

Networks and Distributed Systems:
**Applying Micro Payment Techniques to
Discourage Spam**

by

Shane O'Brien, B.A., B.A.I

Thesis

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

University of Dublin, Trinity College

September 2008

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Shane O'Brien

September 9, 2008

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Shane O'Brien

September 9, 2008

Acknowledgments

Firstly I would like to thank my supervisor, Hitesh, for all his help and guidance throughout the course of my dissertation.

Secondly I would like to thank my family for all their support during the year especially my parents and grandparents.

Finally I would like to thank everyone from the NDS class, especially Tony for his couch and the other lads in the lab with me, Cian, Eric and Dan.

SHANE O'BRIEN

University of Dublin, Trinity College

September 2008

Networks and Distributed Systems:
Applying Micro Payment Techniques to
Discourage Spam

Shane O'Brien, M.Sc.

University of Dublin, Trinity College, 2008

Supervisor: Hitesh Tewari

E-mail spam is a major problem on the Internet where an estimated 75% of all e-mails sent world wide are spam. The main reason that users send spam e-mails is to make money. This is done by sending millions of e-mails and even if one person responds, the spammer will profit.

This thesis aims to remove the profit that a spammer makes by adding an initial charge to send e-mails thus discouraging them for sending any to begin with. Normal users will not want to pay for their e-mails however so the systems also aims to be cost neutral to the average user. This is achieved by implementing a micropayment system for e-mail where the sender pays a small fee to send the e-mail while the receiver of an e-mail receives the same amount. This way the normal user remains with the same amount of money or more while a spammer who is sending millions of e-mails has to pay for them.

The system is designed with a two tier payment structure. E-mails between the users and mail server are processed and recorded on the mail server, while payments for e-mails

between mail servers is done by using a micropayment technique, called hash chains. This removes the reliance on a central server.

Contents

Acknowledgments	iv
Abstract	v
List of Tables	x
List of Figures	xi
Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Goals	2
1.3 Roadmap	3
Chapter 2 State of the Art	4
2.1 E-mail	4
2.1.1 Mail User Agent - The Client of e-mail	5
2.1.2 Mail Transfer Agent - The Server of E-mail	6
2.1.3 E-mail Protocols: Simple Mail Transfer Protocol	6
2.1.4 E-mail Protocols: Post Office Protocol	8
2.1.5 E-mail Protocols: Internet Message Access Protocol	9
2.2 Spam	10
2.2.1 Types of Spam	10
2.2.2 Methods used by Spammers	12

2.3	Current Spam filtering	13
2.3.1	End-User Methods	13
2.3.2	Simple Filtering Methods	15
2.3.3	DomainKeys Identified Mail	19
2.3.4	Cost Based Solutions	20
2.4	MicroPayments	22
2.4.1	Cryptography: Message Digesting	22
2.4.2	Cryptography: Public Key Encryption	23
2.4.3	Micropayment System using Hash Chains	25
Chapter 3 Design		27
3.1	Requirements	27
3.2	Initial Considerations	27
3.3	Early Designs	29
3.4	Second Iteration Designs	32
3.5	Final Design - CentMail	34
3.6	Other Consideration	36
3.6.1	Internal Mail	36
3.6.2	Mailing Lists	36
3.6.3	Sending Mail from non-CentMail MTAs	37
Chapter 4 Implementation		38
4.1	Main Application	38
4.1.1	Milter API	38
4.1.2	OpenSSL Library	43
4.1.3	XML-RPC Library	43
4.1.4	MySQL Library	44
4.2	The Central Server	45
4.2.1	XML-RPC server	45

4.2.2	User Interface	45
4.3	Databases	45
4.4	Interaction	47
Chapter 5 Evaluation		49
5.1	Testing	49
5.1.1	No Credit Tests	50
5.1.2	Credit Tests	50
5.1.3	Other Tests	52
5.2	User Trials	53
5.2.1	Real User System Trial	53
5.2.2	Live System User Trial	54
Chapter 6 Conclusion		56
6.1	CentMail - A micropayment system for e-mail	56
6.2	Future Work	57
6.3	Final Word	58
Appendix A Abbreviations		59

List of Tables

2.1	Computational Speeds of Various Cryptographic Functions	23
4.1	Milter API callback functions that can be	39
4.2	Values that can be returned from callback functions	39

List of Figures

2.1	Overall E-mail System	5
2.2	Sample SMTP transaction between bob@a.com and alice@b.com	7
2.3	The different parts of an e-mail	8
2.4	Sample of Different Spam E-mails	10
2.5	An Example of an fake phishing E-mail	11
2.6	Examples of Address Munging	14
2.7	Example of Hashcash Stamp	20
2.8	A message being signed and sent by Alice	24
2.9	Hash tokens being exchanged as credits	25
3.1	One of the first designs of the system	30
3.2	Another one of the first designs of the system	31
3.3	Overview of the two layer Hash-Chain payment Scheme	32
3.4	User Commitment Signed by the Central Server	33
3.5	MTA Commitment Signed by the Central Server	33
3.6	An Overview of an E-mail being sent on the final design	34
4.1	Initialising the Milter and starting the main event loop	40
4.2	Private memory being set up for the system	41
4.3	The flow of the code within the eom callback	42
4.4	A XML-RPC function from the system	44
4.5	Communication between services for an internal e-mail	47

4.6	Communication between services for an external e-mail	48
5.1	The application output for sending an e-mail with no credit	50
5.2	Application output from sending an e-mail to another server	51
5.3	Application output from receiving an e-mail from another server	52
5.4	Application output from receiving the same payment token twice	53
5.5	Results from the Real User System Trial	54

Chapter 1

Introduction

E-mail can trace its origins to before the Internet[1]. It was originally used to send text messages between users registered on the same computer, but was later extended to allow messages to be sent to users on other computers as well. In modern times e-mail has become one of the major forms of communication and is popular for people of all ages. It allows text to be sent from one person to any other person world wide, with an e-mail address. E-mails also offer the ability to attach files like pictures and documents. The popularity and growth in e-mail as a form of communication for people in their everyday lives, unfortunately, has resulted in some users taking advantage of this and using e-mail to send unsolicited bulk e-mails to other users. This unsolicited bulk e-mail is known as spam.

1.1 Motivation

Spam is not only an annoyance to e-mail users, but also costs businesses and internet service providers money to transmit, store e-mails and prevent spam from reaching their users. The reason that spam is so common is that the cost to send millions of e-mails is almost nothing. So if a user is trying to sell a product using spam they can send millions of e-mails advertising the product and then if only one person buys the product then the

spammer stands to make a profit.

Existing e-mail filters reduce the amount of spam being received by filtering them on the receiving side of the system; however the cost to transport them is still there. Filters also remove legitimate e-mails which can be, depending on the importance of the e-mail, more annoying to the user than spam. To allow the user to check to see if anything was marked as spam, the spam e-mails are kept on the machine, so that the user can look through them and unmark the ones that are not spam. This means that the provider of the e-mail service has to pay to store the e-mails that are considered spam. The better the spam filter the less likely it is that a user will check their spam folder, because they trust that the spam filterer is doing its job correctly.

1.2 Goals

The overall goal for this dissertation is to implement a micropayment system to be used to send and receive e-mail, which will discourage users from sending excess e-mails or spam. It is hoped that by adding a charge for the user to send an e-mail that spammers will not profit from sending millions of e-mails. Normal users, however, will not want to pay for their e-mails so the system will need to be cost neutral for the average user. The system tries to minimise false positives, i.e. legitimate e-mails marked as spam. The main goals are as follows:

- Identify methods that are used by spammers as well as current anti-spam methods that are used to combat spam.
- Identify current e-mail payment systems that are being used.
- Design and implement a micropayment system for e-mail that is cost neutral for a normal user and has a minimal amount of false positives.
- Evaluate the system to determine its effectiveness.

1.3 Roadmap

The structure of the remainder of this dissertation document is outlined as follows:

Chapter 2 will provide background and context for this study by reporting on the current state of the art. This will contribute to the understanding of the main areas relating to this dissertation.

Chapter 3 describes the considerations and steps involved in the design of this dissertation.

Chapter 4 outlines the implementation of the design as covered in chapter 3. It will also discuss some problems encountered, as well as solutions to them.

Chapter 5 discusses the testing of the final application to assure that it was working correctly, as well as covering the user trials and the results from them.

Chapter 6 presents a conclusion, including a summary of the achievements and covers some possible improvements and future work.

Chapter 2

State of the Art

This chapter will discuss current methods, protocols and implementations that there are for e-mail, spam, spam filtering and micropayments. Each section will cover a different topic and provide insight as to the current state of e-mail and spam.

2.1 E-mail

E-mail or electronic mail is not run using a single server or application; it is instead made up of many clients, multiple servers and various protocols, all of which will be described within this section. The overall system looks like Figure 2.1; a typical series of events for Alice sending an e-mail to Bob and is as follows:

1. Alice writes the e-mail using her client which can be a stand alone desktop application or a web based client. When she is finished writing her e-mail, she puts Bobs e-mail address in the 'to' section of the e-mail and sends it.
2. Alice's client then transmits her e-mail to the local Mail Transfer Agent (MTA), the mail server, using a protocol called Simple Mail Transfer Protocol (SMTP [2]). The local MTA checks that all the necessary information is included with the e-mail, if it is not, it is returned to Alice, who will have to send it again with the necessary

corrections. The server is the part after the '@' in the e-mail address, so if the e-mail is for someone on the same server, then the e-mail is delivered straight away.

3. In this case Bob's e-mail account is on a different server, so Alice's MTA performs a DNS look up for Bob's domain. This will return either the IP of the domain or the MX record if the e-mail server is not the same as the domain. Alice's MTA connects to Bob's MTA using this information and transmits the e-mail message using SMTP.
4. Once the message arrives at Bob's MTA, it is stored until requested by Bob, using a desktop application or web based client. The message is retrieved by these applications using the POP3[3] or IMAP[4] protocols.

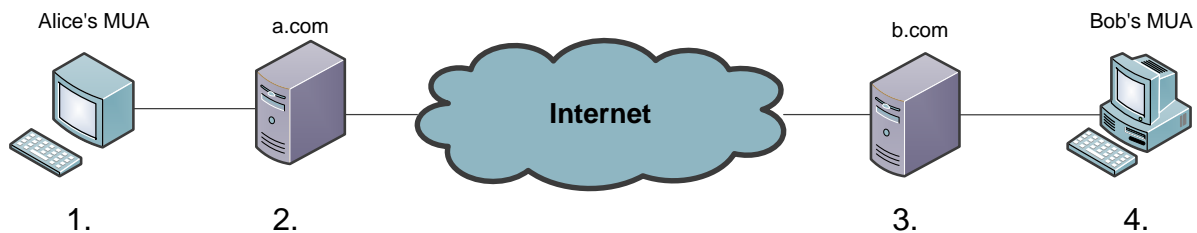


Figure 2.1: Overall E-mail System

2.1.1 Mail User Agent - The Client of e-mail

The Mail User Agent (MUA) is normally the only part of the e-mail system that the end-user sees. It allows users to write, send and receive e-mails. MUAs can be desktop applications that give users lots of different features from HTTP formatting to encryption of e-mails. Some major examples of desktop MUAs are Microsoft's Outlook[5] and Mozilla's Thunderbird[6]. MUAs can also be web based, this web based e-mail, or web-mail, gives users easy access to their e-mail from any internet connection in the world. Most webmail services offer the same features as the desktop applications and are normally easier to use. Some major examples of webmail services are Google's Gmail[7] and Microsoft's Hotmail[8].

Almost all MUAs, web based or desktop based, use the SMTP protocol to send their e-mails to the local MTA servers but offer the user a choice over which protocol, POP3 or IMAP, to use to retrieve their e-mails from the server. Some webmail services that are hosted on the same server as the MTA can avoid using either protocol and just read the e-mails directly from the files on the disk.

2.1.2 Mail Transfer Agent - The Server of E-mail

The Mail Transfer Agent refers to the software that transfers mail from one server to another or receives mail from MUAs. MTAs use the SMTP protocol to transfer e-mail across the internet. The most popular[9] MTA software is Sendmail[10] with the open source server Postfix[11] coming a close second.

On an e-mail server the MTA software is the most important part of it, but MTAs cannot send mail directly to user's computers. To enable on demand retrieval of e-mails the mail server has POP3 and/or IMAP software running as well as the MTA software. Some popular POP3/IMAP software is Courier[12] and Dovecot[13], both of which support POP3 and IMAP access to e-mails. Thus a mail server normally consists of the MTA, Sendmail or postfix etc, and one or more of the protocols to retrieve e-mail, Courier or Dovecot etc.

2.1.3 E-mail Protocols: Simple Mail Transfer Protocol

The main protocol that is used in the transfer of mail between the user and MTA as well as between MTAs is called Simple Mail Transfer Protocol[14], or SMTP. The actual protocol been used to today is called Extended SMTP[2], but is still referred to as SMTP.

SMTP is a simple text based human readable protocol. The commands are similar to English words and the replies to the commands are also in this format. An SMTP server can be used by connecting to the server using Telnet[15] or similar, this will allow you to send e-mails using typed commands, but most of the time is done by the users MUA.

SMTP can only send messages by connecting directly to another SMTP server; it can not receive messages on demand by connecting to another server.

```
S: 220 smtp.example.com ESMTP Postfix
C: HELO a.com
S: 250 Hello a.com, I am glad to meet you
C: MAIL FROM:<bob@a.com>
S: 250 Ok
C: RCPT TO:<alice@b.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: From: "Bob" <bob@a.com>
C: To: Alice <alice@b.com>
C: Date: Tue, 15 Jan 2008 16:02:43 -0500
C: Subject: Test message
C:
C: Hi Alice.
C: This is a test message.
C: Bob
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye
```

Figure 2.2: Sample SMTP transaction between bob@a.com and alice@b.com

An SMTP session starts with an 'HELO' or 'EHLO' command which is the 'hello' or 'extended hello' command, this is accompanied by the host name of the server from which the connection is been made. If a command is accepted by the SMTP server then the server will reply with a '250 ok' which means that its ready for the next command. Once connected and the HELO has been accepted the information for the e-mail can be sent. To start a new e-mail the first thing that needs to be stated is who the e-mail is from. This is done with the 'MAIL FROM:' command accompanied with the e-mail address of the user sending the e-mail. The next thing that is needed is who the e-mail is for this is entered after the 'RCPT TO:' command, there can be as many of these as required. All these commands will be acknowledge by the server with a '250 ok' if they are accepted. Next is the 'DATA' command, this command starts continuous input to the server and

is where any extra e-mail headers as well as the body of the e-mail go (see Figure 2.3 for the different parts of an email). To end the 'DATA' second of the session put a single '.' on its own line now the e-mail will be queued for delivery. At this point 'MAIL FROM:' will start a new e-mail or 'QUIT' will close the connection. See Figure 2.2 for an example of an e-mail being processed by SMTP. The end user will almost never need to use these commands as their MUA will do it for them.

```

Message Headers {
  Received: by 10.210.121.13 with HTTP; Mon, 1 Sep 2008 03:40:40 -0700 (PDT)
  Message-ID: <adf556f90809010340w4ae03cfffv9bb1384d7e7bc2a1@mail.gmail.com>
  Date: Mon, 1 Sep 2008 11:40:40 +0100
  From: "Shane O'Brien" <obries10@tcd.ie>
  Sender: obriensh@gmail.com
  To: "Shane O'Brien" <obriensh@gmail.com>
  Subject: This is the subject line
  MIME-Version: 1.0
  Content-Type: multipart/alternative;
    boundary="-----_Part_41906_9245459.1220265640326"
  Delivered-To: obriensh@gmail.com
  X-Google-Sender-Auth: 2371966d3655f9d6

Message Body {
  -----_Part_41906_9245459.1220265640326
  Content-Type: text/plain; charset=ISO-8859-1
  Content-Transfer-Encoding: 7bit
  Content-Disposition: inline

  Hi shane,

  How are you
  The body of the email goes here.

  From
  Shane

  -----_Part_41906_9245459.1220265640326
  Content-Type: text/html; charset=ISO-8859-1
  Content-Transfer-Encoding: 7bit
  Content-Disposition: inline

  <div dir="ltr">Hi shane,<br><br>How are you
  <br><br>The body of the email goes here.
  <br><br>From<br>Shane<br></div>

  -----_Part_41906_9245459.1220265640326--

```

Figure 2.3: The different parts of an e-mail

2.1.4 E-mail Protocols: Post Office Protocol

The Post Office Protocol version 3[3], or POP3, allows users to retrieve their e-mails from the mail server on demand. This allows an e-mail client to connect to the mail server and retrieve a list of the messages in a user's mailbox; the user can then retrieve them and/or delete them from the server while keeping local copies. POP3 commands, like SMTP commands, are human readable and writeable. A POP3 server can be connected to via telnet or similar and commands are manually entered to check mail.

The POP3 server waits for connections from users on the default port of 110. Once

the user connects they must supply the 'USER' and 'PASS' commands together with their username and password to be allowed to access their e-mails. When the username and password are accepted the POP3 server will display how many e-mails are currently on the server for that user as well as the total size of all the e-mails. The user can use 'LIST' to get a more detailed list of the e-mails that are on the server and the size of each one is. To read an e-mail the user uses the command 'RETR' with the message number of the e-mail they wish to read. They can delete an e-mail from the server with the command 'DELE' and the message number. As with SMTP, the user does not need to use these commands and instead their client will perform them for the user. The e-mail client will normally use the 'RETR' command to save the message to the local drive and then delete them off the server with 'DELE'. The clients can be configured to leave the e-mails on the server until they are deleted locally.

2.1.5 E-mail Protocols: Internet Message Access Protocol

Internet Message Access Protocol[4], or IMAP, is similar to POP3 as it allows a user to retrieve their e-mail on demand from an e-mail server. A major difference from POP3 is that IMAP supports both online and offline modes and all e-mails are left on the server until manually deleted by the client. In online mode the client is automatically notified of new messages which can then be requested from the server. Any changes that are made to the e-mails, deleting or reading etc, are instantly reflected on the server. In offline mode changes and new e-mails are queued until the client connects again. IMAP has support for multiple clients for a single mailbox and changes are reflected between the clients. For example checking your mail on your mobile phone will mark the messages you've deleted and read, so when you check your mail on your computer they will be already deleted or marked read.

Again like SMTP and POP3, IMAP commands are human readable. However, unlike SMTP and POP3, they are not as easy to read and a lot more text intensive with multiple

places where the mail is stored. IMAP realistically can only be used with client software, and to take advantage of all of its extra features needs client software to use it.

2.2 Spam

Spam, or unsolicited bulk e-mail, is estimated to constitute up to 73%^[16] of total e-mail traffic in the world, that's over 14 billion messages. There are many different types of spam, see Figure 2.4 for a selection of spam, and methods that spammers use. This section will explore the most common forms of spam and methods used.

donn fidel	» N0// save on meds you need_+_ - Hi there, Generic drugs...the only w:
friedrick kenji	» buy cheap c1alis, /i@gra and other best pharmaceuticals =.= gghg
Congratulations	The Enclosed Bonus for dfxsfgx xvf will VOID on 2008/08/23 - To en:
army dorai	» debt counseling - Do Not consolidate your debt Eliminate it! Legally EL
vaz Mascarenhas	» Britney Spears Latest Wardrobe Malfunction Exposes Her Assets - '
isador roman	» Play world's best gambling house - Join me and over 2.400.000 playe
ives Grams	» Britney Spears will play a mutant in a new movie - VIDEO
Visualware Inc.	» Visualware Newsletter, August Issue - MCS 8.1 - VISUALWARE NEV
locke annalena	» colorado debt consolidation - Do Not consolidate your debt Eliminate
me	» RE: SALE 89% OFF
Ila Wilson	Order a Ph.D - WHAT A GREAT IDEA! We provide a concept that will a
cos kyeongso	» ID:16505 Get all your pharmaceutical supplies online from now on
P.K.W.CHAN	» EAGER TO HEAR FROM YOU. - From Patrick KW Chan.(Urgent) ! patric
Vanessa Hudson	OLYMPIANS STARTED TO USE 15X MORE POFERFUL FORMULA - F
christoper boon	» b00st your satisfaction with Cialls! - We're glad to inform you, just bes
brandtr russ	» Cheap Meds Viagr@ Many M3ds QnNXpRy9 - Our special newsletter,
Money and Markets	Gold in the Clouds - MONEYANDMARKETS» Wednesday, August 20,
Yolanda Joann	» Rep1icaRolex full-series: 85% Off LuxuryWatches, Hundred Latest
Octavio Peters	jSxdly yexplccin Ffreep Ppo rnoe K5DeBaSsQrEmQ1.4mvgB0g -

Figure 2.4: Sample of Different Spam E-mails

2.2.1 Types of Spam

One of the most common types of spam is advertising products for sale. In this case, the spammer sends out millions of e-mails in the hope that a small number of those spammed will purchase the product. As the cost to the spammer is almost nothing, anything they sell will generate income. They can expand their product range without any marketing costs. Popular types of products that are advertised through spam include medical drugs,

adult entertainment, and health and weight loss.



Dear valued customer of TrustedBank,

We have recieved notice that you have recently attempted to withdraw the following amount from your checking account while in another country: \$135.25.

If this information is not correct, someone unknown may have access to your account. As a safety measure, please visit our website via the link below to verify your personal information:

<http://www.trustedbank.com/general/custverifyinfo.asp>

Once you have done this, our fraud department will work to resolve this discrepancy. We are happy you have chosen us to do business with.

Thank you,
TrustedBank

Member FDIC © 2005 TrustedBank, Inc.

Figure 2.5: An Example of an fake phishing E-mail

Now a days 'phishing' (pronounced fishing) is used extensively to acquire login details, bank account and credit card details and personal information that can be used for identity theft. Phishers, spammers who send phishing e-mails, try to make their e-mail look like they are coming from a legitimate bank or other website that they want information for, see Figure 2.5 for an example of an phishing e-mail. They include a link in the e-mail that goes to a website which is not run by the bank but looks like it does. Phishers normally scare the user into clicking on this link and entering their details by saying that their account will be disabled if they do not. To make the link to the website look real, phishers can use sub-domain and Internationalised domain names (IDNs). By using a sub-domain, the phisher can set up a website like <http://www.yourbankname.example.com/>, where they own the example.com domain but at a quick glance, a user would see the bank name first. With the release of IDNs, phishers could replace the letter 'a' in addresses,

and some other letters as well, with Cyrillic small letter 'а'. This allowed addresses to look identical to the user but would redirect to a phishing website. This vulnerability has been removed by most of the latest internet browsers.

Advance-fee fraud, Nigerian money offer or 419 frauds refer to a type of spam e-mail where the receiver is offered a large sum of money, but to get the money they have to send a smaller sum of money to the sender. This is a variation of an old confidence trick, which offers the hope of large riches in exchange for a small advance sum of money.

The aim of almost all types of spam is to be profitable for the spammer, and this only takes a few people to open and respond to the e-mail.

2.2.2 Methods used by Spammers

For a spammer to send spam they need to get a list of e-mails. This list can be bought from a list merchant or harvested using many different methods for harvesting e-mails addresses. They can use harvester bots, which can trawl through websites searching for e-mail addresses posted on them. E-mail addresses can also be 'guessed' by using a brute force dictionary attacks. This involves trying every word in front of the domain name and sending a test e-mail to that address, any e-mail address that doesn't exist will bounce and be taken off the list, and any that aren't bounced are added to the list and reused.

Once the spammer has a list of e-mail addresses they can use a variety of methods to avoid being black-listed or prosecuted. Lots of e-mails can be sent anonymously, using free webmail services. Most free services have a limit on the amount of e-mails that can be sent per day, so spammers use web bots to register and manage multiple accounts at the same time.

Proxies are computers that users and services can use to make indirect connections to other computers and services like e-mail servers. Open proxies are proxies that allow any connection through them. They can be used to anonymously route e-mail from the spammer to their destination. This helps to reduce the risk to spammers of getting caught.

The use of viruses that take control of users computers, to send spam from, is another widely used tactic. The spammer can set up a website that infects users that browse to it, or send spam to millions of users with the virus attached. Once the user is infected, their computer becomes part of the spammers' botnet or zombie network whenever it is connected to the internet. Botnets are collections of infected computers that can be used to send spam messages via the infected machines and the users e-mail accounts.

2.3 Current Spam filtering

A lot of research and money has gone into combating spam. There is a wide variety of methods available that attempt to combat spam. These range from simple methods that the end-user can use to cut down on the amount of spam that they get, to powerful filters that process millions of spam emails per day.

The following sections will cover some of the filtering methods currently available.

2.3.1 End-User Methods

There are a number of simple methods that the end user can do to avoid or limit spam in the first place. Most of these methods are free, easy to implement and if everyone adopted them there would be a lot less spam.

The first thing is simply not to post your e-mail address publicly. This is impossible for most people, so a method know as Address Munging can be used to obfuscate the e-mail address, when publicly posting the address. Basic Address Munging involves changing the way the address is written, so that a machine would not be able to read the valid address, but still having it correctly human readable. Examples of this involve adding extra words/spaces/special characters to the address which can be easily seen by humans and removed, but not by machines. The following are some examples of basic Address Munging:

More advanced Address Munging include replacing every characters of an address with

john@NOSPAMexample.com
john@example.com.invalid
john(AT)example.com
john at example dot com
j o h n @ e x a m p l e . c o m

Figure 2.6: Examples of Address Munging

its ASCII equivalent, so that when displayed on a web page it is readable, but if a machine was looking at the code it just looks like a string of numbers and symbols. Another is to use a client side scripting language, like JavaScript, to take parts of the address and display them correctly on the web page, again this is readable by the user but not a machine. The address could also be drawn on an image and the image displayed instead of the text, this will work because the machine won't be able to read the address in the picture, where as a human can.

According to Prince et al. [17] 52% of machines will not recognize an address if the @ sign is replaced with the ASCII equivalent, while almost no machines will recognize an address if it's displayed with JavaScript or in an image.

If a spammer decides to send spam emails to random accounts, if they get replies from the user, then they know that the account is active and that their e-mail got through, what ever filters there is on that server. Replying to spam e-mails, therefore is the wrong thing to do, even to those with an opt out option (e.g. click this link if you do not want to receive further e-mails). The other problem with replying to spam e-mails is that most of the time the reply address is forged, and as it doesn't exist, the e-mail will bounce back to the user causing more spam, or get sent to another user who is unrelated to the spammer. Thus a way to reduce some spam would be to avoid responding to spam e-mails.

It is possible to get a disposable e-mail account for users signing up to certain website, likely to make them more vulnerable to spam or where they need to post their address somewhere with a likelihood of being harvested by a spammer. These disposable accounts are normally used once or a few times and then disposed of once they start receiving

spam. These accounts are not suitable for long term communication between people, as they would have to be replaced every few days and the new address sent on to the people, who they wish to communicate with.

Finally a simple way to reduce the amount of spam overall is to report any spam messages that are received by a user. Spam messages can be reported to many different places, the biggest being is spamcop.net [18] which reported an average of 23 spam messages, a second over the last year. Reported spam is used for Blacklisting, which will be covered in the next section, and also for research into improved ways of fighting spam and new methods implemented by spammers.

2.3.2 Simple Filtering Methods

Most modern servers now have filters to try to eliminate incoming spam. The most common and easiest is called Blacklisting, with White and Gray listing been used at the same time.

Blacklisting[19] uses a list of e-mail addresses and IP addresses from known or reported spammers and does not allow e-mail messages from these to be delivered to the recipient. Blacklisting can use either a local list of spammers or, as discussed in the previous section, can use real time Blacklisting from a central server, which maintains a list of e-mail addresses, IP addresses, DNS server, forwarding server and ISPs which have been reported to be spam sending or relaying sources. If an e-mail is received from an address on the blacklist, it can be discarded or rejected back to the sender[20], rejecting it may help reduce the amount of spam that is coming from that spammer in the future.

Whitelisting[19] is a way of preventing false positives. Instead of discarding or rejecting anything that is on the list, as Blacklisting does, it allows any e-mails from addresses or IP addresses on the list. This is used a lot for contacts in e-mail address books. When used with other filtering methods Whitelisting allows e-mail from your contacts to be delivered even if they normally would be filtered out by another filter.

Graylisting[19] takes advantage of the SMTP protocol[2] being able to temporarily reject incoming messages. Normal MTAs will recognise this temporary rejection and try again at a later time. Graylisting works by recording the sender's address, recipient's address and the sender's IP address (these 3 items together are called a triplet) of incoming e-mails. If this triplet has already been recorded before, then the e-mail is allowed through. If the triplet has not been recorded before then the e-mail is temporarily rejected for a certain amount of time. After this time if the e-mail is retried, it is let through. If the sender is a spammer then they will either not retry sending the message after the first rejection or if they do retry the spammers address will have already been blacklisted by other receivers of the spam. Thus Graylisting is more effective when combined with real-time Blacklisting.

Content Filtering is another common method of filtering e-mails for spam. It filters e-mail based on its content. The easiest way of doing this is to reject e-mails containing words that appear regularly in spam messages. Huge problems can happen here, as e-mails, which are not spam are blocked because of the word 'assign' which has the word 'ass' in it, which would be filtered by the content filter. Content filters can be bypassed by adding extra white space and symbols between the letters in a filtered word. Misspelling words would be ignored as well, meaning that content filtering on its own is a very weak form of filtering.

An improvement on content filtering and one of the main types of filtering used today is statistical content filtering[21]. Statistical content filtering uses Bayesian probability to determine e-mails which are spam, and which are not. It works by scanning the users received e-mails and assigning a probability to each word or token of it been spam, this list is saved as a database for each user. When a new e-mail arrives it is scanned and the top 15 words with a probability furthest from 0.5 are taken and the Bayesian probability of these numbers is calculated[21]. If the resulting probability is greater than 0.9 it is considered spam, anything else is not.

To use statistical content filtering to it's full extent, each user would have a 'Report

as Spam' button which would analyse the e-mail as spam increasing the probability of certain words been spam. All mail, not reported as spam, would be analysed as not spam, decreasing the probability that the contained words are spam. As each user has a different database for their words and probabilities, users who have legitimate e-mails containing words that would otherwise be filtered, would not be filtered using this method. For example a chemist who is sending lots of e-mails referring to drugs, won't have their legitimate e-mails filtered, where as any spam e-mails about drugs would still get filtered because of the difference in the content.

A different type of spam filtering is the challenge/response system. With this type of filtering when an e-mail is received by a server, the server automatically replies to the sender with a challenge which must be responded to in order for the e-mail to be delivered. The challenge is simple to do for a single e-mail, but time consuming for a lot of e-mails, and almost impossible to automate. The challenge can consist of simply replying to the challenge e-mail or clicking a website link within the challenge e-mail. Most challenge e-mails contain some form of Turing test, a test to determine if something is a human or a machine, in order to make sure that the response to the challenge is from a human. This Turing test is normally in the form of a CAPTCHA, which is a word or group of letters slightly distorted in an image, so it's relatively easy for a human to read but very hard for a machine to read. Since a lot of spam e-mails are sent with forged reply addresses, the spammers would not get the challenge at all and thus the e-mail would not be delivered. Even if they did get the challenge, the amount of time it would take to respond to each of the challenges would not make it worth while.

Checksum-based filtering[22] takes advantage of the way that spammers send bulk e-mail. All the e-mails are mostly the same, except for some slight differences. Checksum filtering removes anything that might make these e-mails different and creates a check sum of the remaining information, which it stores in a database. The user can report e-mails as spam, and then if any e-mails with the same checksum are received they are marked as spam. Spammers can get around this by adding large chunks of invisible differences

into each e-mail, making checksum filtering effectively useless.

Most MTA servers do not fully enforce the standards set out in RFC 2821[2] and, instead allow the administrator to allow or disallow features. If the standards are enforced fully, mail coming from servers that do not comply with the standards can be blocked. A lot of spammers use software that does not have all the features and standards and so, would have their e-mails blocked. Some of the main standards that can be used to reduce spam are HELO/EHLO checking, Nolisting and Greeting delay.

HELO/EHLO checking can reduce the amount spam received by a mail server by up to 61.8%[23] by simply checking to see if the domain in the HELO statement is a Fully Qualified Domain Name (FQDN). If it is not, then it is rejected with out being received in the first place, which cuts down on the amount of bandwidth the server has to use. The domain can also be looked up via DNS, to check the IP against the server that is connected to make sure it is from that domain.

Nolisting uses a section in the SMTP protocol that provides a prioritized list of e-mail servers on a particular domain. This list is maintains in the DNS as MX records, so if the main server goes down there can be a backup server to take over, while the main server is restarted. Nolisting takes advantage of this by putting a fake server address as the first record, so that when a spammer, using their own custom software, tries to send mail the attempts will fail, as their software does not try and contact the lower-priority servers on the MX records list.

Greeting delay introduces a deliberate pause between when a client connects to the MTA server and when it sends the greeting. Since no communication can take place with the MTA server until the greeting has been sent, anything sent before will be discarded. Spammers try and send their e-mails as fast as they can and a lot of the time will not wait for the greeting, and instead will start sending as soon as the connection is established. The server will discard anything they send and disconnect them if they try this.

There are many different types of spam filtering available, each has its own strengths and weaknesses, but which provides the most effective means of filtering spam? Hybrid

filtering uses all or some of the different types of filtering that have been covered here. It assigns a numerical score to each test result. Running each test on an e-mail gives that e-mail a total score, which determines if that particular e-mail is spam or not. Since no single test is able to mark an e-mail as spam, the false positives can be reduced while keeping the amount of spam filtered high. Widely used Open Source mail filters such as SpamAssassin[24], and Policyd-weight[25] use this method of filtering.

2.3.3 DomainKeys Identified Mail

DomainKeys Identified Mail[26] or DKIM is a new method for e-mail authentication. DKIM is a hybrid of Identified Internet Mail, created by Cisco[27]; and DomainKeys[28], created by Yahoo[29]. It is an e-mail header based signature system which is supposed to be able to protect the sender from spoofing and cut and paste attacks.

When a user sends an e-mail the following happens:

1. The sender's MTA signs the message and puts the signature into the e-mails header.
2. The recipient's MTA verifies the signature in the header.
3. The recipient's MTA contacts the receiving domains DNS to get its public key from the domainkey sub domain.

When the signature is being verified it must pass two tests before it's authenticated. Firstly it must verify that the message was not edited or changed in any consequential way and secondly the receiving domain must ask the sending domain to confirm that whoever signed the message was authorized to do so.

The main advantages of using DKIM are that it allows the source domain of an e-mail to be identified. This mean that forged e-mail messages can be deleted when they are received. Some weaknesses exist with DKIM, for instance if a message was significantly modified in transit or by another filter the signature may not be valid anymore, even if it is a legitimate message. For example some free e-mail providers add advertisement on the

bottom of incoming e-mails, this would cause the signature to become invalid and thus get rejected.

DKIM itself does not filter spam, but with widespread use it can prevent spammers forging source and return addresses and thus allow existing filtering techniques to work more effectively, by forcing the spammers to use a valid source domain. This would make domain based black and white listing more effective, as well as making phishing attacks easier to spot.

2.3.4 Cost Based Solutions

There have been a few attempts at using cost based solutions to combat spam. Some solutions require cost in the literal sense and attach 'stamps' to the e-mails, where as other solutions have a computational cost instead.

One attempt at implementing a computational cost solution is called Hashcash[30]. The idea behind Hashcash is that 'payment' takes the form of used CPU cycles which, to a spammer sending millions of e-mails, can be monetarily expensive in terms of time and power. The sender generates the Hashcash stamp by using the date, the recipient's e-mail address and the information to verify the hash together with a random number and computes the SHA-1[31] hash of them. If the first n bits are zero, where n is how much work the sender needs to do (a bigger n means more work), then that stamp is acceptable and is attached to the e-mail header as proof of work. If the first n bits are not zero then a new random number is generated and tried again until the first n bits are zero. When the e-mail is received by the recipient, it only has to hash the header value and verify that there are n zeros at the start, if there is then the e-mail is accepted.

1 : 40 : 051222 : *foo@bar.org* :: *Cu2iqc4SmotZ7MRR* : 0000214c3J

Figure 2.7: Example of Hashcash Stamp

The time to compute a stamp for $n = 20$ on a 2.13Ghz Core2 Duo takes 0.16 seconds[32]

doing 6.7 million hashes per second. This would not cause much of a problem for most spammers, but by increasing n by just 1, doubles the length of time it takes to compute the stamp. So by using an n of 30, instead the time would be around 2.5 minutes per stamp, meaning a spammer could only send around 575 per day, instead of the millions they normally send, and to a normal user, this amount of e-mails per day would be more than acceptable.

Certified e-mail[33] is a technique that allows a delivery receipt to be obtained, when an e-mail is received by the recipient, by using a Trusted Third Party (TTP). At the same time a proof of mailing is also available, once the receipt is signed. E-mails sent in this manner can be assured to pass filters and be delivered to the recipient. The TTP, normally called the Post Master (PM), can charge a fee for this service either per e-mail or for a certain time period. The system works by routing all mail through the PM. When a sender(S) sends an e-mail to someone else(R), it is instead sent to the PM, who issues a proof of mailing to S. The PM then encrypts the e-mail and sends it to R, who then issues the receipt to the PM, who in turn gives R the key to decipher the message. The PM then gives the receipt to S. This means that there is a lot of traffic going through the PM, which causes a bottleneck at that point.

Goodmail Systems[34] is currently providing Certified Email services for major companies such as AOL[35] and Yahoo[29]. Goodmail charges users \$2.50 per thousand messages, as well as a \$399 accreditation fee. The accreditation means that only certain people will be able to send mail using there service, they mainly limit it to companies in the UK, US and Canada.

Another current implementation of a money based system uses a payment at risk system, similar to bonds. In this system if the sender is not on a list of approved senders, then they have to attach a payment stamp that is only redeemed if the recipient disapproves of the e-mail. So if a friend who's not on your approved senders list sends you an e-mail, they have to attach a stamp, but since you are happy to receive the e-mail, the stamp is not redeemed, where as if a spammer sends you an e-mail and you report it as

spam, then the spammer loses their stamp.

All of the current implementations of payment based filtering involve use of resources, and in some cases money, has to be paid to the companies, running the service. This means that there is a cost to legitimate users, in terms of time and/or money, to send and/or receive e-mails and this is not acceptable to most users.

2.4 MicroPayments

There are many different methods to pay for goods and services over the internet. These methods include credit cards and other macropayment systems. Macropayment systems allow users to pay for goods and services, without exposing their payment details to the vendor for a fee. However, for low value goods and services, the fee that macropayment systems charge, would eliminate any profit for the vendor. So if a vendor is selling items for a cent or providing a pay per query database, they could not use a macropayment protocol. The vendor could instead use a micropayment[36] system which can deal with very small values of fractions of cents. As the value of each transaction is very small, real time communication with the payment server and high levels of security is not feasible, as it would make the service unprofitable. This means that the micropayment systems have to be quick, low on processing and with no unnecessary communications with a central server.

The following section describes a method of implementing a micropayment system using hash chains. Hash chains are used in the credit based micropayment system called PayWord[37]. The cryptographic techniques used in hash chains will also be introduced.

2.4.1 Cryptography: Message Digesting

Message digesting is a one way transform that takes an input of any size and returns a fixed sized string. The same input will give the same output every time it is used, while at the same time changing a single letter or value in the input will give a completely different

output. Most hash functions are fast and computationally inexpensive, see Table 2.1 for the average speeds on a typical computer (A 1.9GHz AMD-64 PC running Windows XP). Two hash functions that are used a frequently in various applications are MD5[38] and SHA[31].

Operation	Number per second
RSA 1024 Signing	398
RSA 1024 Verification	10,000
DES	480,000
IDEA	780,000
SHA-512 Hash	3,200,000
SHA-1 Hash	15,500,000
MD5 Hash	30,000,000

Table 2.1: Computational Speeds of Various Cryptographic Functions

MD5 outputs a 128-bit string, normally a 32 digit hexadecimal number. MD5 is one of a series designed by Ron Rivest, the others are MD2 to MD4. It was one of the most widely used message digest functions, but since 2006. Its use has diminished as collisions can be found in under a minute[39]. If collisions can be found in a message digest function it is considered insecure. Collisions can be used to substitute an unauthorized message for authorized messages.

SHA, Secure Hash Algorithm, can produce different length outputs depending on the type of SHA function used; they are SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512. The latter four are known as SHA-2. SHA-1 produces a 160 bit output, 40 digit hexadecimal number, and the others outputs are the same as their names. There is also a SHA-0 but that is considered insecure like MD5. The SHA functions were designed by the National Security Agency (NSA) of the U.S.A.

2.4.2 Cryptography: Public Key Encryption

Public Key Encryption, or sometimes called asymmetric cryptography, is a process in which plain text is encrypted using one key and decrypted using a different key. Each user has a public and a private key generated from a large random number, normally a

prime number. The public key is given out to anyone who wants it, so that it can be used to encrypt messages that are going to be sent to the owner of that public key. The private key is kept secret by the user, and is used to decrypt any messages that were encrypted by the public key. One major advantage of using this type of encryption is that once you encrypt a message only the user with the private key pair of the public key can decrypt it.

Using the fact that only the user knows their own private key, it is possible to sign messages so that the identity of the sender of a message can be determined. Instead of encrypting a message with the receivers public key, the senders private key is used, see Figure 2.8 for an example. The receiver can then use the senders' public key to decrypt the message, confirming that it was from that sender. This method also can also be used to make sure a message is not tampered with.

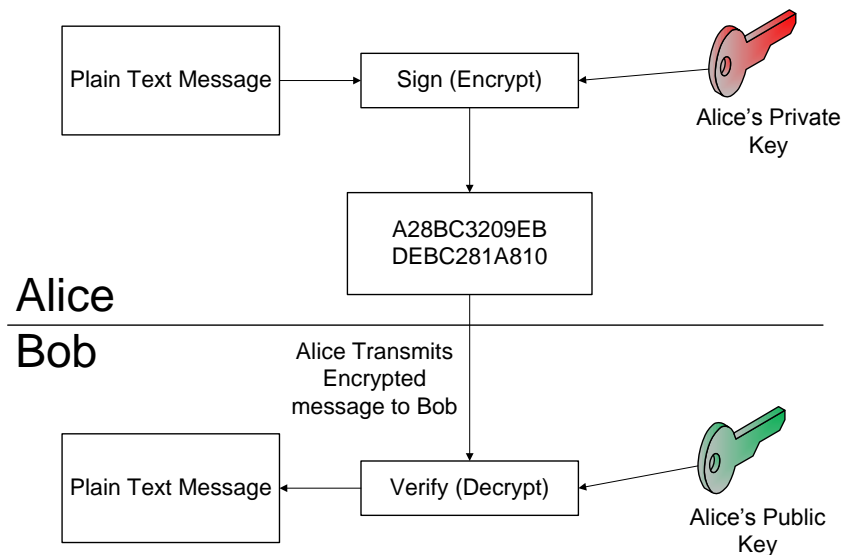


Figure 2.8: A message being signed and sent by Alice

One of the main problems with public key encryption is proving that a public key belongs to a certain user and has not been exchanged for one of a malicious user. The current way of establishing public key ownership is by using a public-key infrastructure [40], which is made up of a number of certificate authorities who certify the ownership of public keys and also provide a place to look up public keys when they are required.

Another big problem is that signing and verifying keys is computationally expensive, Table 2.1 shows the speed of signing and verifying using RSA 1024 which is one of the algorithms used to calculate public and private keys.

2.4.3 Micropayment System using Hash Chains

Hash chains are a collection of hashes where the hash of N in the chain is $N-1$ in the chain, e.g. for a chain of 20 hashes the first one or anchor, N_0 , is the digest of the N_1 and so on until N_{20} which is randomly generated and the digest of it is N_{19} . These hash chains along with a signed commitment can be used to authenticate payments without having to contact a central server. Each chain and commitment is vendor-specific, so they can only be spent at a single place and only by that user.

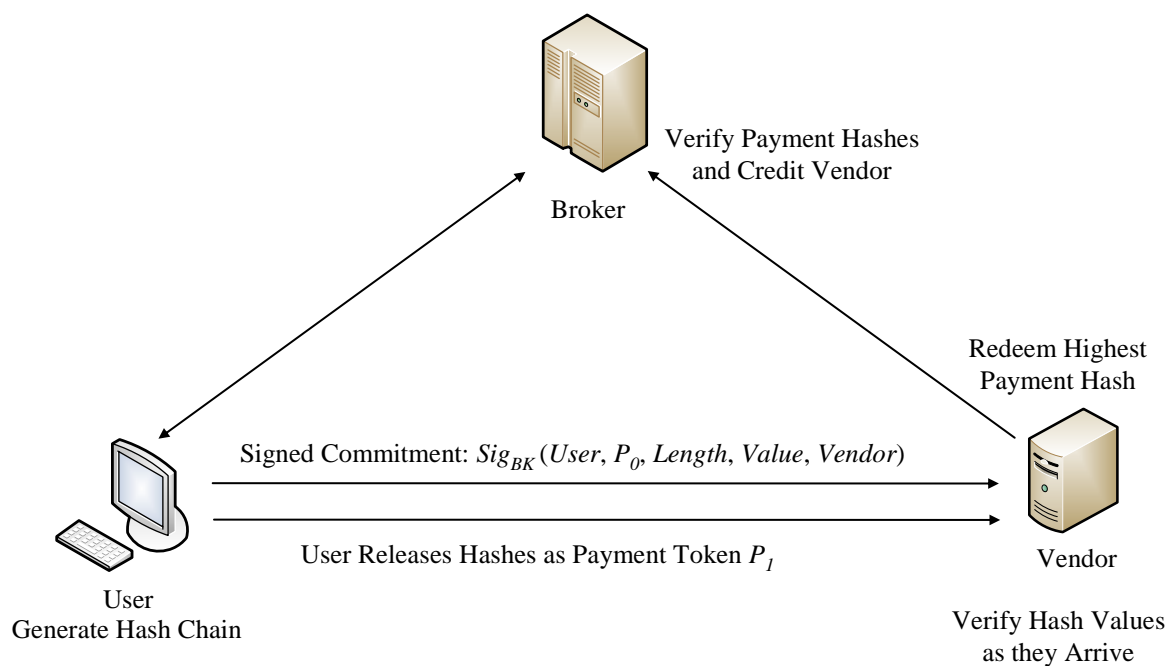


Figure 2.9: Hash tokens being exchanged as credits

When a user wishes to purchase something from a vendor, they generate a hash chain to a value normally more than the user is likely to spend, each hash value represents the same value (normally 1 cent). Unused credits can be discarded as they only represent the users' credit and no value is lost unless they are redeemed. Once the chain has been

generated from W_N to W_0 the user sends the anchor, W_0 , and the length to the Broker who will sign it along with the user, value and the vendor (other values such as expiry date are possible) by digesting the all of these values and signing it with its private key. This signed digest is called the commitment and is sent back to the user. The user can now use the generated hash chain and signed commitment to purchase something from the vendor. This is preformed by first sending the commitment to the vendor who can verify it by getting the Brokers public key. Then the user will release the appropriate amount of hashes for the product or service that they wish to purchase. The released hash is verified offline by the vendor by digesting the received hash until it is equal to W_0 or the last received hash. The payment can be redeemed at a later date by sending the highest received hash along with the commitment to the Broker. This transaction is summarised in Figure 2.9.

Chapter 3

Design

This chapter will outline some of the different designs and iterations of this project and the steps taken from the requirements, initial considerations, ideas and designs to the final design.

3.1 Requirements

The application to be designed needs to be a robust mail filter or mail server that can filter the e-mail, depending on whether or not the e-mail has been paid for. The filtering must be done in real time without delaying the flow of e-mail through the server. The filter must be able to accept and reject e-mails with explanation.

The system needs to use a micropayment technique to process the payments for the e-mails. The payment system needs to be fast, avoid any unnecessary communications and be secure.

3.2 Initial Considerations

Since the application needed to interact with e-mails while they are being sent and received it would be logical to edit the code in the MTA itself. Doing this however, would require either changing the code in an open source MTA, like Sendmail[10], or writing an MTA

completely from the ground up. Both of these would limit the spread of the application with potential users and increase the complexity of installation.

During research for this project an API by Sendmail was found called Sendmail Content Management API or Sendmail Milter[41]. This API allows applications to be built that can access e-mail messages as they are being processed by the MTA. This is done using call backs for all of the SMTP commands that happen on the MTA. This also allows message headers and body to be modified and validated. Applications created with the Milter API can be attached to existing mail servers with a minimal amount of work. The Milter API uses the C programming language for the applications and allows any other commands or functions to be called with in the applications main loop. This API fulfilled the requirements for the mail filter and was chosen for this project.

Although there exists wrappers for the Milter API to allow the use of other programming languages, it was decided to use the C programming language mainly because of the support for C over the other languages. For example the Java version of Milter was last updated in 2005 and a lot of the functionality has been deprecated. C also allows for better memory management which is ideal for server applications, which need to be able to run constantly with out crashing or restarting.

There would be a need to have a server in the system to handle real money payments as well as current account balances for the users and the MTAs. This server does not need to actively do anything, except take requests from the MTAs and take local requests from the web site and process them. The best way to structure the server, in this case, would be to make it a Remote Procedure Call server. Remote Procedure Calls, or RPCs, allows programs to call functions on another computer as if the function was on the local machine. This means that while programming the client, all that will be needed to request information from the server, is to call a local function in the code. This will return the requested information from the server without needing to explicitly program how the client and server interact. There are many different methods of RPC including Java's Java Remote Method Invocation (RMI), Microsoft's .NET Remoting and XML-RPC, which

uses XML to encode its requests.

The type of RPC that was used in this project was XML-RPC. This allowed the server to be a PHP web page on the server thus cutting down on the complexity of the server to being a web server with PHP. There is an open source C library[42] for XML-RPC which allows C applications to call RPC commands, which means that the client can communicate with the server. XML-RPC also offers transmission over SSL[43], which allows for the information to be passed between client and server to be encrypted at the Transport Layer.

User and account data will need to be stored for both the client and server. While it would be less complex to store the information in a file on the machine, sorting and searching the file would be complex and resource heavy. A more efficient way is to use a database, which can be searched, sorted, updated and information inserted anywhere in the data. A database can also be moved to a different machine than the one that the client and server are running on, if more resources are required. It was decided that MySQL from Sun Microsystems would be used because the author was familiar with it and it includes C development libraries as standard with the server installation. MySQL also integrates very well with PHP that the server-side software uses.

The project involves using a micropayment technique for e-mail. It was decided to use the micropayment technique, hash-chains as explained in the previous chapter. As hash-chains requires message digesting as well as public key signing it was necessary to use a cryptographic library. The one that was used is the OpenSSL[44] toolkit. This provides open source implementations of SSL, TLS and many full strength cryptography functions, including message digest and public key signing functions.

3.3 Early Designs

In one of the first designs, pictured in Figure 3.1, the user would use their client to purchase credit from the server. The client would request a hash-chain to the value of the

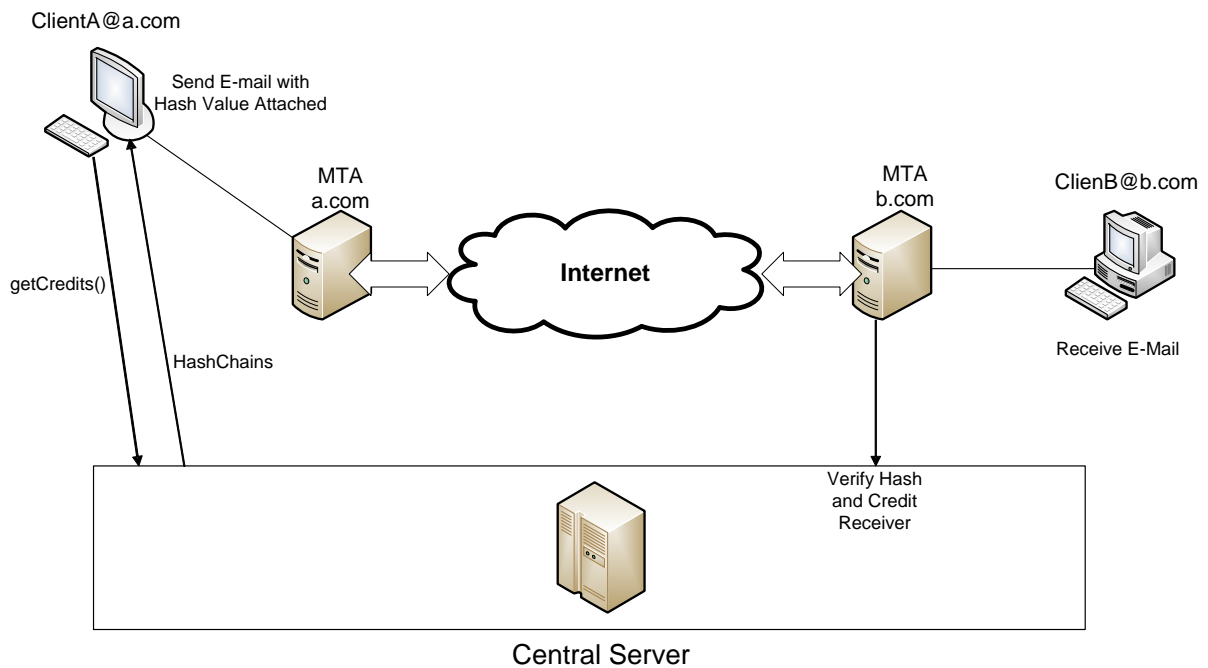


Figure 3.1: One of the first designs of the system

purchase from the payment server. Then when the user sent an e-mail, the client would attach one of their hash tokens to the e-mail in the header and send it to the local MTA. At the senders MTA the e-mail is processed as normal and sent to the receiving MTA. At the receivers MTA the hash token is extracted from the e-mail and sent to the server to be verified and the receivers account is credited, before delivering the e-mail to the recipient.

This design had one major problem along with other design issues that made it harder for users to adopt the system. The main problem is the huge reliance on the central server, it has to create and issue the hash chains to the user's client, it has to verify the hash token at the receivers MTA and also debit the users accounts. A single e-mail has one call to the server and one more call every few e-mails to obtain a hash-chain, with millions of e-mails been sent per day world wide, this would put a huge stress on the server. The design issues that would potentially limit the uptake of the system, is modifying the client, as there are so many different clients, it would be hard to make modifications for them all. This would mean that if someone wanted to use this system, they could only use one

of the clients that had been modified.

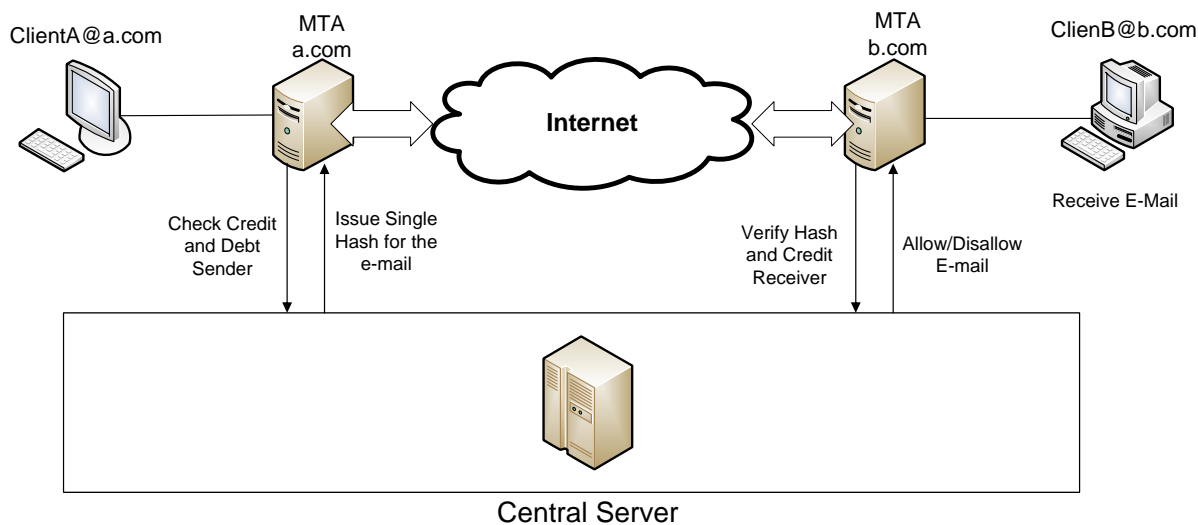


Figure 3.2: Another one of the first designs of the system

Another design, pictured in Figure 3.2, was also considered. In this design the client sends e-mails as normal, but having first credited their account at the central server. Once their e-mail reaches their local MTA, it contacts the central server which checks the sender's credit and if they have enough it debits the sender's account and sends a hash token back to the MTA. The MTA takes the token and attaches it to the e-mail and sends it to the receiving MTA. The receiving MTA sends the token to the server to verify and credits the receiver's account, once the token is verified the e-mail is delivered to the recipient.

This design also has the problem of a central server bottleneck, in this case there are 2 calls for every e-mail that is sent, the server must issue a hash token at one MTA and then verify it at the receiving MTA. This method does however remove the need to modify the client in anyway.

These early designs, while workable, had problems which needed to be addressed by later designs.

3.4 Second Iteration Designs

Taking into account the problems that were encountered with the initial designs the system had to be re-designed. The main problem that occurred in the previous designs was the reliance on the central server to issue and verify the tokens for each e-mail, to remove this reliance it was decided to implement a two layered hash-chains payment scheme. This involves two sets of hash-chains, one set is the users' chains and the other is the MTA chains. The users' chains are used to exchange messages between the user and the MTA, while the MTA chains are only used to exchange messages from MTA to MTA, see Figure 3.3 for an overview of this. To avoid modifying the client to retrieve and store the user's hash chains, the MTA is tasked with getting and holding the hash chains for each user, as well as the hash chains for the other MTAs.

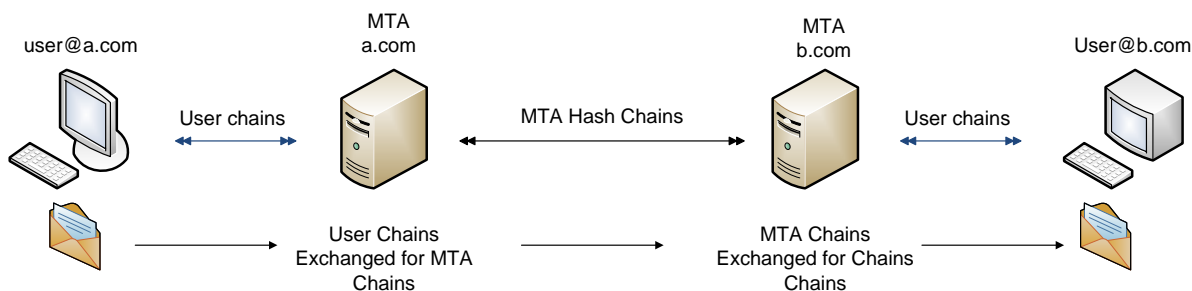


Figure 3.3: Overview of the two layer Hash-Chain payment Scheme

Users, who wish to send e-mails, would have to visit the central server's website and purchase credits using a macropayment solution. These credits would be sent to the user's local MTA in the form of a hash chain and a commitment, which would be stored on the MTA for the user. When the user sends an e-mail, it is sent as normal to the MTA. The MTA would take one of the user's hash tokens as payment for either, another user's hash token if the e-mail is local, or an MTA token if the e-mail is destined for another server. The user's hash chain can only be redeemed at the local MTA. If the e-mail is destined for another server, the MTA purchases a hash chain for that server and attaches one of

the tokens to the e-mail as payment. Once at the other server, the chain can be verified by getting the commitment from the central server, this only has to be done once per hash chain, and checking the token against the commitment. If the token is verified then it is exchanged for a token for the local user and credited to the user.

$\{P_0, \text{Chain Length, User, MTA ID, expiry}\}SK_{\text{Central Server}}$

Figure 3.4: User Commitment Signed by the Central Server

In this design the user's commitment, the commitment that the MTA uses to verify the users hash chains, contains the users address and also the MTA ID that it is valid at. Figure 3.4 shows the users commitment. It is signed by the central server's secret key, so that the commitments integrity and origin can be validated. This is generated at the server, along with the chain, when a user purchases credit then sent to the server via e-mail.

$\{P_0, \text{Chain Length, Sending MTA ID, Receiving MTA ID, expiry}\}SK_{\text{Central Server}}$

Figure 3.5: MTA Commitment Signed by the Central Server

The MTA commitment contains almost the same information as the user one, except that instead of a users address, it has two MTA IDs, sending MTA and receiving MTA shown in Figure 3.5. It is still signed by the central server's secret key. The commitment is generated by the central server, but only after the sending MTA generates the hash chain and sends the central server the chain anchor and length of the chain. Once received the commitment can be generated and stored until requested by the receiving MTA.

This method removes a lot of the server calls that the previous designs had. Each e-mail that is sent to a MTA for the first time has two central server calls, while e-mails sent to the same MTA again will have no central server calls. This design does not modify the client at all, which will allow users to adopt the system easier. The implementation

was started with this design, but was improved into the final design, which is covered in the next section.

3.5 Final Design - CentMail

The final version of the design, also known as CentMail, is an improved version of the previous design. While implementing the previous system, a problem arose when trying to credit a receiver's account on a different server to the sender. The system worked fine, until the MTA hash token that was sent from the sender's MTA could not be exchanged for a user token without generating a hash chain from the MTA to every user. This was unfeasible, both in terms of storage space and extra processing required doing this. It was decided that instead of there being two layers of hash chains, that there would only be one MTA hash chains layer and in the second layer the user's credits would be tracked on their MTA, initially by the CentMail server when a user buys more credit. The MTAs tracking would be backed up by a log of credit history, so that the user could be sure that the MTA was keeping track of the credit legitimately.

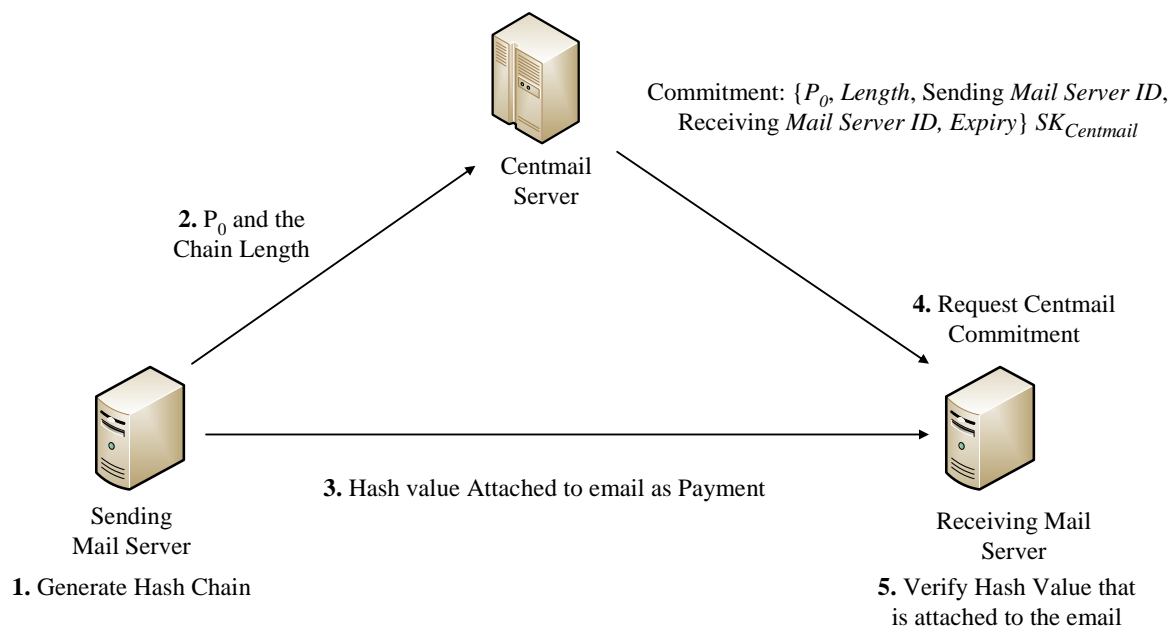


Figure 3.6: An Overview of an E-mail being sent on the final design

When a user sends an e-mail to another user on a different server for the first time, the e-mail is sent to the local MTA, where the users account is debited one credit. The process then follows Figure 3.6:

1. A hash chain is generated on the local MTA by taking an initial random number and, using SHA1, hashes the number to give the first value of the hash chain, P_n . This value is then hashed until the amount of required payment tokens is obtained. The final value is P_0 or the anchor which is used by the receiving MTA for verifying the payments made using this hash chain.
2. Once the local MTA has generated the chain the anchor and the length of the chain is sent to the Centmail Server which will verify that the local MTA has enough credit on account then it will create a commitment this information. The commitment is stored and given to a server when they request it.
3. Now the local MTA has a verifiable hash chain which it can use to send e-mails. So the local MTA will take the next unused hash value and attach it to the e-mail then sends it to the receiving MTA.
4. At the receiving MTA the commitment for the sending server is requested from the Centmail server.
5. The attached hash token is extracted from the e-mail and verified. This is done by hashing the value that was attached to the e-mail. If the resulting value equals P_0 that's in the commitment then the payment is verified.

Once the payment has been verified the receiver is credited with one credit and the e-mail is stored on the receiving MTA, until the recipient connects and retrieves the e-mail. Subsequent e-mails from that server can be verified from the previous hash, until the max value that is stated in the commitment is reached, in which case the new commitment must be retrieved from the Centmail server. Received tokens can be redeemed by sending the last received hash value to the Centmail server.

3.6 Other Consideration

This section will cover some design choices that aims to cover some e-mail options that might cause problems with the CentMail system. Some of these problems cause imbalances, while others cause e-mail to not arrive. They are explained in detail along with their solutions in this section.

3.6.1 Internal Mail

Within large companies and colleges, a lot of the e-mail is sent internally. This includes a large portion of e-mail sent from lecturers and managers to students and co-workers. Using CentMail in this situation would leave the managers and lecturers with a large deficit of credit. To avoid this situation the server administrators can choose to exclude e-mail with in a domain from being processed by CentMail, while still processing mail to and from other domains. Choosing to exclude a domain has a trade off, reducing the effectiveness of the system if one of the accounts with in the domain should get compromised.

3.6.2 Mailing Lists

E-mail users are able to sign up to mailing lists to be notified of new products, events and services, as well as discussion lists on various topics. Mailing lists have a single e-mail address that when an e-mail is sent to it, if the user has permission, forwards the e-mail to all subscribed users. When used with the CentMail system this would cause huge imbalances for the mailing list account, because for every one e-mail it receives, it has to send e-mails to every list member.

To allow mailing lists to still function under the CentMail system, a double 'opt-in' address exclusion is used. The first 'opt in' being that a user has to go to a mailing list website and sign up for the list. The second 'opt in' is an e-mail that is sent to the user from the CentMail server confirming that they want to be on this mailing list. Once the user has responded to this e-mail, all traffic between the users and the mailing list address

will not be charged by the CentMail system.

3.6.3 Sending Mail from non-CentMail MTAs

Users who don't have the CentMail system installed and send e-mail to a server that does have CentMail installed will have their message rejected because it has no payment token. Initially the system could just accept e-mails with no tokens but make the message go through another spam filter with an already high spam score.

A better way is to use manual stamps, the sending user signs up to an account on the central server, deposits some money into this account, then enters the e-mail address they wish to send an e-mail to. This will generate a manual payment token that can be attached to the e-mail and sent. Using this method, users who are not on a CentMail server can get some of the features without having to move server.

Chapter 4

Implementation

This chapter will cover how the system was implemented. It will show how each part interacts with each other and some sample code to explain how the Milter API and other libraries work with the system. Each part will be covered in its own section with the final section explaining how the parts interact.

4.1 Main Application

The main application is based around the Milter API with calls to the OpenSSL library for message digesting and public key signing, the XML-RPC library to communicate with the central server and the MySQL library to store data such as account details and hash chains.

4.1.1 Milter API

The Milter API is the core of the system; it allows access to the e-mails as they are being processed by the mail server via callbacks to each SMTP command. The callbacks are similar to methods that can be overridden with user created code. Each callback is called with arguments that are related to it e.g. the callback `xxfi_envfrom`, which is called at the beginning of each e-mail, contains the senders address as an argument. Table 4.1

shows the different callbacks that the Milter API has and what arguments they offer the developer. By implementing callbacks information about the e-mail can be determined and changed if required.

Callback Function	Description	Arguments
xxfi_connect	Connection Information	Senders Hostname and IP Address
xxfi_helo	SMTP HELO callback	Senders Hostname
xxfi_envfrom	Called at the start of each mail	Senders e-mail address
xxfi_envrcpt	Called for each Recipient	Recipients e-mail address
xxfi_data	SMTP DATA callback	None
xxfi_header	Called for each Header	Header name and value
xxfi_eoh	End of headers	None
xxfi_body	Called for message body	The contents and length of the Body
xxfi_eom	End of Message	None

Table 4.1: Milter API callback functions that can be

Each callback must be returned a value, to tell the mail server what to do with the e-mail e.g. continue processing it with SMFIS_CONTINUE or reject it with SMFIS_REJECT. Table 4.2 shows all the callback return values.

Callback Return Value	Description
SMFIS_CONTINUE	Continue processing the message
SMFIS_REJECT	Reject the current message
SMFIS_DISCARD	Discard the current message
SMFIS_ACCEPT	Accept the message without further filtering
SMFIS_TEMPFAIL	Temporarily Fail the Message

Table 4.2: Values that can be returned from callback functions

To initialise the Milter API the application is required to call 3 functions to configure settings and start the Milter service. Figure 4.1 shows how the Milter is initialised in this project. The application does not need to implement all of the callback commands that are available; the first required start up command tells the Milter what callbacks are implemented in the program and also, what message modification it will be doing during the running of the application. This information is set using the smfi_regisiter function, which takes a struct as an argument containing the function names for the callbacks that

appear in the application, or null. The MTA communicates with the Milter API via a socket which is initialised using the function `smfi_setconn` with the port and hostname. This caused some trouble at the beginning, as the normal way of writing a port and hostname is to put the port after a colon after the hostname, e.g. `host:8080`, where as the Milter API accepts the port first with the hostname following after an `@`.

```
//struct to register callbacks and flags, NULLs are not implemented
struct smfiDesc smfilter =
{
    "CentMail Milter", /* filter name */
    SMFI_VERSION, /* version code -- do not change */
    SMFIF_ADDHDRS, /* flags */
    xxfi_connect, /* SMTP Connect */
    xxfi_helo, /* SMTP HELO command filter */
    xxfi_envfrom, /* envelope sender filter */
    xxfi_envrcpt, /* envelope recipient filter */
    xxfi_header, /* header */
    NULL, /* End of Header */
    NULL, /* Body */
    xxfi_eom, /* EOM */
    NULL, /* Abort */
    xxfi_close, /* cleanup or close */
    NULL, /* unknown */
    NULL, /* data */
    xxfi_negotiate /* Once, at the start of each SMTP connection */
};

int main(int argc, char **argv;)
{
    smfi_setconn("inet:5145@localhost"); //set address and port
    if (smfi_register(smfilter) == MI_FAILURE)
    {
        fprintf(stderr, "smfi_register failed\n");
    }
    return smfi_main(); //start the event loop
}
```

Figure 4.1: Initialising the Milter and starting the main event loop

Once the callbacks and the port have been set `smfi_main` is called, which starts the Milters event loop that triggers the callbacks when they are received from the MTA. After that all the application code takes place within the callback commands.

As the Milter API can process multiple e-mail messages at the same time, storing per e-mail information can not be done using global variables. Instead private memory

must be initialised using the `smfi_setpriv` function within the API. This allows a chunk of memory to be related to the current message being processed. This area of memory can be used to store values between the different callbacks, for instances in this system the memory is used to store the to and from addresses as well as the amount of recipients and the retrieved hash token. This information is arranged in a struct within the memory. See Figure 4.2 to see how the private memory is set up for this system, `ctx` is the unique identifier for the current e-mail and is passed in each call. The memory can be accessed by using the `smfi_getpriv` function, once it has been set.

```

//Private Memory Struct
struct mlfiPriv
{
    int amount;
    char *from;
    char **to;
    char hash[41];
    int n;
};

//function to setup the memory
int setupMemory() {
    struct mlfiPriv *priv;
    priv = malloc(sizeof *priv);
    if (priv == NULL) {
        return 0; //return 0 on fail
    }
    memset(priv, '\0', sizeof *priv);
    smfi_setpriv(ctx, priv);
    return 1; //return 1 on success
}

//define to get currently set private memory
#define MLFIPRIV ((struct mlfiPriv *) smfi_getpriv(ctx))

```

Figure 4.2: Private memory being set up for the system

The callbacks that were needed for this system are `xxfi_envfrom`, `xxfi_envrcpt`, `xxfi_header` and `xxfi_eom`. To get the sending users address and store it in the private memory, `xxfi_envfrom` is used, and `xxfi_envrcpt` does the same for the receiving address. If an incoming e-mail is coming from an external domain then the header is processed to retrieve the payment token and store it in the private memory to be processed in a later callback. The bulk of the code for the system lies within the `xxfi_eom`, end of message, callback. This is mainly because it is the only callback that you can edit the headers and body

from.

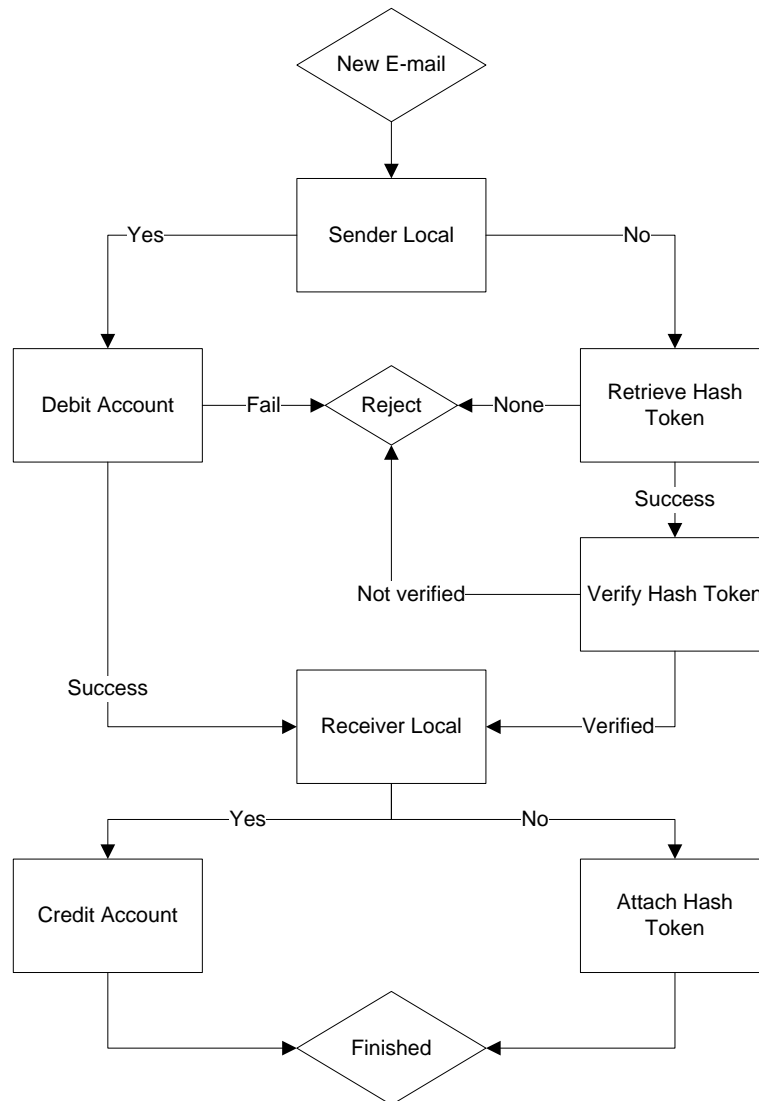


Figure 4.3: The flow of the code within the eom callback

The flow of the code within the eom, as seen in Figure 4.3. It begins by debiting the sending account, checking first if it is local or remote. If it is local then the account is debited if there is enough credit. If it is remote the token that was stored earlier is verified, by checking it against the commitment which, if necessary, is retrieved from the central server.

Once the sender has been debited the receiver can be credited. If the receiver is a local account then the account is credited. If it is on a different server a hash token is

attached to the e-mail using `smfi_addheader` from an existing hash chain or by generating a new one.

4.1.2 OpenSSL Library

The OpenSSL library provides a C language library for cryptography and SSL. In this project we used OpenSSL to provide the message digest function, SHA1, to create and verify the hash chains, as well as the public key cryptography function to sign the commitment. While the public key cryptography could be used as is with out much extra code, the message digest function returned the digest as byte array (unsigned char* in C). To interact with the PHP version of the SHA1 function the digest needed to be in a hexadecimal string. To allow for this a function was created to return the message digest as a string, while also allowing strings to be entered to be digested.

Other then the small digesting problem, the OpenSSL library provided the necessary cryptography features to implement the hash chains processing with signed commitments simply and with out much extra code.

4.1.3 XML-RPC Library

The XML-RPC library proved to be code intensive, when implementing the server calls as local functions. In each method the connection had to be initialised, then the function called, then results parsed and finally the connection closed, see Figure 4.4 for a sample function. The reason the connection had to be made for each function is because of the threaded nature of the Milter API and this also reduces unnecessary connections to the server, which will be handling hundreds of connections per second already. This means putting the connection initialisation at the start of an e-mail would have the client connect, even if it was not going to send anything to the server, instead the connection is only made if the client wants to request something from the server. The connection is established quickly and with only three lines of code makes it faster to implement, then

setting up a socket to the central server.

```
//Given an account name and an mta return the
//amount of amount of credits the user has
int getCredits(const char *accout, const char *mta)
{
    //init the connection
    xmlrpc_env env;
    xmlrpc_value *result;
    xmlrpc_client * clientP;
    xmlrpc_env_init(&env);
    xmlrpc_client_setup_global_const(&env);
    xmlrpc_client_create(&env, 0, NAME, VERSION, NULL, 0, &clientP);

    //make the xml-rpc function call
    xmlrpc_client_call2f(&env, clientP, SERVER_URL, "get_user_credits",
                        &result, "(ss)", name, mta);

    //parse the result
    int output;
    xmlrpc_parse_value(&env, result, "i", &output);

    //shutdown the connection
    xmlrpc_DECREF(result);
    xmlrpc_env_clean(&env);
    xmlrpc_client_destroy(clientP);
    xmlrpc_client_teardown_global_const();

    //return the result
    return output;
}
```

Figure 4.4: A XML-RPC function from the system

4.1.4 MySQL Library

The C library to interact with MySQL came as default with the MySQL server and can be included into applications by including 'mysql.h'. The library includes all the necessary functions to connect to and query the database. When using select queries, queries that get data from the database, the data is returned as a MYSQL_RES type which can be used to get a MYSQL_ROW type, which is an array of the data that was requested. Except for the types that must be used to get the results from select queries, the MySQL library provided fast and easy access to a database, local or remote, to store user account information, commitments and payment tokens.

4.2 The Central Server

The central server consists of two parts, the XML-RPC server and the website for user interaction. Data is provided to both services via a common MySQL server.

4.2.1 XML-RPC server

The XML-RPC server was implemented using PHP with the XML-RPC library that is included with PHP. An XML-RPC server is created by calling the function `xmlrpc_server_create()` and then registering the methods that are to be called from the client. Any PHP method can be registered as an XML-RPC call by using the PHP function `xmlrpc_server_register_method()`. This allowed the methods to be created as functions, which could be tested without having to use XML-RPC calls initially. Data was obtained and stored using a different MySQL server than the client uses. The database was interacted with using the built in MySQL interface that PHP has. This server provides functions to the client to get the server's public key, as well as one to create and get commitments.

4.2.2 User Interface

For the purposes of this project, only a small user interface site was created to allow users to check their current balance and add more credit. The site used PHP to interact with the MySQL server database. The website consisted of two input boxes, e-mail address and amount of credit the user wishes to purchase, and a submit button. By just entering the e-mail address box, the current amount of credits is returned, but by entering both boxes that account would be credited by the amount entered.

4.3 Databases

There are two MySQL databases being used in this project; one is being used with the Militer Application and one is being used on the server. The database for the Militer

application has three tables in it. The first is to store the account details for every user that this Militer is tracking, which is the e-mail address and the amount of credit that they have. To protect from database manipulation a signature is created by combining this information and digesting it with SHA1. This signature will provide an indication of whether or not the information has been changed. The next table that is in the Militer database is for credit transfer logs for keeping track of all the users credit flow. This log is for the users' benefit, so that they can be sure that their MTA is not overcharging or taking credits from them. The final table in the Militer stores local copies of commitments. Each entry in the table needs to store the commitment signature, the chain anchor as obtained from the commitment, the overall length of the chain from the commitment, the MTA it's valid at, the last hash token that was used and the last number of the chain that was used. This information is used to store and verify incoming tokens and attach outgoing tokens.

The database on the server side also has accounts and commitment, as well as a table that tracks the current credits of all MTAs supporting CentMail. The accounts table on the server is tracking the amount deposited by each user and the current credits of all the users on all of the MTAs using the software. To reduce the search time that it would take to find a user by their e-mail address, another field is added to the table which is the MTA name. This way the SELECT statement can search through a smaller subset of results. The commitment table is the same as the commitment table on the Militer, except that it stores both the MTA that is valid from, as well as the MTA that it is valid to. The third table stores the amount of credit that the MTA has overall, this credit is increased when the MTA redeems a hash chain and is reduced when one its hash chains is redeemed.

4.4 Interaction

The system as a whole consists of three parts communicating with each other, the Milter, the server and the MTA that the Milter is running with. All interactions between them take place as some kind of method call, call back or RPC, so the interactions can be seen as calls from the different services. Figure 4.5 shows the interaction that takes place when an e-mail is sent from a local account to a local account. There is no need to connect the central server for local mail, as the credit is just passed from one account to the other. Once the message has being accepted, user B can retrieve it at any time.

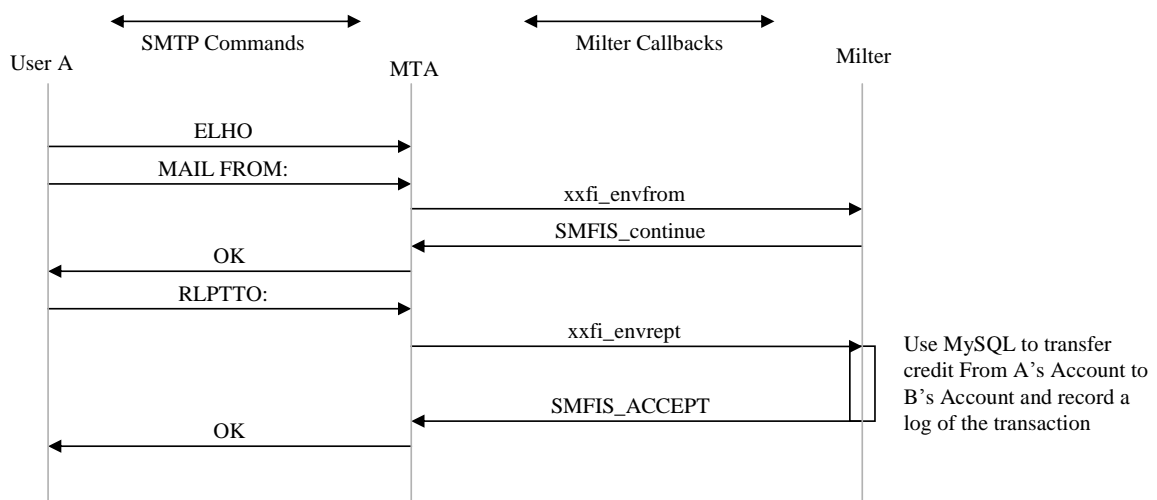


Figure 4.5: Communication between services for an internal e-mail

Figure 4.6 shows the method calls and events that happen during an e-mail being sent from 'User A' on 'MTA A' to 'User B' on 'MTA B'. In this case there are RPC calls to the central server, but only when a new hash chain is created. Most of the communication takes place between the MTA and the Milter application. This figure does not show the SMTP commands that take place for the Milter callbacks to happen.

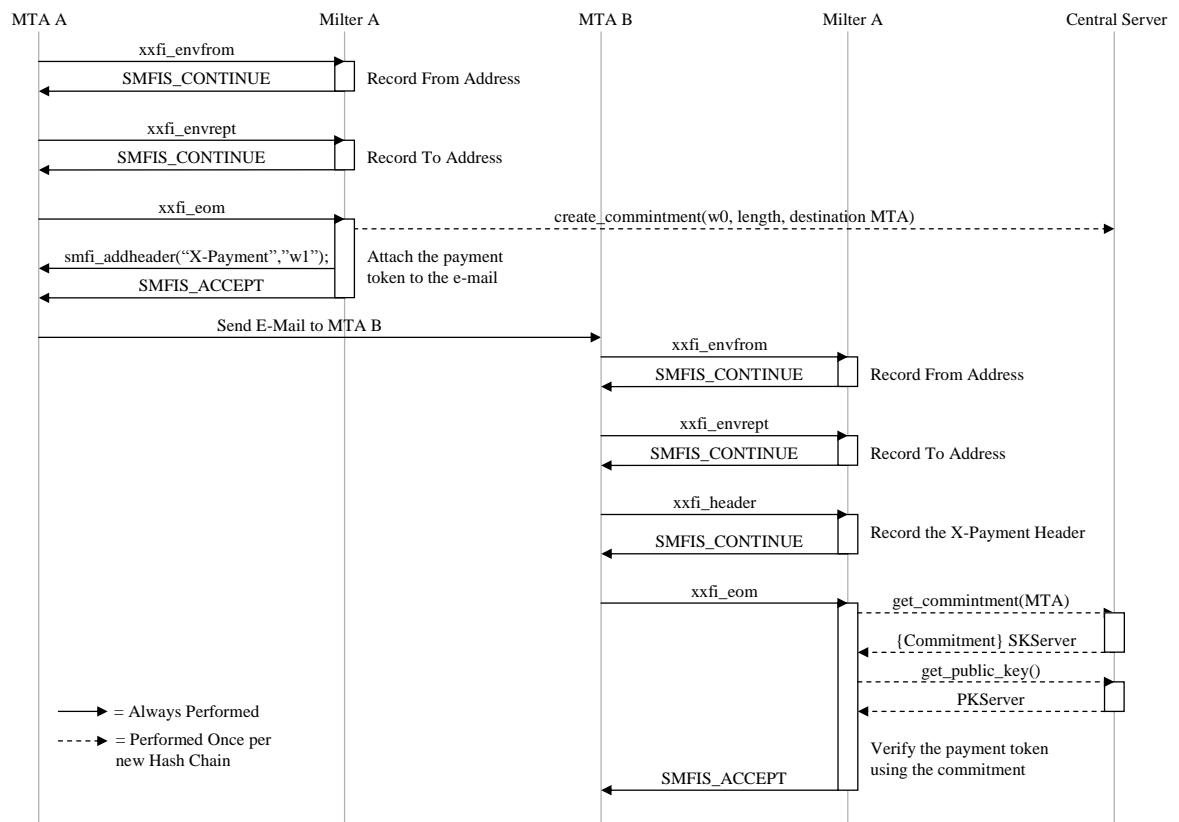


Figure 4.6: Communication between services for an external e-mail

Chapter 5

Evaluation

This chapter will describe the way in which the software was tested to make sure that it worked and the user trials undertaken. The findings of these trials will be reported.

5.1 Testing

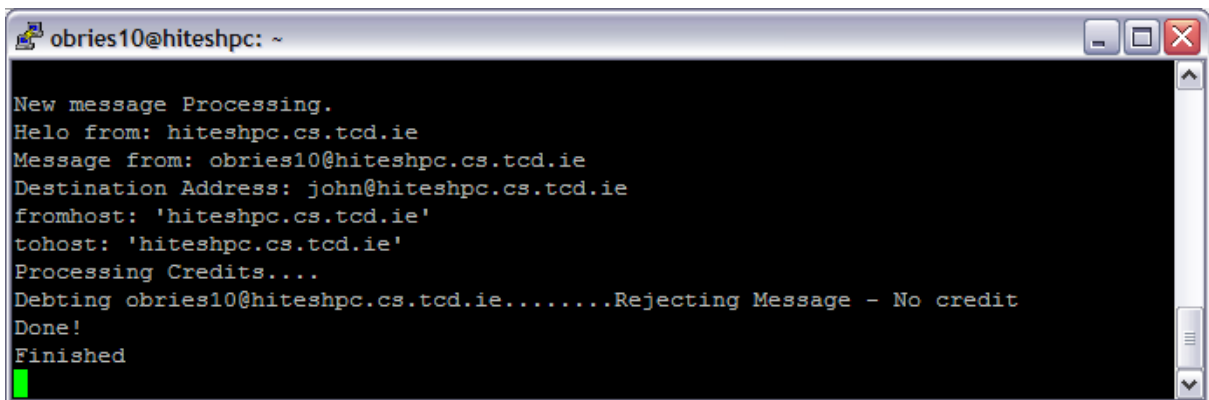
To be sure that the application was running correctly, three sets of tests were carried out to test the different features. For the purposes of testing the software, two machines were set up with Ubuntu server edition operation system with e-mail server and LAMP (Linux, Apache, MySQL and PHP) server picked at install; this installed the postfix MTA software as well as PHP, MySQL and a web server. After installation e-mail could be used without any further configuration. The required libraries were then installed using Ubuntu's built in package management system, apt-get. One of the machines was designated the server and the server side software, the XML-RPC server and user interface, was installed. Both machines were configured with the Milter application. Once everything was running the tests were run one at a time. The first set of tests concerned situations where the sending account had no credit and the second set of tests, where the sending account did have credit. The final set of tests attempted to break or trick the system. For the purposes of these tests the machines are called A, the machine that is designated the server, and B is

the other one.

5.1.1 No Credit Tests

This set of tests contained 2 tests with the sending account having no credit. The results are determined by looking at the account details within the MySQL database, as well as the feedback that the application displays when running.

Test 1 of this set involved using an account on machine A to send an e-mail to another account on A. Both accounts had no credit, so the expected result would be for the e-mail to fail. When the e-mail was sent the application output confirmed that the message had failed due to lack of credit. Figure 5.1 shows the output. This test was a success.

A screenshot of a terminal window titled 'obries10@hiteshpc: ~'. The terminal displays the following text:

```
New message Processing.  
Helo from: hiteshpc.cs.tcd.ie  
Message from: obries10@hiteshpc.cs.tcd.ie  
Destination Address: john@hiteshpc.cs.tcd.ie  
fromhost: 'hiteshpc.cs.tcd.ie'  
tohost: 'hiteshpc.cs.tcd.ie'  
Processing Credits...  
Debting obries10@hiteshpc.cs.tcd.ie.....Rejecting Message - No credit  
Done!  
Finished
```

Figure 5.1: The application output for sending an e-mail with no credit

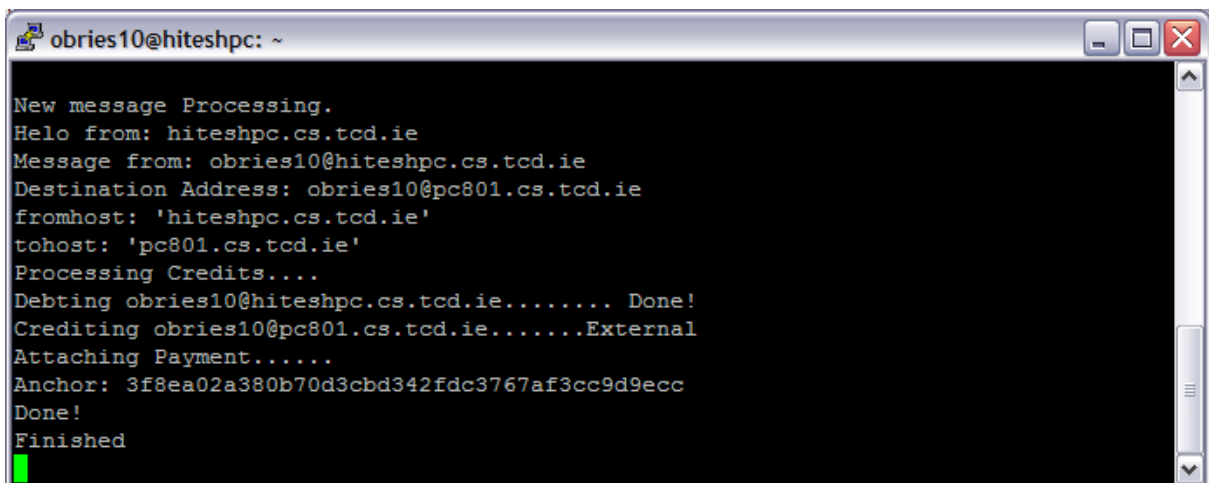
Test 2 was the same as test one, except that the destination was on B instead of been locally on A. The same result was expected and gotten, the message failed on debiting the sending account, so the message was rejected before it left the local server. This test was a success.

5.1.2 Credit Tests

In this set of tests, the sending accounts were given a large supply of credits and were topped up if they ran out. Again the results were determined by monitoring the databases and the output of the application.

Test 1 of this set, had a similar set up as test 1 in the previous set, an e-mail is sent from A to A, but this time it is expected to succeed. After sending the e-mail, the initial response is that the e-mail is sent by looking at the output from the application. This is then confirmed by seeing that the senders account has been debited and the receivers account has been credited on the database as expected. This test was a success.

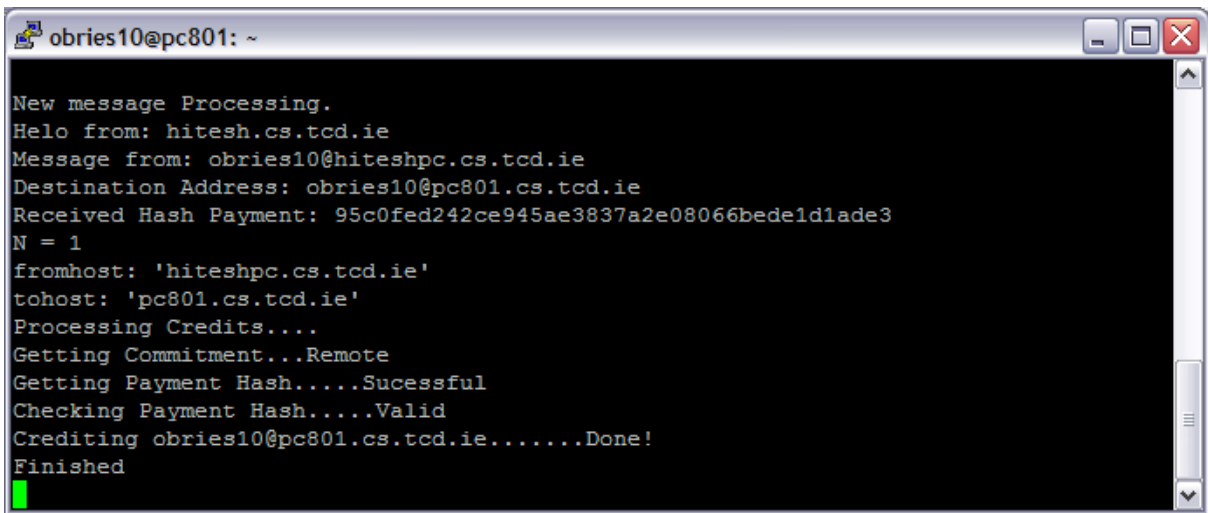
Test 2 is the first test that we expect commitments and hash chains to become involved. This test sent an e-mail from A to B. It was expected that the e-mail would be delivered successfully to B, with a new hash chain being created for the MTA to MTA payment. When the e-mail was sent the application output on A, Figure 5.2 shows that the sender has been debited and when it goes to credit the receiver sees that it is external and creates the hash chain. When the e-mail gets to B the application output, Figure 5.3 shows that the hash token is extracted from the e-mail and verified, both of which are successful and the receiver is then credited. By checking the databases, it can be seen that the sender has spent one credit and the receiver has gained one credit. The hash chain commitment can also be seen, along with the first token that was used. This test was a success.

A screenshot of a terminal window titled 'obries10@hiteshpc: ~'. The terminal displays the following text:

```
New message Processing.  
Helo from: hiteshpc.cs.tcd.ie  
Message from: obries10@hiteshpc.cs.tcd.ie  
Destination Address: obries10@pc801.cs.tcd.ie  
fromhost: 'hiteshpc.cs.tcd.ie'  
tohost: 'pc801.cs.tcd.ie'  
Processing Credits....  
Debting obries10@hiteshpc.cs.tcd.ie..... Done!  
Crediting obries10@pc801.cs.tcd.ie.....External  
Attaching Payment.....  
Anchor: 3f8ea02a380b70d3cbd342fdc3767af3cc9d9ecc  
Done!  
Finished
```

Figure 5.2: Application output from sending an e-mail to another server

Test 3 sends an e-mail from A to B using a different sending account than test 2. This test makes sure that the hash chain that was generated in the last test is used by all accounts, and not just the one that created it. It was expected that the same hash

A terminal window titled 'obries10@pc801: ~' with standard window controls. The terminal displays the following text:

```
New message Processing.  
Helo from: hitesh.cs.tcd.ie  
Message from: obries10@hiteshpc.cs.tcd.ie  
Destination Address: obries10@pc801.cs.tcd.ie  
Received Hash Payment: 95c0fed242ce945ae3837a2e08066bede1dlade3  
N = 1  
fromhost: 'hiteshpc.cs.tcd.ie'  
tohost: 'pc801.cs.tcd.ie'  
Processing Credits...  
Getting Commitment...Remote  
Getting Payment Hash....Sucessful  
Checking Payment Hash....Valid  
Crediting obries10@pc801.cs.tcd.ie.....Done!  
Finished
```

Figure 5.3: Application output from receiving an e-mail from another server

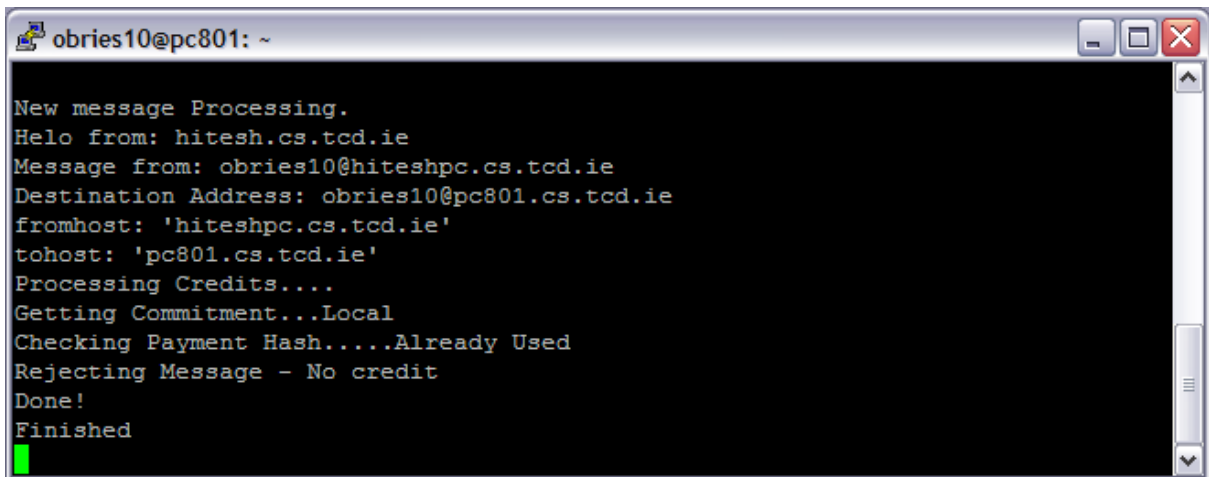
chain would be used and the e-mail sent successfully. This was confirmed by both the application output and the database data which show that A used the same, local, hash chain to send the e-mail. This test was a success.

5.1.3 Other Tests

This set of tests attempted to break the system by resending the same hash token and using a hash token from another MTA.

Test 1 sends the same token twice, the first time legitimately. It was expected that the first e-mail would succeed, while the second e-mail would fail. This is confirmed by the application output, Figure 5.4, which shows that the hash token has already been used and rejects it. This test simulates an e-mail being read on route by packet sniffing and the payment token being copied out of it and reused. This test was a success.

Test 2 takes a token from the chain of one server and tries to use it to send an e-mail to a different server. As the chain is not associated with the receiving server, it is expected that this will fail. By checking the output from the application, it can be seen that the Milter labels the hash token as 'invalid' and rejects it at the receiving server. This test was a success.

A terminal window titled 'obries10@pc801: ~' with standard window controls. The output text is as follows:

```
New message Processing.  
Helo from: hitesh.cs.tcd.ie  
Message from: obries10@hiteshpc.cs.tcd.ie  
Destination Address: obries10@pc801.cs.tcd.ie  
fromhost: 'hiteshpc.cs.tcd.ie'  
tohost: 'pc801.cs.tcd.ie'  
Processing Credits...  
Getting Commitment...Local  
Checking Payment Hash....Already Used  
Rejecting Message - No credit  
Done!  
Finished
```

Figure 5.4: Application output from receiving the same payment token twice

5.2 User Trials

Once the system passed all the tests and was considered stable, two user trials were carried out. It was hoped that these trials would show that normal users remain cost-neutral. The trials would also allow the system to be tested in a live e-mail environment.

5.2.1 Real User System Trial

The first trial that was set up invited peers to take part. The trial gave each user a new e-mail address pre-topped up with 10 credits with the option to topup to 40 credits. The users were asked to use the e-mail account as they would their own and send e-mails to each other. E-mails to and from accounts outside the server were ignored. The trial lasted for 2 weeks and 15 users took part with a 16th account set up to act as a spammer. This account sent spam e-mails to the other users, until it used up all of its credits.

At the end of the trial, over 100 e-mail messages had been sent through the system to the different users. The balance for each user was calculated by taking their current credit balance and taking away the amount that they topped up with during the trial. This gave each user a positive number if they ended up with more credits than they started with, a zero if they remained the same or a negative number if they finished with less credits

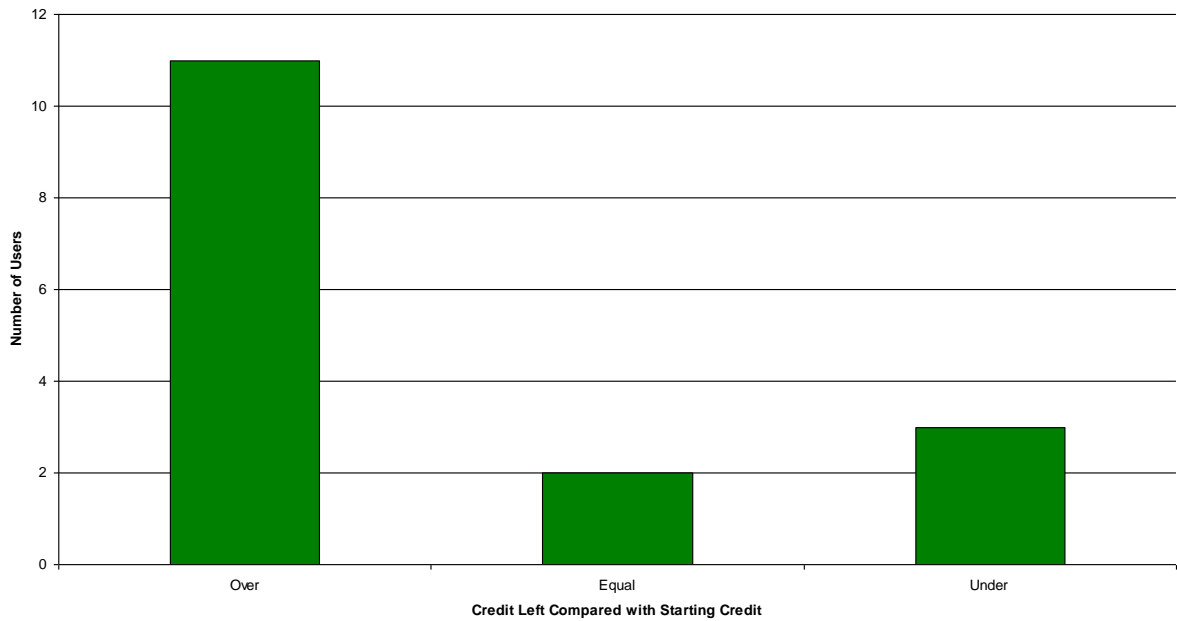


Figure 5.5: Results from the Real User System Trial

then they started with. The amount of users that ended up at each was graphed as in figure 5.5. As can be seen over 80% of the users finished the trial with the same amount of credit or more, while the spammer’s account ended up as one of the accounts with less than their starting amount. As no one replied to the spammer’s e-mails, the only way that the spammer could keep sending messages was by topping up their account with more credit.

5.2.2 Live System User Trial

The second trial carried out used an existing mail server and users. The system administrator of the mail server set it up so that existing users could opt into the trial by routing their mail to another server. The trial was carried out as a blind trial, where the user would not notice any changes to their e-mail. Instead of rejecting e-mails when the user had no credits, they would be allowed to go into a negative balance. This would allow their overall balance to be monitored and recorded like the previous trial.

Unfortunately it turned out that outgoing mail could not be routed through the sys-

tem and thus all of the account had a positive balance equal to the amount of e-mails they received making the results obtained from this trial useless. This trial did however demonstrate that the application could be run on a live server running a different operating system, Solaris, successfully.

Chapter 6

Conclusion

This chapter will start by summarizing the CentMail system, what it can do, the key design features, the results obtained from user trials and how it reduces and discourages spam. It will then discuss some possible future work that can be undertaken.

6.1 CentMail - A micropayment system for e-mail

Users send spam e-mails because they make money from doing it. Millions of e-mails cost nothing to send, so if one person responds to a spam message the spammer is in profit. CentMail removes the profit from sending spam by adding a cost to send e-mail. At the same time, normal users will not have to pay for e-mail as the system is cost neutral for them. This is achieved by charging a user to send an e-mail, while paying a user the same amount to receive an e-mail.

The system was successfully implemented by using a micropayment system which makes use of hash chains. The system works on two tiers, the first being the hash chain exchange between different servers and the second is the credits that the users have, which is monitored and managed by each MTA locally. Overall this cuts down on the reliance on communication with central servers, the system only needs to contact the central server to create or obtain the commitment for the hash chains.

The the system user trials revealed that most normal user maintain the same amount of credit or gain more. It also showed that users who send excess e-mails, such as spammers, would have to pay for their e-mails, if they receive no replies. The tests preformed on the system showed that simply reusing a payment token or using a token from another server will not work, and that mail that is not paid for, will be rejected at the sending server, cutting down on the amount of bandwidth that is used.

CentMail can cut down on the amount of spam and discourage spammers from sending spam to begin with by using some of the features mention above. The main reason that spammers will be discouraged is by the charge to send the e-mail, which for them can become very large. Since they can no longer be assured a profit from their spam, they will not send e-mails to servers that have CentMail. If they hijack an account, the amount of damage that can be done is limited to the amount of credit that account has currently. Trying to reuse credits will also fail as illustrated by the tests.

6.2 Future Work

The software at the moment is capable of running on a live e-mail server, as was seen in the second of the user trials. As it is a C application and contains a lot of pointers, there is a small bit of memory leaks that need to be plugged. Other then that, the system administrator should be able to select various options to suit their servers, like using a file instead of MySQL. The install process needs to be simplified and automated to make it easier to make a server, CentMail enabled.

On the central server, the main thing that has to be done is to integrate a macropayment system and a proper user interface. Users should be able to graphical see what their current credit status is, as well as past spending.

Currently the Milter API covers the two main types of MTA software. These two types of MTA cover a large proportion of the MTAs on the internet, but do not cover them all. The application can be written as a proxy that will route mail through itself

before going to the MTA. Although this makes it more complex to install, it means that it will have a high compatibility rate than just having the Milter on its own.

Once the system is completely finished the next step would be to release it to the public, while maintaining control of the central server.

6.3 Final Word

We have successfully implemented a micropayment system that can be used for e-mails. This system discourages spam by adding a cost to the spammer, who would otherwise be sending millions of e-mails without any cost. It is easy to install and requires no modification to existing servers or clients.

Appendix A

Abbreviations

Short Term	Expanded Term
DNS	Domain Name System
ASCII	Dynamic Host Configuration Protocol
IP	Internet Protocol
ISP	Internet Service Provider
SMTP	Simple Mail Transfer Protocol
MTA	Mail Transfer Agent
MUA	Mail User Agent
POP3	Post Office Protocol version 3
IMAP	Internet Message Access Protocol
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
HELO	Hello command for MTA
EHLO	Extended Hello command for MTA
FQDN	Fully Qualified Domain Name
MX record	Mail Exchanger record
DKIM	DomainKeys Identified Mail
CPU	Computer Processing Unit

Short Term	Expanded Term
SHA-1	Secure Hash Algorithm
MD5	Message Digest version 5
TTP	Trusted Third Party
PM	Post Master
HTTP	HyperText Transfer Protocol
IDN	Internationalised Domain Names
RPC	Remote Procedure Call
SSL	Secure Sockets Layer

Bibliography

- [1] Tom Van Vleck. The ibm 7094 and ctss, Sept 2004. URL <http://www.multicians.org/thvv/7094.html>.
- [2] J. Klensin. Simple Mail Transfer Protocol. RFC 2821 (Proposed Standard), April 2001. URL <http://www.ietf.org/rfc/rfc2821.txt>.
- [3] J. Myers and M. Rose. Post Office Protocol - Version 3. RFC 1939 (Standard), May 1996. URL <http://www.ietf.org/rfc/rfc1939.txt>. Updated by RFCs 1957, 2449.
- [4] M. Crispin. INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1. RFC 3501 (Proposed Standard), March 2003. URL <http://www.ietf.org/rfc/rfc3501.txt>. Updated by RFCs 4466, 4469, 4551, 5032, 5182.
- [5] Microsoft Outlook. URL <http://www.microsoft.com/outlook/>.
- [6] Mozillas Thunderbird 2. URL <http://www.mozilla.com/thunderbird/>.
- [7] Google Mail, Gmail. URL <http://www.gmail.com/>.
- [8] Windows Live Hotmail. URL <http://www.hotmail.com/>.
- [9] Mail Server Statistics. URL <http://www.mailradar.com/mailstat/>.
- [10] Sendmail. URL <http://www.sendmail.com/>.

- [11] Postfix Sendmail program. URL <http://www.postfix.org/>.
- [12] Courier-IMAP. URL <http://www.courier-mta.org/imap/>.
- [13] Dovecot Secure IMAP and POP3 server. URL <http://www.dovecot.org/>.
- [14] J. Postel. Simple Mail Transfer Protocol. RFC 821 (Standard), August 1982. URL <http://www.ietf.org/rfc/rfc821.txt>. Obsoleted by RFC 2821.
- [15] J.B. Postel, J. Reynolds, Network Information Center, and SRI International. Telnet Protocol Specification. 1983.
- [16] Spam statistics and facts, . URL <http://www.spamlaws.com/spam-stats.html>.
- [17] M. Prince, B. Dahl, L. Holloway, A. Keller, and E. Langheinrich. Understanding How Spammers Steal Your E-Mail Address: An Analysis of the First Six Months of Data from Project Honey Pot. *Second Conference on Email and Anti-Spam*, 2005.
- [18] SpamCop.net, . URL <http://www.spamcop.net/>.
- [19] I. Miszalska, W. Zabierowski, and A. Napieralski. Selected Methods of Spam Filtering in Email. *CAD Systems in Microelectronics, 2007. CADSM'07. 9th International Conference-The Experience of Designing and Applications of*, pages 507–513, 2007.
- [20] T. Salmi. Youve got email... again! Protecting ones mailbox from spam with automatic filtering.
- [21] Paul Graham. A Plan for Spam. URL <http://www.paulgraham.com/spam.html>.
- [22] F.D. Garcia, J.H. Hoepman, and J. van Nieuwenhuizen. SPAM FILTER ANALYSIS.
- [23] A. Treviño and JJ Ekstrom. Spam Filtering Through Header Relay Detection.
- [24] Apache SpanAssassin Project, . URL <http://spamassassin.apache.org/>.
- [25] policyd-weight - A policy daemon for Postfix. URL <http://www.policyd-weight.org/>.

- [26] E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton, and M. Thomas. DomainKeys Identified Mail (DKIM) Signatures. RFC 4871 (Proposed Standard), May 2007. URL <http://www.ietf.org/rfc/rfc4871.txt>.
- [27] Cisco Systems Inc. URL <http://www.cisco.com/>.
- [28] M. Delany. Domain-Based Email Authentication Using Public Keys Advertised in the DNS (DomainKeys). RFC 4870 (Historic), May 2007. URL <http://www.ietf.org/rfc/rfc4870.txt>. Obsoleted by RFC 4871.
- [29] Yahoo! URL <http://www.yahoo.com/>.
- [30] A. Back et al. Hashcash-A Denial of Service Counter-Measure. URL: <http://www.hashcash.org/papers/hashcash.pdf>. August, 2002.
- [31] PUB FIPS. 180-2:Secure Hash Standard,. *US Department of Commerce, Technology Administration, National Institute of Standards and Technology*, 2002.
- [32] Hashcash Website. URL <http://www.hashcash.org/>.
- [33] A. Bahreman et al. Certified electronic mail. Master's thesis, Carnegie Mellon University, 1992.
- [34] Goodmail Systems, Home of CertifiedEmail. URL <http://www.goodmailsystems.com/>.
- [35] AOL. URL <http://www.aol.com/>.
- [36] M. Peirce and H. Tewari D. O'Mahony. *Electronic Payment Systems for E-Commerce*, chapter 7. 2nd edition, 2001.
- [37] R.L. Rivest and A. Shamir. PayWord and MicroMint: Two simple micropayment schemes.

- [38] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), April 1992. URL <http://www.ietf.org/rfc/rfc1321.txt>.
- [39] Vlastimil Klima. Tunnels in hash functions: Md5 collisions within a minute. Cryptology ePrint Archive, Report 2006/105, 2006. <http://eprint.iacr.org/>.
- [40] C. Adams, S. Farrell, T. Kause, and T. Mononen. Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP). RFC 4210 (Proposed Standard), September 2005. URL <http://www.ietf.org/rfc/rfc4210.txt>.
- [41] Milter.org. URL <http://www.milter.org/>.
- [42] A lightweight rpc library based on xml and http for c and c++. URL <http://xmlrpc-c.sourceforge.net/>.
- [43] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. URL <http://www.ietf.org/rfc/rfc5246.txt>.
- [44] Openssl: The open source toolkit for ssl/tls. URL <http://www.openssl.org/>.