國立交通大學

資訊工程學系

碩士論文

限制送信方以阻擋廣告信的郵件控管機制

Registration-Based Mail Access Control Scheme With Encapsulated Addresses

研　究　生：林雲太

指導教授：謝續平　博士

中華民國九十三年六月

限制送信方以阻擋廣告信的郵件控管機制

Registration-Based Mail Access Control Scheme With Encapsulated Addresses

研　究　生：林雲太　　Student: Win-Tai Lin

指導教授：謝續平　博士　　Advisor: Dr. Shiuh-Pyng Shieh

國　立　交　通　大　學

資　訊　工　程　學　系

碩　士　論　文

A Thesis
Submitted to
Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
in
Computer Science and Information Engineering

June 2004
Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 三 年 六 月

限制送信方以阻擋廣告信的郵件控管機制

研究生：林雲太　　　　　　　　指導教授：謝續平 博士

國立交通大學資訊工程學系

摘要

廣告信的問題近幾年來顯得越來越嚴重。由於電子郵件信箱的高暴露頻率，使用者每天都必須收下平均百封以上的廣告信件。不幸地，一旦某個電子郵件信箱被列入寄送廣告信的名單中，從此便不會被刪去，這使得廣告信的問題只會越來越嚴重，不會有減緩的一天。

爲了解決這個問題，我們提出了一套和郵件伺服器結合、藉由限制送信方身分以阻擋廣告信的郵件控管機制。這個機制會要求第一次送信進來的通信者註冊，一旦對方註冊，使用者便能決定是否要讓對方註冊，若同意的話，對方便會拿到一個只有他能用的電子郵件位址，從此，若該通信者要寄信給這個使用者，他便需要寄信到這個特殊的電子郵件位址。這套郵件控管機制不僅能抵擋所有假冒或捏造送件者的廣告信，也能抵擋使用者不願通信的人。更重要的是，使用者甚至不需要親自記得對方給予的特殊郵件位址，一樣能像往常一般寄信或收信。最後，我們也實作了一個系統原型來測試，看看搭配了這套郵件控管系統的郵件伺服器改變後的效率，結果是令人滿意的。

Registration-Based Mail Access Control Scheme With Encapsulated Addresses

Student: Win-Tai Lin Advisor: Shiuh-Pyng Shieh

Department of Computer Science and Information Engineering

National Chaio Tung University

Abstract

Due to the unwitting exposure of email addresses to the Internet, almost every user will be forced to receive a large amount of spam everyday. Once an email address is found in the spammers' list, unfortunately, it will never be removed from the list. Besides, spam could even be regarded as a denial of service attack against the whole Internet. Therefore, for fighting spam, we propose a scheme - *Mail Access Control System (MACS)*, a system that uses a registry mechanism to block illegal emails or spam. Besides, in the MACS, there is a cooperative mechanism called *Specific Sender Addresses (SSA)* which are email addresses only given to the correspondents a user wants to communicate with and only allowed to be used by these specific correspondents. The length of SSA will be longer than the original one, so the transparency achieved by keeping correspondents' SSA for users will be another important feature in the MACS. We will analyze the effectiveness of our scheme and evaluate the MACS by comparing the processing cost with the original mail server.

誌　　謝

我畢業了！首先想感謝我的指導教授謝續平老師，在這兩年來受到老師諸多的照顧，在指導教誨中獲得了許多寶貴意見與經驗，還有楊明豪和李富源學長的細心指導，沒有他們的耳提面命，沒有今日能畢業的我。另外我也要感謝我的父母，我愛他們，永遠愛。絕對不能漏掉我實驗室的一群好友，阿喬、小銘、OAK 和鉉王，謝謝阿喬的精神鼓勵、小銘老是找我打球、OAK 抓歌天王和鉉王的早睡早起教育。另外是實驗室助理 clancy 和 olga，雖說前者凶巴巴、後者大笨蛋，但是還是要謝謝他們的包容和讓我偷吃東西。最後，謝謝分散式系統與網路安全實驗室這個大家庭兩年來的磨練。

感激在心裡，一生也不會忘記。

Table of Contents

List of Figures

List of Tables

1. Introduction

Spam, also known as unsolicited commercial email or unsolicited bulk email, is currently one of the severe nuisances on the internet. People are forced to receive a large amount of spam everyday but senders only pay very little cost per email. Besides receiving, people will also spend time on reading and deleting all the spam. Gradually, it may decrease users' confidence in using email as their medium of communication. Actually, spam can even be considered a denial of service attack against the whole Internet. They cause the bandwidth loss and unnecessary disk usage to their targets. Therefore, due to the huge but invisible damage to the whole world, it will be very urgent and worthful for fighting spam.

1.1 Requirements

Our overall goal is to perform an effective and efficient spam-filtering system. For this kind of systems, the essential factors can be characterized by the following requirements.

- **Accuracy:** The false positive and the false negative rate of a good spam-filtering system must be very low, even can be zero. The way of judging emails by human will be the most accurate.
- **Incremental Deployment:** A good spam-filtering system can be deployed gradually on the whole world rather than asking all servers in a field should be cooperated to work in the beginning.
- **Backward Compatibility:** If a spam filtering system need to modify the base protocol (e.g., SMTP[1][4]) to achieve its goal, it will be hardly

adopted.

- **Against malicious mail DDoS:** A spam filtering system usually costs more resources or database storage than the original mail server, consuming its resource by DDoS will be easier. There must be an additional mechanism against spam DDoS in this spam filtering system.
- **Low storage cost:** To a mail server, the amount of users and their own correspondents will be very large. If each user or each their correspondent takes up a part of storage in database, it will cost the whole spam-filtering system or say, file system hugely.
- **Transparency:** To the users, if the spam filtering system works like original mail server, it will be very convenient. Users send and receive emails in the same way and don't need to learn how to use this new technology for preventing the spam mail.

1.2 Related Work

There are many researches and methods have been proposed for fighting spam, from government regulation [6] to software approaches. In last few years, a filtering method is very famous. This new method which uses the techniques of probability theory originally invented by Thomas Bayes, the 18th century mathematician, to analyze entire email messages [10]. There are many freely available open-source Bayesian-based email filtering software [12][13][14][15][16][17][18] designed by different programming language, e.g., SpamAssassin [7] which is a Perl-based software. Even a Bayesian-like machine learning software has been included in several email products from Microsoft Corp [11]. False positive is the disadvantage of this kind of software, because Bayesian-based analysis uses the concept of probability to flag messages as spam. Therefore, there may be this kind of situation that some legitimate emails are judged as spam but users even don't know. To make matters worse, one of them maybe is very important for the user. It's the false-positive problem.

Another kind of methods for fighting spam is the concept of Single-Purpose Address (SPA) [8]. SPA is an email address which encrypts policy in it for the purpose of filtering spam. The policy specified may be an expiration date and/or the email address of the allowed sender or a full-fledged program. The advantage of it is it doesn't cost additional storage in the file system, but it cannot solve the guessing problem well. We will describe the guessing problem in section 2.3. Like SPA, the Tagged Message Delivery Agent (TMDA) [9] also has the concept of using correspondents' email address to create single-purpose addresses. TMDA doesn't encrypt policies into email addresses (except for their Dated Addresses) but takes the address as a rule to look up its local tables. Due to this reason, TMDA will keep the rule and the corresponding allowed sender in its database. For the whole mail server, that will be a big cost to the database storage. The challenge-response system of TMDA is another feature. This system blocks spam by sending the original sender a short puzzle that a human can answer quickly and accurately but a machine can not. It's effective for fighting spam, but it will make an auto-sending email to be unable to go through the mail server, like electronic greeting cards sent from any website which doesn't ask users to give them senders' email addresses or some important notification emails sent from an auto-sending program.

Recently, there is a way proposed and cooperated by most of ISPs for fighting spam. They thought a part of spam were usually sent from spammers' own mail servers over dynamic IPs so they add a keyword "dynamic" to each dynamic IP's Fully Qualified Domain Name (FQDN), e.g. 1-1-1-1.hinet.net becomes 1-1-1-1.dynamic.hinet.net. Afterwards they recommended all mail servers to block the emails come from these dynamic IPs by searching the keywork "dynamic". Actually, these may mitigate the problem of spam, but spammers still can deliver spam by the mail server of their own ISPs. Finally, if a large amount of spam were sent though the mail server of one ISP, this ISP may reopen their dynamic IPs due to the overhead to their mail servers.

1.3 Contribution

The objective of this paper is to design an effective and practical spam-filtering scheme which can work with the current mail servers over the SMTP. Unlike the Bayesian-based email filtering software, false-positive problem will not happen in our scheme. Besides, our scheme requires lower storage cost than the TMDA in storing the special email addresses given to correspondents and solves the guessing problem in the situation not affecting other correspondents, which is just the disadvantage of the SPA system. Finally, the transparency in our scheme makes users the most convenient.

1.4 Synopsis

The rest of this paper is organized as follows. In section 2, we will describe the proposed scheme for fighting spam. In section 3, a concept which is used in our proposed scheme called "Specific Sender Addresses" will be presented in great detail. Many operational issues regarding our scheme will be discussed in section 4. In section 5, we will show the evaluation of our scheme, comparing the performance with the original mail server. Finally, section 6 will talk about the future work and then include this paper.

2. Proposed Scheme

Although the way whether an email is spam or not judged totally by the machine is convenient for users, it will be the most effective way if they are judged by human being. Seriously, to a user, any email he doesn't want to receive can be called spam. Therefore, we proposed a scheme which can not only block all unsolicited commercial emails but also let the user decide whose emails he wants to receive. In the front of this scheme, it's a flexible registry-required mechanism which will block any email sent by unknown users and then send an email back for asking the sender to register. In the back of this scheme, registering emails from each sender will be put in user's mailbox, waiting for his agreement and authorization. This makes the spirit that a user can actually decide whose emails he wants to receive rather than let all incoming emails be judged totally by the machine. Besides, we also designed a Specific Sender Addresses (SSA) mechanism which belonged to our scheme to restrict the senders' identity. If a user agrees one correspondent's registry, our scheme will give that person a SSA which is an email address alias of the user and only that correspondent can send emails to this SSA. The appearance of a SSA will be like "user_name.Extension@domain_name". The "Extension" part is the appended string which is encrypted with the allowed sender email address as a rule by each user's encryption key in our scheme and the "." is a symbol separates the original username and the Extension part. So, a SSA will mean only the sender email address encrypted in the Extension part can send emails to this SSA. Emails sent from other persons will be considered as illegal emails. In Section 3 we will describe the SSA mechanism in great detail.

Our scheme is base on SMTP and runs as a part of the mail server, so it is totally backward compatible to current environment. We call it the Mail Access Control System (MACS). Figure 1 shows the architecture of a mail server with the MACS.

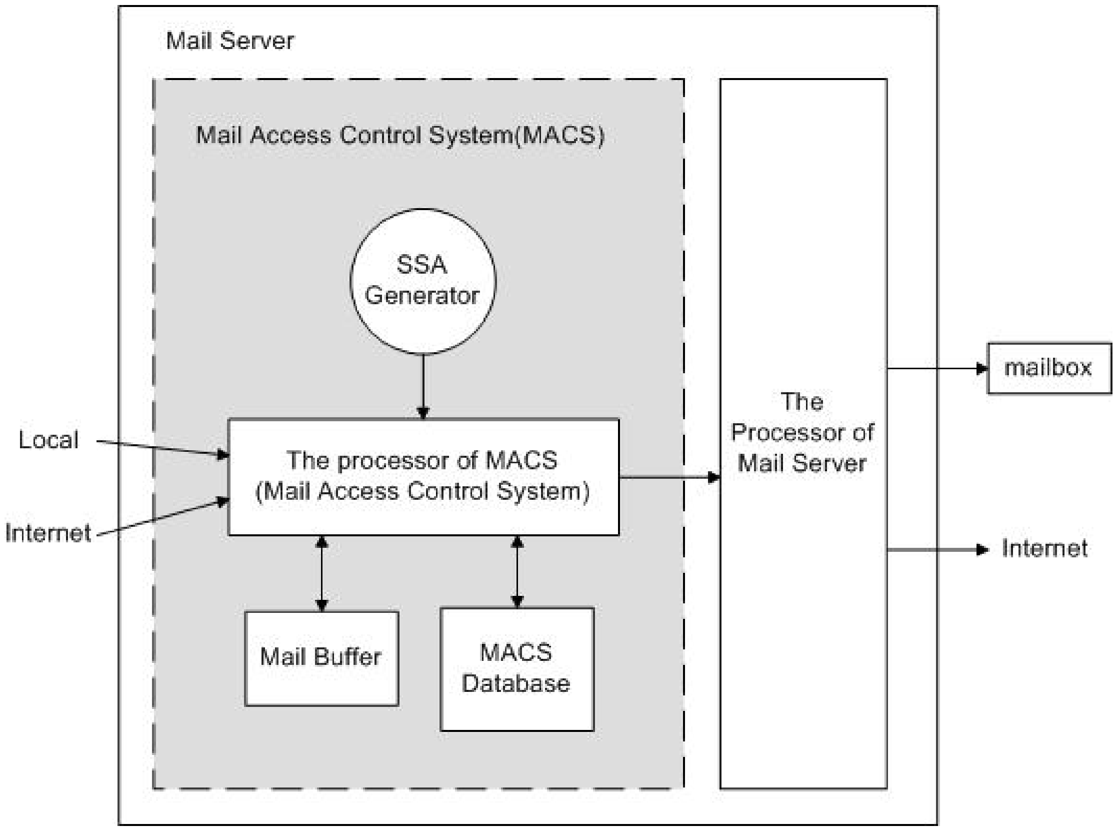

Mail Server
Mail Access Control System(MACS)
SSA Generator
Local
Internet
The processor of MACS (Mail Access Control System)
Mail Buffer
MACS Database
The Processor of Mail Server
mailbox
Internet

Figure 1. Architecture of MACS with the mail server

When an email arrives from Local or the Internet, MACS will check it first. If that email is sent to a legal SSA, it will be recognized as a legal email by the MACS and be delivered into the incoming queue of the mail server. MACS will let the processor of the mail server take over the legal emails. Besides examining the legality of emails, MACS can create different SSAs for each user’s correspondents by calling the SSA Generator if a user wants to receive any correspondent’s emails from now on. In addition to the SSA Generator, there are two major storage in the MACS : the MACS Database and the Mail Buffer. The MACS Database stores all information needed to be checked when any email comes in or goes out, e.g., each user's encryption key and the SSAs given by other correspondents. There are also a database of whitelist and blacklist in the MACS Database. Users can set up their own whitelist and blacklist. As the literal meaning shows, the whitelist stores the allowed correspondents' email addresses or domain names. Emails matching any rule in the whitelist will be allowed to come in without registration. On the contrary, the blacklist stores the unwelcome correspondents' email addresses or domain names. Emails matching any rule in the blacklist will be discarded by the MACS directly. Finally, the Mail Buffer stores all emails which are waiting for the user's agreement of their registry in a limited period of time.

In Section 2.1 we start with the base scheme of MACS in two kinds of situations : only receiver has MACS-support and both sender and receiver have MACS-support; in Section 2.2 we solve three problems by proposing an enhanced scheme of MACS; for another different problem – guessing problem, we will need a SSA-update mechanism and that will be discussed in Section 2.3.

2.1 The Base Scheme of MACS

There are two situations in our base scheme : only receiver side has MACS-support and both sender and receiver side all have MACS-support.

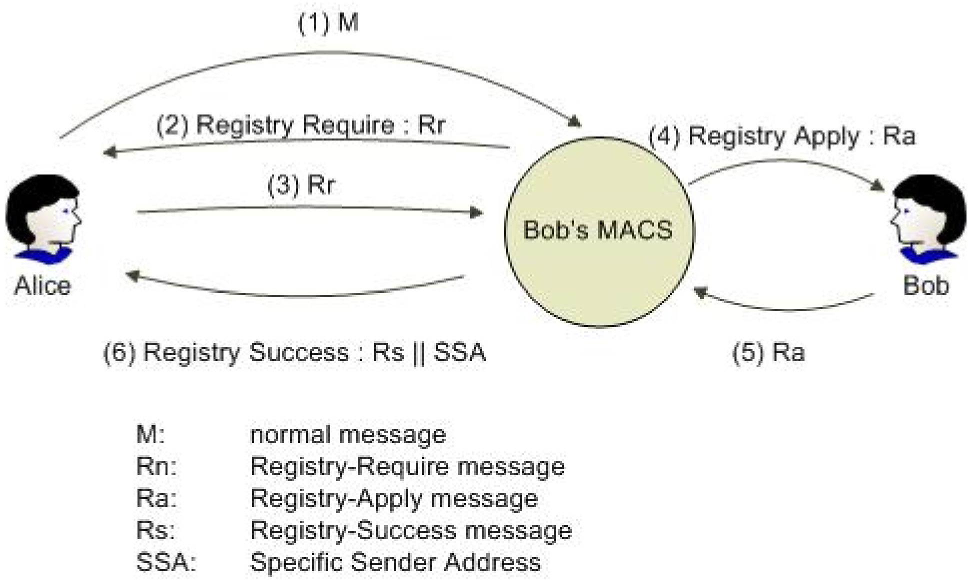

(1) M
(2) Registry Require : Rr
(3) Rr
(4) Registry Apply : Ra
(5) Ra
(6) Registry Success : Rs || SSA
Alice
Bob's MACS
Bob
M: normal message
Rn: Registry-Require message
Ra: Registry-Apply message
Rs: Registry-Success message
SSA: Specific Sender Address

Figure 2. Dealing with a normal email in single support environment

Figure 2 shows the situation that a sender (we give it a name Alice) sends a normal email to a receiver (we give it a name Bob) and only receiver side has MACS-support. Because the email from Alice is a normal email, which means Alice doesn't send this email to a legal SSA, Bob's MACS will block it and send back another email with Registry-Require message notifying Alice she needs to apply a registry to Bob by simply introducing herself on this Registry-Require Email and replying it to Bob. If Alice follows the registry way and does reply the email, when Bob's MACS receives this replied email, it will keep the Alice's introduction, modify the email content with Registry-Apply message to let Bob know someone wants to send emails to him, and then put this Registry-Apply Email into Bob's mailbox waiting for his agreement. If Bob doesn't want to receive Alice's emails, he won't do anything. If he does, replying this Registry-Apply Email with nothing modified is the only thing he needs to do. When Bob's MACS gets the replied Registry-Apply Email from Bob, it will send an email with Registry-Success message and a new generated SSA to Alice notifying her that her registry has been allowed by Bob, if she wants to send emails to Bob, she needs to send emails to this SSA from now on. Figure 3 shows the situation of using SSA. If Alice sends an email to a legal SSA which ever got from Bob, Bob’s MACS will let this email pass through.

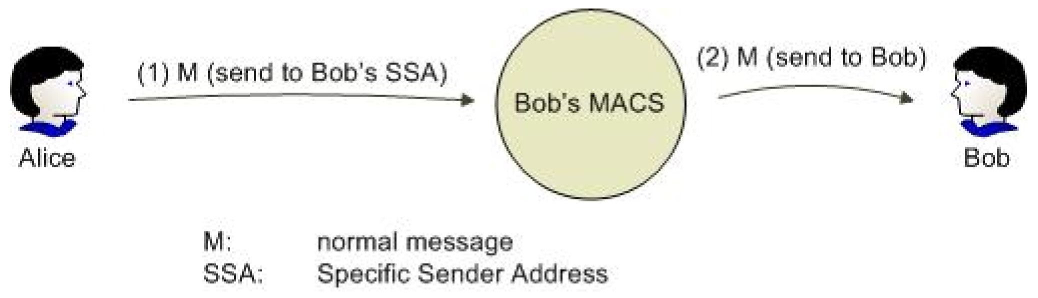
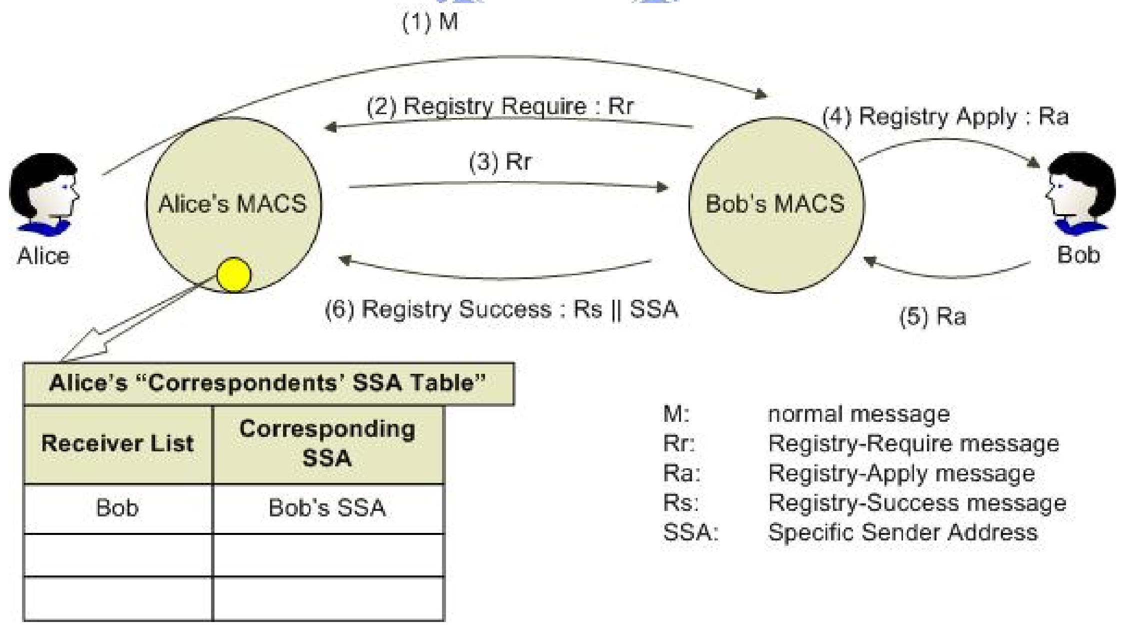


Figure 3. Dealing with a SSA email in single support environment

An idea approach would be communicating with each other in a both support environment. It will be very convenient for Alice, if her mail server also has MACS-support. If it has, the process of applying registry will be automatically done. Figure 4 shows that there are two different places between it and Figure 2.


(1) M
(2) Registry Require : Rr
(3) Rr
(4) Registry Apply : Ra
(5) Ra
(6) Registry Success : Rs || SSA
Alice
Alice's MACS
Bob's MACS
Bob
M: normal message
Rr: Registry-Require message
Ra: Registry-Apply message
Rs: Registry-Success message
SSA: Specific Sender Address

Alice’s “Correspondents’ SSA Table”

Receiver List	Corresponding SSA
Bob	Bob’s SSA

Figure 4. Dealing with a normal email in single support environment

The first difference is when Alice’s MACS receives a Registry-Require Email from Bob's MACS, it will automatically fill out this email with Alice's introduction which are written by Alice in advance and then reply the email. Another difference is when Alice's MACS receives a "Registry Success Email" by Bob's MACS, it will store the SSA appended to the email in Alice's "Correspondents' SSA Table" which is actually one member in the MACS Database. So, if Alice's mail server has MACS-support, she will not be annoyed by the registry procedure and doesn't need to keep Bob's SSA by herself.

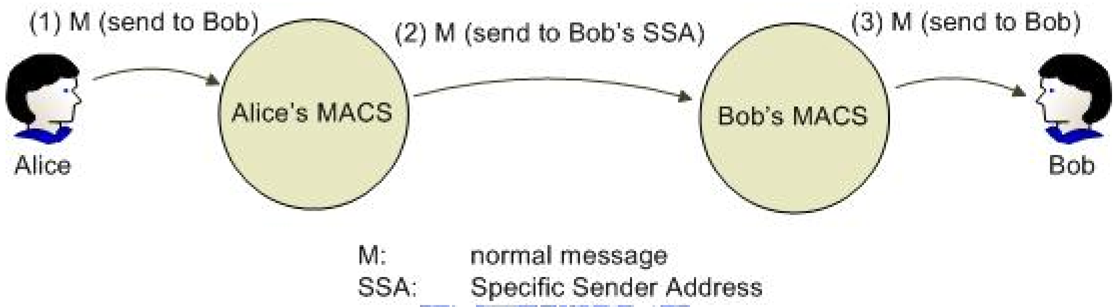


Figure 5. Dealing with a SSA email in both support environment

In this both-support scheme, Alice can send emails as usual. MACS will block her outgoing emails and automatically transform the envelope recipient address into the corresponding SSA format like Figure 5, if Bob's SSA is found in her "Correspondents' SSA Table". So, with MACS-support, a user does even not need to know how the SSA looks like and indeed, it's very convenient.

2.2 Attack-Resistant Scheme of MACS

In the base scheme, three kinds of attack will occur. We first illustrate these problems and then propose an enhanced and attack-resistant scheme of MACS.

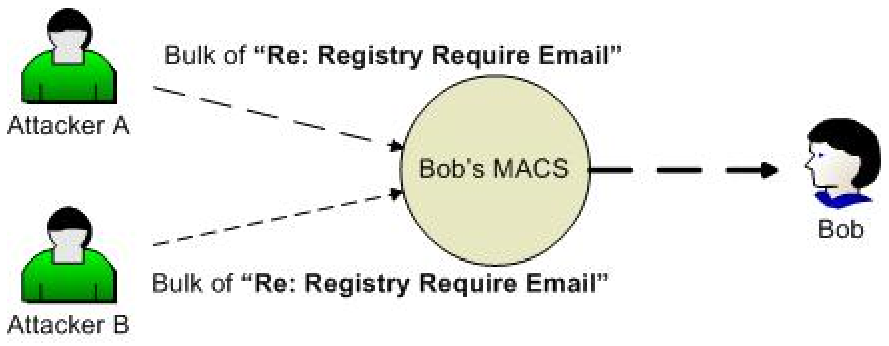
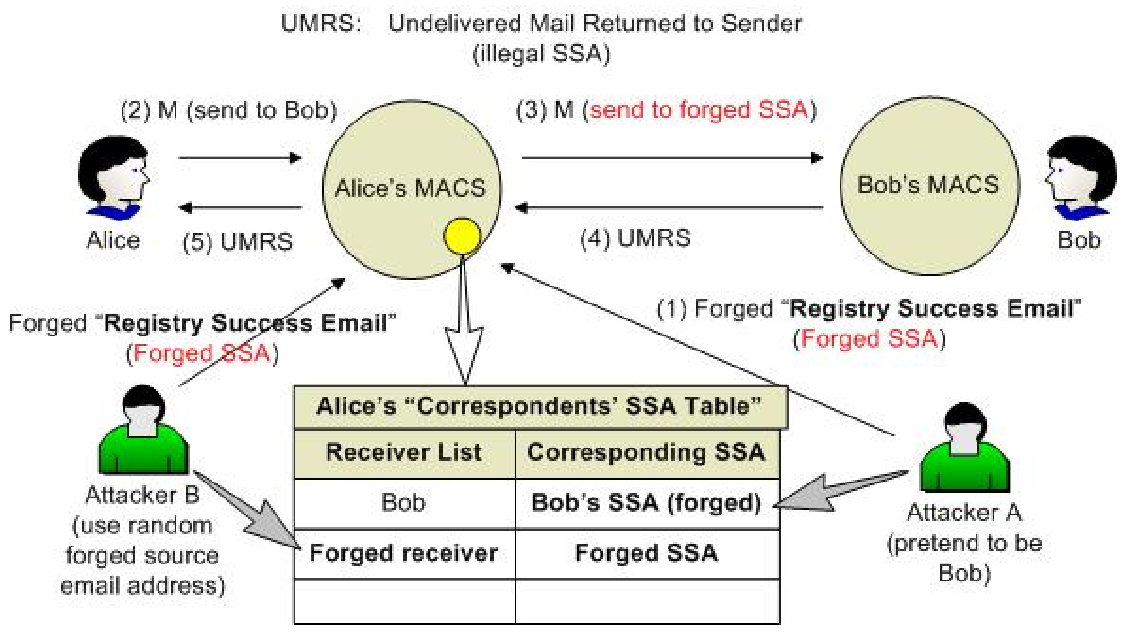

Bulk of "Re: Registry Require Email"
Attacker A
Bob's MACS
Bob
Attacker B
Bulk of "Re: Registry Require Email"

Figure 6. Bulk of "Re: Registry Require Email" attack

First attack is Bulk of "Re: Registry Require Email" attack. If we don't design a prevention way, anyone can follow the format of "Registry Require Email" and directly send bulks of "Re: Registry Require Email" to the victim. In our base scheme, MACS will let the "Re: Registry Require Email" pass through without checking it, so the victim will see bulks of "Re: Registry Require Email" waiting for his agreement. This could be another form of spam, because spammers can use this attack and paste commercial words as their self-introduction on the "Re: Registry Require Email". We limit the length of self-introduction (e.g. 30 characters) to decrease the effect of commercial words, make it hard to show their product but easy to do self-introduction to a human being. Despite we can decrease the effect of making "Re: Registry Require Email" as spam, victims still may be annoyed by bulks of "Re: Registry Require Email" with nonsense. **Figure 6** shows this attack.


UMRS: Undelivered Mail Returned to Sender (illegal SSA)
(2) M (send to Bob)
(3) M (send to forged SSA)
Alice's MACS
Bob's MACS
Alice
(5) UMRS
(4) UMRS
Bob
Forged "Registry Success Email" (Forged SSA)
(1) Forged "Registry Success Email" (Forged SSA)
Attacker B (use random forged source email address)
Attacker A (pretend to be Bob)
Alice's "Correspondents' SSA Table"

Receiver List	Corresponding SSA
Bob	Bob's SSA (forged)
Forged receiver	Forged SSA

Figure 7. Forged "Registry Success Email" attack

The second attack is Forged "Registry Success Email" attack. In our base scheme, anyone could send a forged "Registry Success Email" to the victim, making victim's MACS change or add SSAs in the victim's "Correspondents' SSA Table". For example, in **Figure 7**, Attacker A may pretend to be Bob (it means forge the source email address as Bob's email address) and send a "Registry Success Email" to Alice with a forged SSA. Hereafter, emails which Alice sends to Bob will send to another place. In the base scheme, we solve this problem by checking incoming SSA's username and domain name, seeing whether these names match the source email address of the Registry Success Email. Due to this checking mechanism, attackers could only forge the extension part of a SSA, and this makes the victim still can send emails to the original SSA's mail server but gets a return email (Undelivered Mail Returned to Sender) notifying him that he sends to an illegal SSA. Another attack is an attacker may send many Forged "Registry Success Email" to waste the victim's storage of MACS database, like Attacker B does in **Figure 7**.

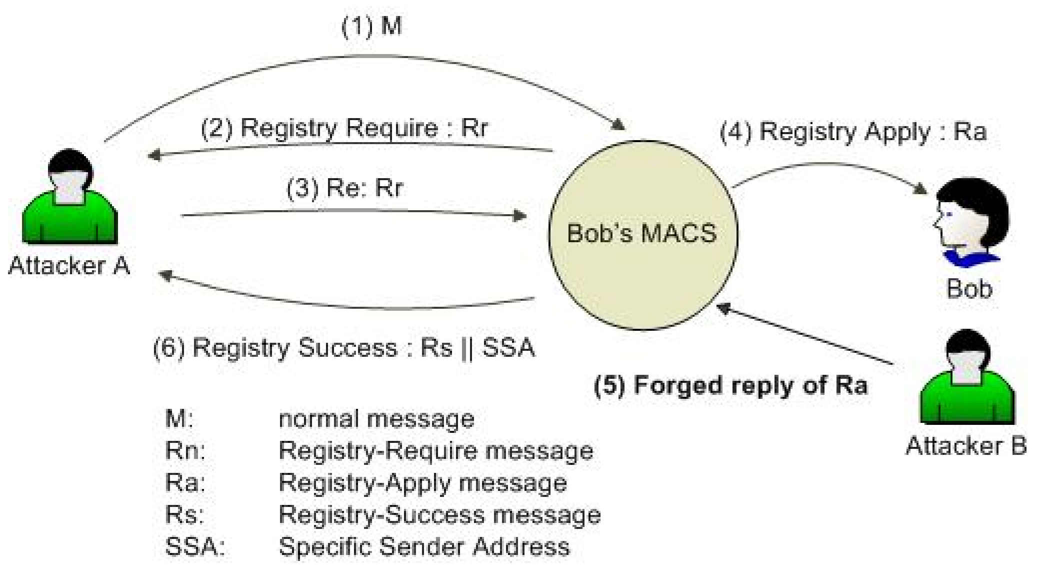

(1) M
(2) Registry Require : Rr
(3) Re: Rr
(4) Registry Apply : Ra
Bob's MACS
Attacker A
Bob
(6) Registry Success : Rs || SSA
(5) Forged reply of Ra
Attacker B
M: normal message
Rn: Registry-Require message
Ra: Registry-Apply message
Rs: Registry-Success message
SSA: Specific Sender Address

Figure 8. Forged "Re: Registry Apply Email" attack

The last attack is Forged "Re: Registry Apply Email" attack. **Figure 8** shows this attack. Attacker A wants to send emails to Bob. He follows the registry mechanism and finally replies Registry Require Email to Bob's MACS. Bob receives the Registry Apply Email from his MACS, but he doesn't want to receive Attacker A's emails, so he doesn't reply it. After a few days, Attacker A doesn't get the Registry Success Email. He guesses Bob doesn't want to receive his emails, so he requests Attacker B to help him. Attack B is a user in the same domain with Bob, so he can pretend to be Bob sending a forged Re: Registry Apply Email to Attacker A. Once Bob's MACS gets this forged email, it will generate a legal SSA and send this SSA with a Registry Success Email to Attacker A. From now on, Attacker A can send emails to this Bob's SSA, and poor Bob would be forced to receive them. Actually, Bob can delete this SSA given to Attacker A afterwards and put Attacker A into his blacklist forever. Although this way is useful enough to solve the Forged "Re: Registry Apply Email" attack, it's still not the most effective way.

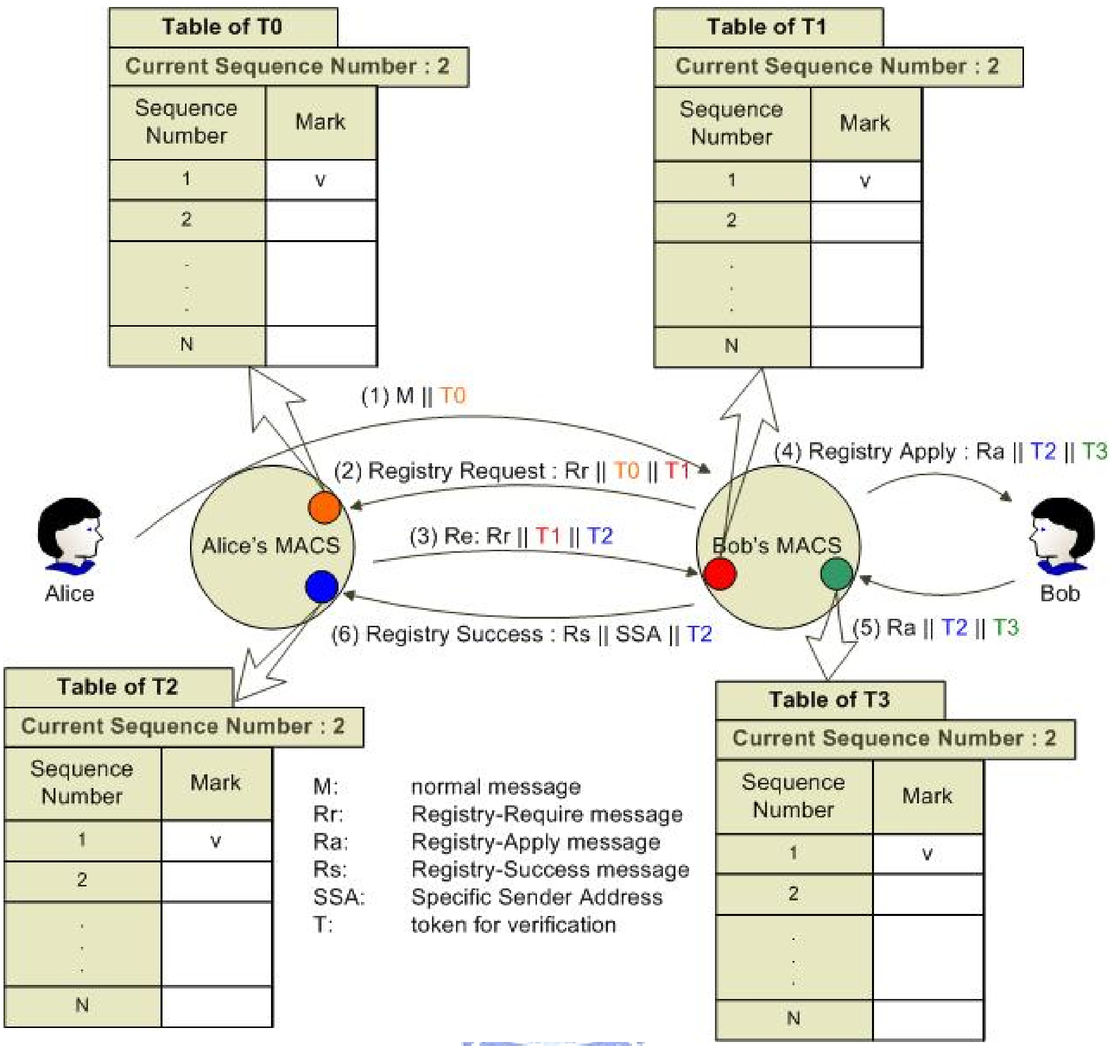


Table of T0
Current Sequence Number : 2
Sequence Number
Mark
1
v
2
N
Table of T1
Current Sequence Number : 2
Sequence Number
Mark
1
v
2
N
(1) M || T0
(2) Registry Request : Rr || T0 || T1
(3) Re: Rr || T1 || T2
(4) Registry Apply : Ra || T2 || T3
(5) Ra || T2 || T3
(6) Registry Success : Rs || SSA || T2
Alice
Alice's MACS
Bob's MACS
Bob
Table of T2
Current Sequence Number : 2
Sequence Number
Mark
1
v
2
N
Table of T3
Current Sequence Number : 2
Sequence Number
Mark
1
v
2
N
M: normal message
Rr: Registry-Require message
Ra: Registry-Apply message
Rs: Registry-Success message
SSA: Specific Sender Address
T: token for verification

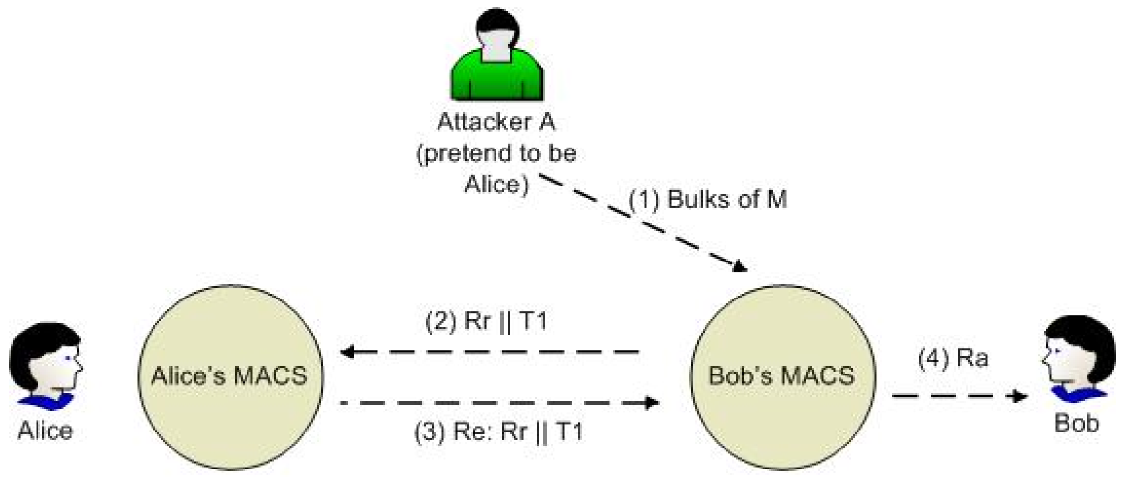
Figure 9. The enhanced and attack-resistant scheme of MACS

After illustrating three kinds of attack, we propose an enhanced and attack-resistant scheme with the token checking mechanism which can prevent these attacks in advance. We use the token T and Table of T to achieve our goal and save the database storage at the same time. T is an encryption of “sender email address”, “receiver email address”, “a sequence number” and “an expiration date”. Before MACS sends a special format email out, it will create this kind of T and append it to this email. After MACS receiving the replied email from corresponding user, it will check whether there is a T. If there isn’t a T found in the replied email, this email will be discarded. If there it is, MACS will decrypt this T and check whether this is an illegal T by examining its format. Comparing sender and receiver email address pair in T with those in the email header will know whether it's a stolen T. Checking the sequence number in T can prevent DoS attack with the same email (replay attack). Moreover, checking the expiration date in T can prevent attackers from reusing this T.

As it shows in **Figure 9**, each "Table of T" holds a current sequence number whose value is one in the beginning. Whenever MACS creates a T, it will encrypt this sequence number with other information needed to be hidden in T. Besides, it will use the current sequence number as an index to search the "Table of T" and clear the original mark in that index site if there is. Finally, MACS will make the current sequence number plus one. These are things MACS needs to do when creating a T. Afterwards, when MACS decrypts the T in the replied email and find it's a legal T, it will use the sequence number decrypted from T as an index to check the "Table of T". If that index site has been marked, this replied email will be seen as a duplicate mail and be discarded by MACS. If there isn't any mark in it, MACS will mark it in that index site preventing later DoS or replay attack. The current sequence number has a maximum N and it is also the maximum storage of the Table of T. When it reaches its maximum, it will become one again. Actually, this makes a problem. If two special format emails which hold the legal but different T with the same sequence number in them are sent to the MACS during their expiration date, MACS will judge one of them as an attack email. We must estimate the average time the sequence number was ran out and set the expiration date of T less than this time.

In **Figure 9**, T1, T2, T3 are designed individually for "Bulk of Re: Registry Require Email Attack", "Forged Registry Success Email Attack" and "Forged Re: Registry Apply Email Attack" introduced previously. If an attacker sends bulks of "Re: Registry Require Email" to the victim, all emails will be discarded by the victim's MACS due to the lack of T1. Likewise, "Forged Registry Success Email Attack" and

“Forged Re: Registry Apply Email Attack” cannot happen due to attackers’ lack of T2 and T3.



M:	normal message
Rr:	Registry-Require message
Ra:	Registry-Apply message
T:	token for verification

Figure 10. A type of the Reflector Attack

Till now, all attackers can do is following the normal sending flow to get legal T1 or T2. An attacker can set up his own MACS and use different forged username to send the victim many normal emails. When the victim’s MACS sends back corresponding “Registry Require Email”s, this attacker’s MACS will reply all of them and make the “Bulk of Re: Registry Require Email Attack”. If this really happens, the victim can put this attacker’s email address or private domain name into his MACS Blacklist and then he will not receive any email from that attacker again. Actually, an attacker usually doesn’t want to use their true domain name but may use a forged identity to send bulks of emails to the victim to make another type of the Reflector Attack. Reflector attack is a kind of attack which an attacker forges his identity as the victim sending many packets to many legitimate servers and all the reply packets from the servers will send to the victim. We show a similar type of the Reflector attack on our MACS scheme in **Figure 10**. Despite the appearance of T1, an attacker still can imitate the Reflector attack to make the "Bulk of Re: Registry Require Email" attack. In this "Alice-Bob" example, the attacker A may pretend to be Alice sending bulks of emails to Bob. Bob's MACS, of course, will send "Registry Require Email"s to Alice. Alice's MACS then will reply all of them and this will make the "Bulk of Re: Registry Require Email Attack".

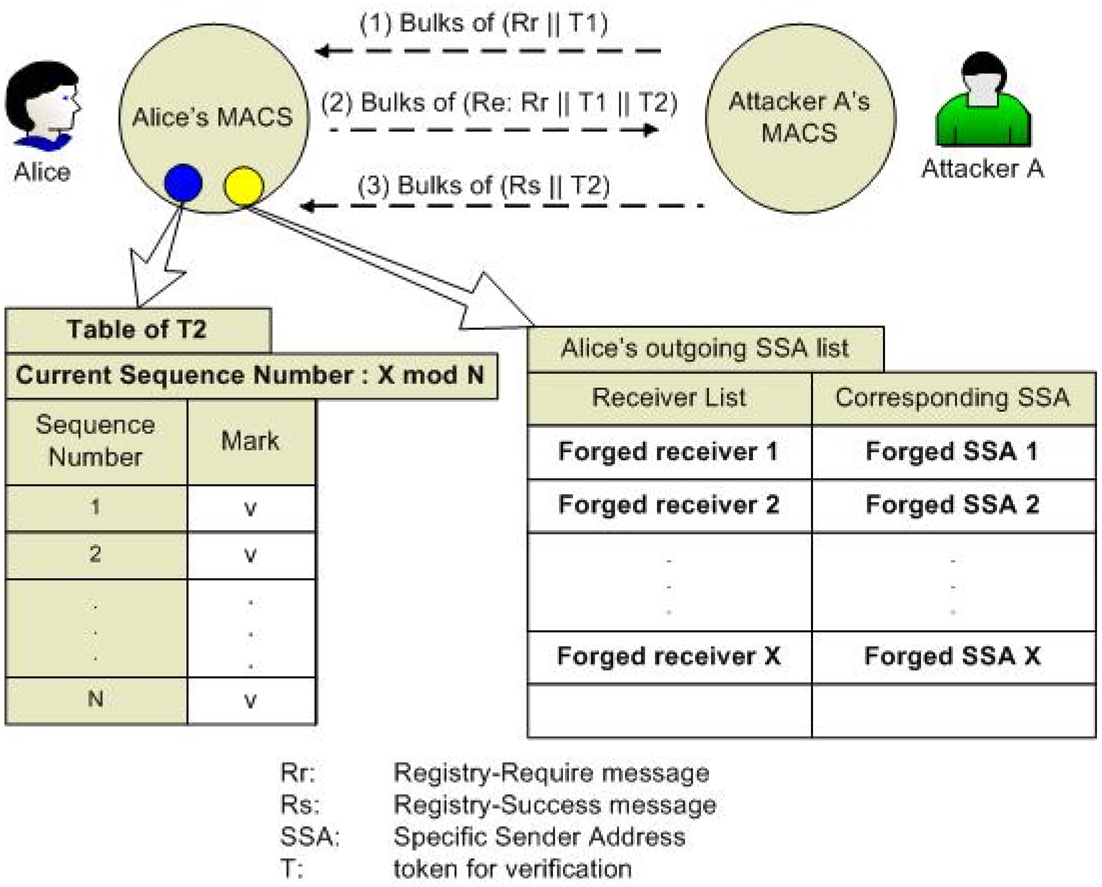


Table of T2

Current Sequence Number : X mod N

Sequence Number	Mark
1	v
2	v
. . .	. . .
N	v

Alice's outgoing SSA list

Receiver List	Corresponding SSA
Forged receiver 1	Forged SSA 1
Forged receiver 2	Forged SSA 2
. . .	. . .
Forged receiver X	Forged SSA X

Rr:	Registry-Require message
Rs:	Registry-Success message
SSA:	Specific Sender Address
T:	token for verification

Figure 11. An attack making the DoS and wasting the victim's database storage

Similarly, an attacker may set up his own MACS and use different forged username to send the victim bulks of "Registry Require Email"s. We show this attack in **Figure 11**. After receiving replied "Registry Require Email"s from the victim, the attacker's MACS may directly send "Registry Success Email"s back with each T2 got from each replied “Registry Require Email”. This will still make the “Forged Registry Success Email Attack” to waste the victim’s database storage with all forged receivers and SSAs. Moreover, it will also make the Dos attack to the victim’s Table of T2, especially when X is bigger than N. (X is the number of bulks emails, N is the maximum storage number of the Table of T2)

In **Figure 9**, the token T0 can make a big help in preventing this kind of attack. When MACS sends a non-SSA email out, it will create a T0 and append to it. After receiving any Registry Require Email, MACS will check the T0 and then decide to reply or to discard it. This will let the attacks we discussed above become hard to work.

We have completely defined the scheme of MACS. In the next section, we will talk about another issue – update of the SSA and we will see another problem – the guessing problem and how does MACS solve it.

2.3 SSA Update

There is still one problem may happen indirectly to users. If an attacker guessed, got from the internet or was told any SSA and the corresponding user’s email address which is allowed to send emails to this SSA, he may pretend to be this victim user sending emails to the SSA. Of course, the victim will receive all this attacker’s emails because to the victim’s MACS, this attacker’s identity is totally legal. Despite receiving many junk emails, the victim still cannot put this correspondent’s email address into his blacklist, or he will lose all emails truly sent by that correspondent. We call this kind of problem the guessing problem. Guessing problem is troublesome. It also happens in the whitelist mail filtering system. If a user uses his whitelist to limit correspondents allowed to send him emails, once the guessing problem happens (an attacker got any permitted email address in this user’s whitelist and uses this email address as the envelope sender address to send emails to this user), he still cannot remove that victim correspondent's email address from his whitelist. It is possible to guess one user's whitelist, especially an attacker guessed the name who is in the same domain as the victim.

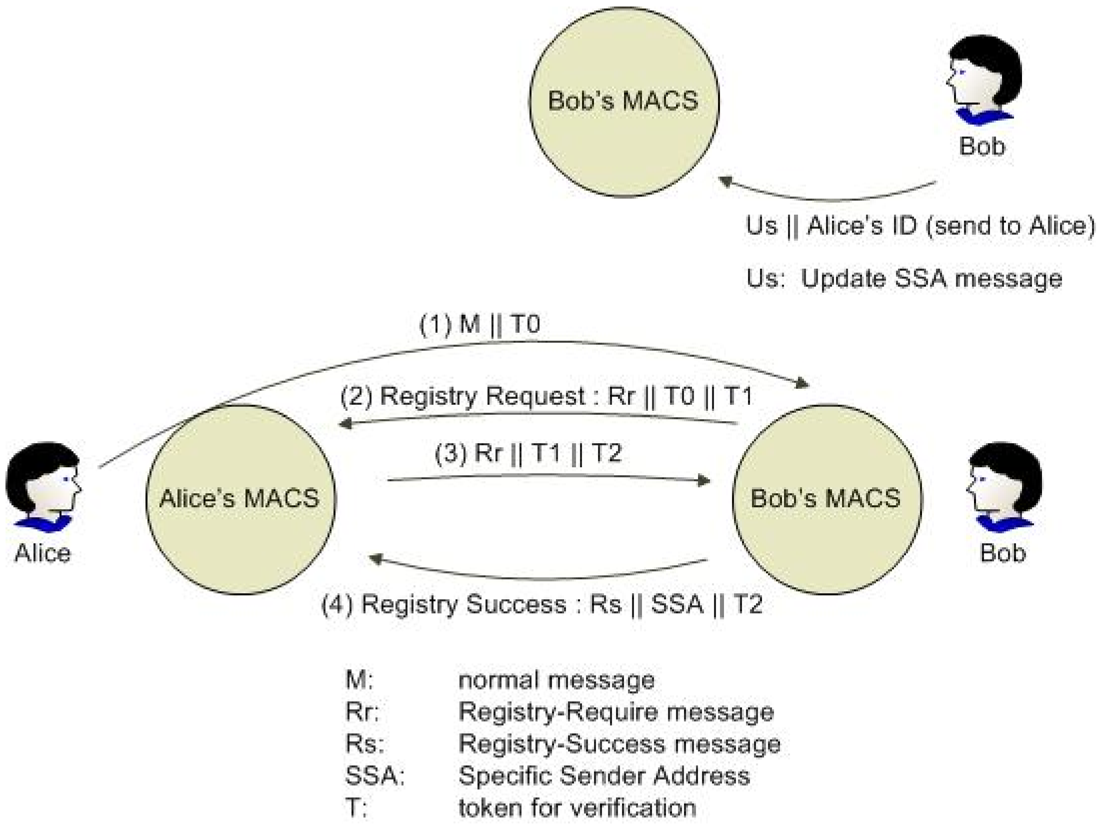

Bob's MACS
Bob
Us || Alice's ID (send to Alice)
Us: Update SSA message
(1) M || T0
(2) Registry Request : Rr || T0 || T1
(3) Rr || T1 || T2
(4) Registry Success : Rs || SSA || T2
Alice
Alice's MACS
Bob's MACS
Bob
M: normal message
Rr: Registry-Require message
Rs: Registry-Success message
SSA: Specific Sender Address
T: token for verification

Figure 12. The process of SSA update

Our scheme can solve guessing problem by updating the SSA ever given to others if a user finds any SSA has been used by spammers. Besides, once we update a SSA, the old one will automatically become invalid, we will discuss this property in detail in section 3. **Figure 12** shows our update scheme. If Bob wants to update the SSA which ever give to Alice. He just needs to send an Update SSA Email to Alice. When his MACS gets this kind of email, it will keep a mark. Next time, when Alice sends an email to Bob, it will start the same Registry Require mechanism, and finally Alice will get a Registry Success Email with a new SSA. The only difference in this

Registry Require mechanism is that Bob isn’t involved, because Bob has agreed the update mission.

3. SSA Mechanism

In section 2, we have simply introduced how a SSA looks like and how we use it. A SSA is an email address alias of a user. It encapsulates an allowed sender's email address in it and only that sender can send emails to this SSA. In a word, it's a limited alias email address and only one specific sender can send to it. Now we want to describe how and why the MACS creates a SSA.

3.1 Creation and Processing of SSA

Once we create and deliver a SSA to a person, we must have a relative examining mechanism to check the legality of this SSA used in the incoming email. Till now, we always talk about the spirit of SSA : only a specific sender can send emails to it. Now, we want to describe how we did it and what the reason is.

There are many ways may achieve this goal, but the most convenient way is storing all information in the MACS database. For example, we keep each allowed correspondent's email address and the corresponding SSA in the MACS database. When a SSA email arrives, we just search all pairs of allowed-correspondent and SSA in our database. If any pair matches the pair in the incoming SSA email header, we judge it as a legal email. Undoubtedly, the computation of this way is light, but the database storage cost will be very huge. For each correspondent of a user, MACS may need to keep a pair of the correspondent's email address and the corresponding SSA. We are not always sure how many correspondents a user has and how many users will be under a MACS. So we try to use a way which need not cost any database storage. We encrypt data needed to be check into the SSA. Here we call the data a rule which is an allowed sender email address. When a SSA email arrives, we just decrypt it and see whether the rule matches the sender email address in the mail header. Because data needed to be checked all transferred from the database to the SSA itself, this way will not cost any other database storage but only an encryption key of MACS. Although this seems perfect, it still has a drawback. It cannot solve the guessing problem very well. After the happening of the guessing problem, indeed, we can update a brand-new SSA for others by changing the encryption key of MACS but that will make all other SSAs become invalid. We may adopt two ways after changing the encryption key of MACS. One is updating all other SSAs at the same time. Actually, it is impossible because MACS doesn’t know any information of correspondents. Another way is keeping a number of old encryption keys at the same time. Adopting this way we won’t need to update all other SSAs immediately but cannot solve the guessing problem immediately either. Eventually, we still need to let all legitimate correspondents use new SSAs gradually, and discard all old encryption keys. So we will need another way to verify a sender’s legitimacy. If a sender sends to a SSA encrypted by an old key, we can send an email to him and wait his reply. If he replies, we will give him a new SSA. Actually, to all other correspondents of users in MACS, it’s an annoying solution, because they must change the SSA (if they don’t have MACS-support) only due to an unrelated person. Besides, what is the best time to discard an old encryption key will be a very troublesome problem, because there isn’t any information letting MACS know how many correspondents are still using an old SSA.

We proposed a better way to solve the problems mentioned above. In this way, MACS will create a N-byte (in our scheme, default value of N is 2) TTL (Time to Live) for each allowed correspondent of users before it calls MACS Generator to generate each SSA. This N-byte TTL will be involved in the creation of the SSA and be seen as an identity of a correspondent. Once we need to change a SSA, we can just change the corresponding TTL. Of course, we need to store each TTL in the MACS database, but we use a way to let the storage cost be minimal as possible as it can. We will show what the way is and why we create and store each TTL in section 3.3.

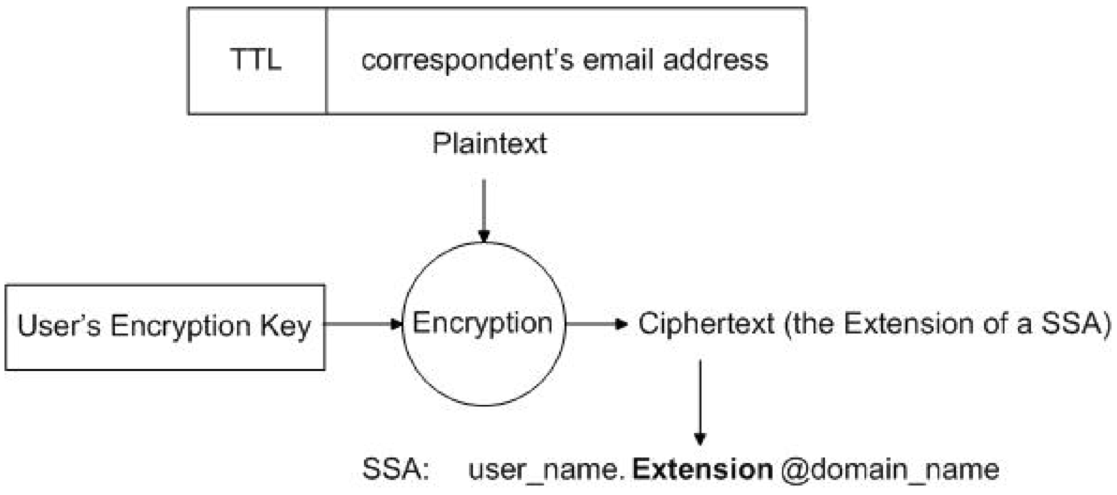

TTL
correspondent's email address
Plaintext
User's Encryption Key
Encryption
Ciphertext (the Extension of a SSA)
SSA: user_name. Extension @domain_name

Figure 13. The producing process of the Extension part of a SSA

Now, let's see the producing process of a SSA first. The format of a SSA is "user_name.Extension@domain_name". The major part of it which we need to create is the Extension part. **Figure 13** shows the way how the MACS Generator generates the Extension part of a SSA. In the MACS, each user has a different encryption key. To achieve the goal of specific sender addresses, the Extension is an encryption of the TTL and the correspondent's email address which are encrypted by each user's encryption key. Therefore, after decrypting the Extension part of a SSA, the MACS can examine whether the sender of this email is legal. Besides, the MACS encapsulates the correspondent's email address into the SSA for saving the storage cost. The MACS doesn't store any SSA in its database, because the rule has been encrypted into the SSA. The encryption algorithm, for example, the Advanced Encryption Standard (AES) with 256-bit key may be chosen for the creation of SSA. Because the length of an encryption block in AES is 16-bytes and we don't want the length of a SSA being too long, we may decide to let the plaintext has the same size with the encryption block, e.g., if the length of an encryption block is 16-byte, we can combine the 2-byte TTL and the 14-byte hash value of the correspondent's email address as our plaintext. This hash function can be chosen from any collision-free and one-way hash function, e.g., MD5. The main purpose of this hush function is to make the secret plaintext of the encryption algorithm. If an attacker gets any user's encryption key and any SSA this user gave to his correspondent, this attacker still cannot know the correspondent's email address. Till now, actually, the Extension is still not complete. In most cases, there may be some unprintable character in the final ciphertext. We can use Base64 [5] (encoding with character [A-Z][a-z][+][/]) to encode the ciphertext. If we choose the AES as the encryption algorithm, after encoding the 16-byte ciphertext, the length of it will become 22 bytes. Due to the case-insensitivity of the email address, a potential problem will occur if any email software changes the case of a SSA, e.g., lowercases all email addresses. For not taking any risk, we decide to choose Base32 [5] (encoding with character [A-Z][2-7]) encoding, although the 16-byte ciphertext will become a 26-byte text string. Eventually, with this Extension and a separate symbol ".", here the SSA is.

When MACS receives a SSA email which is sent to a local user, it will first use Base32 to decode the Extension part and get a ciphertext. Then it will decrypt the resulting ciphertext with the user's encryption key and then get a plaintext. After that, MACS will find out the sender's N-byte TTL in the database and then combine this N-byte TTL with the sender's email address as a string. Finally, if this string matches the plaintext got from the Extension, MACS will judge this SSA email as a legitimate email and pass it into the incoming queue of the mail server. If it doesn't, MACS will send an email back notifying the original sender his email has been discarded due to his improper use of SSA.

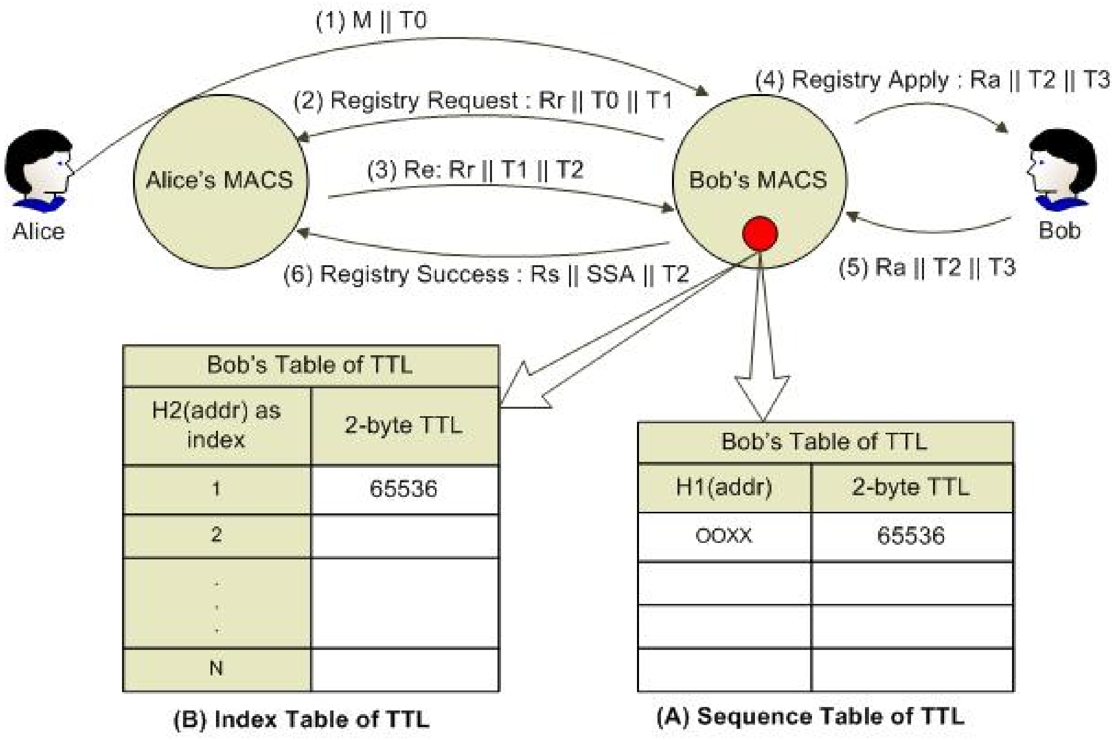
3.2 Issues about SSA

In section 3.1, we ever mentioned two major issues needed to be cared when designing a SSA mechanism. They are the guessing problem and storage cost in database.

Our N-byte TTL is just designed for solving the guessing problem. By changing this N-byte TTL of a correspondent, MACS can create a brand-new SSA and update it for that correspondent with the way described in section 2.3. The initial value of N-byte TTL will be set to its maximum, 2^N, so the easiest way to change this N-byte TTL is subtract one from it each time. Most importantly, SSAs which are hold in each other correspondent of users won't be affected by this update, because each correspondent has his own N-byte TTL. Besides, when MACS changes a N-byte TTL, the old SSA will become invalid immediately and that is what we want to see most.

The length of TTL is an important issue. If it is too big, it will cost unnecessary storage in database. If it is too small, the times for updating SSA will be not enough. For example, if we set the TTL to only 1 byte, the initial value of it will be 256, it means we only can update a SSA 256 times. One day, we will run out of any one (when the TTL becomes zero) and have no choice but to change this user's encryption key to make another brand-new SSA for the same correspondent. Of course, this will be an annoying thing for other correspondents of this user, because they need to change the SSA received before just due to the exhaustion of one correspondent's TTL. We set the length of TTL 2-byte. The initial value of TTL will be 65536 and we think it's a proper size.

A collision-free hash function and a table for storing TTLs are used in our scheme. To minimize the storage cost in database, there are two methods can be used for storing TTLs. The major difference between these two methods will also be the design way of this collision-free hash function and the structure of the table used in database. We illustrate them with different table names : Sequence Table of TTL and Index Table of TTL individually in **Figure 14**.


(1) M || T0
(2) Registry Request : Rr || T0 || T1
(3) Re: Rr || T1 || T2
(4) Registry Apply : Ra || T2 || T3
(5) Ra || T2 || T3
(6) Registry Success : Rs || SSA || T2
Alice
Alice's MACS
Bob's MACS
Bob
(B) Index Table of TTL
(A) Sequence Table of TTL

Bob's Table of TTL	
H2(addr) as index	2-byte TTL
1	65536
2	
. . .	
N	

Bob's Table of TTL	
H1(addr)	2-byte TTL
OOXX	65536

M:	normal message
Rr:	Registry-Require message
Ra:	Registry-Apply message
Rs:	Registry-Success message
SSA:	Specific Sender Address
T:	token for verification
H(addr):	Hash value of correspondent's email address

Figure 14. Two methods used for storing the TTLs

In the Sequence Table of TTL, there are two kinds of strings needed to store. One is the hash value of the correspondent's email address and another is the 2-byte TTL of that correspondent. For example, in **Figure 14**, before Bob's MACS wants to generate a SSA for Alice, it will compute the hash value of Alice's email address and generate a 2-byte TTL for Alice. Then Bob's MACS will store these two strings in Bob's Sequence Table of TTL. Afterwards, if needed, Alice's TTL will be found by matching the hash value of Alice’s email address sequentially in this Table by Bob’s MACS.

The table for storing TTLs in our second method is called Index Table of TTL. The most different place from the first method is the return value of that collision-free hash function. Suppose we get a collision-free hash function whose return value falls in 1 to a maximum N. Then we can use these return values as the index of our table array to get the data. Therefore, due to this kind of return value, there will be only the TTL part needed to store in our Index Table of TTL. The search time of the Index Table is O(1), faster than the O(n) of the Sequence Table and the storage cost of the Index Table will be less than the Sequence Table, because the former doesn’t store all hash value. The only disadvantage of the Index Table is that Index Table limits the number of correspondents a user has in the beginning. If the maximum number N is 2000, it means a user only can have 2000 correspondents at most.

4. Discussion

There are several operational issues of MACS need to be discussed. We will describe the functionality of some components and the solution for some special situation.

4.1 White-list and Blacklist

We have ever simply introduced the white-list and the blacklist in the proposed MACS scheme. Users can actively put his correspondent's email address into his white-list in advance, and then this correspondent's emails will pass through the SSA examination mechanism directly. Besides, using white-list can solve the problem happened during subscribing electronic newspapers. We will mention it after. In the blacklist, users can put any target whose emails he doesn't want to receive. The target it blocks can be a complete email address, a username or a whole domain name. Actually, the whole scheme we described in section 2 and section 3 can block almost all emails a user doesn't want to see, but the blacklist will be useful in the following situation. If a spammer sends many spam mails to a user with forged source email address, this user won't receive any spam mail due to lost of the Registry Require Email. If a spammer sends spam mails with a true domain name and a random-forged username, his MACS will receive the Registry Require Email sent from the user's MACS. This will make the user get many junk Registry Apply Emails sent from different usernames but the same domain name. This user, now, can put this domain name into his blacklist and never get them again. The way how does a user add or delete targets in his white-list and blacklist will be another issue. The easiest way, like the way we used in the update of SSA in section 2.3, a user needs to send a special format email to the target person, and this email will be processed and then discarded by the user's MACS. In the end of this section, we will give all this kinds of special format emails a different solution.

4.2 Subscribing Electronic Newspaper

Nowadays, many people are used to subscribe various kinds of electronic newspapers. The way they subscribe them is filling out the form on the subscribing webpage with their personal information like name, birthday and email address. If the user tells a newspaper publisher his email address, the newspaper publishing agent will use this email address to send electronic newspapers to the user, and then receive the Registry Require Emails sent from the user's MACS. Of course, the user cannot see the electronic newspapers because they will be queued in the MACS. Here comes the problem. The newspaper publishing agent is not a person but just a service program. It cannot understand the content of Registry Require Email and reply it until the person who is responsible for the electronic newspaper service checks the service account's mailbox and replies these emails. Unfortunately, we are always not sure whether anyone might check the service account's mailbox. If no one checks the service account's mailbox, this subscriber will not see any newspapers.

There are three methods can solve this problem. The first method which is also the best way is installing a MACS into the electronic newspapers publisher's mail server. A MACS can automatically reply each Registry Require Email to each subscriber and store all SSAs given from the subscribers' MACS. Therefore, no problem will occur during subscribing and publishing electronic newspapers.

Second is recommending the subscriber to check his MACS mail queue after using his naked email address to subscribe an electronic newspaper. MACS mail queue is put all the emails which are waiting for the registry in a period, including all the spam mails. Therefore, the MACS mail queue will be designed as a space putting another mailbox of users. Users can use the normal way (e.g. mutt –f filename) to open this mailbox and see all the potential spam mails. If the user finds one newspaper in his MACS mail queue, he will know the sender’s email address of this newspaper and then add this email address into his whitelist in MACS. Afterwards, he will receive all these electronic newspapers come from this allowed email address. In addition to using the whitelist, MACS provides an additional protocol which allows a user to ask his MACS to create a SSA for his specific correspondent. Therefore, when the subscriber knows which email address the newspaper publisher will use to send him emails, he can ask his MACS to create a SSA for this publisher and modify the information of the subscriber’s email address to this brand-new SSA on the website of that newspaper. From now on, all newspapers will send to this SSA directly.

The last method involves an additional rule in the SSA mechanism. A user can ask his MACS to create and delete a SSA-for-ANY which is also a SSA but allows anyone to send to it. Each user can have many different-looking SSA-for-ANYs. So, an electronic newspaper subscriber can get a SSA-for-ANY from his MACS and fill out the subscribing form with this SSA-for-ANY. Afterwards, the publishing agent will send electronic newspapers to this SSA-for-ANY and these emails will be judged as legitimate emails by the subscriber’s MACS. Actually, this subscriber can always receive electronic newspapers by the SSA-for-ANY, but because everyone can send to it, the Guessing problem will occur easily. So, like the second method we described, this subscriber can ask his MACS to create a SSA for the publisher once he knows the publisher’s email address or just put the publisher’s email address into his whitelist, and then ask his MACS to delete this SSA-for-ANY.

4.3 Processing the Group email

When MACS receives a reply of Registry Require Email which is sent to a group email address, it will pass it into the incoming queue and let the mail server handle it. The mail server will then give each member of this group a duplicate email. This will make a problem : who needs to reply this email or who has the right to decide. If everyone needs to reply it for agreeing to receive this correspondent's emails, it will take too much time to wait all replies. MACS lets the administrator of the mail server decide who has the right to agree the registry from correspondents. As long as these users who have the decision right all reply, the emails which are sent from that correspondent will be seen by every members of the group.

4.4 Changing User's Encryption Key

Despite the hardness of stealing or guessing a user's 32-byte encryption key, it is still possible to get a user's encryption key for an attacker. If this happens, actually, the user won't suffer any attack directly. If the attacker wants to send forged emails to the victim, he must guess the TTL of one of this victim's correspondents and use the TTL and the encryption key to make the legitimate SSA. A TTL is 2-byte in our scheme, so the attacker must guess 2^{16} times at most. Therefore, a user should have the right to ask MACS to change his encryption key, if he feels many of his SSAs have been compromised. Actually, we can prevent this kind of attack by increasing the length of the TTL, e.g. 4-byte TTL. It will let the guessing times increase substantially. But comparing with the possibility of being guessed a 32-byte encryption key, saving the storage of the database in storing many TTLs will be more valuable for the MACS.

4.5 A Solution to all kinds of Special Format Emails

So far, there are many kinds of special format emails which users need to remember by themselves. When a user wants to ask his MACS to do something, he will need to send the corresponding format email to his MACS. All of them are for different purposes, e.g. update the SSA for one correspondent (section 2.3), add a member to the whitelist, add a member to the blacklist, get a SSA-for-ANY, delete a specific SSA-for-ANY, get a SSA for one specific sender, change the encryption key of self. We can make it more convenient by reducing the complexity of these special format emails. At the same time, we can write a little tool which can send these special format emails for users. Users only need to choose their purposes (e.g. adding whitelist) and type the target email address. This tool will operate all details for them. It will be more convenient if every user downloads this additional tool of the MACS.

There is a problem hidden in sending these special format emails. Everyone may pretend to be others to send these emails, for example, you can arbitrarily add any members to other users' whitelists just like the Forged "Re: Registry Apply Email" attack showed in Figure 8 of section 2.2. Doing authentication will be what we need. We may force users to type their passwords on POP3[2] or IMAP4[3] server when they are sending these special format emails. We may also do the authentication by a CA built up for a local network or a domain. When MACS receives a special format email, it will check whether this email comes from the true person, not a forged one. Besides, there are still many feasible ways. Briefly, if only the way can do the authentication well, it will be a suitable way to this problem.

5. Evaluation and Comparison

When an email comes into our MACS, more time will be needed to process this email, e.g. decode and decrypt. Therefore, we want to know how much percentage of time was increased. We built our MACS prototype by modifying one component of the Postfix version 2.1. This component is called “Cleanup” in the architecture of the Postfix and is responsible for receiving all emails come from Local or Internet and then delivering them to the incoming queue of the Postfix. For evaluating the effectiveness and efficiency of our MACS prototype, we compared the difference between the original mail server and the new mail server with our MACS prototype.

We prepared two computers and used the Postfix 2.1 official release as our mail server. These two computers were our sender site and receiver site. The sender site was equipped with AMD Athlon XP2800+, 2 G RAM running FreeBSD 4.10-STABLE. The receiver site was equipped with double Intel P3 733 MBHz processors, 256 MB RAM running Linux 2.4.20 (Red Hat 9.0). We let the receiver site install the Postfix 2.1 without and with our MACS individually and let the sender site send 1000 emails sequentially as fast as it can to the receiver.

Figure 15 shows the total cost of processing 1000 emails in the “Cleanup” component in two situation : mail server without MACS and with MACS. The x-axis stands for the number of processed emails and the y-axis stands for the total time spent in the “Cleanup” component. In the situation without MACS (original Cleanup), passing 1000 emails through the “Cleanup” component totally cost 10.93427 seconds in average. Passing 1000 SSA emails through our MACS (new Cleanup) totally cost 12.96729 seconds in average. Due to the decoding and decryption to a SSA, our MACS increases 18.5% processing time to process a SSA email in average.

For processing a single email, this result is not satisfying, but actually, to the whole mail server, our MACS won't down too much efficiency. **Figure 16** shows this result. The x-axis stands for the number of processed emails and the y-axis stands for how much time passed when "Cleanup" finished processing emails. In the situation without MACS, 182 seconds passed when "Cleanup" finished the 1000th email. In the situation with MACS, 195 seconds passed when "Cleanup" finished the 1000th email. To the whole mail server, it only increased 7% on the cost. We think it's an acceptable result.

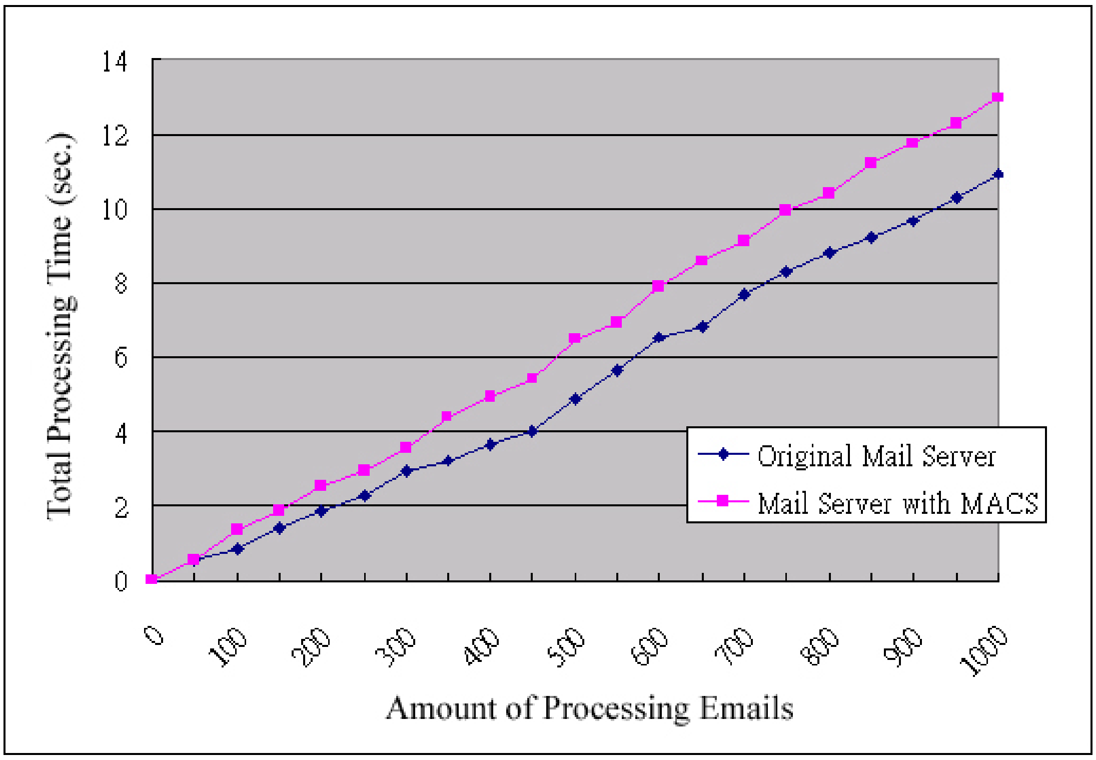


Figure 15. Total time spent on processing emails in mail server and mail server with MACS

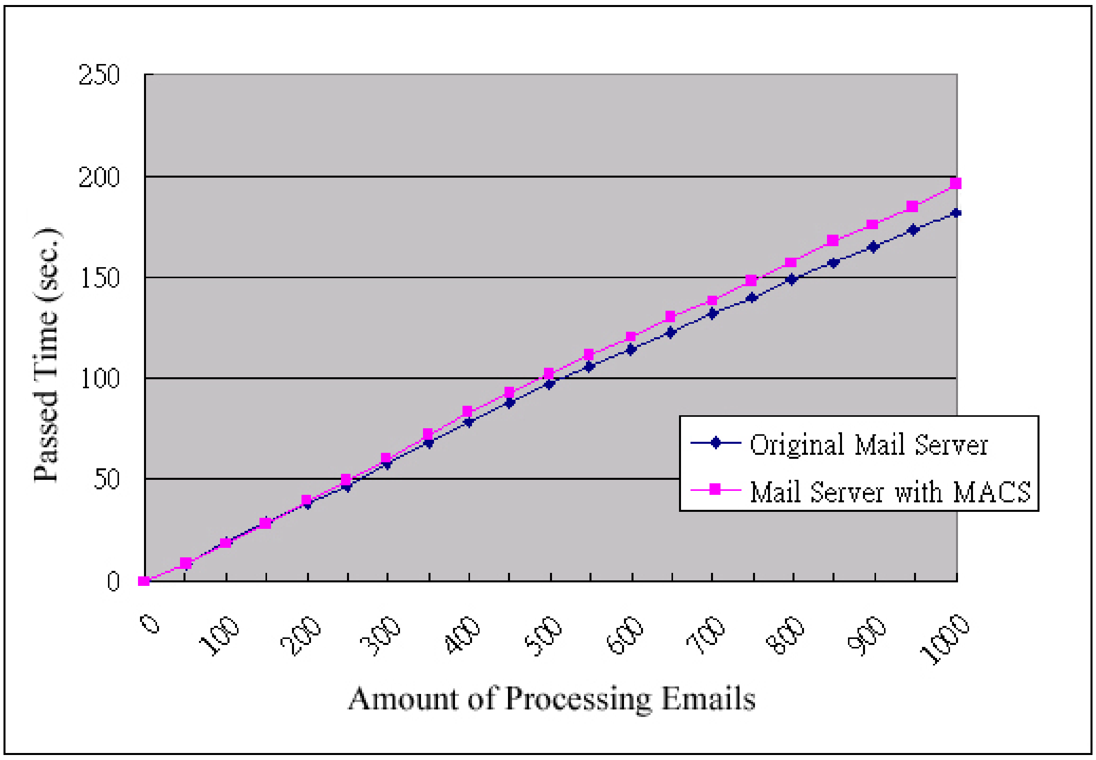


Figure 16. Time passed from the first to the 1000th email in mail server and mail server with MACS

In addition to the efficiency, we evaluate the database storage cost of the MACS by a specific example. Suppose there are 1,000,000 users in a MACS and each user has 1,000 correspondents. For a user in the MACS, we need to count the size of encryption key, the size of Index Table of TTL and the size of SSA Table. Each user's encryption key is 32-byte. A TTL for each correspondent is 2-byte. Each row of a SSA Table is put two strings : correspondent's email address and the SSA the correspondent gives to the user. We suppose the size of a normal email address is 30 bytes and the size of a SSA is (30+1+26) bytes in average. Besides, for a whole MACS, we need to count its 32-byte encryption key and the size of all Table of T (T0, T1, T2, T3). Each mark in a Table of T is 1 bit : 0 or 1, so there are total N bit used in a Table of T. We suppose the maximum N in the Table of T will be (number of users) * (number of T-corresponding emails each user will send or receive in one day in average) * (number of days the Token T exists).

	Single user	**All users in a MACS**	**Storage**
Each user's encryption key	32 bytes	32 bytes * #u	32 MB
Each user's Index Table of TTL	2 bytes * #c	2 bytes * #c * #u	2 GB
Each user's Correspondent's SSA Table	(30+57)bytes * #c	(30+57)bytes * #c * #u	87 GB
	Whole MACS		**Storage**
MACS's encryption key	32 bytes		32 bytes
Table of T0	1 bit * #u * 50 * 2		12.5 MB
Table of T1	1 bit * #u * 200 * 7		175 MB
Table of T2	1 bit * #u * 25 * 7		21.875 MB
Table of T3	1 bit * #u * 25 * 7		21.875 MB
#u = number of users = 1,000,000 #c = number of correspondents each user has = 1,000			

Table 1. The storage cost in database of the MACS

To the Table of T0, we suppose a user sends 50 normal emails out everyday and our MACS only keeps each T0 for 2 days. That is because if the receiver site doesn't have MACS-support, the T0 will never come back, but if the receiver site has MACS-support, the T0 will come back soon with the Registry Require Email. So, for the T0, two days will be much sufficient. To the Table of T1, we suppose a user receives 200 normal emails everyday (including spam mails) in average and our MACS keeps each T1 for a week. It means the correspondent may be able to reply the "Registry Require Email" during one week. To the Table of T2 and T3, we suppose a MACS may receive 25 "Registry Require Email"s and 25 "Re: Registry Require Email"s everyday in average and our MACS will keep each T2 or T3 for a week. This means the user may be able to decide whether he gives a SSA to the correspondent (reply the "Registry Apply Email") during one week.

Table 1 shows the maximal storage cost of the MACS which are used by 1,000,000 users and each user has 1,000 correspondents. We noticed that all "Correspondent's SSA Table"s cost the database 87 GB for achieving the goal of transparency. Actually, we could reduce the storage cost by using a collision-free hash function like the Sequence Table of TTL or the Index Table of TTL we described in section 3.3. Through the hash function, we can shorten the length of the correspondent's email address substantially, even let it disappear. Furthermore, despite the large database storage cost for achieving the transparency, it is still acceptable for current disk storage.

	Bayesian-Based Scheme	SPA	TMDA	MACS
False Negative Problem	Yes	No	Yes	No
False Positive Problem	Yes	No	No	No
Database Storage Cost in Fighting Spam		Low	High	Medium
Feasible Solution to Guessing Problem		No	Yes	Yes
Against Spam Flooding	Yes	No	Yes	Yes
Receiving Automatic-Sending emails	Yes	Yes	No	Yes
Transparency to Users		No	No	Yes

Table 2. Comparison between spam-filtering schemes

Table 2 shows the comparison between all kinds of spam-filtering schemes. Due to the probability theory, even though a Bayesian-based scheme can improve its accuracy on judging spam to 99%, the false negative and false positive problem will still occur. False Negative problem also occurs in TMDA, because if a spammer answers the puzzle in the challenge-email, users in TMDA system will still receive spam. MACS cost more database storage than SPA but less than TMDA. Actually, MACS costs a little more database storage for solving the guessing problem well but SPA cannot solve it. Besides, in the scheme of SPA, users still need to see bulks of spam finding whose emails they want to receive and then create the corresponding SPAs for them. Moreover, users in TMDA may not receive automatic-sending emails like some notifying emails sent by a sending program because a machine cannot solve their puzzles. Finally, if a user receives a SPA or a TMDA, he must keep it or change the old email address of the corresponding alias by himself. The worst is that if he doesn't keep them with, he cannot send emails to those SPAs or TMDAs by the Mail User Agent (MUA) in other person's computer. So, transparency is also a feature of the MACS.

6. Conclusion and Future work

The MACS can keep all correspondents' SSA ever gave to a user for auto-transforming the recipient email address into the SSA format in the future when the user sends an email to the correspondent. It's convenient for the user, but in the meanwhile this user will be restricted only to use his mail server with the MACS to send emails out. If someday this user goes to other place and suddenly he needs to send an email to his friend ever gave him a SSA but the request of sending an email in this area is not allowed by his mail server, a problem will occur. Because the SSA this user's friend ever gave him stored in the user's MACS, if he doesn't send emails through his mail server, the recipient of email address won't be transformed and this user will be required to register to his friend again but he will not know however. There are some methods can solve this kind of situation. The first method is giving the user his correspondents' SSA as well as storing the SSAs in the MACS. Then the user may take all his correspondents' SSA with him and use them arbitrarily around the world. The second method will be installing a webmail-like software over SSL in the user's mail server. Then the user can use this webmail-like software to send emails out around the world after typing his password. The last method, which is also the most interesting method, will be designing a new authentication protocol over SMTP. Like the scheme of Mobile IP, when the user sends emails out by the foreign mail server, this foreign mail server will create a tunnel to his mail server (home server) and check his identity. Finally, this email may be directly sent from home server or sent from the foreign server after getting the corresponding SSA he should sent to. This protocol designing for the mobility is needed to think in detail and it will be the future work of the MACS.

In conclusion, we have described the proposed scheme - *Mail Access Control System (MACS)*, a system that uses a registration mechanism to block unknown emails first and later lets users decide whose registry they want to accept. We have also described the mechanism of *Specific Sender Addresses (SSA)* which is an email address only allowed to be used by one specific sender. We take this specific sender's email address as a rule and encrypt it into the SSA for limiting the owner of this SSA. When a user decides to accept one correspondent's registry, the MACS will create a brand-new SSA and give this SSA to that correspondent. Thus, our system will stop spam and emails sent from persons our users don't want to communicate with. Besides, the encryption in the SSA can save the database storage of the MACS and the TTL generated for each correspondent can solve the guessing problem very well. Moreover, for making a stable and robust system, we also analyzed and solved the DoS attack, the Reflector attack and the forged email attack of special format emails. The transparency is also achieved by the "Correspondents' SSA Table". Finally, the evaluation of the MACS showed that the cost in processing 1000 emails increases only 7% in average and this result will be acceptable to the whole mail server in the respect of fighting spam.

References

[1] J. Klensin, Editor. Simple Mail Transfer Protocol. Request for Comments 2821, Internet Engineering Task Force, April 2001.

[2] J. Myers and M.Rose. Post Office Protocol – Version 3. RFC 1939, http://www.rfc-editor.org/, May 1996.

[3] M. Crispin. Internet Message Access Protocol – Version 4rev1. RFC 2060, http://www.rfc-editor.org/, December 1996.

[4] J. Postel. Simple Mail Transfer Protocol. Request for Comments 821, Internet Engineering Task Force, August 1982.

[5] S. Josefsson, Editor. The Base16, Base32, and Base64 Data Encodings. RFC 3548, http://www.rfc-editor.org/, July 2003.

[6] http://www.spamlaws.com/.

[7] http://spamassassin.org/.

[8] J. Ioannidis. Fighting Spam by Encapsulating Policy in Email Addresses. In *Proceedings of NDSS 2003*.

[9] http://www.tmda.net/.

[10] Vaughan-Nichols, S.J. Saving Private E-mail. Spectrum, IEEE, August 2003, 40-44.

[11] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk email., *AAAI Workshop on Learning for Text Categorization*, July 1998, Madison, Wisconsin. AAAI Technical Report WS-98-05

[12] http://bogofilter.sourceforge.net/.

[13] http://www.squirrelmail.org/.

[14] http://spambayes.sourceforge.net/.

[15] http://crm114.sourceforge.net/.

[16] http://sherpafilters.sourceforge.net/.

[17] http://popfile.sourceforge.net/.

[18] http://www.mozilla.org/mailnews/spam.html.

[19] http://www.postfix.org/.