

#### 2.4.2.1 Anti-spoof Techniques

These techniques avoid spoofing of email addresses and force spammers to send mail from their own domain. In these techniques the validation of the source of the mail is done at the receiver side. The techniques discussed here are SPF [21], Yahoo domain keys [10], sender ID [24], and TEOS [33].

SPF stands for Sender Policy Framework. Figure 2.3 shows the operation of SPF. In general, every domain has its own designated mail servers that send the SMTP traffic from that domain and act as mail exchangers. In SPF, every domain should publish the SPF records for each mail server of the domain. The format of SPF records is described in the Internet draft: each record should consist of the IP address of the mail server that delivers the mail. The SPF records are published in the DNS. The existing DNS records require modifications to include the SPF records. This makes the sending domain SPF compliant.

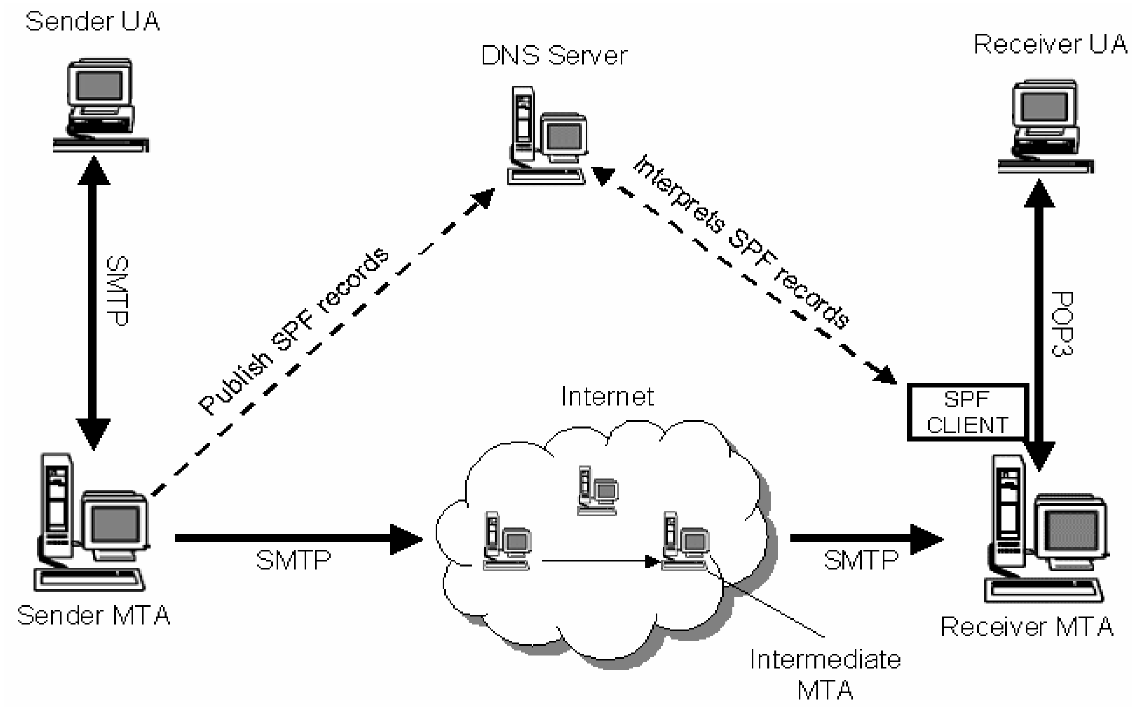


Figure 2.3: Overview of SPF

An SPF client should be installed on the receiver's side either on the receiver's user agent or mail server. An SPF client is capable of interpreting the SPF declarations for a domain. After receiving the mail, an SPF client extracts the sender's IP address from the "`Received`:" header and the domain name from the "`From`:" header of the email and performs the DNS lookup for the SPF record for that domain. Depending on the response from the DNS, it validates the sender. However SPF can detect only domain level spoofing. The user within the domain can forge other users in the same domain.

Sender ID [24] is a new proposal by Microsoft that relies on SPF. It addresses some of the issues that are not dealt with in SPF. Sender ID enhances SPF by allowing mail forwarding, enhancing the mechanism for mailing lists and mobile users. In Sender ID, the SPF records are published using XML format, unlike in SPF, where records are published in plain text. Sender ID doesn't provide more protection than SPF.

Yahoo, Inc. proposed another solution called Yahoo Domain Keys [10]. This technique uses PKI (Public Key Infrastructure) for sender authentication. In this technique, every domain should generate a public/private key. The public key should be published in the DNS. The mail sent from the domain is digitally signed with the private key of the domain and the signature is sent along with the mail. Figure 2.4 shows the operation of Yahoo Domain Keys. On the receiver side, the receiver extracts the domain name from the mail, and obtains the public key corresponding to that domain. Using the obtained public key the receiver signs the received mail and compares the signature with the one obtained in the mail. If both signatures match the mail is not spoofed. Even

Yahoo domain keys like SPF and Sender ID detects only domain level spoofing. None of these techniques can detect spoofing within the domain

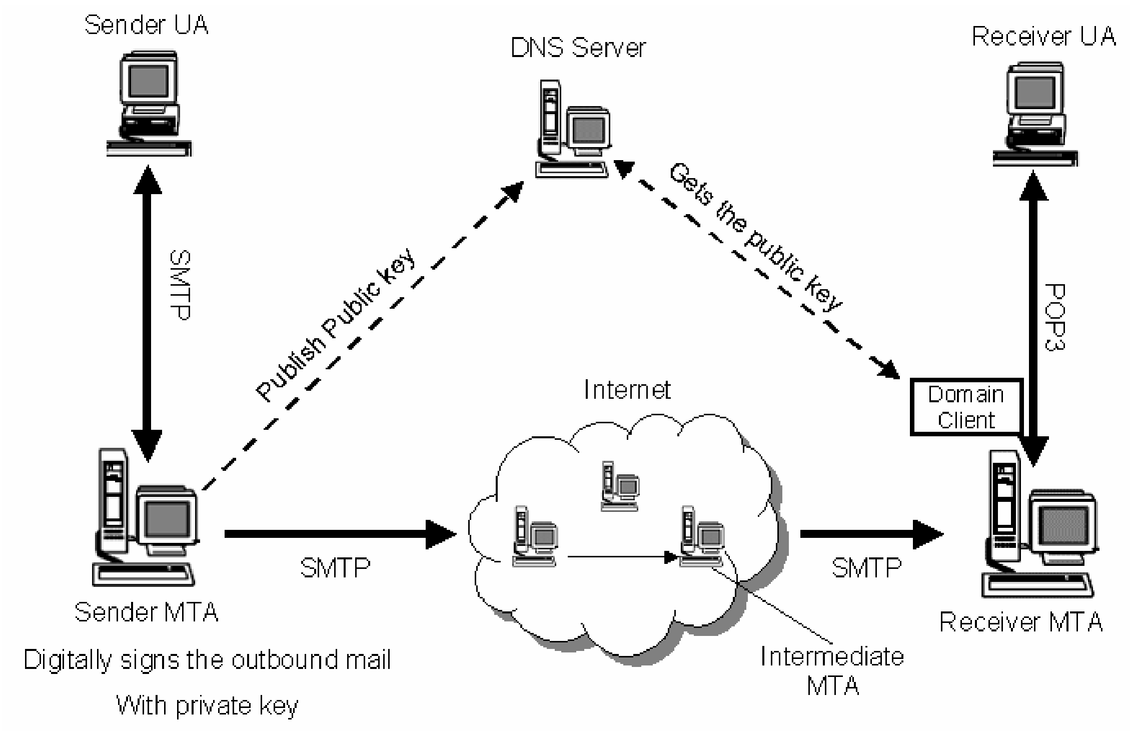


Figure 2.4: Overview of Yahoo Domain Keys

TEOS [33] stands for Trusted Email Open Standard. This standard describes a new sender authentication policy and message assertion system for getting accurate information about the sender and the message. It describes three levels of security: for normal senders, bulk senders and commercial senders. In the first level of security, the sender SMTP server should run Trusted Email Send Engine (TESE) and the receiver should run Trusted Email Receive Engine (TERE). The TESE generates a securely verifiable statement of sender domain identity before sending the mail. The statement is verified by the TERE. If the identity fails the sender domain is revoked. The statement is sent as an SMTP header. This process avoids domain level. In the second level security, along with first level, a Trusted Third Party (TTP) certificates are used to avoid both domain and user level spoofing. The third security level adds more accountability of the sender by using fully verified digital certificates for each email generated. The receiver should validate the certificate, and if the validation fails the mail is discarded.

#### 2.4.2.2 Trusted Third Party (TTP) - Certifying Authority

In trusted third party techniques, both the sender and receiver of the mail trust a central authority, which validates both the sender and the receiver. The central authority can pre-sign the mail or check the validity of the sender at the receiver's request once the mail has been received.

Microsoft has proposed a technique called Ticket server [1], which acts as a central authority. In this technique, every sender must have a ticket in order to send mail. Later, the receiver cancels the ticket and refunds the sender if the mail is from a trusted source. This technique uses a PKC. The sender, receiver, ticket and ticket server have a unique key for identification. The ticket server issues a challenge to the sender. After getting the appropriate response, a ticket is issued to the receiver. The sender encrypts the message with the obtained ticket and transmits the encrypted mail. After receiving the mail, the receiver authenticates it with the ticket server using an identification key and obtains the key of the sender. Later, the receiver encrypts the message with the sender's key to obtain the actual message. Optionally, the receiver can refund the key to the sender.

Pretty Good Privacy (PGP) [5] is another technique, which uses a PKC for authorizing the sender and the contents of the mail. In this technique, each sender should have a unique public-private key pair; before the mail is sent out, the message is signed with the sender's private key. The public key is published in the PGP key server, or on the sender's website. After receiving the mail, the receiver decrypts the mail with the public key. This technique avoids spoofing at the user level.

### 2.4.3 Payment-based Methods

Payment-based methods make spam expensive to send. The motivation behind such methods is that spam is used as a source of advertising and marketing because it is the most economic way to reach people around the world. Payment based methods can be classified as monetary payment schemes, CPU cycle payment schemes, and memory payment schemes.

#### 2.4.3.1 Monetary Payment Schemes

In monetary payment schemes, to make spam expensive to send, real currency mechanisms were proposed. Basically, the sender should spend money to send mail. Reference [29] describes Payword and Micromint, two micro-payment protocols. Reference [38] describes an enhanced micro-payment scheme. The practical implementation of such schemes for email systems did not seem feasible, since the mail had no currency and country limits. Cashramspam [9] was the first monetary payment scheme to be successfully implemented, and it was done so in Australia. In this system, users create accounts with Cashramspam and maintain a balance using credit cards. Though this may reduce spam, it is by nature susceptible to several forms of identity theft, as the credit card and other personal information need to be provided for transactions with Cashramspam.

**2.4.3.2 CPU Cycle Payment Schemes**

In CPU payment schemes, the sender of an email pays in terms of CPU cycles. In these techniques, a sender is issued puzzles that are CPU intensive. The result the sender sends to the receiver acts as a proof of work. If the result is acceptable, the mail is viewed by the receiver. With CPU payment schemes some of the resources on the sender side will be utilized for solving the puzzles, which may result in reducing the rate of spamming. The selection of the puzzles should be such that a normal user does not incur much load while sending mail, whereas for spammer, who is sending millions of emails per hour, the puzzles should cause an overload. In addition the puzzles should be easy to generate, easy to verify for the receiver, but time-consuming to solve for the sender. Reference [3] describes the hashcash scheme, which proposes the hashcash cost-function that can be used as a proof of work done by the sender of an email. Camram [8] is another payment scheme, which implements the concept of postage for electronic mail. One important feature the puzzle issued to the sender, should exhibit is a predictable solving time. Reference [30] proposes Time-Lock puzzles, whose solving time is predictable and are thus well suited for email systems. The amount of time required to solve the puzzles is deterministic. These puzzles are easy to create and verify and they cannot be parallelized.

### 2.4.3.3 Memory Payment Schemes

Memory payment schemes are similar to CPU cycle payment schemes but the puzzles that are issued to the sender are memory access intensive rather than CPU cycle intensive. The basic motivation for such schemes is that the time taken for solving the CPU intensive puzzles depends on the speed of the processor. For faster processors, time taken is less when compared to slower processors. Reference [2] proposes that the memory access speeds vary across machines much less than do CPU speeds; it also proposes that memory-bound functions may behave more similarly than CPU-bound functions. Reference [13] describes a few memory intensive puzzles, and proves by experimental results that memory-bound functions show more similarly than CPU- bound functions with slow and fast processors.

IM2000 [7] is a new protocol proposed by D. J. Bernstein. The gist of the protocol is that after the sender sends email it is not delivered to the receiver. Instead, it is stored under the sender's disk quota in the sender's mail server and a brief notification is sent to the receiver that an email is waiting from the sender. The receiver can download the mail directly from the sender's mail server if he or she trusts the sender. This protocol forces the sender to store all his sent mail. This makes it impossible to send millions of emails due to physical space constraints in any realistic sender. The receiver downloads the mail from sender's server directly. So all the spam mails would not be stored up by receivers.

## 2.5 Summary of Existing Methods of Controlling Spam

This chapter classified the existing solutions to three categories: filtering techniques, sender authentication techniques, and sender payment techniques. The figure 2.5 below shows the detailed classification of existing solutions to control spam.

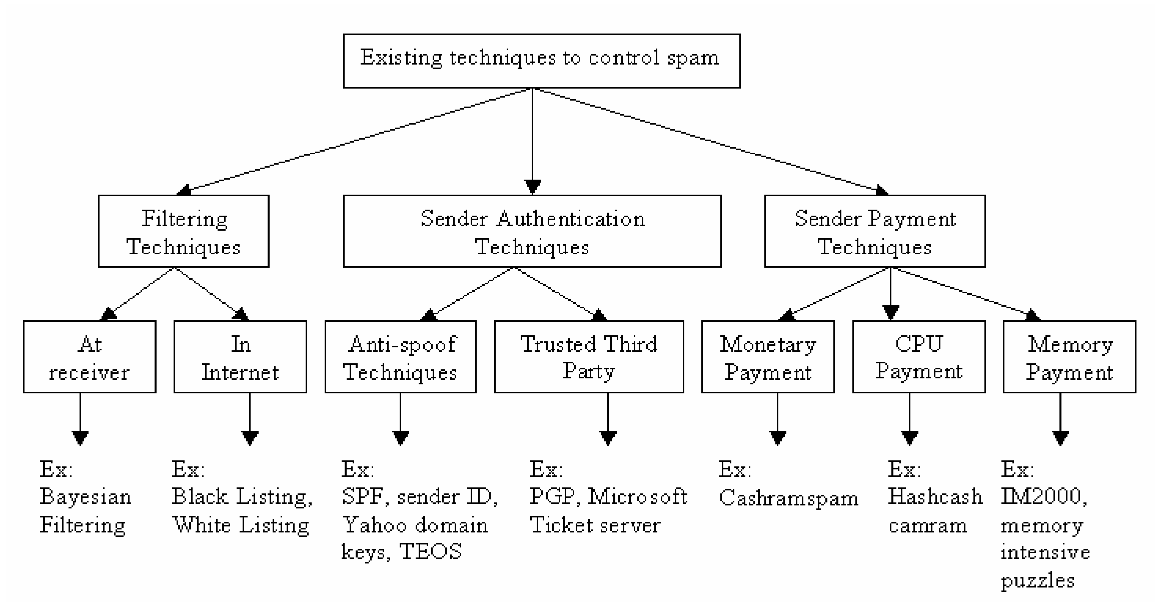

Existing techniques to control spam
Filtering Techniques
Sender Authentication Techniques
Sender Payment Techniques
At receiver
In Internet
Anti-spoof Techniques
Trusted Third Party
Monetary Payment
CPU Payment
Memory Payment
Ex: Bayesian Filtering
Ex: Black Listing, White Listing
Ex: SPF, sender ID, Yahoo domain keys, TEOS
Ex: PGP, Microsoft Ticket server
Ex: Cashramspam
Ex: Hashcash camram
Ex: IM2000, memory intensive puzzles

Figure 2.5: Classification of Existing Techniques to Control Spam

The conclusion that can be made after analyzing all the techniques is that no single solution can completely prevent or block spam. The filtering techniques either at the receiver or in the Internet needs constant maintenance with updating of filtering rules; Sender authentication techniques can solve the problem of spoofing, which helps in mitigating the problem of spam but doesn't solve the spam problem. The trusted third party sender authentication techniques are helpful, if deployed widely by all the existing servers. The sender payment schemes make spam expensive to send, but are difficult to implement.

# CHAPTER 3

# EMAIL++: NEW METHOD TO AUTHENTICATE EMAIL SENDER

This chapter describes the design, implementation, operation and validation of a new email sender authentication mechanism called Email++.

## 3.1 Objective

The objective is to design a new mechanism for authenticating the sender of an email. This mechanism can detect spoofing of email addresses and thus prevent an email receiving user agent from actually receiving a spoofed email. This makes SMTP a more secure communication format for email. The mechanism should conform to the requirements specified below.

## 3.2 Requirements of Sender Authentication System

This section describes the requirements of the new authentication mechanism.

- Compatible to the existing email system, with minimal changes to the existing SMTP.
- Capable of detecting both domain level and user level spoofing.
- Should not degrade the throughput or increase resource utilization and latency of the mail server or the mail client.
- Should not introduce any extra security threats that are not there in the existing email system.

- Should be easily implementable over the existing system without involving complex changes to the design of existing system.

## 3.3 Overview of Email++

This section presents an overview of Email++. For each email sent out from an authentic user of the domain, the sender's SMTP server calculates the MD5 hash [28] of the email and stores it in the user's account. The receiver, after receiving the email, calculates the MD5 value and sends it back to the sender's SMTP server. This communication is achieved via an out-of-band TCP connection made on some pre-determined port number. The receiver stores the mail in a temporary folder before delivering it to the inbox. If the mail is legitimate, then the MD5 value of the sender and the receiver should match. If this is the case, the sender's SMTP server sends "EMAIL_OK" to the receiver, and the mail is moved from the temporary folder to the receiver's inbox. In the case of spoofed email, the MD5 hash value of the receiver cannot be found at the sender and the mail is deleted from the temporary folder without being sent to the receiver's inbox.

Using an out-of-band TCP connection for sender validation is the key concept in Email++. To accomplish this, the sender's SMTP server runs a service for validating users of its domain. The receiver takes the domain information of the sender from the SMTP headers of the received mail and uses this domain information to query the DNS servers for the valid IP address of the sender's SMTP server. The receiver connects to the sender's SMTP server on a predetermined port number.

One more advantage to using this out-of-band connection is that the sender's SMTP server will be overloaded with a huge amount of queries from all of the users who received the email. If a spammer bursts out millions of emails, then his or her SMTP servers will have to answer millions of queries, which may result in Denial of Service (DoS) [16]. This forces the SMTP servers to validate their users before accepting an email, or place a limit on the amount of mail that a user can send.

## 3.4 Design of Email++

The existing SMTP protocol does not validate the sender's identity before accepting email for forwarding. This makes it easy for the sender to forge any domain or user identity within the domain. This vulnerability made spoofing and in turn spamming easy. Email++ addresses this issue by adding sender authentication capabilities to SMTP, making it more secure. This authentication is incorporated by making the receiver query the sender SMTP server to make sure the mail was in fact sent from the valid user of the domain. The figure below shows the normal email flow shown in Chapter 2 along with the incorporated new Email++ authentication capability.

From the Figure 3.1 we can see that the sender SMTP server and receiver host are involved in the authentication process. Both SMTP server and receiver host need modifications in order to incorporate the new authentication capability. This is achieved by adding the server modifications to the sender SMTP server, through an Email++ server, and the client modifications at the receiver through an Email++ client. The Email++ server is responsible for running the authentication service. The Email++ client is responsible for authenticating the sender of an email. The Figure 3.2 shows the new added components.

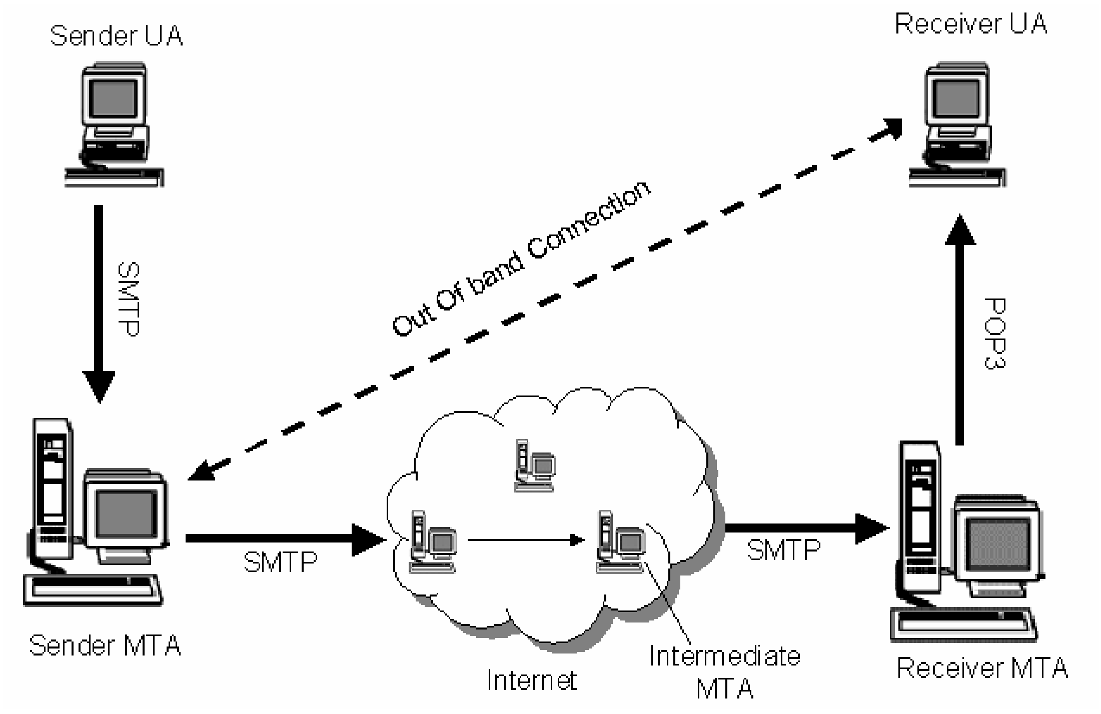


Figure 3.1: Email Flow Along with Email++ Authentication Capability

The dotted lines in Figure 3.2 indicate the new components added in Email++ that does not exist in the normal SMTP. The design of a new authentication mechanism is discussed separately for Email++ client and Email++ server.

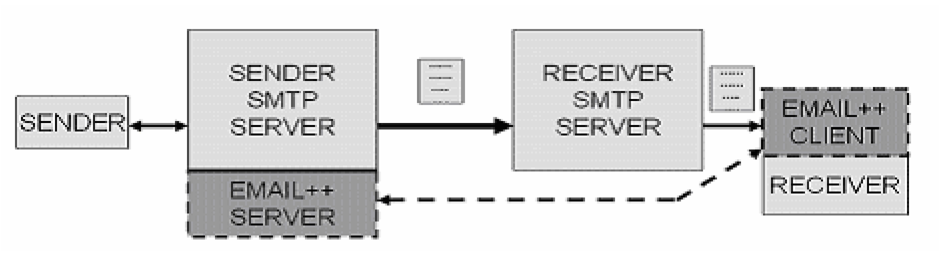


Figure 3.2: Top-level Design of Email++

### 3.4.1 Design of Email++ Server

This section describes the design of Email++ server. The Email++ enabled SMTP server must authenticate the sender before accepting the mail. If the authentication is successful then the email information is stored in the user's account. The Email++ enabled SMTP server should also host a validation service that accepts a request from the receiver and sends a response back. The response should indicate whether the mail was really sent from the valid user of that domain. Figure 3.3 shows the functionality of the Email++ server.

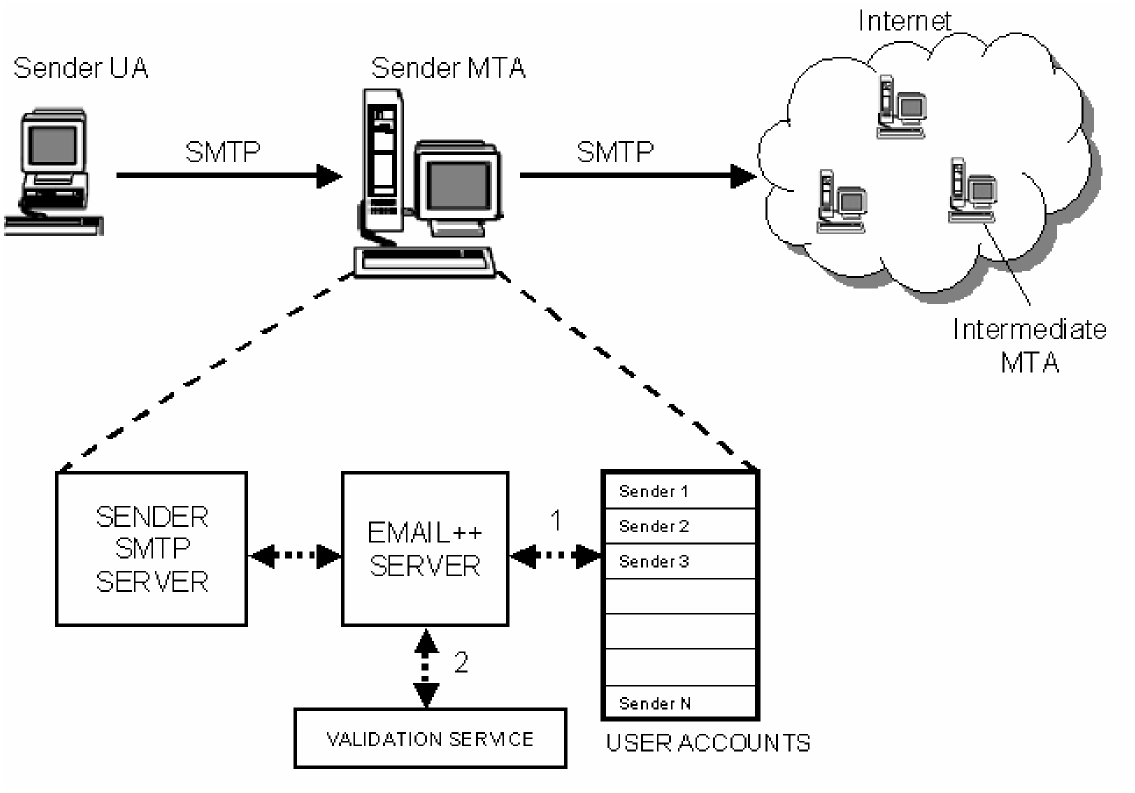


Figure 3.3: The Functionality of Email++ Server

In order to identify the valid senders of the domain, the SMTP server issues some challenges. This challenge response can be a simple username and a password. All users should be authenticated before they are allowed to use any services from the SMTP server.

### 3.4.1.1 Generating MD5 Hashes for Authenticated Users

Once the user is authenticated successfully, the Email++ server module extracts the required data from the SMTP headers for generating the MD5 hashes and saves the hash information in the user accounts. This functionality is shown as 1 in Figure 3.3. Only the email message and the receiver's email address are used in generating the hashes, because these contents are not modified in the email transit. One advantage of using the receiver's email address for generating the MD5 hash is that a sender who is sending to multiple recipients needs to generate multiple hashes. If the sender is a spammer relaying millions of emails, MD5 hash generation may cause an extra overhead. Figure 3.4 below shows a sample SMTP session. The underlined contents are used for generating the MD5 hash value.

```
220 localhost.localdomain ESMTP Sendmail 8.12.5/8.12.5; 14:36:18 -0400

HELO MICKEY

250 localhost.localdomain Hello giga4.csee.usf.edu [131.247.2.17], pleased to meet

MAIL FROM:sender@mydomain.com

250 2.1.0 ssakamur@mickey.csee.usf.edu... Sender ok

RCPT TO:receiver@somedomain.com

250 2.1.5 ssakamur@mickey.csee.usf.edu... Recipient ok
Figure 3.3        The functionality of Email++ server
DATA
354 Enter mail, end with "." on a line by itself

This is the body of the message
.
250 2.0.0 i94IaIWI004759 Message accepted for delivery
```

Figure 3.4: Contents Used for Generating the MD5 Hash

The RCPT TO: value and the DATA value are sent as input to the MD5 hash generating algorithm, which generates the 128-bit hash value. Figure 3.5 shows the sample hash generation for the above mail.

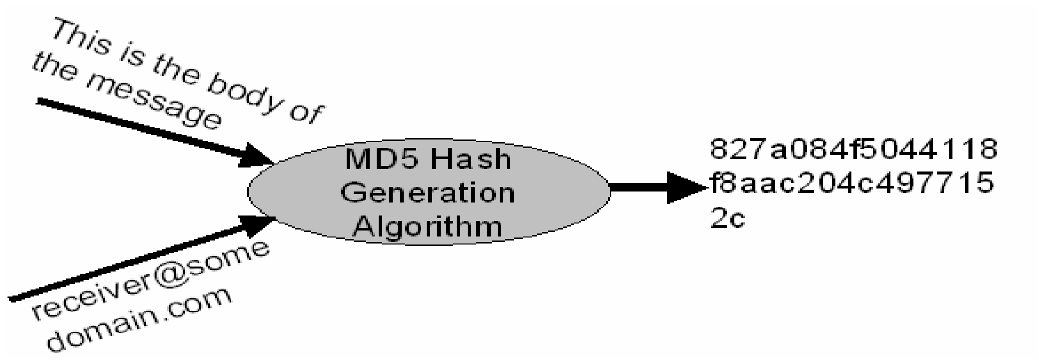

This is the body of the message
receiver@somedomain.com
MD5 Hash Generation Algorithm
827a084f5044118f8aac204c4977152c

Figure 3.5: Hash Generation Process

The hash value that is generated is a constant 128-bit value for any size of message. This hash value and the timestamp showing when the mail was accepted for delivery are stored in a separate log in each user account. The Email++ server accesses the account of the sender and saves the information for each email sent out by the sender. The stored information is of the format shown in Figure 3.6.

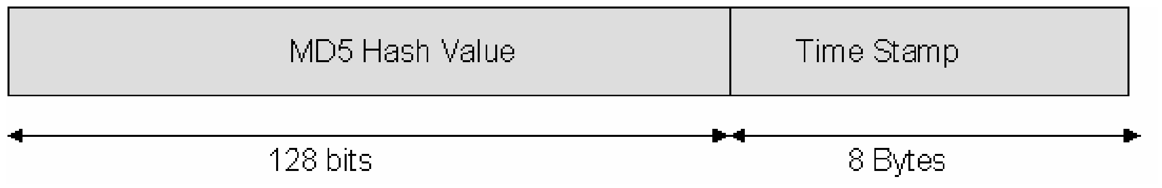


Figure 3.6 : Stored MD5 Message Format

The timestamp value is added to the contents of the mail and sent to the receiver. The overall functionality of hash generation and storing of information is shown in figure 3.7

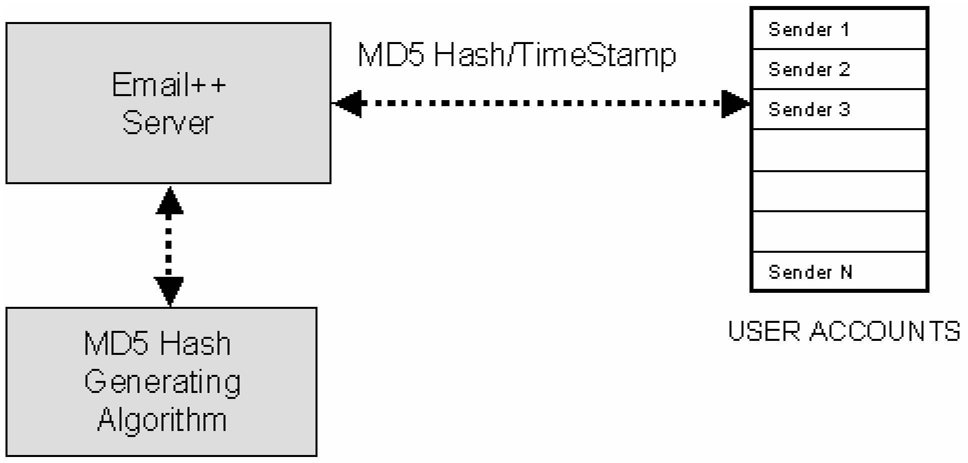


Figure 3.7: MD5 Hashing and Storing Process

All mail that reaches the SMTP server is forwarded in the network. The mail received from authenticated users (i.e. those users who authenticated successfully with username and password) is hashed and the information is stored in the respective user accounts and forwarded in the network. Even the mail received from users who did not authenticate successfully is forwarded in the network, assuming these users are using this domain mail server as an intermediate MTA. Thus Email++ allows open relaying and at the same time distinguishes the valid users of the domain.

#### 3.4.1.2 Running the Sender Validation Service

The sender SMTP server should host a validation service on some pre-determined port number for validating its domain users. This service is shown as 2 in Figure 3.3. The service should accept the out-of-band TCP connections from the receivers and obtain the account information and MD5 hash of the received mail. This information is compared against the information in the corresponding user's account. If a match is found an "EMAIL_OK" message is sent back to the receiver. If not, the message "EMAIL_FAIL" is returned to the receiver. "EMAIL_OK" implies that the mail originated from that specific domain and that the user was authenticated with the mail server before sending the mail. "EMAIL_FAIL" implies that the mail apparently originated in that specific domain, but that the user was not authenticated with the mail server, which means that the mail may be spoofed. Figure 3.8 below shows the communication between the receiver and the sender SMTP server.

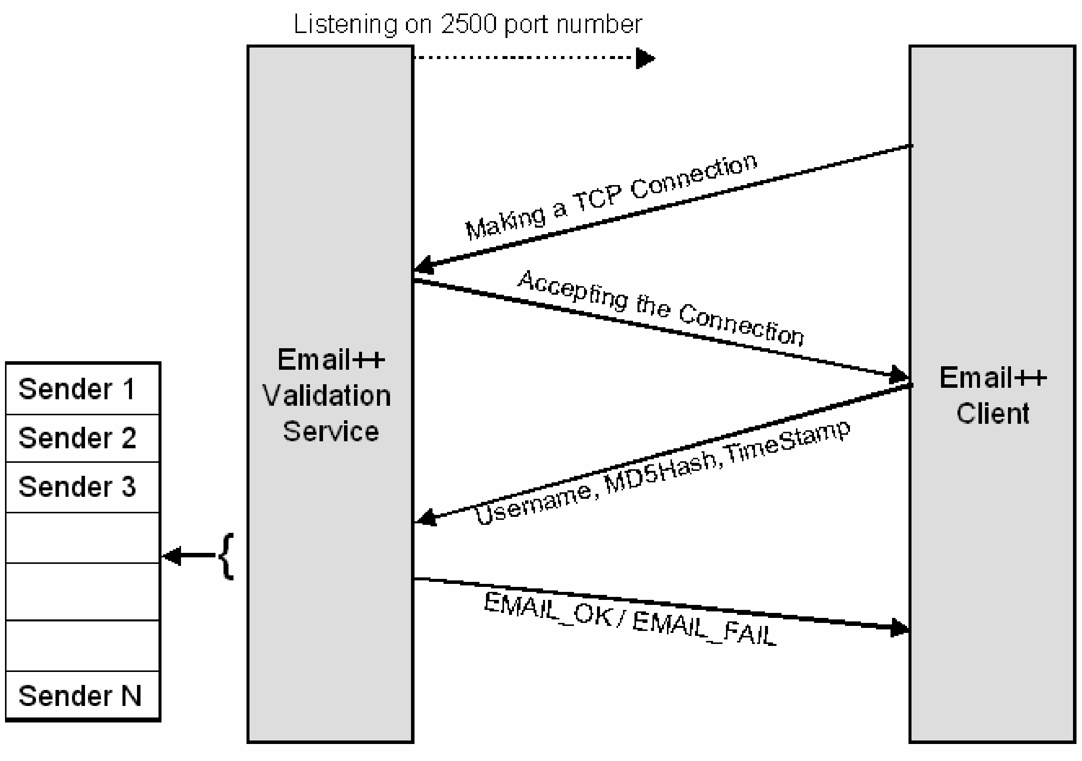


Figure 3.8: Out of Band TCP Connection

### 3.4.2 Design of Email++ Client

On the receiver side, the Email++ client receives the mail from the receiver's mail server before it is delivered to the receiver's inbox. The Email++ client implements a temporary mailbox, which stores all the new mail that is to be validated. Only the validated mail is placed in the receiver's mailbox. The process of sender authentication involves (1) obtaining the sender SMTP server's IP address to establish an out-of-band TCP connection; and (2) using the established out-of-band connection to inquire the sender's mail server whether the sender is a valid user of the domain, and whether the received mail was delivered from that domain. Figure 3.9 shows the functionality of the Email++ client.

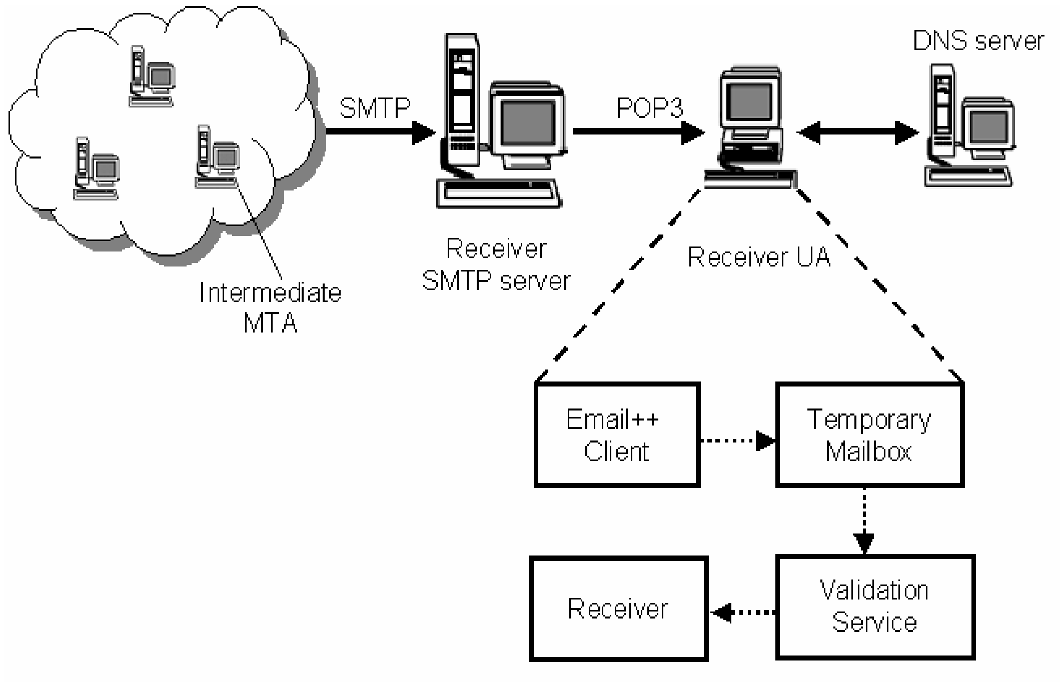


Figure 3.9: Functionality of Email++ Client

### 3.4.2.1 Obtaining the Sender Mail Server's IP Address

The most important issue for the receiver is getting the IP address of the sender's SMTP server. This function is labeled as 1 in Figure 3.9. The IP address can be obtained from the SMTP headers in the mail, but the headers are not reliable as they are susceptible to spoofing. In Email++, the receiver gets the IP address of the mail server by contacting the DNS [25]. The receiver extracts the domain name of the sender from the "`Apparently-From`" SMTP header and uses that domain name and queries the DNS server for the mail server's IP address. In the DNS records, the MX record consists of the IP address of the mail server for a given domain.

### 3.4.2.2 Authenticating the Sender

The receiver uses the IP address obtained by querying the DNS server to make an out-of-band TCP connection to the server on some predetermined port number. This port number should be the same as the port number that the Email++ server runs the validation service. The receiver also extracts the mail contents and the time stamp from the email. It appends the recipient mail address to the mail contents and generates the MD5 hash value. The MD5 hash, the time stamp and the username of the sender are sent to the sender's email server on the established TCP connection. This is done by the validation service in figure 3.9. The sender's email server extracts the hash value and searches for the matching MD5 value in the user's account. If a match is found, the sender is considered authenticated and the mail is not spoofed. If the mail is spoofed, the sender's address will be contacted but no MD5 match will be found in the sender's account as the sender did not send the mail, and an "EMAIL_FAIL" will be sent back to the receiver. This mechanism effectively detects spoofing both at the domain and user level. The message formats used for the communication are shown below. The time stamp is extracted from the mail contents and the sender name is taken from the "`Apparently-From:`" header, which contains the sender's email address.

If the response from the sender's server to receiver is "EMAIL_OK", the mail is removed from temporary mailbox and placed in the user's mailbox. If the reply is EMAIL_FAIL, the Email++ client deletes the mail before it is sent to the receiver's inbox. The message formats used for communication are shown in Figure 3.10.

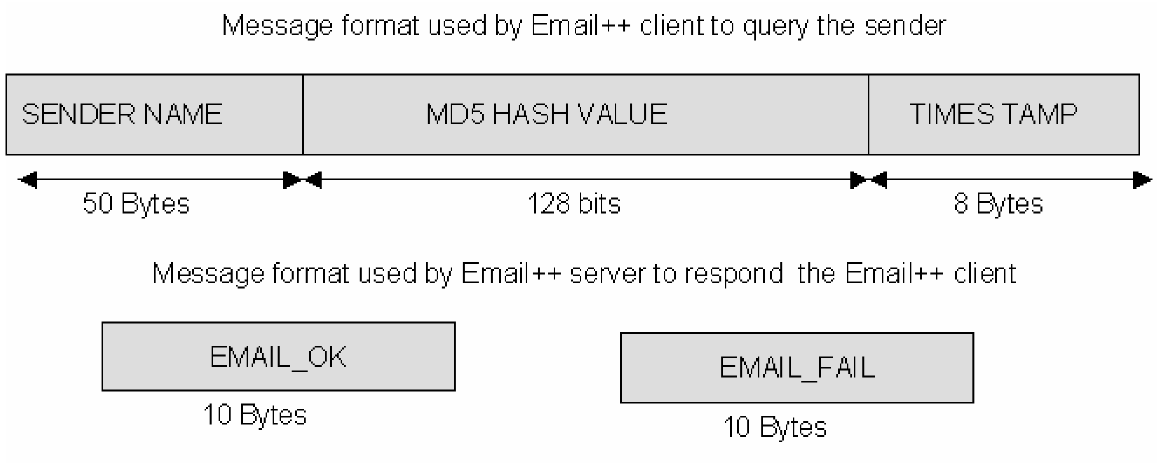

Message format used by Email++ client to query the sender
SENDER NAME
MD5 HASH VALUE
TIMES TAMP
50 Bytes
128 bits
8 Bytes
Message format used by Email++ server to respond the Email++ client
EMAIL_OK
10 Bytes
EMAIL_FAIL
10 Bytes

Figure 3.10: Message Formats for Out of Band Communications

## 3.5 Implementation of Email++

The Email++ mechanism can be incorporated into the existing SMTP framework by adding the sender authentication functionality and MD5 hash management functionality to the SMTP server. The validation service can be implemented as a separate server, but should be running in the same host as the SMTP server. The sender authentication is done using the Validate_Sender ( ) procedure. The procedure is shown in Figure 3.11

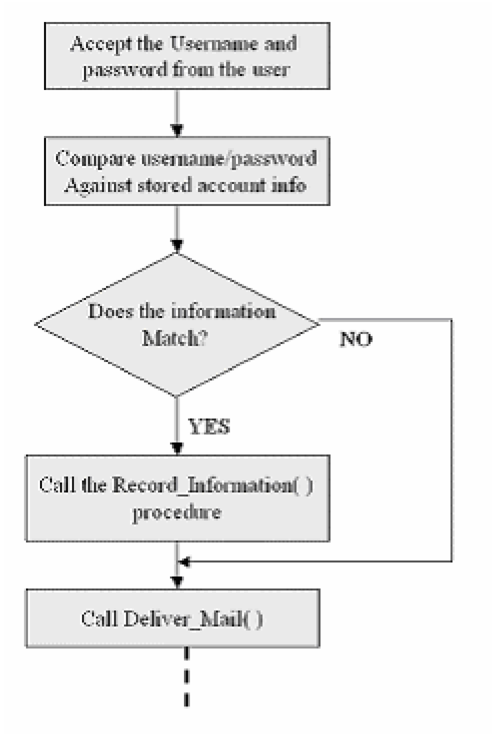

Accept the Username and password from the user
Compare username/password Against stored account info
Does the information Match?
NO
YES
Call the Record_Information( ) procedure
Call Deliver_Mail( )

Figure 3.11: The Validate_sender Procedure

The Record_Information() procedure takes the contents of the mail (MAIL_MESSAGE) and the recipient email address (RCPT_ADDRESS) extracted from DATA and RCPT TO: SMTP commands. Figure 3.12 shows the flow chart for the Record_Information( ) procedure.

The Generate_MD5_Hash( ) shown in figure 3.12 can be any MD5 implementation that is in conformance with the RFC 1321.

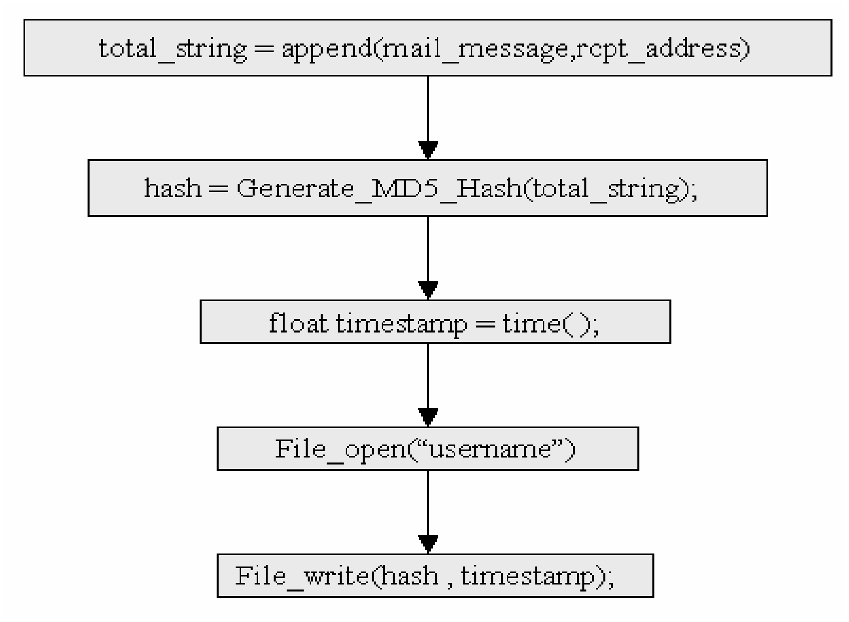

total_string = append(mail_message,rcpt_address)
hash = Generate_MD5_Hash(total_string);
float timestamp = time( );
File_open("username")
File_write(hash , timestamp);

Figure 3.12: Record_information Procedure

The Email++ client part can be added to the receiver's UA. The client part should establish an out-of-band TCP connection to the sender's SMTP server, and authenticate the sender. It should maintain a temporary mailbox for storing the mail initially. The receiver should query the DNS to get the IP address of the sender. The query/response can be a standard DNS query for MX record. Figure 3.13 shows the implementation of flow of communication between the sender's SMTP server and the receiver.

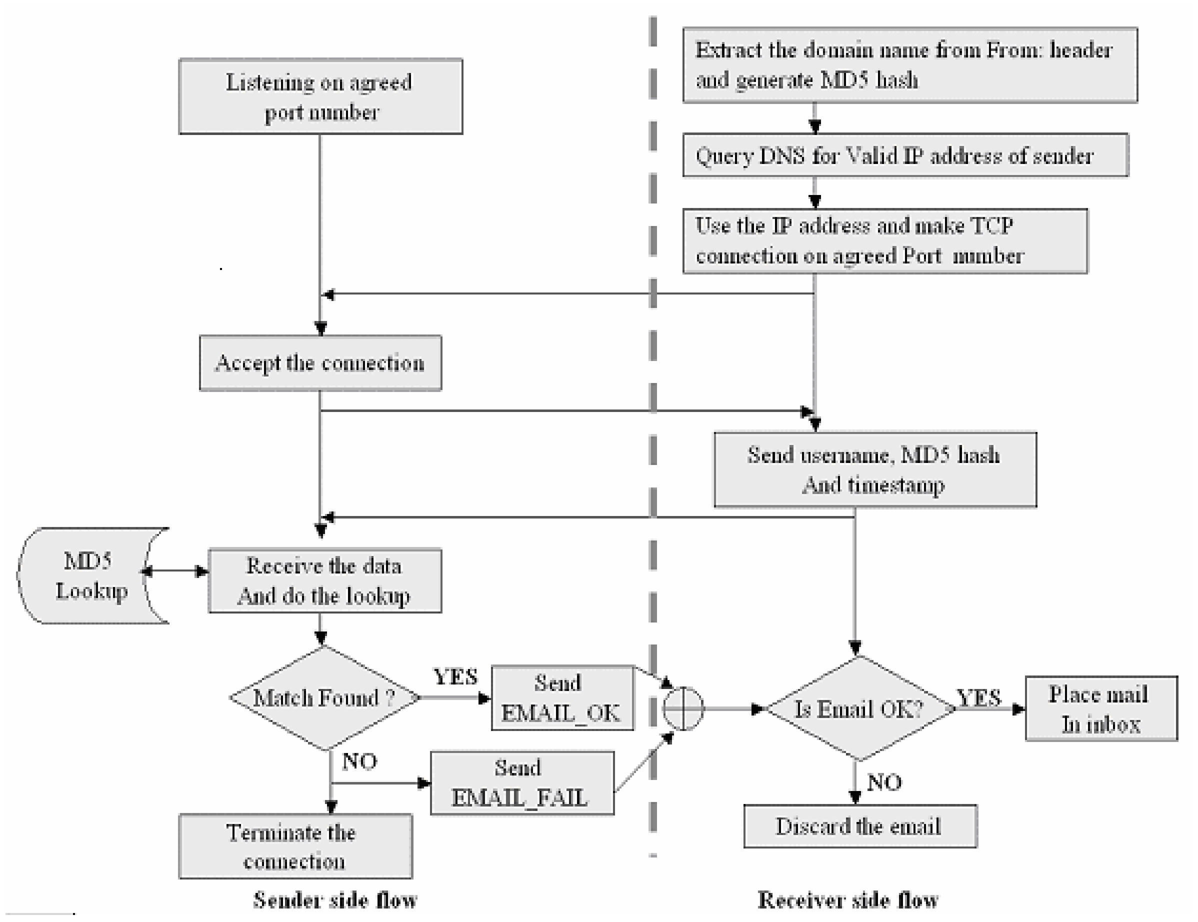

Listening on agreed port number
Extract the domain name from From: header and generate MD5 hash
Query DNS for Valid IP address of sender
Use the IP address and make TCP connection on agreed Port number
Accept the connection
Send username, MD5 hash And timestamp
MD5 Lookup
Receive the data And do the lookup
Match Found ?
YES
Send EMAIL_OK
NO
Send EMAIL_FAIL
Terminate the connection
Is Email OK?
YES
Place mail In inbox
NO
Discard the email
Sender side flow
Receiver side flow

Figure 3.13: Implementation of Communication Between
Sender's SMTP Server and the Receiver

If a sender payment scheme is to be implemented over the Email++ architecture, CPU-intensive puzzles like TimeLock Puzzles could be used. Reference [13] describes the application of TimeLock puzzles. The established out-of-band connection can be used by the receiver to issue the puzzle. The sender can send back the result on the same TCP connection

## 3.6 Operation of Email++

This section summarizes the operation of Email++ by showing the sequence of steps involved in sending an email.

- The sender MTA validates the sender by username/password, and accepts the mail for delivery.
- The validated sender information is stored in the MD5 log for each user. The MD5 hash of the message and the timestamp showing when the message was accepted for delivery are stored.
- All the mail received at the SMTP server is relayed in the network.
- Several intermediate MTA's may be involved in routing the mail to its final destination.
- When the receiver checks the mail, all the new emails are initially directed to the Email++ client, which stays in-between the receiver's SMTP server and the receiver. The Email++ client calculates the MD5 hash, and extracts the timestamp.
- Email++ client extracts the domain name from the MAIL FROM: SMTP header and queries the DNS for the IP address.
- The IP address obtained in Step 6 is used for establishing an out-of-band TCP connection. The Email++ client sends the MD5 hash, timestamp and username to the sender SMTP server. The MVS system checks for the matching MD5 in the users account. If there is a match, it sends "EMAIL_OK". If there is not a match, it sends "EMAIL_FAIL".

- Depending on the response, the mail is delivered to the receiver's inbox. In case of EMAIL_FAIL, the local policy (eg. To discard the email) will be applied to the mail.

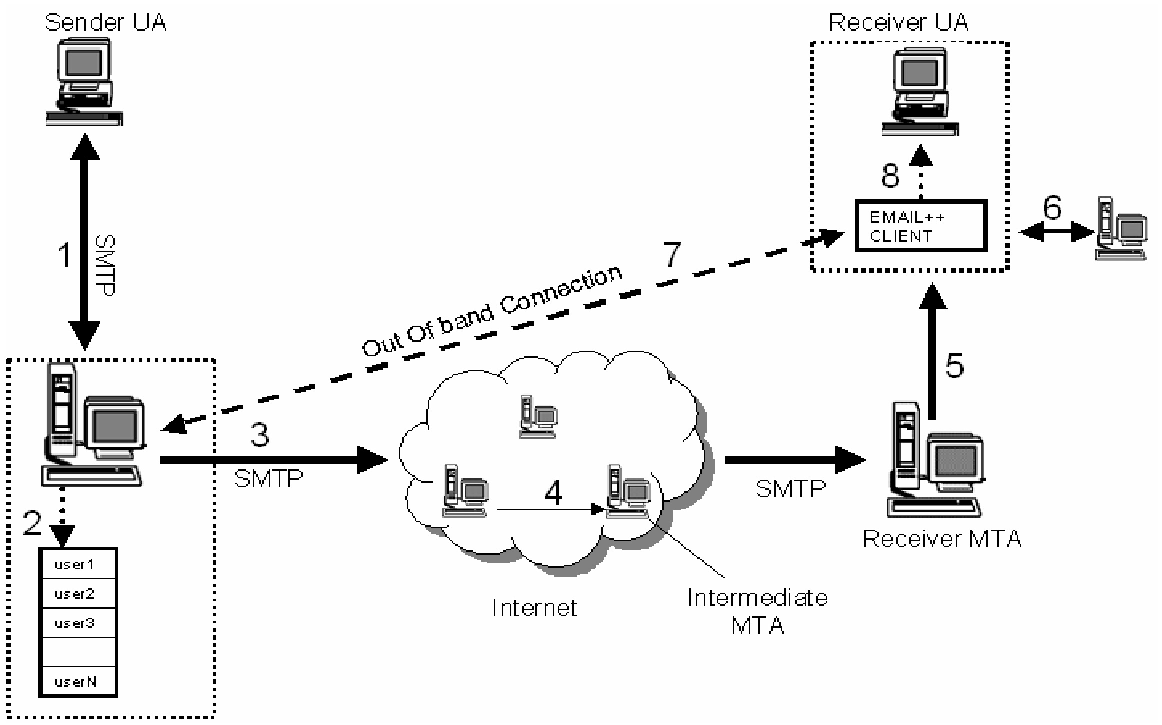


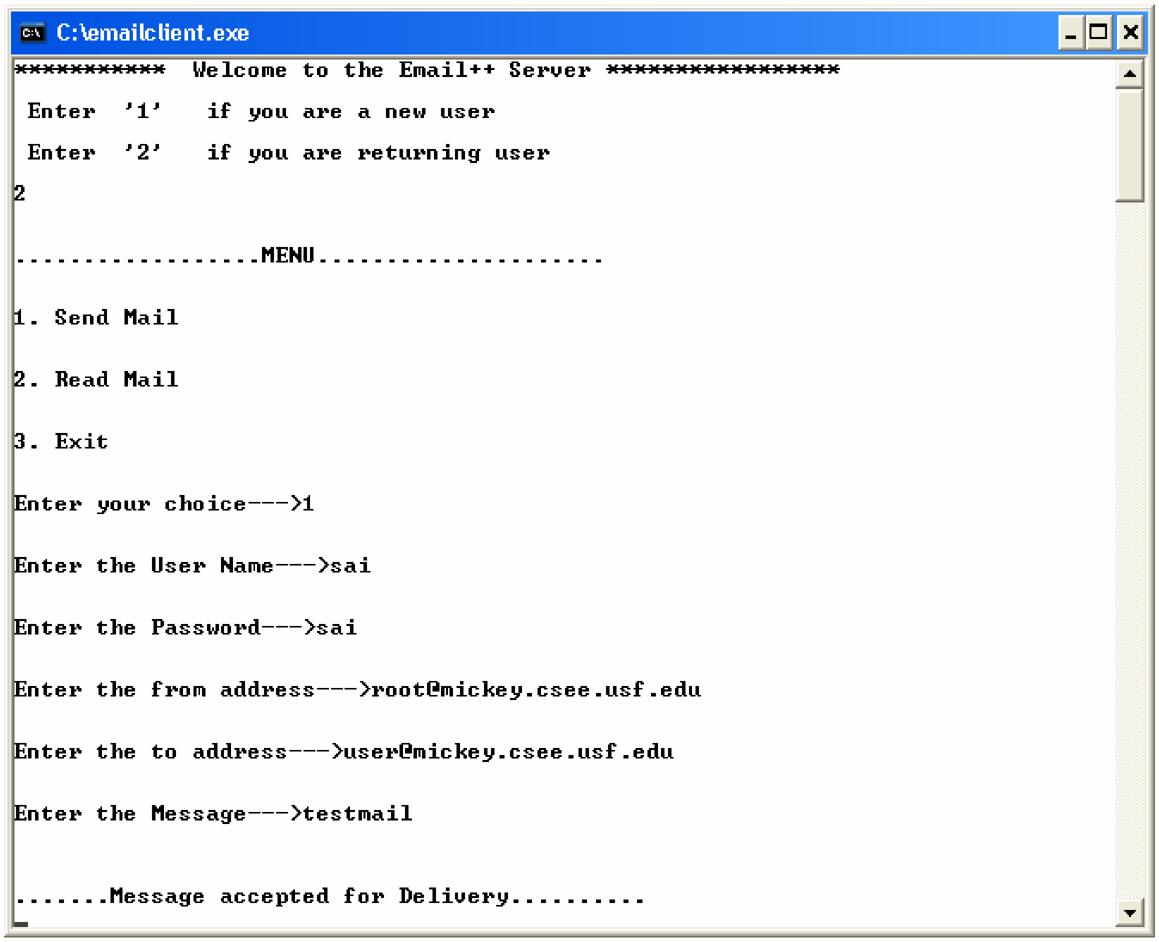
Figure 3.14: Steps Involved in Sending Email in Email++

## 3.7 Validation of Email++

This section describes the validation of Email++. The Email++ design is validated by implementing both Email++ client and server programs in C language. The Email++ server is targeted to the Linux operating system.

A separate mail server for the csee.usf.edu domain is hosted and runs Sendmail software. The client program is developed in C language and is targeted to Windows operating systems. The figures 3.15, 3.16 and 3.17 shows the sequence of execution steps for non-forged email. The “root” is sending email to “user”.

The username and password are “sai”. Figure 3.15 shows the client interface for sending the email. The client is invoked by “root” to send mail to the “user”.

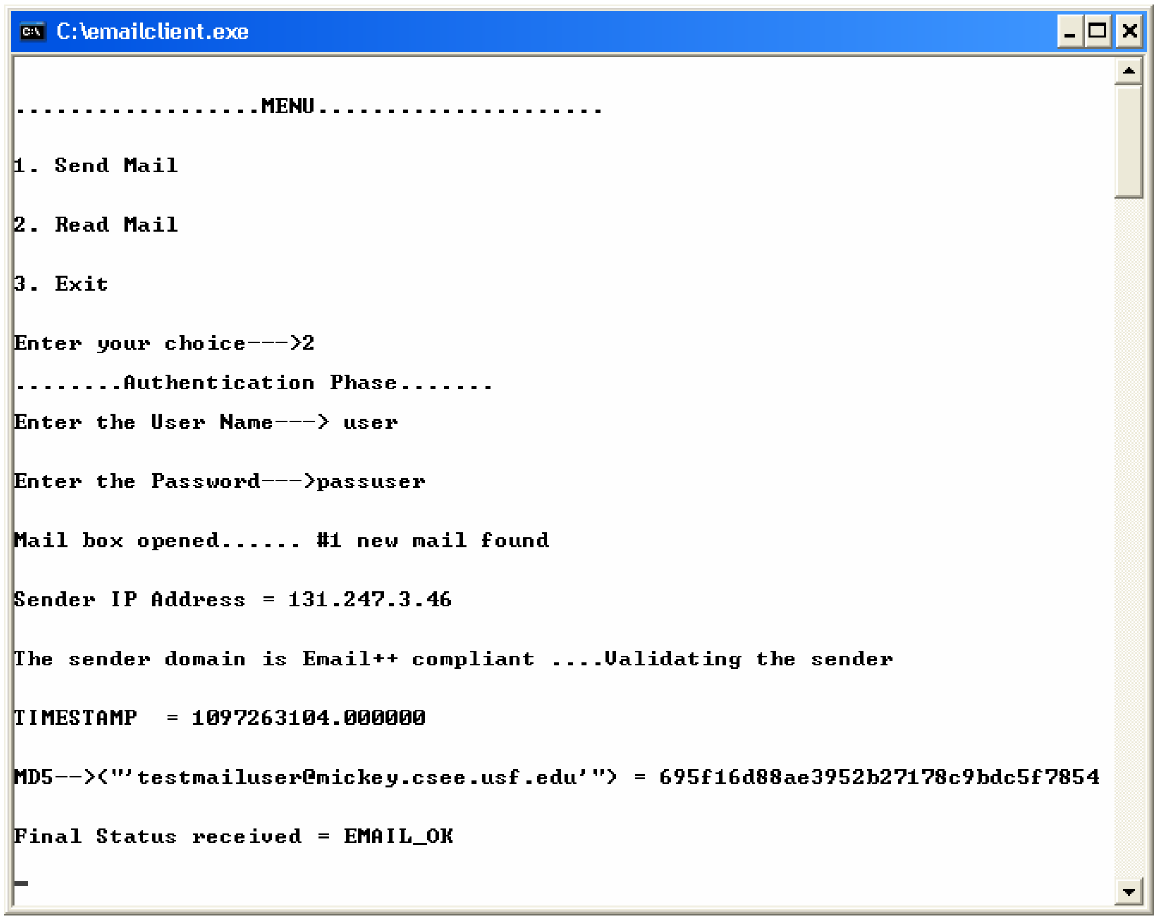


```
************  Welcome to the Email++ Server *****************
 Enter  '1'   if you are a new user
 Enter  '2'   if you are returning user
2

..................MENU......................

1. Send Mail

2. Read Mail

3. Exit

Enter your choice--->1

Enter the User Name--->sai

Enter the Password--->sai

Enter the from address--->root@mickey.csee.usf.edu

Enter the to address--->user@mickey.csee.usf.edu

Enter the Message--->testmail


.......Message accepted for Delivery..........
```

Figure 3.15: The Client Interface Used by “root” for Sending Email

The mail server validates the “root” and accepts the mail for delivery to the “user”. The receiver of the mail invokes the Email++ client for checking the email. The Email++ client gets the mail from the receiver’s mail server and performs the out-of-band validation before placing the mail in user’s inbox. Figure 3.16 shows the output from the

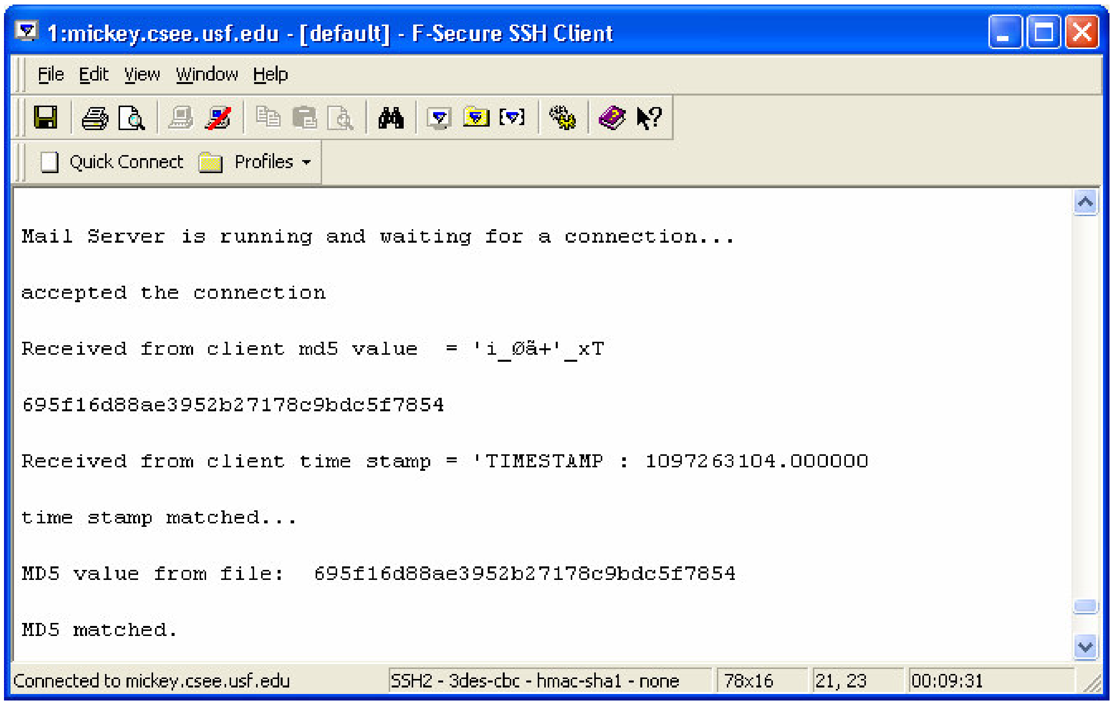
Email++ client. After receiving the EMAIL_OK command from the sender's mail server, the mail is moved to the receiver's inbox and is mail is visible to the receiver.



```
..................MENU......................

1. Send Mail

2. Read Mail

3. Exit

Enter your choice--->2
........Authentication Phase.......
Enter the User Name---> user

Enter the Password--->passuser

Mail box opened...... #1 new mail found

Sender IP Address = 131.247.3.46

The sender domain is Email++ compliant ....Validating the sender

TIMESTAMP  = 1097263104.000000

MD5-->("'testmailuser@mickey.csee.usf.edu'") = 695f16d88ae3952b27178c9bdc5f7854

Final Status received = EMAIL_OK

_
```

Figure 3.16: The Client Interface Used By "user" for Checking Email

Figure 3.17 shows the sequence of events that occurs in Email++ server, which accepts the TCP connection and receives the MD5 hash, timestamp and username, and compares the hash value and the timestamp against the already stored values in the user's account. In this case as the email is not spoofed, an exact match for timestamp and MD5 hash were found in the root's account. If the match is found "EMAIL_OK" command is sent to the reciver.


1:mickey.csee.usf.edu - [default] - F-Secure SSH Client
File Edit View Window Help
Quick Connect
Profiles

```
Mail Server is running and waiting for a connection...

accepted the connection

Received from client md5 value  = 'i_Øã+'_xT

695f16d88ae3952b27178c9bdc5f7854

Received from client time stamp = 'TIMESTAMP : 1097263104.000000

time stamp matched...

MD5 value from file:  695f16d88ae3952b27178c9bdc5f7854

MD5 matched.
```

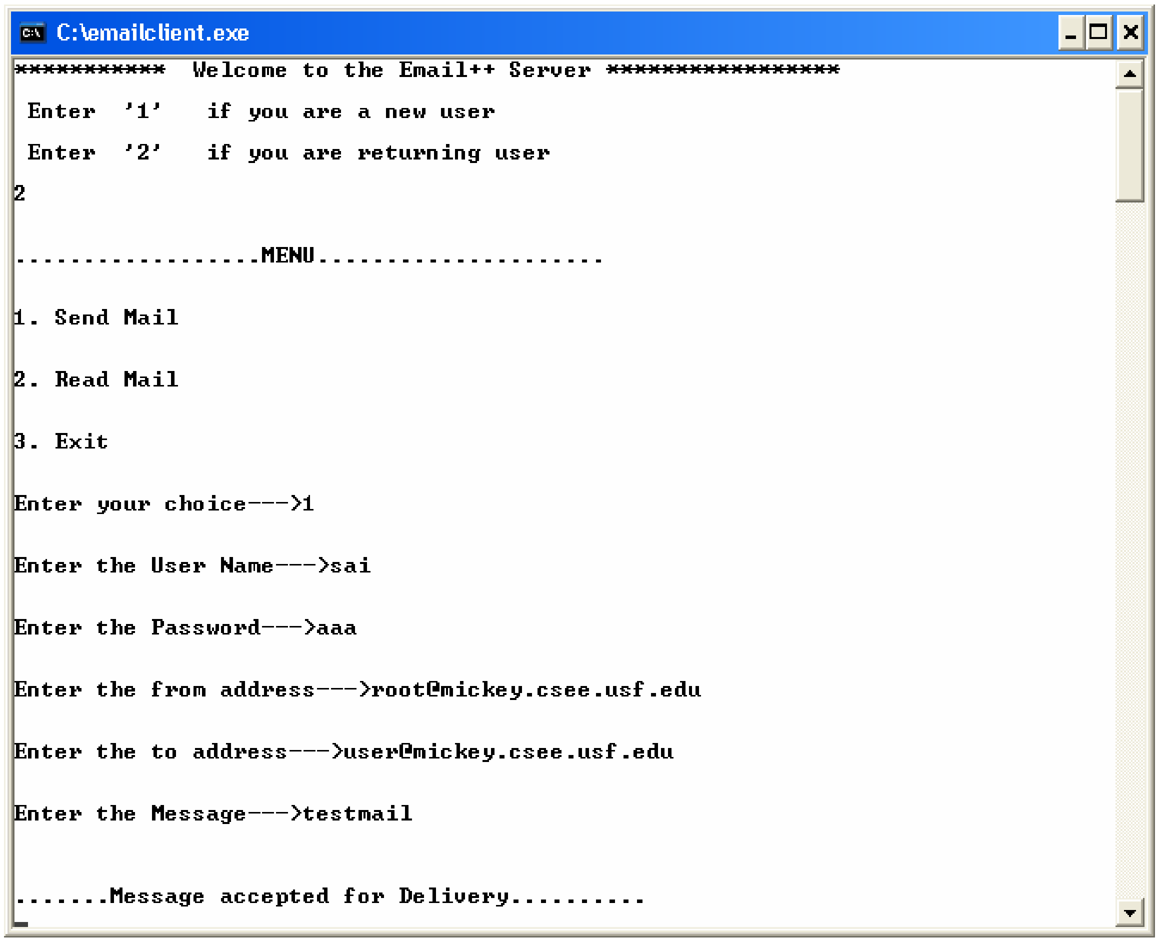
Connected to mickey.csee.usf.edu | SSH2 - 3des-cbc - hmac-sha1 - none | 78x16 | 21, 23 | 00:09:31

Figure 3.17: Output from Email++ Sender of Root's Mailserver

### 3.7.1 Prevention of Spoofing

Consider the scenario where "root" is the valid sender of the domain mickey.csee.usf.edu; "spoofer" from a different domain tries to spoof as "root".

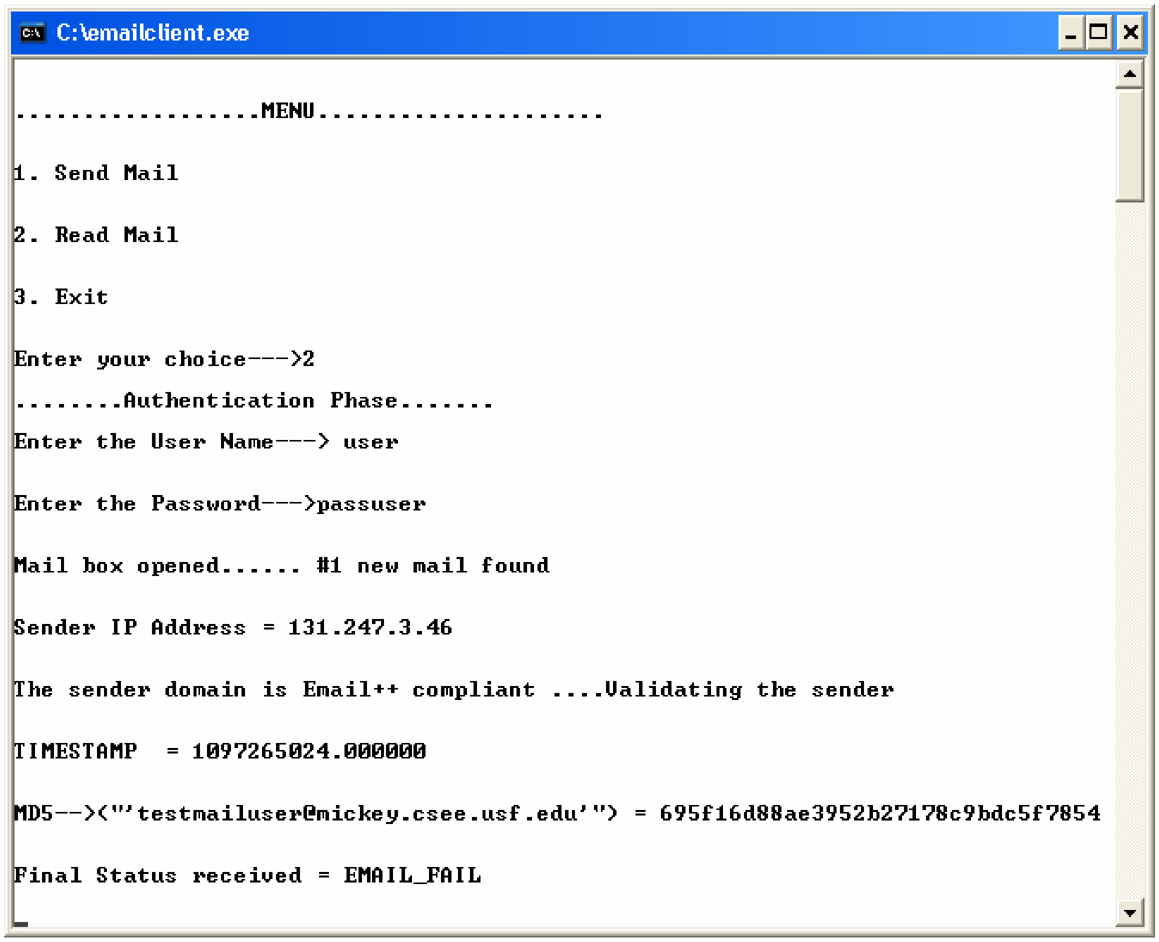
In the client program, which accepts the mail for delivery, the sender name is given as root@mickey.csee.usf.edu. But in sender authentication, a valid username is given and an invalid password is given (as "spoofer" does not know the password of "root"). Figure 3.18 shows the client interface used by "spoofer" for sending email as "root".



```
***********  Welcome to the Email++ Server *****************

 Enter  '1'   if you are a new user

 Enter  '2'   if you are returning user

2

..................MENU.......................

1. Send Mail

2. Read Mail

3. Exit

Enter your choice--->1

Enter the User Name--->sai

Enter the Password--->aaa

Enter the from address--->root@mickey.csee.usf.edu

Enter the to address--->user@mickey.csee.usf.edu

Enter the Message--->testmail


.......Message accepted for Delivery..........
_
```

Figure 3.18: The Client Interface Used by “spoofer” for Sending Email as “root”

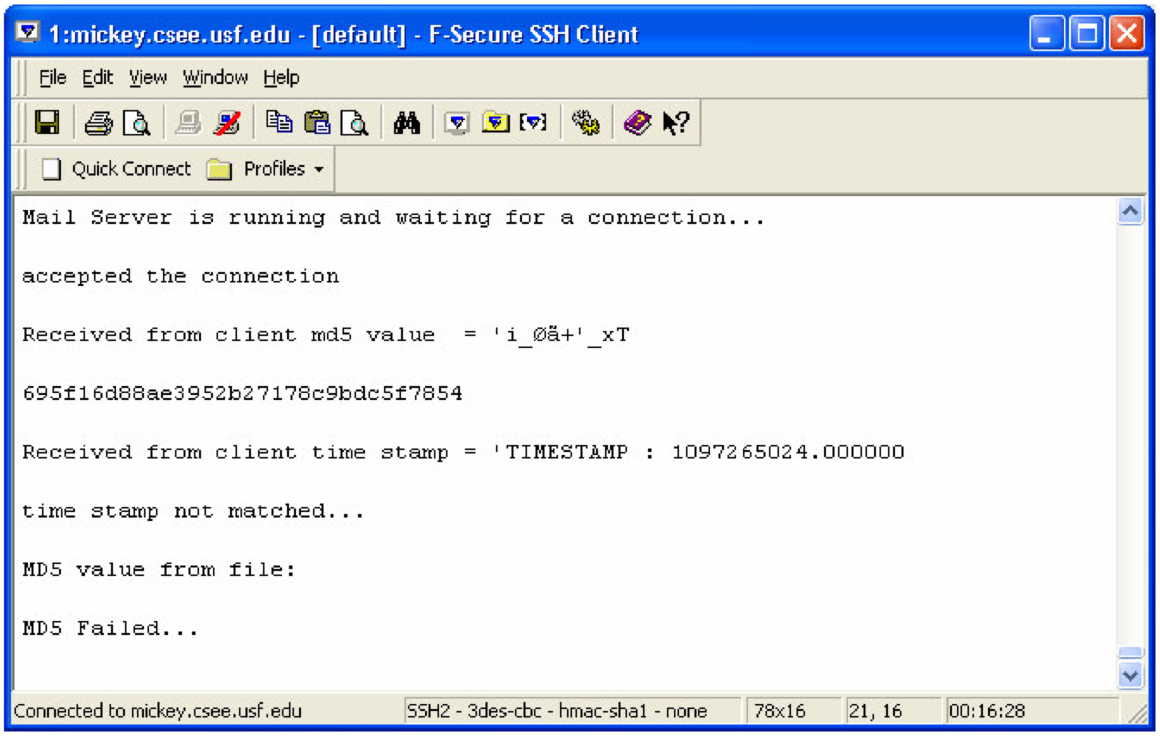
Figure 3.19 shows the screen shot of the interface used by “user” for checking email. In this case the final status received is “EMAIL_FAIL”, which is the expected result for spoofed email.



```
..................MENU......................

1. Send Mail

2. Read Mail

3. Exit

Enter your choice--->2
........Authentication Phase.......
Enter the User Name---> user

Enter the Password--->passuser

Mail box opened...... #1 new mail found

Sender IP Address = 131.247.3.46

The sender domain is Email++ compliant ....Validating the sender

TIMESTAMP  = 1097265024.000000

MD5-->("'testmailuser@mickey.csee.usf.edu'") = 695f16d88ae3952b27178c9bdc5f7854

Final Status received = EMAIL_FAIL

_
```

Figure 3.19: The Client Interface Used by “user” for Checking Email

Figure 3.20 shows output from the Email++ server. When the wrong password is entered, the sender’s information is not stored in the email server, since it is assumed that he or she is not the valid user of the domain. Hence, when the authentication is done, EMAIL_FAIL is received from the sender. Figure 3.20 shows the output from the mail server.


1:mickey.csee.usf.edu - [default] - F-Secure SSH Client
File Edit View Window Help
Quick Connect
Profiles
Connected to mickey.csee.usf.edu
SSH2 - 3des-cbc - hmac-sha1 - none
78x16
21, 16
00:16:28

```
Mail Server is running and waiting for a connection...

accepted the connection

Received from client md5 value  = 'i_Øâ+'_xT

695f16d88ae3952b27178c9bdc5f7854

Received from client time stamp = 'TIMESTAMP : 1097265024.000000

time stamp not matched...

MD5 value from file:

MD5 Failed...
```

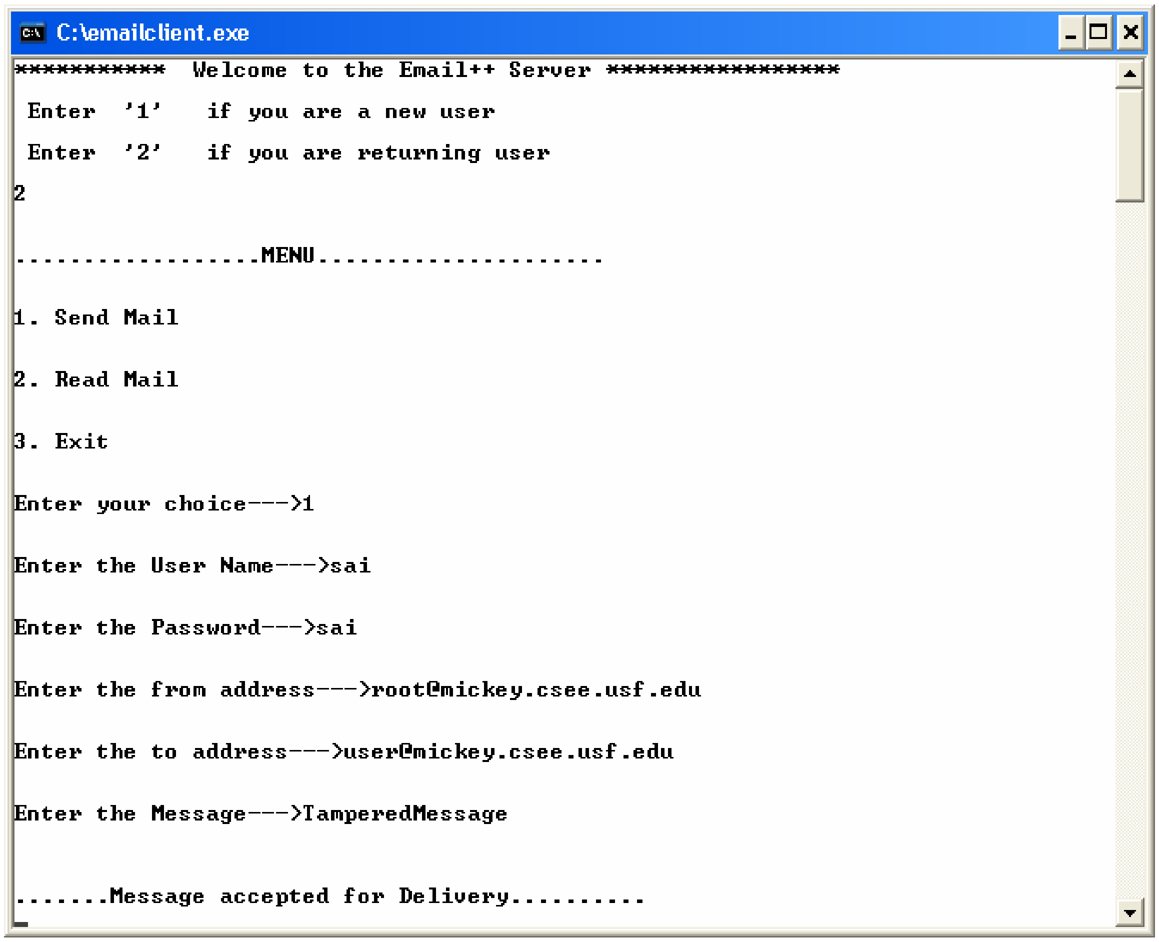
Figure 3.20: Output from the Email++ Server.

It is observed that the spoofer’s mail MD5 and timestamp are not stored in the root’s account. The Email++ server returns EMAIL_FAIL to the receiver.

### 3.7.2 Prevention of Tampering

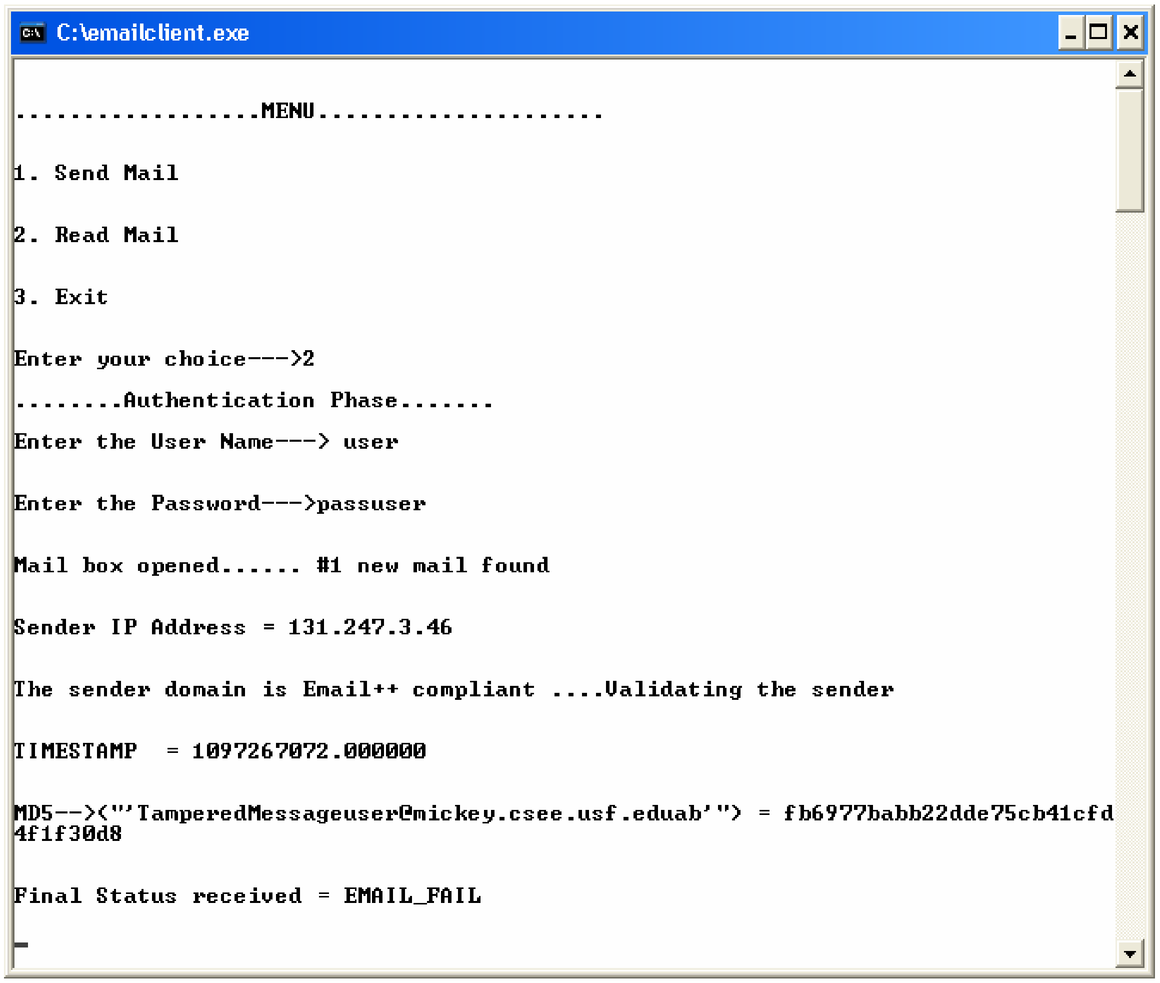
Consider the scenario where “root” is the valid sender of the domain mickey.csee.usf.edu. The contents of the mail sent by “root” are modified in transit and tested to see if the Email++ server identifies it.

After the mail is received, the contents of the mail are modified prior to the generation of the hash value in the Email++ client program; This emulates the contents being modified in transmission. The MD5 hash of the modified contents is sent in the query to the sender’s Email++ server. Figure 3.21 shows the client interface for sending mail. Figure 3.22 shows the output from Email++ client.


C:\emailclient.exe

```
************  Welcome to the Email++ Server ******************

 Enter  '1'   if you are a new user

 Enter  '2'   if you are returning user

2

..................MENU.....................

1. Send Mail

2. Read Mail

3. Exit

Enter your choice--->1

Enter the User Name--->sai

Enter the Password--->sai

Enter the from address--->root@mickey.csee.usf.edu

Enter the to address--->user@mickey.csee.usf.edu

Enter the Message--->TamperedMessage


.......Message accepted for Delivery..........
```

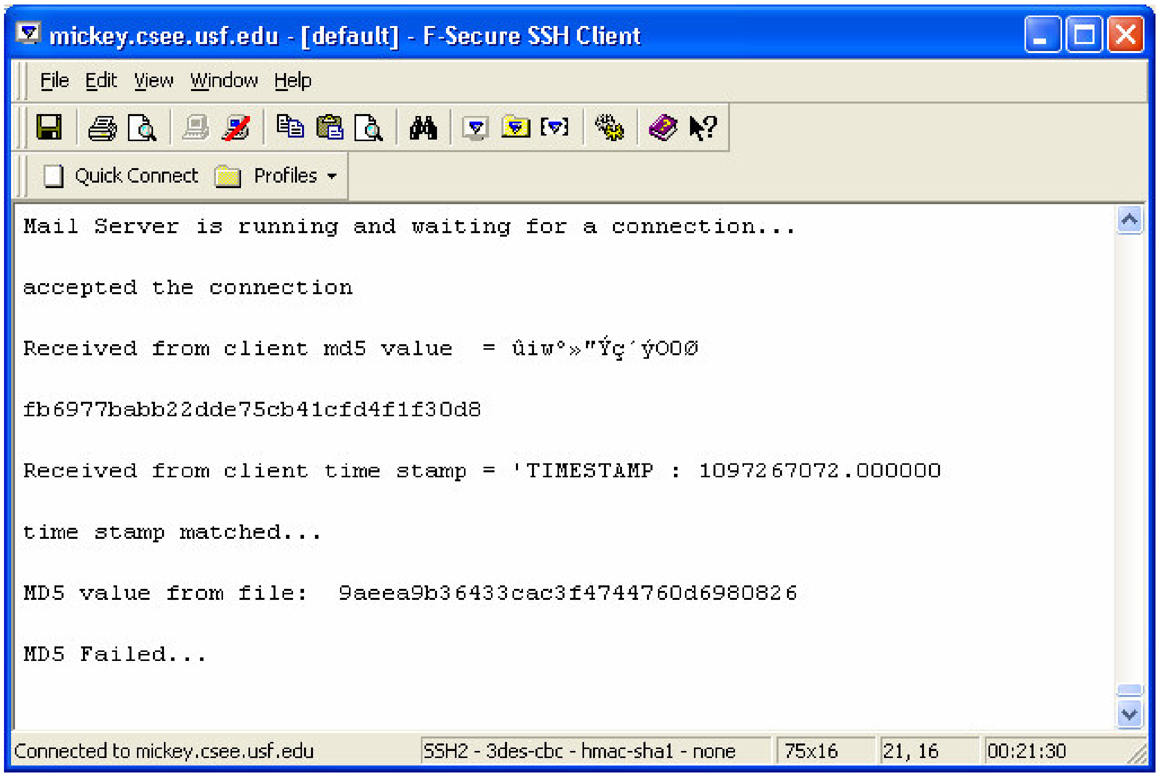
Figure 3.21: The Client Interface Used by “root” for Sending Email



```
..................MENU.....................

1. Send Mail

2. Read Mail

3. Exit

Enter your choice--->2
........Authentication Phase.......
Enter the User Name---> user

Enter the Password--->passuser

Mail box opened...... #1 new mail found

Sender IP Address = 131.247.3.46

The sender domain is Email++ compliant ....Validating the sender

TIMESTAMP  = 1097267072.000000

MD5-->("'TamperedMessageuser@mickey.csee.usf.eduab'") = fb6977babb22dde75cb41cfd
4f1f30d8

Final Status received = EMAIL_FAIL

_
```

Figure 3.22: The Client Interface Used by "user" for Checking Email

The MD5 value of the TamperedMessageuser@mickey.usf.edu should be generated, instead extra characters "ab" are appended at the end and the MD5 hash is generated. This emulates a message being tampered with in transit. Figure 3.23 shows the output from the sender's mail server.


mickey.csee.usf.edu - [default] - F-Secure SSH Client
File Edit View Window Help
Quick Connect
Profiles

```
Mail Server is running and waiting for a connection...

accepted the connection

Received from client md5 value  = ûiw°»"Ýç´ýOOØ

fb6977babb22dde75cb41cfd4f1f30d8

Received from client time stamp = 'TIMESTAMP : 1097267072.000000

time stamp matched...

MD5 value from file:  9aeea9b36433cac3f4744760d6980826

MD5 Failed...
```

Connected to mickey.csee.usf.edu
SSH2 - 3des-cbc - hmac-sha1 - none
75x16
21, 16
00:21:30

Figure 3.23: The Output from Email++ Server

It is observed that the hash value obtained doesn't match with the hash value that is stored, even though the timestamp matches. The Email++ server returns EMAIL_FAIL to the receiver. Hence, the Email++ mechanism effectively identifies the tampered mail.

# CHAPTER 4

# COMPARISON OF EMAIL++ TO EXISTING METHODS

Chapter 3 described the new authentication mechanism for email, called Email++. It described the design, operation, implementation and validation of Email++. This chapter compares Email++ with existing sender authentication techniques. Email++ is compared with SPF (Sender Policy Framework) [21], Microsoft's Sender ID [24], Yahoo Domain Keys [10], Microsoft's ticket server [24], ePrivacy's Trusted Email Open Standard (TEOS) [33], Cashramspam [9], and PGP (Pretty Good privacy)[25].

## 4.1 Classification of Sender Authentication Schemes

Sender authentication protocols considered in this chapter can be primarily categorized (1) based on whether they use DNS or central authority for authenticating the sender and (2) based on the level of the authentication such as domain level or user level. The detailed classification is shown below.

- Authentication strategy
  - DNS-based authentication.
  - Central authority-based authentication
- Granularity of authentication
  - Domain level authentication
  - User level authentication
  - Handling temporary SMTP servers

- Security vulnerabilities
    - ➢ Susceptible to DNS attacks
    - ➢ Extra DNS load
    - ➢ Extra traffic load into network
- Complexity of implementation
    - ➢ Needs a flag day
    - ➢ Needs modifications to SMTP protocol
    - ➢ Needs modifications to DNS
- Miscellaneous features
    - ➢ Validates message content
    - ➢ Allows forwarding and relaying of mail

### 4.1.1 Classification Based on Authentication Strategy

Authentication strategy refers to the strategy followed by the authentication scheme to check the validity of the sender information. The protocols discussed here depend widely on either DNS or a central authority system for checking the correctness of the sender's information.

#### 4.1.1.1 DNS-based Authentication

In DNS-based authentication schemes, the correctness of the information in the SMTP headers is crosschecked with the information in the DNS, which is more reliable than the SMTP headers. SPF, SenderID, Yahoo domain keys, Email++, and TEOS uses DNS-based authentication and need modifications to DNS, whereas Email++ uses the existing DNS system.

#### 4.1.1.2 Central Authority-based Authentication

Some of the proposals for sender authentication such as Microsoft Ticket server, Cashramspam, and PGP use a central certifying authority for validating the sender. The central authority either pre-authorizes the mail by digitally signing the email, or authorizes the sender based on the receiver's request. In Microsoft's ticket server, the sender is authorized based on the user's request. In Cashramspam, each email is authorized before it is actually delivered. In PGP, the centralized key server is used for publishing the public key of the sender.

Both of the strategies described above have pros and cons. DNS-based systems are advantageous because they use the existing system, but they are vulnerable to DNS attacks. Centralized systems are advantageous because they are more secure, but they are harder to implement than the DNS-based systems.

### 4.1.2 Classification Based on Granularity of Authentication

The sender authentication schemes can also be classified based on the granularity of authentication, that is, to what extent the authentication is done and the levels of forgery the scheme can detect.

#### 4.1.2.1 Domain Level Authentication

Domain Level Authentication is the verification of the validity of the domain name of the sender. This helps to identify domain level spoofing, and this is a significant factor to consider, as it helps domains to safeguard their identity from spammers and hackers. This authentication also inhibits bounced SMTP traffic from congesting the domain. There are certain tradeoffs involved in achieving this authentication. The process may add latency in email delivery; some existing proposals require modifications to DNS standards, and the process of authentication for each email consumes extra resources including CPU, memory, and network bandwidth. SPF, Yahoo Domain keys, and Sender ID use domain level authentication.

#### 4.1.2.2 User Level Authentication

In user level-authentication, along with the domain name the actual sender is also validated. In addition to avoiding domain-level forging, the user within the domain is also validated before accepting an email. In addition to the tradeoffs associated with domain level authentication, this mechanism can impose mandatory user validation features, like username and password authentication, before the mail is accepted for delivery. This is optional in current systems. Email++, Cashramspam, PGP, and Ticket server all use user level authentication. Yahoo domain keys can be made more granular, but the current proposal supports up to domain level only.

#### 4.1.2.3 Temporary SMTP Servers

Temporary SMTP servers are the mail servers that are not always available. One of the techniques used by spammers to avoid bounced SMTP traffic is to blast out millions of emails and clog SMTP servers. If the sender's server is temporary and not available, the probability of the sent mail being a spam is high. So, the proposed mechanism should ensure that the sender server is available before delivering the mail to the receiver's mailbox. This authentication process may add latency to the delivery of mail. The process of authentication for each email adds some extra load to the receiver. Only Email++ handles temporary SMTP servers.

The schemes that are more granular are better. Email++ is better than the other proposed techniques, as its granularity is up to user level. In addition, it handles temporary SMTP servers.

### 4.1.3 Classification Based on Security Vulnerability

Another way to classify the techniques is based on their security vulnerabilities.

#### 4.1.3.1 Susceptible to DNS Failures

Many anti-spoofing proposals like SPF, sender ID, Yahoo Domain Keys, and Email++ completely rely on DNS for sender authentication. DNS itself is susceptible to hacking, resulting in Man-In-The-Middle attacks, DNS spoofing, and DNS Hijacking [6]. The effect of all these DNS vulnerabilities on the proposed system should be carefully explored.

#### 4.1.3.2 Adds Extra DNS Load

DNS load is an important constraint to be considered. The extra load that the sender authentication techniques add on the DNS server should be predictable. The new system should not add a lot of DNS load. If the load added is too heavy for the DNS server to handle, the system may crash and affect the other systems that use DNS for their proper functioning. SPF, Sender ID, Yahoo Domain Keys, Email++, and TEOS all add extra load on DNS.

#### 4.1.3.3 Adds Extra Network Traffic

Schemes like Ticket Server, PGP, Email++, and Yahoo Domain Keys adds extra traffic to the network. The extra traffic should be predictable, comparatively low, and it should not degrade the performance of the existing system by introducing delays in transmission. Email++ adds more traffic into the network because of the out-of-band TCP connections from receivers to senders.

### 4.1.4 Classification Based on Complexity of Implementation

One of the important requirements for the sender authentication scheme is its complexity of implementation

#### 4.1.4.1 Need a Flag Day

Flag Day can be described as a day on which there should be a complete adaptation of the new proposed mechanism for the existing system to work properly from the day after Flag Day. It can be described as a day on which all the existing email systems should change to the new proposed system. Furthermore the email system is down that day. This is the most important parameter to consider, because such drastic adaptation may not be feasible in a real world implementation. A new system should co-exist with the already prevalent email infrastructure. The mechanism should work even if the majority of the email systems are not prepared to adopt the new system. The new authentication mechanism should not be expected to bring about an abrupt change in the overall existing system, but should rather bring about gradual change. All of the schemes reviewed are designed to be gradually adoptable.

#### 4.1.4.2 Modifications to SMTP Protocol

The new proposed mechanism should not modify the standard SMTP protocol. As SMTP is widely deployed, a minor modification may result in an effect on the majority of existing systems. If the modification to SMTP is mandatory, in order to make it more secure, then the practical implementation of the scheme should be evaluated more thoroughly and carefully. The tradeoffs in implementing this include changes in the current SMTP standards, which affect current systems. Email++, Yahoo Domain Keys, and Ticket Server need modifications to SMTP protocol. Sender ID needs modifications to the SMTP server in order to support forwarding.

#### 4.1.4.3 DNS Modifications

Some of the sender authentication mechanisms described in Chapter 2 use DNS records for the validation of the sender identity. Some mechanisms propose modifying the DNS protocol by adding extra records or data to achieve this. This modification may also impact other systems that use DNS. The proposed mechanism should not modify the DNS or, if the modification is mandatory, the impact of the modification should be inspected carefully. Implementing such a mechanism may result in changes in current standards. SPF, Sender ID, Yahoo Domain Keys need modifications to DNS. Email++, even though it uses DNS-based authentication, needs no modification to DNS.

The authentication mechanism should be simple and easily implementable. It should not require a flag day, but should favor gradual adaptation. It should propose no or minimal changes to the SMTP and DNS protocols, and the modifications should be backward compatible.

### 4.1.5 Miscellaneous Features

#### 4.1.5.1 Validates Message Content

Some of the mechanisms like SPF and Sender ID validate the sender, but not the contents sent. Content validation is important, as the data may be tampered with in the transmission, either intentionally or accidentally [31]. Email++, PGP, Yahoo Domain Keys, and Ticket Server validate the message content.

#### 4.1.5.2 Allows Relaying

Stopping open relaying is one of the common security measures taken by domain administrators to protect their domain. But when SMTP was designed, relaying was included on purpose. There are several advantages of relaying as described in [42]. For this reason, a new mechanism that favors relaying is better than the one that hinders it. Another research study at Internet Mail Consortium shows that the amount of spam is not related to the number of open relays [12]. Sender ID and Email++ allow for the relaying of mail.

## 4.2 Comparison to Existing Sender Authentication Techniques

This section tabulates the summary of the comparison of SPF, Sender ID, Yahoo Domain Keys, Microsoft Ticket Server, TEOS, Cashramspam, PGP, and Email++. Tables 1 and 2 show the comparison charts.

Table 4.1 Summary of Existing Anti-spoof Techniques

| Name | SPF | Sender ID | TEOS | Yahoo Domain keys | Email++ |
|---|---|---|---|---|---|
| Detect Domain Level spoofing | Yes | Yes | Yes | Yes | Yes |
| Detect user level spoofing | No | No | Yes | No | Yes |
| Doesn't Need a Flag Day | No | Yes | Yes | Yes | Yes |
| Doesn't need modification to SMTP Protocol | Yes | No | No | No | No** |
| Doesn't need DNS modifications | No | No | No | No | Yes |
| Not susceptible to DNS failure | No | No | No | No | No |
| Validates message content | No | No | Yes | Yes | Yes |
| Doesn't need central certifying authority | Yes | Yes | Yes | Yes | Yes |
| Handles temporary SMTP servers | No | No | No | No | Yes |
| Allows open relaying | No | Yes | No | No | Yes |

** In Email++ the SMTP protocol needs no modification as Email++ doesn't add any extra headers to the existing protocol. The Email++ server module can run as a separate service, over an existing mail server.

Table 4.2 Summary of Existing Anti Spam Techniques

| Name | Microsoft Ticket Server | CashRamSpam | PGP (Pretty Good Privacy) |
|---|---|---|---|
| Does user level authentication | Yes | Yes | Yes |
| Doesn't need Flag Day | No | Yes | Yes |
| Doesn't need modification to SMTP Protocol | Yes | Yes | Yes |
| Doesn't need DNS modifications | Yes | Yes | Yes |
| Not susceptible to DNS failure | Yes | Yes | Yes |
| Validates message content | Yes | No | Yes |
| Doesn't need Central certifying authority | No | No | Yes** |
| Email sent in clear text | Yes | Yes | Yes |
| Public Key Cryptography | Yes | No | Yes |

** PGP doesn't require central certifying authority, if the public key is not published in the PGP key server.

## 4.3 Summary of Comparison

- Email++ is the better than other DNS-based authentication schemes as it utilizes DNS, but requests no modification to the existing DNS.
- Email++, PGP, Cashramspam, and Ticket Server that provide user-level authentication are more advantageous than SPF, Yahoo Domain Keys, and Sender ID.
- SPF does not propose any modifications to SMTP; Email++ proposes no modifications to DNS. Sender ID and Yahoo domain keys, proposes changes to both SMTP and DNS.
- All the DNS-based schemes are vulnerable to DNS attacks and introduce DNS load. Email++, Ticket Server and PGP introduce more traffic in the network than other schemes like SPF, Sender ID, and Yahoo Domain Keys, and TEOS.
- Email++, Yahoo Domain Keys and PGP provide extra features, such as validation of the message content. Email++ and sender ID support mail relaying.

# CHAPTER 5

# PERFORMANCE EVALUATION OF EMAIL++

## 5.1 Objectives

This chapter describes the performance evaluation of Email++. It describes the metrics considered for evaluating the performance and the significance of each metric. It also describes the experimental setup, design of each experiment and observations from each experiment. It concludes with the summary of the performance evaluation.

## 5.2 Performance Measures of Interest

This section summarizes the most important criteria for the performance evaluation of Email++.

### 5.2.1 CPU Demand

Email++ implementation in the sender SMTP server requires extra CPU resources for MD5 calculations before sending mail and for satisfying out-of-band validation requests from mail recipients. This is an important metric, which should be carefully evaluated. If Email++ is CPU intensive, the net result may impact the performance of the mail server. The extra overload on the server can lead to low throughput of the mail server, and unpredictable latency in email delivery.

### 5.2.2 Memory Demand

The Email++ server at the sender side and the client at the recipient are not very memory intensive. Extra disk space is required on the server side for maintaining the

MD5 lists of mail for each user. The Email++ server and client modules require some disk space.

### 5.2.3 Bandwidth Demand

Email++ adds extra traffic to the existing system by its out-of-band TCP connections between client and server for validating the sender. For obtaining the IP address of sender's email server an additional DNS query increases the traffic further. However the amount of extra traffic in out of band connections is predictable, as the size of the hash value and timestamp are always constant for any size email. At the receiver, the DNS query from receiver to its name server adds traffic equal to one normal DNS query.

### 5.2.4 Increased Email Latency

The hashing of email, before delivery at the sender side and the validating of email before placing it in the actual inbox of the receiver at the receiver side, adds some latency to the actual delivery time. This latency is not exactly predictable, as it depends on the server load, distance between the sender and receiver, bandwidth available, and network conditions.

### 5.2.5 DNS Load

The Email++ mechanism adds an extra DNS request for each email that is received for that domain. This may increase the load on the DNS server.

## 5.3 Experimental Setup

To test the specification of Email++, the Email++ client and Email++ server modules were developed in C language. A new mail server called Email++ mail server was setup in the USF Computer Science and Engineering domain (csee domain). The server module of Email++ was installed in the Email++ mail server. The Email++ client module is installed and executed from the sender user agent, which generated the email traffic. The mail server is configured in such a way that if the mail is for the local user of the domain, it is kept in the user's account otherwise it is forwarded to the common gateway of the Computer Science and Engineering Domain network.

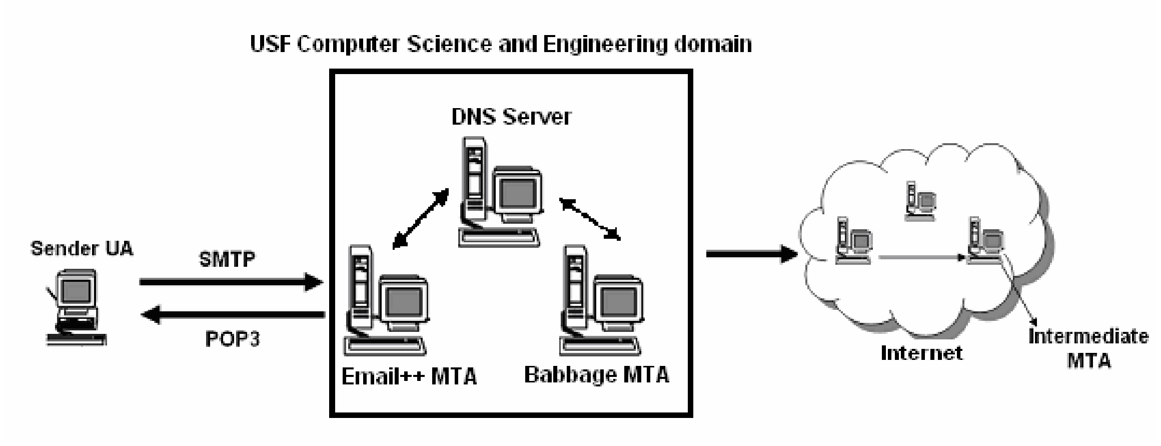


Figure 5.1: Experimental Setup

## 5.4 Design of Experiments

This section describes the design of experiments for evaluating the performance of an Email++ client and server.

This section describes the experiments conducted to evaluate the performance of the sender's mail server. The performance metrics evaluated are CPU demand, memory demand, and network bandwidth demand.

- CPU utilization experiment: In this experiment, the sender sends 500 mails of 450 bytes each to the mail server and monitors the percentage of CPU utilized and the time taken for delivering the mails. The Linux tools SAR, vmstat and top were used to get the monitoring information. The percentage of CPU utilized is measured with the Linux perfmon tool. This experiment was run for both regular SMTP and for Email++ enabled SMTP. The control variable is the number of mails and the response variable is the percentage utilization of CPU.

- Memory utilization experiment: This experiment evaluates the extra memory resources required by the Email++, when compared to regular SMTP. In this experiment the sender sends 500 mails of 450 bytes each and monitors the percentage of main memory utilized by using the Linux monitoring tool top. The control variable is the number of mails and the response variable is the percentage utilization of main memory.

- Bandwidth utilization experiment*:* This experiment measures the percentage of extra bandwidth utilization of Email++. The client receives 500 mails of varying size 0.5 KB, 1.5 KB, 2.5 KB, 3.5 KB, 4.5 KB, 5.5 KB, 6.5KB to measure how the percentage of extra traffic generated by Email++ varies with the size of the mail.

## 5.5 Experimental Results

This section describes the results from the experiments described in the previous section. This section also describes additional experiments conducted to further understand or support the results from the defined experiments.

Figure 5.2 shows the result from CPU utilization experiment. The results show that Email++ enabled mail server takes more time for 500 mails than the normal SMTP server.

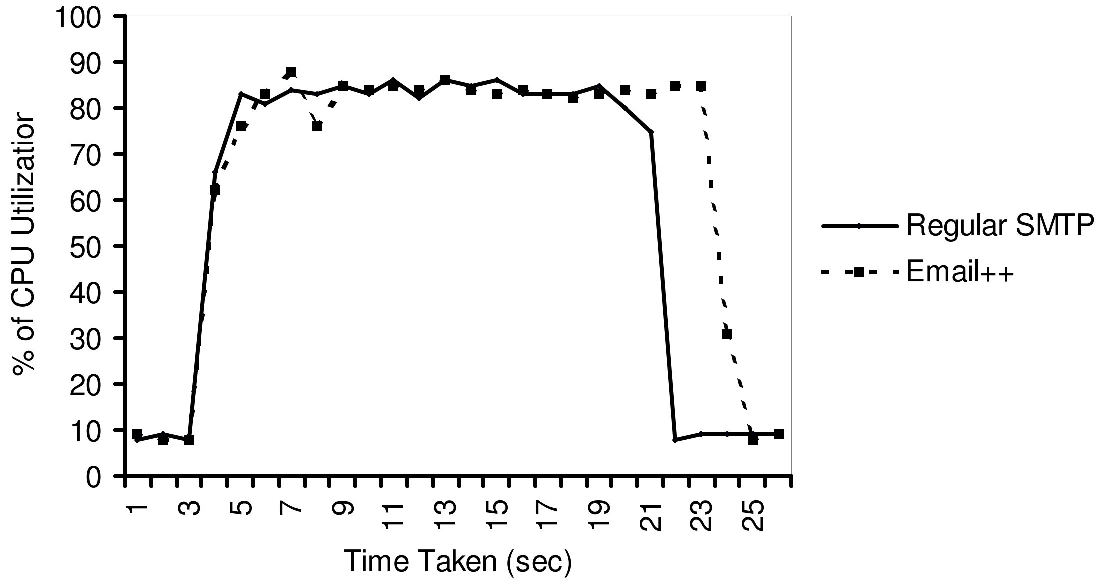


Figure 5.2: Comparison of CPU Utilization of Regular SMTP and Email++

The percentage of CPU utilized is almost the same for both regular SMTP and Email++ enabled SMTP**.** The Email++ enabled SMTP server consumes CPU resources for longer time than the normal SMTP. The percentage increase in time for 500 mails is about 11%. Hence per mail the percentage increase in time is 11 %. Email++ takes more time to execute and hence uses CPU resources for longer time.

Figure 5.3 shows that the percentage of memory utilized by Email++ enabled mail server is slightly larger than that of the regular SMTP. The maximum difference between the percentages of memory utilization is nearly 3%.

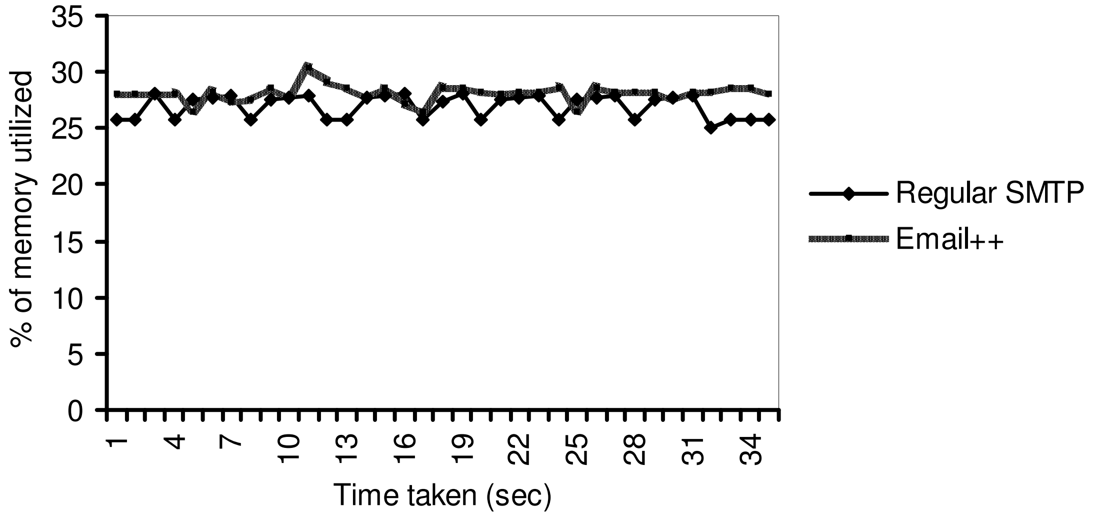


Figure 5.3: Comparison of Memory Utilization of Regular SMTP and Email++

Figure 5.4 shows the percentage of extra traffic generated decreases as the size of the mail increases. So as the mail size increases the relative amount of extra traffic generated by Email++ decreases. In a worst case when the email size is less than 1KB the percentage of extra traffic is about 40%. In an average case when the email size is 3.5KB the percentage of extra traffic is around 15%.

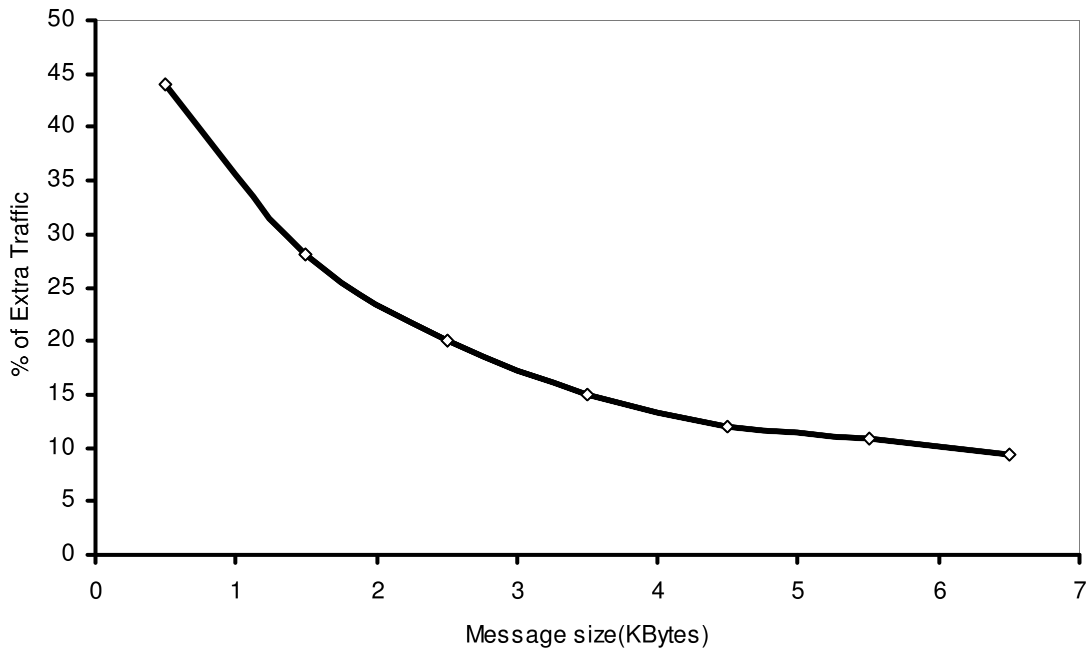


Figure 5.4: Percentage of Extra Traffic as Mail Size Increases

## 5.6 Discussion of Results

The CPU utilization experiment shows that the Email++ enabled mail server takes 11% more time for sending 500 emails of 450KB each. The extra time incurred was from the MD5 calculations and the out of band connections. CPU utilization experiment #1 and CPU utilization experiment #2 were conducted to evaluate the time taken for MD5 calculations and out of band connections respectively.

- CPU utilization experiment #1: In this experiment 500 mails of 450 bytes each were sent to the mail server and the percent of CPU utilized and the time taken for delivering the mails is monitored by turning off the out-of-band validations. So the extra time incurred in this case should be from the MD5 calculations alone.

Figure 5.5 shows the results of CPU utilization experiment #1. It suggests that the amount of time and CPU utilized by MD5 calculations is negligible. The amount of time incurred from MD5 computations for 500 emails of 450 bytes each is less than 1 sec.

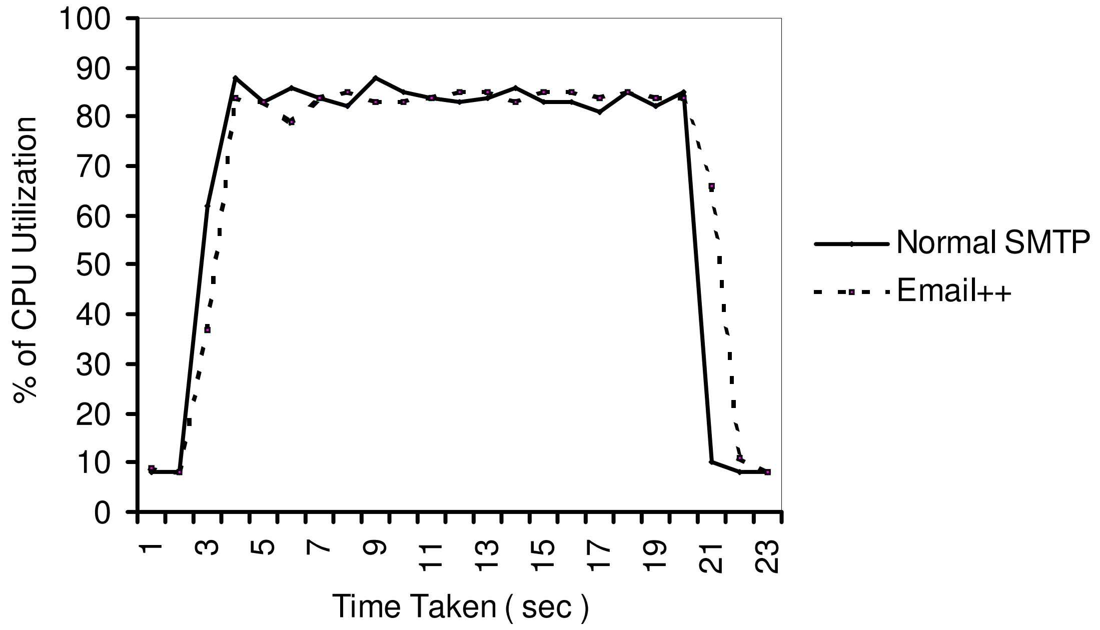


Figure 5.5: Percentage of CPU Utilization for MD5 Calculations Alone

CPU utilization experiment #2: This experiment measures the CPU utilization and time taken for the out of band connections alone, by turning off MD5 calculations. In this experiment 500 mails of 450 bytes each were sent to the mail server and the percent of CPU utilized and the time taken for delivering the mails is monitored by turning on the out-of-band validations. So the extra time incurred in this case should be from the out-of-band calculations alone.

Figure 5.6 shows the results of CPU utilization experiment #2. The result suggests that major portion of extra time is spent in out of band connections and validations than the MD5 calculations.

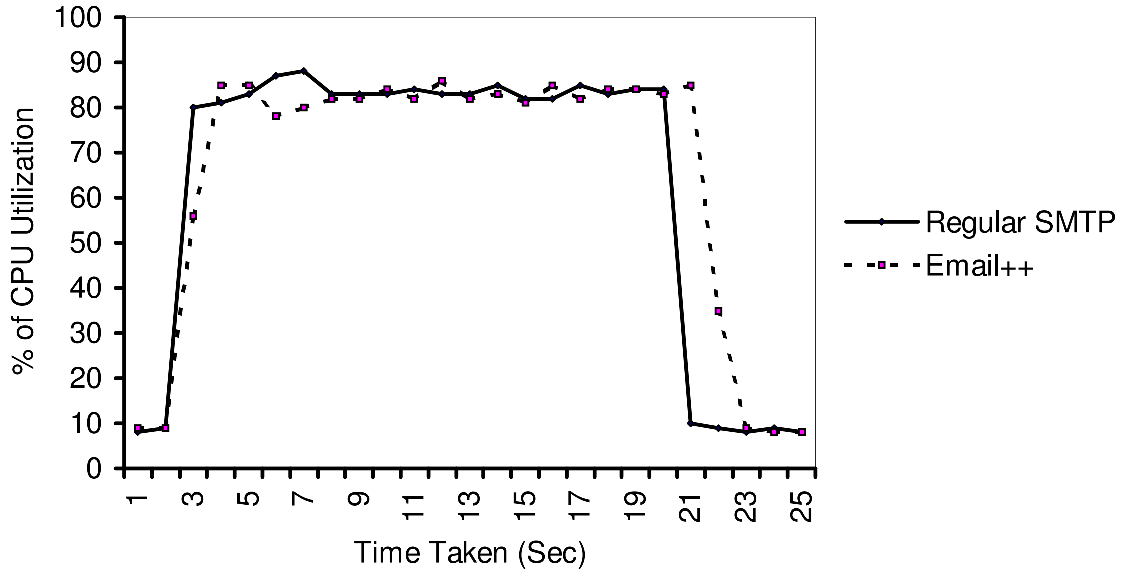


Figure 5.6: Percentage of CPU Utilization for Out-of-band Connections Alone

CPU utilization experiment #1 shows that the time taken for MD5 calculations of emails of 450 bytes is negligible. To check how the time taken for MD5 calculations varies for larger emails, the MD5 experiment was conducted.

MD5 experiment: This experiment computes the amount of time taken for 500 MD5 computations by increasing the email size to 1KB, 2KB, 4KB, 8KB, 16KB, and 32KB

Figure 5.7 shows the results of MD5 experiment. It shows that as the size of the message increases from 1KB, 2KB, 4KB, 8KB, 16KB, and 32KB the amount of time required for MD5 computation is approximately doubled. For 500 mails of 32Kbytes the time taken is nearly 1 second. The amount of time taken for 1 message of 1KB is 0.00006 seconds (60 microseconds).

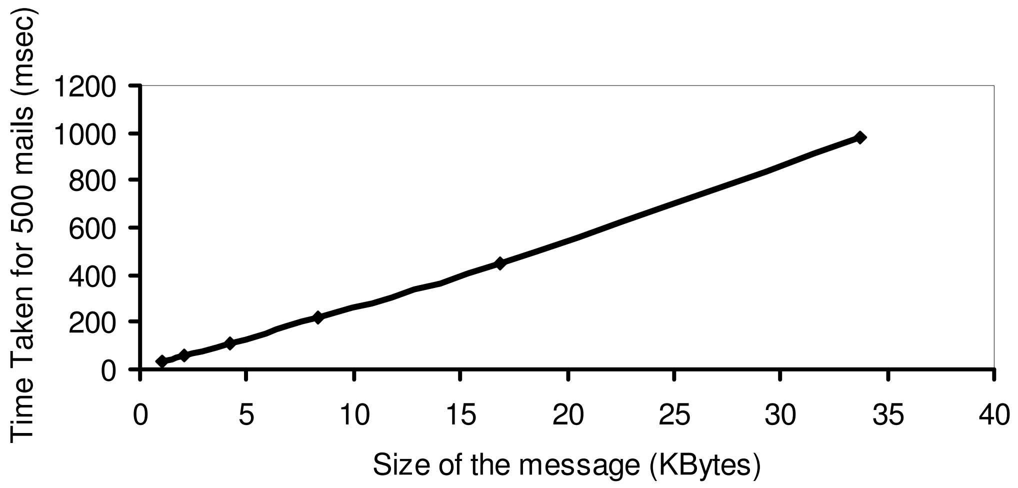


Figure 5.7: Time Taken for MD5 Calculations as Message Size Increases

The results from memory utilization experiment shows that the percentage of memory utilized by Email++ server is nearly 3% more than the regular SMTP server. From this we can conclude that Email++ is not very memory intensive. The amount of disk space required for installing the Email++ server is 35KB. The amount of disk space required for storing one MD5 record is 35 bytes.

The results from the bandwidth utilization experiment shows that as the size of the email increases the percentage of the extra traffic decreases. This behavior is obvious as Email++ adds a constant amount of traffic for any mail size. An experiment was conducted to check how the throughput of the email server varies with increasing message size.

Throughput experiment: This experiment compares the throughput of regular SMTP and Email++ enabled SMTP by increasing mail size. This experiment was conducted by varying the mail size as 0.5KB, 1.5KB, 2.5KB, 4.5KB, and 6.5KB. The throughput is calculated as number of mails per second.

Figure 5.8 show that there is a constant decrease in the throughput of regular SMTP and Email++ enabled SMTP. The results can be justified as follows. The extra overhead to the server comes from MD5 computations and out-of-band validations. As the mail size increases the overhead incurred from MD5 validations is negligible as shown in MD5 experiment. For out-of-band validations the amount of data sent is same for any sized mail hence it takes same amount of time and resources. As the size of the mail increases the percentage decrease in throughput is nearly constant.

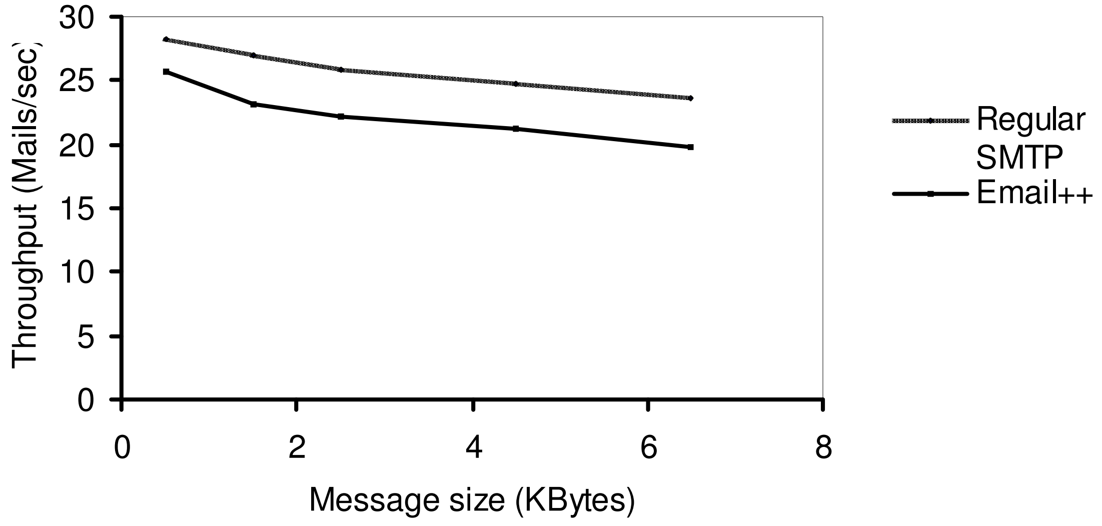


Figure 5.8: Comparison of Throughput of Regular SMTP and Email ++

No extra load is added on DNS at the sender side. But the number of connections that server should hold per email increases from two (one for SMTP and one for DNS) to three (an extra connection for out-of-band connection.)

The extra CPU is required for calculating the MD5 calculations and the out-of-band connections. Each recipient should calculate one MD5 hash for each mail. From the results of MD5 experiment, we can conclude the amount of time for one MD5 is negligible.

Extra disk space is required for installing the MD5 client, which takes 92Kbytes The Email++ client queries the DNS to get the IP address of the sender's mail server. Extra traffic generated by this query is equal to the traffic of one DNS query.

## 5.7 Summary of Performance Evaluation

The performance evaluation of Email++ shows that an Email++ mailserver takes 11% more time than the regular SMTP server to process the same amount of mail. Most of the extra time is spent in out-of-band validations and the time incurred from MD5 calculations is insignificant. The experiment results shows that Email++ is not very memory intensive. The maximum memory utilization percentage difference between regular SMTP server and Email++ mail server is shown to be approximately 3%. Email++ is bandwidth intensive for small mails less than 1KB. It is shown that for smaller mails the percentage of extra traffic is above 40%. But as the mail size increases in the average case when the mail size is 3.5 KB the percentage of extra traffic is nearly 15%. The experimental results prove that Email++ adds minimal burden on the existing system without degrading the performance of that system.

# CHAPTER 6

# SUMMARY AND FUTURE WORK

## 6.1 Summary

Email has become a vital mode of communication and SMTP is the standard protocol for its implementation. SMTP does not perform sender authentication and is thus vulnerable to sender identification spoofing. This vulnerability is exploited by spammers. In the current SMTP standard the sender is not accountable to the mail he/she sends. Hence techniques to make SMTP more secure are required. This thesis describes a new authentication mechanism called Email++ for validating the sender of an email.

The new authentication mechanism detects spoofing of the sender's address by verifying the address using an out-of-band connection to the sender's mail server. This mechanism also ensures that the contents of the mail are not tampered with in transit. The authentication mechanism is implemented and validated. The impact of the mechanism on the performance of the existing system is evaluated. The results show that the new mechanism imposes 11% extra CPU overhead. The Memory demand is negligible. The amount of extra traffic imposed is 15%. The extra load on DNS is one extra connection per email sent.

## 6.2 Enhancing Email++

One important issue is the consideration of multiple email servers for a single domain. The current design of Email++ considers a single mail server per domain. This should be extended to multiple email servers per domain environment.

A possible enhancement to Email++ would be to implement a sender payment scheme over the existing system. The out-of-band communication channel can be used by the receiver to issue a puzzle to the sender. The sender can send the results back to the receiver using the same connection. The receiver can check the result and if its valid, the mail is forwarded from temporary mailbox to the actual mailbox of the receiver. Having the out-of-band connection mechanism in place makes it easy to implement the sender payment schemes. In addition to that, once the sender gets through the out-of-band authentication phase, its proved that both the sender and receiver are real not faked. This helps to implement the sender payment scheme more securely.

## 6.3 Adaptation of Email++ to Other Systems

The basic idea in Email++ can be adapted to other systems that are susceptible to spoofing. Source address spoofing is possible in cases where the sender does not require any information back from the receiver. The idea in Email++ is to make a connection back to the sender with the address the sender provided to make sure if it is valid. If the sender provides a forged address, the forged address will be contacted, and hence the source of the forgery can be identified.

The DNS system is also susceptible to spoofing. DNS servers constantly update their database with the latest information sent from other DNS servers. Some DNS servers accept a response from any server and update its cache with the relevant information. It does not really authenticate whether the response came from the true domain. This also results in cache poisoning. The idea of an out-of-band connection can be applied in this context where after the update is received by the DNS server, it contacts the source to make sure that the data was in fact sent from the domain.

SMS messages can be classified as cell phone originated SMS, that is sending SMS from cell phone to other cell phone, or web based SMS, sending SMS through web interface to a cell phone. Both cell phone originated SMS and web based SMS are susceptible to spoofing, but web based spoofing is easy to do. But web based SMS are more susceptible to spoofing. The idea of an out-of-band connection for sender authentication can be used to avoid SMS web based spoofing, where the web site domain maintains a validation service, and the receiver makes an out-of-band connection to the sender domain to make sure the message was in fact sent from the valid user of the domain.

# REFERENCES

[1] M. Abadi, A. Birrell, M. Burrows, F. Dabek and T. Wobber, “Bankable Postage for Network Services,” *Proceedings of the 8th Asian Computing Science Conference*, Vol. 2896, pp. 72-90, 2003.

[2] M. Abadi, M. Burrows, M. Manasse and T. Wobber , "Moderately Hard , Memory-bound Functions," *Proceedings of 10th Annual Network and Distributed System Security Symposium (NDSS)* , pp. 25-39,2003.

[3] Adam Back "Hashcash - A Denial of Service Counter-Measure,”
URL : http://www.cyberspace.org/hashcash.

[4] ARPANET Network.
URL: http://www.darpa.mil/.

[5] D. Atkins, W. Stallings and P. Zimmermann, “PGP Message Exchange Formats,” *RFC 1991*, 1996.

[6] “Attacking the DNS Protocol-Security Paper,” Security Associates Institute, 2003.
URL: http://www.sainstitute.org.

[7] D. J. Bernstein, “Internet Mail 2000,”
URL: http://cr.yp.to/im2000.html.

[8] “Camram”
URL : http://www.camram.org/.

[9] “Cashramspam,”
URL: http://www.cashramspam.com/home.phtml.

[10] M. Delany, “Domain-based Email Authentication Using Public-keys Advertised in the DNS (DomainKeys),” *Internet Draft*, 2004.

[11] R. Deng, L. Gong, A. Lazar and W. Wang, “Practical Protocols for Certified Electronic Mail,” *Journal of Network and Systems Management*, Vol. 4, No. 3, pp. 279-297, 1996.

[12] Distributed Checksum Clearinghouse
URL : http://www.rhyolite.com/anti-spam/dcc/.

[13] C. Dwork, A. Goldberg, and M. Naor, "On Memory-Bound Functions for Fighting Spam," *Proceedings of 23rd Annual International Cryptology Conference,* Vol. 2729, pp. 426-444, 2003.

[14] “Email Address Harvesting: How spammers reap what you sow,” *Federal Trade Commission survey,* 2002.

[15] “False Claims in Spam,” A report by the FTC’s Division of Marketing Practices, April, 2003.

[16] P. Ferguson and D. Senie , “Network Ingress Filtering: defeating Denial of Service Attacks which employ IP source Address Spoofing,” RFC 2267.

[17] P. Graham, “Better Bayesian Filtering,” *Proceedings of 2003 Spam Conference*, 2003.

[18] P. Hoffman, “Allowing Relaying in SMTP: A Series of surveys,” *Internet Mail Consortium Report, IMCR-016*, 2002.

[19] K. Johnson, “Internet Email Protocols A Developer’s Guide“.

[20] J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker, “SMTP Service Extensions,” *RFC 1425*, 1993.

[21] M. Lentczner and M. W. Wong, “Sender Policy Framework (SPF),” *Internet Draft*, 2004.

[22] G. Lindberg, “Anti-Spam Recommendations for SMTP MTAs ,” *RFC 2505*, 1999.

[23] Louise Kehoe, “Email Spoofing on the Rise,” appeared in National Post, article no. 784, 2001.

[24] “Microsoft Sender IP framework,” *Microsoft Corporation*, 2004.

[25] P. Mockapetris, “Domain Names – Implementation and specification,” *RFC 1035*, 1987.

[26] Open relay black listing database
URL: www.ordb.org.

[27] J. Postal, “Simple Mail Transfer Protocol,” *RFC 821*, 1982.

[28] R. Rivest, “The MD5 Message-Digest Algorithm,” *RFC 1321*, 1992.

[29] R.L.Rivest and A. Shamir, “Payword and Micromint : Two Simple Micro Payment Schemes,” *Proceedings of Security Protocols International Workshop,* pp. 69-87, 1997.

[30] R. L. Rivest, A. Shamir and D. A. Wagner, "Time-lock puzzles and timed release crypto," *LCS technical memo MIT/LCS/TR-684,* 1996.

[31] P. Robichaux, “Secure Messaging with Microsoft Exchange Server,” *Microsoft Press*, 2004.

[32] Sandvine
URL: http://www.sandvine.com/news/in_print.asp.

[33] V. Schiavone, D. Brussin, J. Koenig, S. Cobb and R. Church, "Trusted Email Open Standard," *ePrivacy Group White Paper*, 2003.

[34] Spam Statistics at verisign.
URL: http://www.itfacts.biz/index.php?id=P2038.

[35] “The cost of spam false positives,” *Ferris Analyzer Information Service, Report No. 385*, 2003.

[36] Verisign email phishing statistics
http://www.scamsafe.com/scamsafe/news_general/.

[37] Vipuls Razor
URL: http://razor.sourceforge.net/.

[38] S. M. Yen and P. Y Kuo, "Improved micro-payment system," *Proceedings. of the Eighth National Symposium on Information Security*, pp. 175-186, 1998.