Secure Public-Key Services For Web-based Mail
A sketch of a secure e-mail environment

Evan Knop
Advisor: Prof. Sean Smith

Summer 2001

Abstract

Many people believe that electronic mail ought to have the same privacy as paper mail; strong encryption may be used to insure that messages which are sent across public channels are protected against examination or modification. For this purpose, several standards have been proposed and implemented which use public-key cryptography to provide authentication and privacy to RFC822 messages. The two most popular standards are S/MIME and OpenPGP, each of which is supported by a variety of mail clients.

As the internet becomes more ubiquitous, and people want to access their e-mail from any client, web-based interfaces to e-mail have become more popular. Instead of using specialized mail clients, users want to be able to access their messages from anywhere, using a web browser. Unfortunately, this move to more centralized services is in direct conflict with the desire to keep mail private by keeping private keys on a small number of secured machines.

In particular, the user may not wish to trust the provider of the web-based e-mail service with their private key. In this situation, a trusted computing environment is needed to ensure that the web server cannot compromise the private key, while allowing the user to authorize decryption and signing of messages through a web-based interface. The IBM 4758 provides a secure hardware coprocessor which can be used to protect the user's secrets from the system administrator and provide a trusted computing platform for decrypting and managing the user's keys. Building on Shan Jiang's master's thesis, WebALPS, we have researched and attempted to build a web-based S/MIME user agent which would keep the user's keys secured from the server operator. Although the sofware was not completed, we learned several software engineering lessons and created a paradigm for indicating the source and trust level of displayed HTML source from a trustworthy and untrustworthy host sharing an HTTP connection.

Contents

Chapter 1

Introduction

1.1 Background

With the dramatic increase of Internet connectivity and e-mail use in the past few years, there has been increasing demand to enable electronic communications to have the same security guarantees that exist for physical correspondence. Furthermore, because of the heterogeneous nature of the Internet, any new solutions must inter-operate (or at least not interfere with) existing communications.

Current e-mail communication is based on the SMTP protocol described in RFC 821 [12]. The format of e-mail headers is defined in RFC 822, and message bodies general conform to the MIME standard described in RFCs 2045, 2046 and 2047 (Multipurpose Internet Mail Extension) [3, 4, 5]. Any replacement electronic mail system must either conform to the guidelines of the RFCs, or must define its own transport medium, message syntax, and naming scheme. Furthermore, the transition between this new system and the current system may also cause user confusion and system incompatibility. For all of these reasons, most attempts to add capabilities have chosen to extend the 7-bit ASCII protocol specified in RFC 822.

Two of the areas in which RFC 821/822 e-mail falls short of physical mail are in securing and authenticating messages. Because RFC 821 does not specify any method of verifying the identity of message sender , it is simple[1] for a malicious host to send mail with a forged `To:` header. Similarly, RFC

[1]The most obvious example of this phenomenon is unsolicited commercial e-mail, often called “SPAM.”

	S/MIME	**PGP**
old	*S/MIME v2* RFCs 2311-2315, 2268	*PGP/MIME* RFCs 1991, 2015
new	*S/MIME v3* RFCs 2630-2633	*OpenPGP* RFC 2440

Table 1.1: Old and new public-key e-mail solutions

821 makes no provision for the transmission of encrypted or blinded messages, "sending e-mail is like sending a postcard" — anyone along the delivery path (which could include the administrators of third-party systems, or network eavesdroppers) can examine and alter the contents of an e-mail message.

Both security and authenticity can be achieved using cryptographic methods. Using public-key cryptography, users can encrypt their messages so that they cannot be read except by the possessors of certain private keys. Similarly, e-mail senders can use digital signatures to prove their identity and prevent tampering with messages which they have sent.

1.2 Current Solutions

Currently, there are two competing standards for formatting and encoding public-key protected e-mail: S/MIME (Secure MIME, proposed by RSA, Inc.) and PGP (based on the Pretty Good Privacy software). Each of these standards has two versions currently in use, as shown in Table 1.1 [1].

These two solutions approach the goal from different ends of the software spectrum: S/MIME is based on the X.509 certificate standard and the ASN.1 syntax defined by the ITU (International Telecommunications Union) and the ISO (International Organization for Standardization), while PGP is based on the binary format of the popular PGP application. Table 1.2 shows some of the differences between the S/MIMEv3 and the OpenPGP message formats [1].

Currently, there is no clear industry support for one format over the other — plugins exist to enable PGP support in almost every mail client, but products such as Netscape Communicator and Microsoft Outlook have chosen to adopt S/MIME as their "built-in" public-key message format. One advantage of S/MIME is that it supports the X.509 certificate structure which

Mandatory Features	**S/MIMEv3**	**OpenPGP**
Message format	Binary, based on CMS	Binary, based on PGP
Certificate format	Binary, based on X.509	Binary, based on PGP
Symmetric encryption algorithm	Triple DES (DES EDE3 CBC)	Triple DES (DES EDE3 CFB)
Signature algorithm	Diffie-Hellman (X9.42) with DSS	El Gamal with DSS
Hash algorithm	SHA-1	SHA-1
MIME encapsulation of signed data	Choice of multipart/signed or CMS format	multipart/signed with ASCII armor
MIME encapsulation of encrypted data	application/pkcs7-mime	multipart/encrypted

Table 1.2: Features of S/MIME and OpenPGP message formats

is considered the standard for most PKI (Public-Key Infrastructure) efforts.

1.3 Current Status

Although there exist client programs which will support either S/MIME or PGP, these programs require that the user have local access to their public/private key pair in order to perform e-mail functions. Because this key pair acts as a digital "proof of identity", it should be well-protected, which argues for putting it on as few machines as possible. The goal of protecting private keys unfortunately restricts the number of clients machines from which an e-mail user can access their mail. In the increasingly decentralized Internet environment where many users may access their e-mail on many computers (corporate desktop, laptop, home computer, public terminals, etc.), the restriction that only certain "secured" computers can allow a user to access their messages is restrictive.

For many users where e-mail privacy and security is not an issue, web-based e-mail interfaces have proved very successful in providing client-independent access to e-mail. However, web-based e-mail interfaces do not offer the sort of private-key assurances which users would expect from client-side software. Because web services cannot offer these guarantees, there are few web-based email services which provide support for PGP or S/MIME [2]. As digital signatures become legally binding, it becomes increasingly problematic to have private keys stored on a third-party server – an unscrupulous server operator could easily sign financially binding documents with the "unforgable" digital signature of a user.

For these reasons, we decided to attempt to secure the private keys using the IBM 4758 secure co-processor platform (hereafter known as "the 4758" or "the card" [18, 13]. Sean Smith suggested many of these directions in his WebALPS paper, in which he proposed using the 4758 to provide a remote trusted computing environment [17]. We will building on the work of Shan Jiang's master's thesis, in which he implemented a co-server called on the 4758 which would allow the co-server to participate in an SSL connection between a client machine and a web host running the apache http server [11]. Using the trusted hardware/software platform of the 4758 and WebALPS, we could assure that only the co-server and the client can perform private-key operations. By using the 4758 to intercept only the necessary requests, much of the load can be offloaded from the co-processor to Apache server software.

[2]On Freshmeat.net[6], ICEMail, a Java e-mail client, was the only package that provided any private-key options, and it included only S/MIMEv2

Chapter 2

Selection of Problem

2.1 Selection of Technology

The first decision we needed to make when we began this project was whether we would support the S/MIME or OpenPGP format. Eventually, it would be very satisfying to have a web-based mail client which could seamlessly interoperate with both of these specifications, but it would be impractical for our first attempt to support all the options of either interface, and attempting to split our work between the two would simply degrade both.

One of the motivations for this project is that secure e-mail is one of the obvious applications of a public key infrastructure. More generally, secure web-based e-mail would allow testing of both the WebALPS framework developed by Shan Jiang and of the capabilities of the IBM 4758. We were also concerned that there would need to be sufficient deployment to test the interoperability of our solution with other implementations of the same protocol.

In the end, we decided to implement S/MIMEv3, because of its integration with the X.509 certificate structure which forms the basis for most PKI efforts. Also, there are several software packages which all serve to implement the S/MIME standard, while OpenPGP is based mostly on two implementations, GnuPG and the PGP-6.5 applications.

Because several major vendors recently began shipping S/MIMEv3-compliant mail user agents. In particular, both Microsoft Outlook 2000 SR1a and Netscape Communicator offer some degree of support for S/MIMEv3. There also exist plugins for existing mail clients (Baltimore's MailSecure and SSE's

TrustedMIME). Therefore, we will be able to test our implementation with several other implementations.

2.2 Other Work

Before we began building our own secure e-mail service, we surveyed the current options for securing e-mail without relying on a secure local disk to store private keys. We found only one web-based public-key e-mail solution, which actually relied on a Java applet to perform the actual private-key operations. The only other non-client-based solution we found was a product that claimed to offload the public and private key operations to a trusted server, but these products seem intended for use in a trusted-server environment.

2.2.1 Hushmail

Hushmail is a free web-based mail service which uses the OpenPGP format to allow it's users to create and manage public and private keys and to send and recieve e-mail. Hushmail's software is based on signed Java applets; all public-key operations are actually done in the client's JVM. Users create keys using Hushmail's account sign-up applet, which creates their public and private keys, and then encrypts the private key using the AES cypher, with a key based on an iterated SHA hash of the user's passphrase. The public and private keys are then stored on the Hushmail servers. Although this mechanism seems relatively secure, the storage of private keys on third-party controlled servers is problematic. Although the key which encrypts the private keys is known only to the user, many of the private keys could probably be recovered through a dictionary attack if the keys could be accessed from the servers.

2.2.2 TFS Secure Messaging-Server

The TFS Secure Messaging-Server promises to allow organizations to apply public-key encryption at the mail server, based on rules and policies defined by administrators. The Messaging-Server offers support for both S/MIME and OpenPGP, and supports both generation and import of X.509 Certificates. This product does not fall into the same category as Hushmail, because it seems to be intended to allow e-mail encryption on the mailserver before mail is sent through a firewall to the internet at large. However, this product might also be a candidate for application of the secure coprocessing technology of the 4758, to protect the device from insider attack. (If one applied the WebALPS method of an untrusted server and a secure co-server to SMTP-over-SSL, this type of product could be the result.)

Chapter 3

Software Support

3.1 Library support

To perform the actual S/MIME decoding on the 4758, we needed to use a cryptographic library to decode the X.509 certificates and (possibly) for the operations to encode and decode the S/MIME envelopes and signatures.

3.1.1 cryptlib

We began the project by looking at cryptlib, a cryptographic library with some support for the IBM 4758, as well as a high-level software interface to S/MIME and X.509 certificates. Cryptlib is implemented entirely in C, and it supports a large number of encryption modes and standard formats. Cryptlib is commercial software, however the source code is available, and there is a free license for non-commercial use of cryptlib.

When using cryptlib, we discovered a few limitations in the cryptlib software, in particular, the need to use a keyset to store private keys. Keysets are a collection of public and private keys and certificates which cryptlib can store and access on a variety of media, including databases, LDAP directories, and fies. Unfortunately, the CP/Q embedded operating system running on the 4758 has no support for any of these abstractions.

3.1.2 Crypto++

Crypto++ is a free library which provides access to a large number of conventional and public-key encryption algorithms. Crypto++ is intended to serve as a library for encryption source, and does not provide high-level handling of services such as X.509 certificates or S/MIME signatures. Also, the interface to Crypto++ is implemented in C++, which is not available in the 4758 development environment. For all these reasons, Crypto++ was removed from consideration.

3.1.3 Snacc

Snacc is a tool for producing C parsers and data representations automatically from ASN.1 syntax trees. Snacc is short for "Sample Neufeld ASN.1 to C Compiler". Because of the need to use X.509 and the Cryptographic Message Syntax, both of which are specified in ASN.1, we considered using snacc to process these data structures, which would then be supplied to custom software libraries and the 4758's crypto hardware to perform the actual encryption and decryption. Unfortunately, the size of the ASN.1 specifications for X.509 and CMS, in addition to the lack of documentation and the difficulty of reading or using the final flex/yacc output, caused us to abandon this solution.

3.1.4 certlib

Certlib is a free software library for reading X.509 certificates, released by Sun Microsystems for use in their SKIP Secure IP initiative. Certlib contains code for reading and writing X.509 certificates used by S/MIME, but it is unfortunately written in C++. Also, much of certlib is based on Sun's underlying C++ API, which would also need to be ported to C on the 4758.

3.1.5 Getronics S/MIME Freeware Library

Getronics has been contracted by the U.S. government to produce a free S/MIME library in conjunction with the S/MIME v3 standards process. The library is designed to be able to use a variety of cryptographic back-end libraries to perform it's processing. The library itself implements all of RFCs 2630 and 2634, and parts of RFC 2633 and 2632. The underlying crypto library is responsible for implementing RFC 2631 and the remainder of RFCs 2632 and 2633. Unfortunately, the current back-end library modules exist only for Crypto++, RSA BSAFE, and hardware crypto engines. Also, the library only provides a very limited C interface, with most of the functionality being provided by a C++ API.

3.2 Web interface

3.2.1 WebALPS

We have decided to build upon the foundations of the WebALPS project, as this has already provided a platform where the 4758 can participate meaning-fulling in a SSL-encrypted connection. In addition, there already exist some hooks to allow for co-server filtering of requests to the web server, which will be necessary for requests which should be fulfilled by the co-server itself.

3.2.2 SquirrelMail

SquirrelMail is a PHP e-mail client based on IMAP. SquirrelMail was selected from a variety of Open-Source web-based e-mail clients on the basis of: mail attachment handling, no use of Javascript or Java, and the ability of source code. There exist several other web-mail packages which may fulfil similar requirements; since the focus of this project is not on the e-mail interface, SquirrelMail seemed to be a reasonable solution. SquirrelMail is particularly useful in that it uses IMAP features to selectively retrieve portions of a MIME message for display. This functionality can be used by the co-server to download only the necessary attachments for verification or decryption.

Chapter 4

Securing the Interface

4.1 Controlling content from the server

If we want to protect users' private keys and private-key functions from unauthorized access by the untrusted host software, we need a clear way to indicate to the user the origin of each element, and prevent the host server from being able to display malicious content. One example of such malicious content would be a page from the host server which would allow the user to apply their private key, but which also had a java applet which would report the user's keystrokes to the host. Because of the arbitrary nature of Java and Javascript, the entire secure-mail system should neither depend upon nor contain any Java or Javascript.

4.1.1 Filters

In order to prevent the server from providing malicious content, the co-server must assure that all data that is sent to the web client has been scanned and that the content has been removed or neutralized in some way. Also, the co-server may want to protect certain information from the client before it is accesses by the host server. For example, the text of an e-mail message which is to be encrypted should not be passed to the untrusted host until it has been encrypted.

Both of these tasks (and perhaps others, such as controlling the caching of pages) call for the application of co-sever-side filters. By designing these filters in such a way that each filter can be considered independently, we can practice software re-use by writing a filter interface once, and then writing

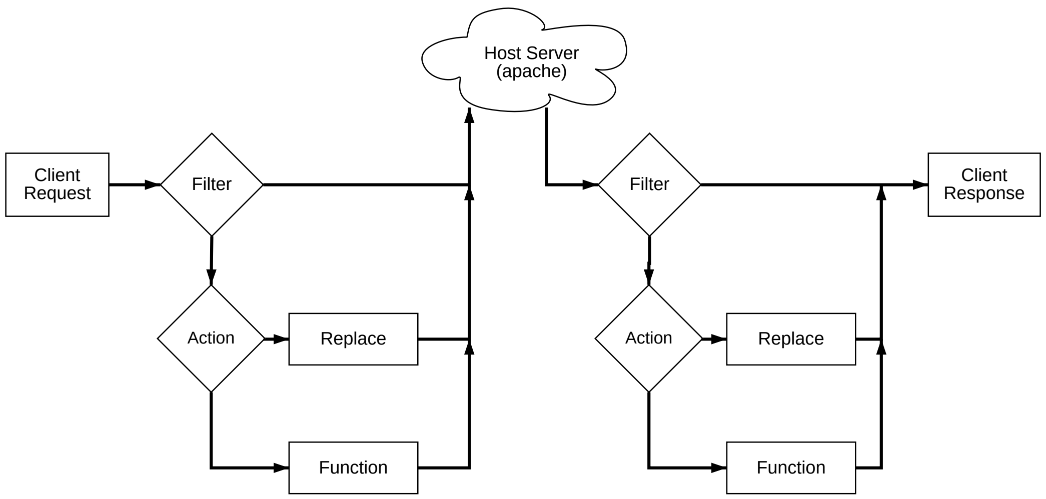


Figure 4.1: Overview of the filtering process

each filter independently, rather than adding filtering code piecewise to the rest of WebALPS. At a minimum, the filter interface would need to supply the ability to run one or more co-server-supplied functions with the ability to modify the input and output strings from the server. To simplify filter programming, it should be possible to load several filters and have them operate sequentially. To simply common tasks, the filter interface could specify a text string or filter action which would indicate whether or not the filter should be run. Because a common task is fixed-string replacement, we should make it possible to write a fixed-function filter which replaces a server text string with a string from the co-server. This leads us to a filter model similiar to that of Figure 4.1.

4.1.2 Filter Interface

The actual filter interface is written in C, and supports three types of filters: input filters, which operate on the client input string after it is decrypted but before it is passed to apache; output filters, which operate on the server output string before it is encrypted and sent to the client; and session filters, which are described in Section 4.2.

To accommodate our goals from Section 4.1.1, we needed to design an interface which allows the programmer to create and add a filter, and to be able to remove the filter if it is no longer needed. We also need to establish two different “filter queues”: one, called “input”, is applied to all data coming from the user to the server, and the other, called “output”, which is applied to data being sent to the user from the server. Both of these queues operate in the same manner: each filter is applied sucessively, until the filter list is exhausted, or until one of the filters signals an error.

Each filter is allowed to modify the data string before control is passed to the next function. The order of the queue is based on the time that the filter is registered, with filters being applied from most-recently registered to least-recently-registered. Each filter consists of three items: a search pattern (currently, only case-sensitive and case-insensitive exact-string matches are available), an “action specifier”, which specifies which queue the filter should be added to, and what sort of action the filter should take when a match is found (string replace or run a supplied function), and a data element, which can be a replacement string or a pointer to a programmer-supplied function.

4.2 Allowing the co-processor to serve pages

To allow the co-processor to service client requests itself, rather than passing along filtered content from the web browser, we have specified a third type of filter, which completely replaces the server output with the output of a programmer-supplied function. These filters are called session filters, and they allow the co-processor to begin it’s own dialogue with the client by creating it’s own content. The server is completely unaware of this re-writing, because it must trust the co-server to encrypt the SSL pages. One example of this re-writing occurs in Figure 4.16, where the server believes that it is instructing the client to prepare one frameset, while the co-server actually sends a different frameset (to establish the security connection).

Session filters exist in both the input and output filters, although they only match on the output filter if their input filter matched. Furthermore, if a session filter matches on the output, no other output filters are run. Effectively, if the input function on the session filter accepts the session, then the filter software provides a direct session with the client’s web browser, unaffected by the surrounding context. Although the session filter acts as if it has a direct connection to the client, it must still run as an input and an output filter, connecting to the host web server between the two.

This intermediate connection to the web server can also be used to exchange other data. For example, in Figure 4.14, the request for the server to provide `/null.html` could be replaced with a call to a server-side cgi which would look up the public keys of the message recipients for an encrypted message. In the example transactions that follow, most of the transactions in which the co-server requests `null.html` represent the result of a session filter.

4.3 Indicating the status of the connection

Because there is no clear way for the user to be certain as to whether the content they are seeing in the web browser window is a provided by the apache web server or output from the 4758 co-server, we needed to find some way to present an "unforgable" graphical tag which would indicate to the user whether the page(s) they were viewing had originated on the (untrusted) host or from the trusted co-server. Without this information, the host could simply forge a public-key login window, and use this information to launch a man-in-the-middle attack by connecting back to the co-processor and using the password to later connect to the co-server and sign arbitrary data.

While working with SquirrelMail, we realized that HTML 4.0 Frames allowed us to enclose the host's "access area" by limiting it to one frame in a frameset. By not naming the other frames, and by removing any host-provided links which had a `TARGET="_top"` specification, the host would have no way to refer to other content in the browser window beyond its own frame. By using further co-server filters registered before (thus executed after) the `TARGET="_top"` filter, we could selectively set the `TARGET` specification on links which referenced the co-server to allow us to update the "security status" frame of the web interface.

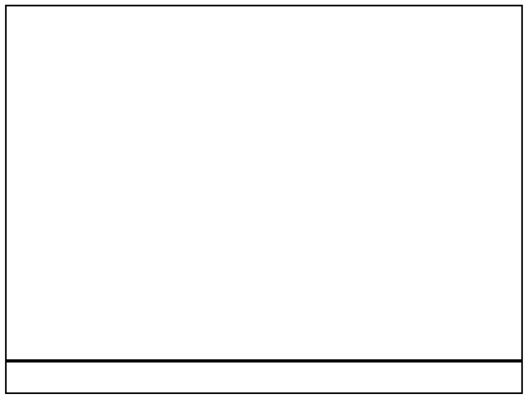
4.3.1 Session example

To illustrate the way in which this filtering scheme would would with the secure mail application, Figures 4.2 – 4.16 provide a graphical representation of the user's screen and the corresponding HTTP traffic between the client, co-server, server, and back. The sequence represented here roughly corresponds to the following sequence:

- A user logging in
- Reading a message which is encrypted with their public key

- Sending a signed message

In particular, the stages of navigation within the upper (content) frame that occur entirely on the host are sketched, because the possible richness of a web-based mail system would be more than could be sketched in this format..

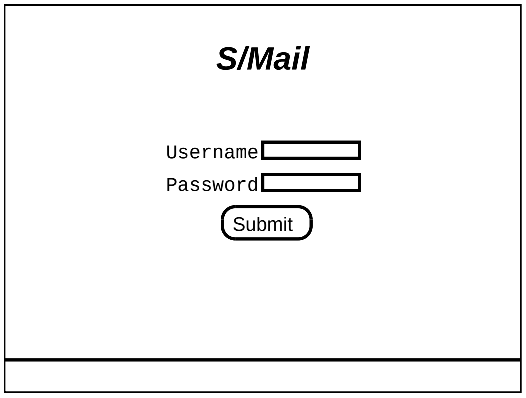


```
Client       GET / HTTP/1.0
Co-server:   GET /null.html HTTP/1.0
Server:      Content-Type:  text/html


Co-server:   Content-Type:  text/html

             <HTML>
             <HEAD><TITLE>S/Mail</TITLE></HEAD>
             <FRAMESET COLS=*,60>
             <FRAME SRC="apachelogin.html" NAME="_toy">
             <FRAME SRC="insecure">
             </FRAMESET>
             </HTML>
```

Figure 4.2: The initial request/response filter



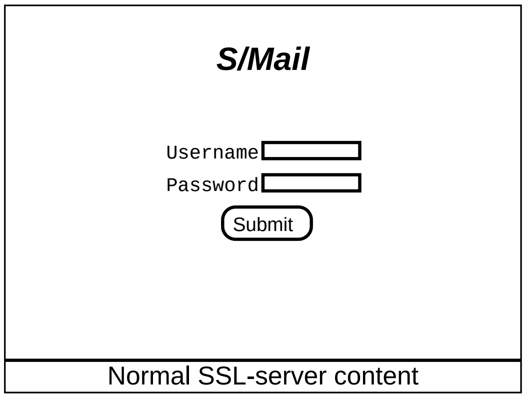
```
Client       GET /apachelogin.html HTTP/1.0
Co-server:   GET /apachelogin.html HTTP/1.0
Server:      Content-Type:  text/html

             <HTML>
             <HEAD><TITLE>S/Mail</TITLE></HEAD>
             <BODY>
             <H1 ALIGN="center">S/Mail</H1>
             <FORM ACTION="main.php" METHOD="get" TARGET="_top"> Username:
             <INPUT TYPE="text" NAME="user"><BR>
             Password:  <INPUT TYPE="password" NAME="pass"><BR>
             <INPUT TYPE=submit NAME=login> </FORM>
             </BODY>
             </HTML>

Co-server:   Content-Type:  text/html

             <HTML>
             <HEAD><TITLE>S/Mail</TITLE></HEAD>
             <BODY>
             <H1 ALIGN="center">S/Mail</H1>
             <FORM ACTION="main.php" METHOD="get"> Username:  <INPUT
             TYPE="text" NAME="user"><BR>
             Password:  <INPUT TYPE="password" NAME="pass"><BR>
             <INPUT TYPE=submit NAME=login> </FORM>
             </BODY>
             </HTML>
```

Figure 4.3: The server's login page

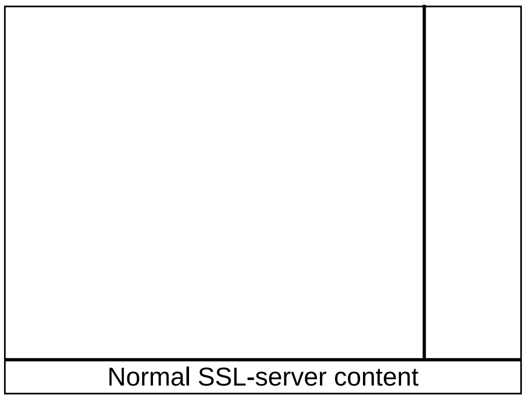


```
Client       GET /insecure HTTP/1.0
Co-server:   GET /null.html HTTP/1.0
Server:      Content-Type:  text/html


Co-server:   Content-Type:  text/html

             <HTML>
             <HEAD><TITLE>Insecure (SSL-only) page</TITLE></HEAD>
             <BODY>
             <STRONG>Normal SSL-server content</STRONG>
             </BODY>
             </HTML>
```

Figure 4.4: Loading the insecure service banner from 4.2



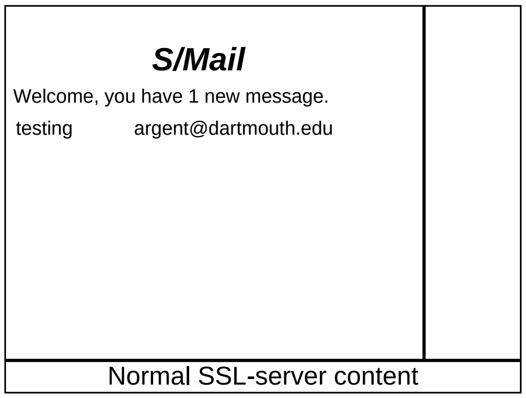
```
Client      GET /main.php?user=user&pass=FooBAr HTTP/1.0
Co-server:  GET /main.php?user=user&pass=FooBAr HTTP/1.0
Server:     Content-Type:  text/html

            <HTML>
            <HEAD><TITLE>S/Mail</TITLE></HEAD> <FRAMESET ROWS=*,100>
            <FRAME SRC="mailbox.php?new" NAME="content">
            <FRAME SRC="sidebar.php" NAME="sidebar">
            </FRAMESET>
            </HTML>

Co-server:  Content-Type:  text/html

            <HTML>
            <HEAD><TITLE>S/Mail</TITLE></HEAD> <FRAMESET ROWS=*,100>
            <FRAME SRC="mailbox.php?new" NAME="content">
            <FRAME SRC="sidebar.php" NAME="sidebar">
            </FRAMESET>
            </HTML>
```

Figure 4.5: S/Mail loading apache server contents


S/Mail
Welcome, you have 1 new message.
testing argent@dartmouth.edu
Normal SSL-server content

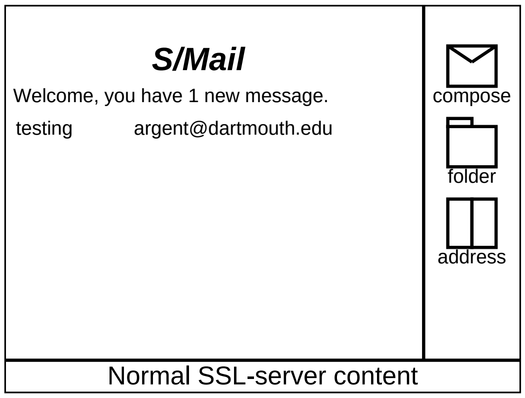
```
Client      GET /mailbox.php?new HTTP/1.0
Co-server:  GET /mailbox.php?new HTTP/1.0
Server:     Content-Type:  text/html

            <HTML>
            <BODY> <H1 ALIGN="center">S/Mail</H1>
            Welcome, you have 1 new message.
            <TABLE>
            <TR><TD><A HREF="smime?message=1">testing</A></TD>
            <TD>argent@dartmouth.edu</TD></TR>
            </TABLE>
            </BODY>
            </HTML>

Co-server:  Content-Type:  text/html

            <HTML>
            <BODY> <H1 ALIGN="center">S/Mail</H1>
            Welcome, you have 1 new message.
            <TABLE>
            <TR><TD><A HREF="smime?message=1" TARGET="_top">testing</A></TD>
            <TD>argent@dartmouth.edu</TD></TR>
            </TABLE>
            </BODY>
            </HTML>
```

Figure 4.6: S/Mail loading front page


S/Mail
Welcome, you have 1 new message.
testing argent@dartmouth.edu
compose
folder
address
Normal SSL-server content

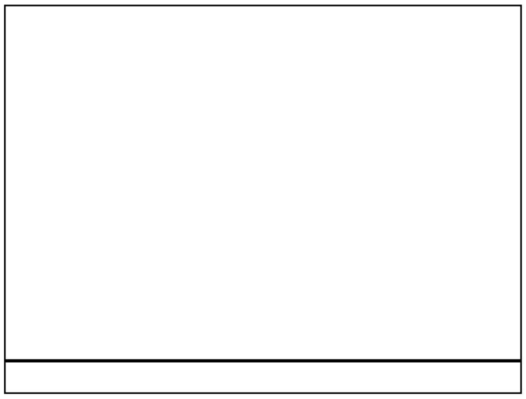
```
Client      GET /sidebar.php HTTP/1.0
Co-server:  GET /sidebar.php HTTP/1.0
Server:     Content-Type:  text/html

            <HTML>
            <BODY>
            <A HREF="compose.php" TARGET="content">
            <IMG SRC="compose.gif"><BR>compose</A><BR>
            <A HREF="folders.php" TARGET="content">
            <IMG SRC="folders.gif"><BR>folders</A><BR>
            <A HREF="addresses.php" TARGET="content">
            <IMG SRC="addresses.gif"><BR>addresses</A><BR>
            </BODY>
            </HTML>

Co-server:  Content-Type:  text/html

            <HTML>
            <BODY>
            <A HREF="compose.php" TARGET="content">
            <IMG SRC="compose.gif"><BR>compose</A><BR>
            <A HREF="folders.php" TARGET="content">
            <IMG SRC="folders.gif"><BR>folders</A><BR>
            <A HREF="addresses.php" TARGET="content">
            <IMG SRC="addresses.gif"><BR>addresses</A><BR>
            </BODY>
            </HTML>
```

Figure 4.7: Loading the S/Mail sidebar

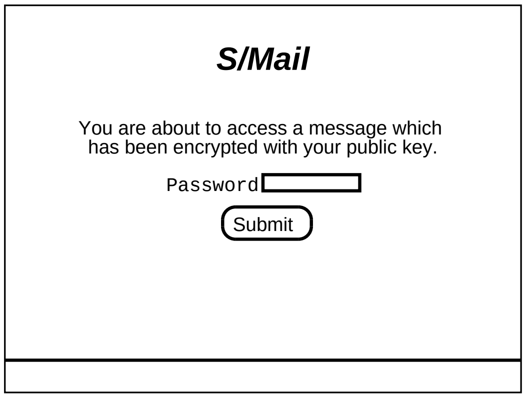


```
Client      GET /smime?message=1 HTTP/1.0
Co-server:  GET /null.html HTTP/1.0
Server:     Content-Type:  text/html


Co-server:  Content-Type:  text/html

            <HTML>
            <HEAD><TITLE>S/Mail</TITLE></HEAD>
            <FRAMESET COLS=*,60>
            <FRAME SRC="smime_decode_pass?message=1" NAME="_toy">
            <FRAME SRC="secure">
            </FRAMESET>
            </HTML>
```

Figure 4.8: Accessing private-key operations

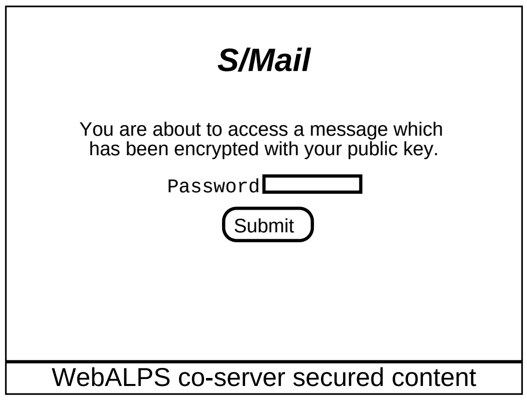

S/Mail
You are about to access a message which has been encrypted with your public key.
Password
Submit

Client:	`GET /smime_decode_pass?message=1 HTTP/1.0`
Co-server:	`GET /null.html HTTP/1.0`
Server:	`Content-Type: text/html`

Co-server:

```
Content-Type:  text/html
<HTML>
<BODY>
<H1 ALIGN="center">S/Mail</H1>
You are about to access a message which has been encrypted
with your bublic key</BR>
<FORM ACTION="smime_display?message=1" METHOD="get">
Password:  <INPUT TYPE="password" NAME="pass"><BR>
<INPUT TYPE="submit">
</FORM>
</BODY>
</HTML>
```

Figure 4.9: Private key verification



You are about to access a message which has been encrypted with your public key.

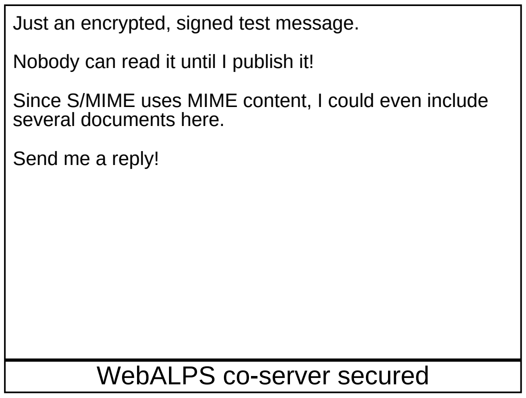
WebALPS co-server secured content

```
Client      GET /secure HTTP/1.0
Co-server:  GET /null.html HTTP/1.0
Server:     Content-Type:  text/html


Co-server:  Content-Type:  text/html

            <HTML>
            <HEAD><TITLE>WebALPS secured page</TITLE></HEAD>
            <BODY>
            <STRONG>WebALPS co-server secured content</STRONG>
            </BODY>
            </HTML>
```

Figure 4.10: Displaying the secure banner while decoding messages


Just an encrypted, signed test message.

Nobody can read it until I publish it!

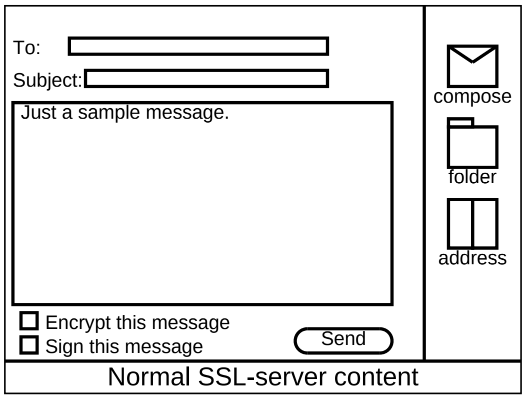
Since S/MIME uses MIME content, I could even include several documents here.

Send me a reply!

WebALPS co-server secured

Client:	`GET /smime_display?message=1 HTTP/1.0`
Co-server:	`GET /download.php?message=1 HTTP/1.0`
Server:	`Content-Type:  application/pkcs7-mime`
	`encrypted attachment data`
Co-server:	`Content-Type:` *`decrypted content-type`*
	`decrypted content`

Figure 4.11: Displaying the encrypted content



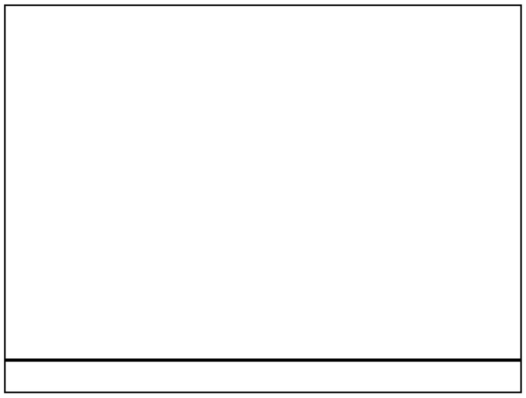
```
Client:     GET /compose.php HTTP/1.0
Co-server:  GET /compose.php HTTP/1.0
Server:     Content-Type: text/html

            <HTML><BODY>
            <FORM ACTION="/mail.php" METHOD="get">
            To: <INPUT TYPE="text" NAME="to"><BR>
            Subject: <INPUT TYPE="text" NAME="subj"><BR>
            <TEXTAREA NAME="mess"></TEXTAREA><BR>
            <INPUT TYPE="checkbox" NAME="enc">Encrypt this message<BR>
            <INPUT TYPE="checkbox" NAME="sign">Sign this message<BR>
            <INPUT TYPE="submit" NAME="Send">
            </FORM>
            </BODY></HTML>

Co-server:  Content-Type: text/html

            <HTML><BODY>
            <FORM ACTION="/mail.php" METHOD="get" TARGET="_top">
            To: <INPUT TYPE="text" NAME="to"><BR>
            Subject: <INPUT TYPE="text" NAME="subj"><BR>
            <TEXTAREA NAME="mess"></TEXTAREA><BR>
            <INPUT TYPE="checkbox" NAME="enc">Encrypt this message<BR>
            <INPUT TYPE="checkbox" NAME="sign">Sign this message<BR>
            <INPUT TYPE="submit" NAME="Send">
            </FORM>
            </BODY></HTML>
```

Figure 4.12: The standard compose window (from 4.7)

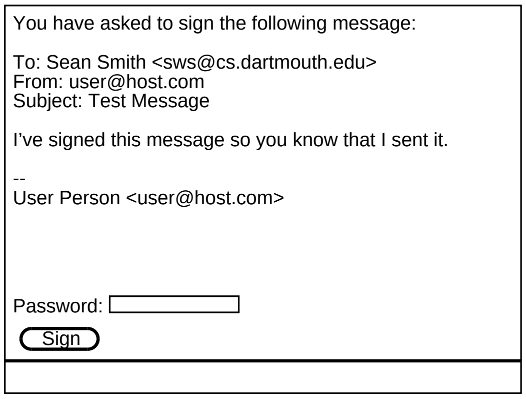


```
Client      GET /mail.php?to=user&subj=subject&...&mess=message HTTP/1.0
Co-server:  GET /null.html HTTP/1.0
Server:     Content-Type:  text/html


Co-server:  Content-Type:  text/html

            <HTML>
            <HEAD><TITLE>S/Mail</TITLE></HEAD>
            <FRAMESET COLS=*,60>
            <FRAME SRC="send?to=user&subj=subject&...&mess=message"
            NAME="_toy">
            <FRAME SRC="secure">
            </FRAMESET>
            </HTML>
```

Figure 4.13: Intercepting a send mail message


You have asked to sign the following message:

To: Sean Smith <sws@cs.dartmouth.edu>
From: user@host.com
Subject: Test Message

I've signed this message so you know that I sent it.

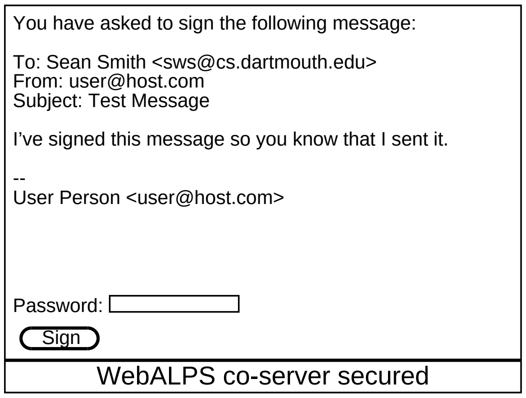
--
User Person <user@host.com>

Password:

```
Client:     GET /send?to=user&subj=subject&...&mess=message HTTP/1.0
Co-server:  GET /null.html HTTP/1.0
Server:     Content-Type:  text/html

Co-server:  Content-Type:  text/html
            <HTML>
            <BODY>
            You have asked to sign the following message:<BR>
            <FORM ACTION="/smime_send" METHOD="get">
            <PRE>
            To:  user
            From:  user@host.com
            Subject:  subj
            message
            </PRE>
            Password:  <INPUT TYPE="password" NAME="pass"><BR>
            <INPUT TYPE="submit">
            </FORM>
            </BODY>
            </HTML>
```

Figure 4.14: Verifying message signing


You have asked to sign the following message:

To: Sean Smith <sws@cs.dartmouth.edu>
From: user@host.com
Subject: Test Message

I’ve signed this message so you know that I sent it.

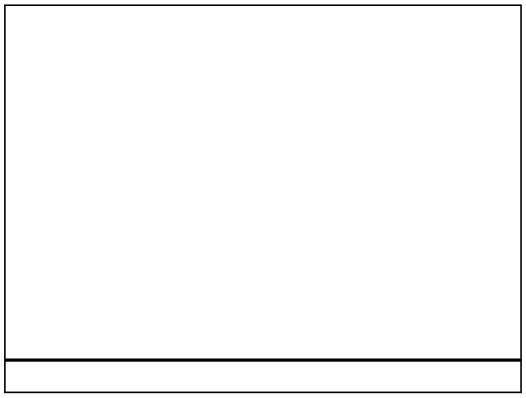
--
User Person <user@host.com>

```
Client      GET /secure HTTP/1.0
Co-server:  GET /null.html HTTP/1.0
Server:     Content-Type:  text/html


Co-server:  Content-Type:  text/html

            <HTML>
            <HEAD><TITLE>WebALPS secured page</TITLE></HEAD>
            <BODY>
            <STRONG>WebALPS co-server secured content</STRONG>
            </BODY>
            </HTML>
```

Figure 4.15: Displaying the secure banner for message signing



```
Client       GET /smime_send?pass=pass HTTP/1.0
Co-server:   GET /mail.php?to=user&subj=subject&...&signed message
             HTTP/1.0
Server:      Content-Type:  text/html

             <HTML>
             <HEAD><TITLE>S/Mail</TITLE></HEAD>
             <FRAMESET ROWS=*,100>
             <FRAME SRC="mailbox.php?new" NAME="content">
             <FRAME SRC="sidebar.php" NAME="sidebar">
             </FRAMESET>
             </HTML>

Co-server:   Content-Type:  decrypted content-type

             <HTML>
             <HEAD><TITLE>S/Mail</TITLE></HEAD>
             <FRAMESET COLS=*,60>
             <FRAME SRC="main.php" NAME="_toy">
             <FRAME SRC="insecure">
             </FRAMESET>
             </HTML>
```

Figure 4.16: Returning to the main (insecure) page

Chapter 5

Results

5.1 S/MIME implementation

Due to the complexity of the S/MIME specification and the time constraints of this researcher, it was not possible to produce an implementation of the S/MIME mail client running on the 4758. Several factors combined to prevent this outcome; I have attempted to categorize the barriers from fundamental to awkward. A programmer attempting to implement this mailer would be advised to familiarize him- or her-self with these problems before attempting to implement the S/MIME functionality.

5.1.1 Operating system support

Although the embedded CP/Q operating system on the 4758 supports a rich variety of cryptography primatives, many of the other operating system limitations (such as the lack of any file system functions or networking primitives) make it difficult to move software from host-side code to coprocessor-code. Also, CP/Q supports only a (mostly-ANSI) subset of the C programming language, reducing the possible range of tools for programming to only the IBM toolset with one of three or four supported compilers. Because of the embedded nature of CP/Q, the on-card debugging tools are also much less featureful than traditional debuggers.

5.1.2 S/MIME specification

The four RFCs which make up the S/MIME specification [9, 16, 14, 15] total 118 pages, which includes several pages of ASN.1 definitions for the Cryptographic Message Syntax. CMS further relies on X.509 certificates, which are detailed in the 129 pages of RFC 2459 [10]. The ASN.1 format which is used by CMS is very detailed, and is capable of expressing arbitrary binary data in variety of "compact" forms based on a textual specification. Unfortunately, the flexibility of ASN.1 makes it difficult to write a parser for X.509, necessitating the use of a library to process the X.509 and CMS formats.

5.1.3 Library support

Many of the libraries listed in Section 3.1 are implemented in C++, at least in part. In particular, the arbitrary length of most ASN.1 representations and underlying data makes the use of an object-oriented language attractive to the developer. However, because CP/Q does not support C++ development tools, these libraries are almost completely worthless, except as references on the standards.

Of the remaining libraries, only cryptlib has been ported to the 4758. All of the other libraries would require various degrees of re-writing to be able to compile in the file-descriptor-less world of CP/Q. While Cryptlib has been built to run on the CP/Q, it has not been extensively tested, and some of the major functionality (such as keysets, which are used to store public and private keys) does not have any way to operate with it's backing store inside the 4758. Furthermore, Peter Gutmann, the developer of cryptlib, does not consider the CP/Q inside the 4758 to be a "supported environment", and there are currently no instructions on building and linking cryptlib for use on the 4758 [7].

5.2 Future Directions

As it currently exist, the S/MIME implementation for the secure mail project consists of a few utilities written in C for the linux operating system which have been compiled with cryptlib, but which have not been tested. In this programmer's experience, the process which would likely be most fruitful in producing a working S/MIME mailer running on the 4758 would begin with a working set of host S/MIME utilities in written in C.

These utilities can be used to test the S/MIME implementation and to discover any library problems before the application reaches the 4758 environment. These utilities should be written in such a way that they perform most of their S/MIME operations on memory buffers not associated with a file. This method of operation mimicks the environment of the 4758.

For testing purposes, it may also be useful to write some host-side scripts with the names and functionality of which will later be ported to the card side. The utilities from the previous step should be used here, to test both the implementation details and correct operation of the utilities.

Because the co-server filters can be used to prevent (override) access to files (or arbitrary URL patterns), filters can be written to intercept and overwrite the host-side processing directives as each utility is ready to be ported into the card. For the initial implementations, it may be useful to print the input and output from the card to a debugging channel, particularly if the results from the card are unexpected.

Obviously, before the co-server filters can actually be directed at the working S/MIME implementation, any necessary support libraries must be ported to run on the 4758. Unfortunately, this author cannot give much more advice than this. If cryptlib is chosen for this effort, it is possible that cryptlib's keyset support could be modified to allow a memory buffer to act as a backing store. If another library is selected, research should be done as to the best way to remove file and socket dependencies from the library.

Chapter 6

Conclusions

Although this thesis work has not resulted in a working public-key e-mail system implemented on the IBM 4758, it has led to several conclusions about the software engineering process and software design. These conclusions are based on the obstacles listed in Chapter 5.1.

6.1 Library support

One of the barriers to the use of several potentially useful libraries in this project was the limitation that the IBM 4758 development toolkit could only support a subset of ANSI C and the cryptographic functions provided by the modified CP/Q operating system. Although the 4758 is quite limited, it can certainly provide an example of the sort of environment which might be encountered when working on an embedded hardware platform.

If library writers want their software to be usable on embedded systems, they should write their libraries to the minimum set of C features necessary, and provide functions which use memory buffers for input and output, rather than using file descriptors. Although this abstraction might reduce speed slightly, it has the additional advantage that the C code (separated from the input/output routines) could also be used in other applications which do not provide a file descriptor, such as some network packages and device I/O. By using more recent languages (such as C++) and less portable features (files instead of memory buffers), library developers may exclude their library from consideration in non-traditional applications.

6.2 Standards design

After the decision to support S/MIME over OpenPGP was made, I did more investigation into the specifications of S/MIME. The Cryptographic Message Syntax which is used for S/MIME allows for a large range of encryption methods and optional attributes.

Many of the data types in the CMS format are defined as an ASN.1 SEQUENCE, which (approximately) consists of a data length followed by the concatenated representations of each object. To make the data representations shorter, often attributes are specified as OPTIONAL, which requires the parsing routines to infer the existence or non-existence of those attributes by the existence or absense of that attribute at the appropriate location in the SEQUENCE. Because of this representation, any element following an OPTIONAL attribute in a SEQUENCE must have an identifying prefix which allows it to be differentiated from preceding OPTIONAL elements. This additional complexity is due to the way in which ASN.1 attempts to provide both flexible and compact data storage for arbitrary data structures. In contrast, most existing applications specify a much more fixed format for data, which simplifies reading and parsing at the expense of complete generality.

Similarly, to represent data of arbitrary length, ASN.1 provides three methods of encoding the length of an element: two prefix methods, and one arbitrary-length (ended by a terminator) method. This form of encoding (of which recording the length is only part) is known as the Basic Encoding Rules (BER), which applies to all elements, from integers to complex SEQUENCE composites. Although there exists a simplified form of BER known as DER (the Distinguished Encoding Rules) which allows only the shortest encoding, this format is not required by most standards. In contrast, the OpenPGP standard specifies the representation of integers as two big-endian octets representing the exact bit length followed by the number in big-endian form, specifying *exactly* the encoding for all implentations to use. From an implementation point of view, the OpenPGP rules are both more concise and easier to implement.

In general, excess mechanism in standards documents may be considered harmful, as it is rarely possible to correctly predict the direction of future enhancements. On the other hand, by specifying a simple and robust framework without limiting the data contents uneccessarily, it is possible for later designers to expand the original mechanism without breaking backwards compatibility or unnecessarily complicating the mechanism. A good example of this is RFC 822, which attempts to specify the format of message headers while leaving room for later customization (as has repeatedly occurred in the last 19 years).

6.3 WebALPS design

Although the use of the IBM 4758 provides a secure computation platform to host WebALPS, the current implementation of WebALPS shifts much of the burden of establishing and maintaining the SSL connection to the host-side apache, mod_ssl, and OpenSSL software for performance reasons. Unfortunately, the decision to allow the host to fragment the HTTP response before requesting that the co-server encrypt the message may have security implications if a filtered string can be broken into two fragments by the host software, allowing some portion of an unacceptable string to be transmitted to the client. Unfortunately, it might prove difficult to integrate the filtering software with coprocessor-side fragmentation, particularly if the total size of the response is larger than the memory of the coprocessor.

Bibliography

[1] The Internet Mail Consortium. *S/MIME and OpenPGP*. World Wide Web, `http://www.imc.org/smime-pgpmime.html`, 1999.

[2] David H. Crocker. *RFC 822: Standard for ARPA Internet Text Messages*. Internet Society, `ftp://ftp.isi.edu/in-notes/rfc822.txt`, 1982.

[3] N. Freed and N. Borenstein. *RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. Internet Society, `ftp://ftp.isi.edu/in-notes/rfc2045.txt`, 1996.

[4] N. Freed and N. Borenstein. *RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. Internet Society, `ftp://ftp.isi.edu/in-notes/rfc2046.txt`, 1996.

[5] N. Freed and N. Borenstein. *RFC 2047: Multipurpose Internet Mail Extensions (MIME) Part Three: Message Header Extensions for Non-ASCII Text*. Internet Society, `ftp://ftp.isi.edu/in-notes/rfc2047.txt`, 1996.

[6] Freshmeat. World Wide Web, `http://www.freshmeat.net/`.

[7] Peter Gutmann. Personal communication. Discussion on porting cryptlib inside the 4758.

[8] Peter Gutmann. Cryptlib 3.0 beta, 2001.

[9] R. Housley. *RFC 2630: Cryptographic Message Syntax*. Internet Society, `ftp://ftp.isi.edu/in-notes/rfc2630.txt`, 1999.

[10] R. Housley, W. Ford, W. Polk, and D. Solo. *RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile.* Internet Society, `ftp://ftp.isi.edu/in-notes/rfc2459.txt`, 1999.

[11] Shan Jiang. WebALPS implementation and performance analysis: Using trusted co-servers to enhance privacy and security of web interactions. Master's thesis, Dartmouth College, 2001.

[12] Jonathan B. Postel. *RFC 821: Simple Mail Transfer Protocol.* Internet Society, `http://www.rfc-editor.org/rfc/rfc821.txt`, 1982.

[13] Proceedings, 22nd National Information Systems Security Conference. *Validating a High-Performance, Programmable Secure Coprocessor*, 1999.

[14] B. Ramsdell, editor. *RFC 2632: S/MIME Version 3 Certificate Handling.* Internet Society, `ftp://ftp.isi.edu/in-notes/rfc2632.txt`, 1999.

[15] B. Ramsdell, editor. *RFC 2633: S/MIME Version 3 Message Specification.* Internet Society, `ftp://ftp.isi.edu/in-notes/rfc2633.txt`, 1999.

[16] E. Rescorla. *RFC 2631: Diffie-Hellman Key Agreement Method.* Internet Society, `ftp://ftp.isi.edu/in-notes/rfc2631.txt`, 1999.

[17] Sean W. Smith. WebALPS: Using trusted co-servers to enhance privacy and security of web interactions. Technical Report Research Report RC-21851, IBM T.J. Watson Research Center, 2000.

[18] Sean W. Smith and Steve Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, (Special Issue on Computer Network Security):31:831–860, April 1999.