Dartmouth College

Dartmouth Digital Commons

Dartmouth College Undergraduate Theses

Theses and Dissertations

5-1-2003

Trusted S/MIME Gateways

Mindy J. Pereira
Dartmouth College

Trusted S/MIME Gateways

Mindy Pereira
Mindy.Pereira.03@alum.dartmouth.org

Senior Honors Thesis: Winter/Spring 2003
Department of Computer Science
Dartmouth College

Advisor: Prof. Sean Smith
sws@cs.dartmouth.edu

Dartmouth Computer Science Technical Report TR2003-461
May 2003

Abstract:

The utility of Web-based email clients is clear: a user is able to access their email account from any computer anywhere at any time. However, this option is unavailable to users whose security depends on their key pair being stored either on their local computer or in their browser. Our implementation seeks to solve two problems with secure email services. The first that of mobility: users must have access to their key pairs in order to perform the necessary cryptographic operations. The second is one of transition: initially, users would not want to give up their regular email clients. Keeping these two restrictions in mind, we decided on the implementation of a secure gateway system that works in conjunction with an existing mail server and client. Our result is PKIGate, an S/MIME gateway that uses the DigitalNet (formerly Getronics) S/MIME Freeware Library and IBM's 4758 secure coprocessor. This thesis presents motivations for the project, a comparison with similar existing products, software and hardware selection, the design, use case scenarios, a discussion of implementation issues, and suggestions for future work.

Acknowledgements

This thesis would not have been possible without the kind assistance of many people. First and foremost, thank you to Grace, Esther, Kristen, and Reid for reading and editing the many incarnations of this thesis and participating in my use case studies. Thanks to Neha for the daily sanity checks and emergency food and beverage supplies. A big thank you to Alex and John for all the help with coding, Linux, and the many other problems I encountered. Also, thank you Mike and Ling for your help with Sherlock. And last, but certainly not least, thank you to Professor Sean Smith for giving me an interesting problem to work on and all the support I needed to make things happen.

1. Preliminaries

1.1 Introduction

Currently, a person using an email program that allows them a degree of mobility must choose between convenience and security. The utility and convenience of Web-based mailers is clear: they give a user the ability to operate her email account from any Internet-connected computer at any time. However, if a user is tied to security that depends upon her key pair being stored either on her local computer or in her browser, she loses the ability to send or receive secure mail while mobile. Our solution for making public-key cryptography work for mobile email is the use of a secondary trusted server that stores and manages the user's key pair and performs all operations associated with user authentication, encryption and decryption of text, and the addition and verification of digital signatures. PKIGate fills the role of the trusted server, providing a set of S/MIME and secure cryptographic services while still allowing the user to retain the use of her preferred client.

1.2 Organization

This thesis is organized into six sections. The first section introduces the problem, its motivations, and the interesting case of email at Dartmouth. Section 2 provides information on the MIME and S/MIME standards, compares existing technologies with the design for ours, and explains the technical concepts and technologies we plan to use. The third section discusses the thought process that went into selecting software and hardware for the project as well as describing the software we used in our final design.

Section 4 lays out the hi- and lo-level design for the entire PKIGate mail system. Use case scenarios for the design from Section 4 are diagramed and detailed in Section 5. We conclude in Section 6 with a discussion of implementation difficulties, suggestions for future work, and an evaluation of our solution and the project as a whole.

1.3 Motivation

Since public key cryptography requires no prior contact between sending and receiving parties, as long as the recipient's public key information is published, it is an ideal solution for email. Using other cryptographic standards that depend on the exchange of an agreed upon key requires the implementation of a number of security protocols, with each step opening the key to access by an adversary.

The utility of Public Key cryptography in email situations is easy to see. A user is able to digitally sign email, allowing the recipient to verify the sender. Imagine that the college application process takes a digital and paranoid turn and admissions offices wish to verify every letter of recommendation. If a letter writer uses an email program with public key functionality, he can digitally sign the letter, allowing the admissions office to verify he sent it, and satisfy their paranoia.

The case for encryption is even easier to make, but we will give a particularly pertinent example. Medical offices must now meet Condition 3 of HIPAA, the Health Insurance Portability & Accountability Act, which states that security standards must be put in place "protecting the confidentiality and integrity of 'individually identifiable health information'"[9]. Assume that a doctor at Dartmouth Health Services needed to contact a student with confidential health information. In order for the email communication to be HIPAA compliant, the information must be secured. If there is a certificate server set up on the Dartmouth campus, the doctor can easily access the student's public key and send him an encrypted message without the additional administrative tasks of creating and sharing an encryption key.

1.4 Email at Dartmouth

The email system at Dartmouth College provides a particularly difficult environment in which to offer secure mobile email access. Students, faculty, and staff at the college enjoy the use of Blitzmail, a homegrown email client that allows a user to access his account from any computer on which the program is installed without altering the application. Additionally, Dartmouth is home to at least two incarnations of Web-based mailers, Netblitz and Webblitz. Since a majority of communication on the campus by is done through email, the campus is home to hundreds of public computers, fondly referred to as "Blitz terminals," from which users often check their email.

Obviously in this environment, public key cryptography tied to a user's computer would be inconvenient, at the least. If the user's key needed to be on the machine hosting the email client, we would have two options: either we would be forced to install the key pair of every student, faculty and staff member on every computer, including personal computers, at Dartmouth or we would have to restrict the number of computers a user could operate her email account from. At a minimum, the first option would require the upgrade of the 1,000+ personal computers belonging to the incoming freshmen every year. However, Webblitz and Netblitz give students the ability to check email from anywhere in the world. Unless the student had a way of transporting her key pair, we could not guarantee her secure email services away from the campus. Therefore, a reasonable solution, at least for PKI services at Dartmouth, is to store user's key pairs on a secure central server.

2. Background

Before beginning the discussion of our implementation of PKIGate, it is necessary to provide some background information. A description of the MIME and S/MIME standards, existing technologies, and definitions of terms we will be using throughout this thesis follow.

2.1 Standards

2.1.1 MIME

MIME, or *Multipurpose Internet Mail Extensions*, is a standard for Internet messages that ensures that messages from different email systems are exchanged successfully [19]. Messages in MIME format can contain files of varying types, including types defined by users. The file type is described in a “content-type” header, which the recipient email program (or browser) uses to determine the application to use in opening the enclosure. The binary data that makes up the enclosure is transformed into ASCII text via the base64-encoding scheme.

2.1.2 S/MIME

S/MIME, *Secure/Multipurpose Internet Mail Extensions*, is a standard designed to add authentication and encryption services to MIME formatted email messages [14]. It is based on the X.509 certificate standard and the ASN.1 syntax. In the standard, digital signatures and key transport both depend on public key cryptography. Encryption in S/MIME is a two-stage process referred to as creating a *digital envelope*. First, a symmetric cipher using the *content encryption key (CEK)* is applied to the message. Next, the CEK is encrypted using the recipient's public key. Both the message and encrypted key are sent.

S/MIME Encryption

1) Create CEK
2) Use CEK in symmetric cipher encryption of message
3) Encrypt CEK with recipient's public key
4) Send encrypted CEK and message

Figure 2.1

2.2 S/MIME Gateway

Sending an email message in a simple email system involves several steps. First, the user composes a message in his *email client*, a program installed on his computer that allows him to connect to the mail server [4]. The client sends the message to his *email server*, where the user's account information is stored. There, the message recipient's address is broken into two parts: the user and domain names (Ex: user@domain) [5]. The sender's server forwards the message to the server where *domain* is located. The *domain* server then delivers the message to *user's* account. Later, when that user logs on to his email account, his client will issue a request to have the mail in his account sent to his computer.

A *gateway* is not an email server. Rather, as defined by the marketers of the S/MIME gateway applications we are about to discuss, it serves as a layer between the email server and the outside world that allows the email client to remain unchanged. Outgoing email is piped through the gateway, where S/MIME wrapping and cryptography are applied before the message reaches the outside world. Incoming mail passes through the gateway where S/MIME unwrapping and decryption or validation are applied before the message reaches the mail server.

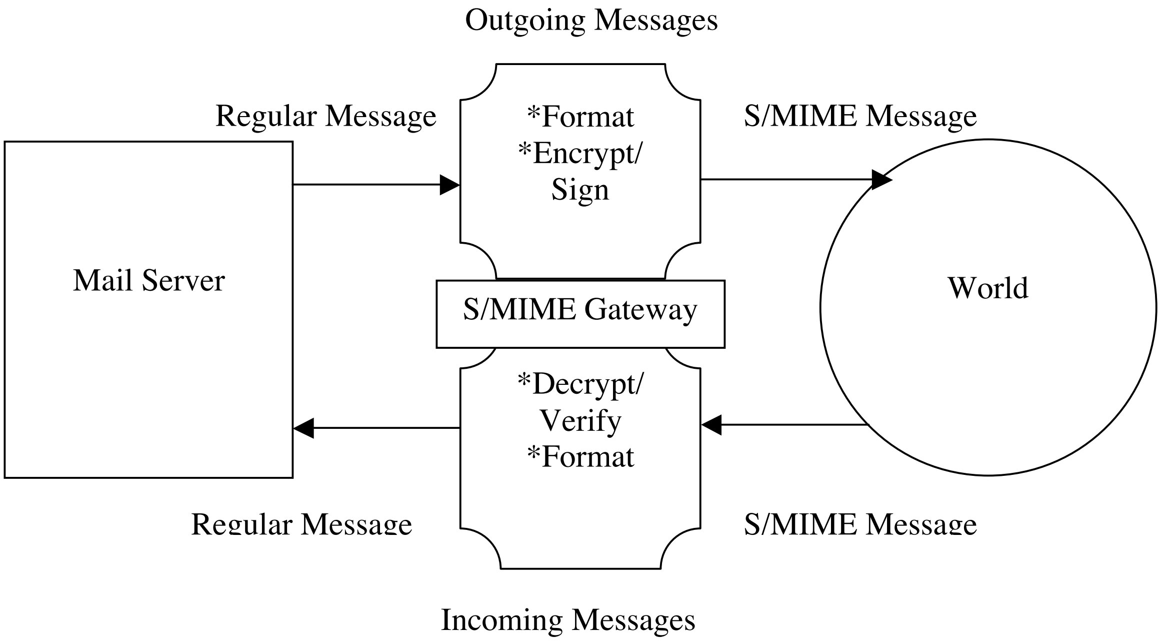

Figure 2.2: S/MIME Gateway

2.3 Existing S/MIME Gateway Applications

Before creating the design for a new S/MIME gateway, we researched existing products similar to the technology we intended to develop. This allowed us to see what features were being offered, how other groups made their designs secure and understandable, and provided some initial ideas on how our own application should be constructed. Before beginning the comparison, it is essential to understand that none of the applications we discovered clearly state how a user's private key or cryptographic operations are made secure. Our PKIGate is the only one that offers a tamper-proof hardware platform to protect sensitive operations and information. The following section introduces some of the major ideas behind the design of our S/MIME gateway-style application, PKIGate, as it relates to existing products, with the actual design to follow in Section 4.

2.3.1 "Seamless" S/MIME Gateway Applications

I will use the term *seamless gateway* to refer to an S/MIME application that the user does not need to know exists. The use cases of these applications are most similar to those corresponding to our `secure` level functionality. For example, the ZipLip *Email Gateway* and BayCorp *MailMarshal* are each responsible for all key management tasks and act irrespective of the client's S/MIME or cryptographic abilities [22, 1]. The gateway application itself decides whether an outgoing message should be signed or encrypted, and automatically decrypts and verifies incoming messages. With *S/MIME Stripper*, all outgoing emails are signed. The first time a user sends an email, *S/MIME Stripper* generates a private key and an X.509 Certificate [15].

PKIGate is different from the applications above in that it is unclear whether or not the user has any control over the application's function. In PKIGate, even a user at the `secure` level has the option of setting account or recipient preferences via the Security Manager (A full discussion of this follows in Section 5).

2.3.2 User controlled S/MIME Gateways

User controlled refers to the ability of the user to affect the way in which the gateway operates. The IAIK *S/MIME Mapper* is an example and signs and encrypts email messages based on user-specified preferences that the user has defined [20]. For decryption, the email client requests messages from the mail-server, at which point *S/MIME Mapper* connects to the server, receives the messages, decrypts those that were encrypted, and verifies signatures. The application generates and attaches a message containing information about authentication or encryption schemes and the validity of the signature, if there was one to verify, before the message is transferred to the email client.

Much like what is described for *S/MIME Mapper*, a user at the `secure` level of PKIGate can set preferences for the entire account. We add information messages to `secure` level user's messages and display them for `paranoid` level users in the Security Manager window, a secure client connected to the application.

2.3.3 Mixed-mode S/MIME Gateways

Applications that fall into this category are closest to our design for PKIGate because they allow the user to select the level of security at which he/she wishes to operate his/her account with the gateway. The CryptoEx *Selflearning Gateway for Email Security* supports both normal users and "end-to-end encryption for special users" via scripting [2]. Emails can be processed according to any set of rules that can be expressed in a script language, such as those based on the message sender. Key pairs are generated on demand, rather than being created in advance by an administrator and manually maintained.

PKIGate will have three levels of security, to be explained later, that allow the application to determine what, if anything, should be done to incoming and outgoing messages. Instead of scripting, however, the user will be allowed to set a series of preferences. Current plans for PKIGate do not call for the automatic generation of certificates or key pairs, and instead assume they will come from outside the application or be managed by someone in an administrative role.

2.4 Secure Coprocessors

A secure coprocessor is a tamper-proof, sealed device with a processor, memory storage, and, (in some cases) fast cryptography support [21]. It is trusted to protect the information stored on it and its computations from a set of physical attacks by an adversary. Any attempt at penetration will result in the erasure of all critical memory. A secure coprocessor serves as the trusted server for PKIGate by providing an area in the system on which to perform sensitive cryptographic operations. For a more in depth explanation of using a secure coprocessor in a practical security solution, see Smith and Weingart's "Using a High-Performance, Programmable Secure Coprocessor" [17].

3. Software and Hardware Selection

When developing any new application, it is essential that the developer research tools, libraries, or other existing ideas that can aid in development. Given the timeline for this project, the growth of S/MIME as an internet standard, and the availability of outside resources, it seemed prudent to spend time investigating the available resources and selecting from amongst them to use in the implementation of PKIGate rather than coding all components from scratch. For fulfilling the requirements of the S/MIME standard, open source options existed in the form of the pre-written utility, OpenSSL S/MIME, or as a set of libraries from DigitalNet that provided all the necessary functionality [13, 6]. For MIME formatting, a number of open source options existed. We chose to explore two: Mpack, a Linux utility, and the S/MIME Freeware MIME library [12, 6]. The design called for a secure platform on which all cryptographic operations could be performed. For that, we chose the IBM 4758 secure coprocessor, used in many other PKI Lab projects. We now discuss the details of these implementation options and their differences and provide the reasons for the selections we eventually made.

3.1 Choosing an S/MIME Implementation: OpenSSL vs. SFL

3.1.1 The OpenSSL S/MIME Utility

OpenSSL is a software package often associated with creating secure socket layers [13]. However, it also contains a utility for handling S/MIME email processing. The S/MIME utility operates via the command line and offers encryption, decryption, signing, and verification of messages. Certificates, which the S/MIME utility uses in performing cryptographic operations, are sent to the application as another input parameter. The utility supports DES, TDES, RC2-40, RC2-64, and RC2-128 (where the number in each of the RC2 algorithm names refers to the key length) algorithms, through its cryptographic library, crypto.

3.1.2 The S/MIME Freeware Library

The DigitalNet Government Solutions (formerly Getronics) S/MIME Freeware Library (SFL) is a set of libraries that, when used in conjunction, provide cryptographic message syntax (CMS) wrapping and can interface with any type of underlying cryptography [6,7]. When used in combination with other DigitalNet libraries, all necessary services for creating an S/MIME application are provided: the DigitalNet Enhanced SNACC library provides ASN.1 encoding and decoding and the Certificate Management Library provides X.509 Certificate Processing. (See Figure 3.1 for the overall architecture).

Cryptographic functionality is provided by loading “instances” of crypto-tokens, in the form of dynamically linked libraries, which can be thought of as sessions with the selected Cryptographic Token Interface Library (CTIL) [8]. A user is allowed multiple active crypto-tokens from which the application selects the instance that provides the appropriate functionality for decrypting or verifying incoming messages. The user (or application) may also set a preference for which underlying crypto-token he/she would like to use in outgoing message operations.

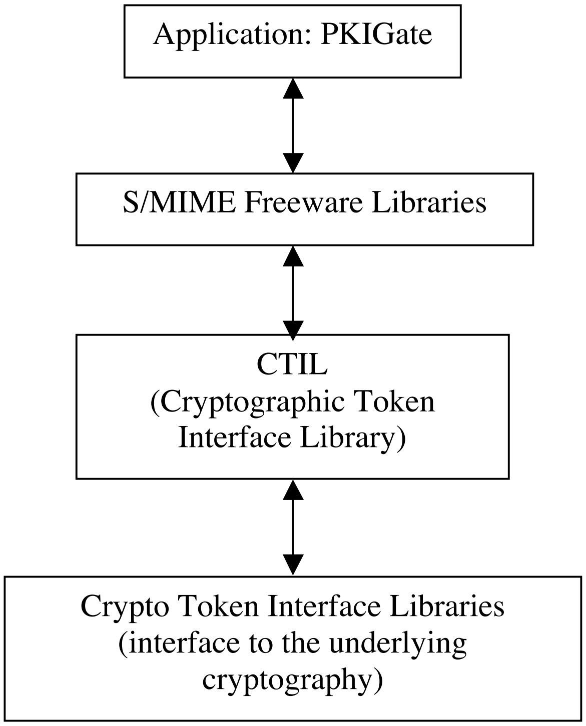
Application: PKIGate
S/MIME Freeware Libraries
CTIL
(Cryptographic Token
Interface Library)
Crypto Token Interface Libraries
(interface to the underlying
cryptography)

Figure 1. S/MIME Freeware Library Architecture
(Note: only those pieces from the library essential to our implementation are shown)

3.1.3 Selection of SFL

There were two main factors in the decision to use the S/MIME Freeware Library (SFL) rather than the OpenSSL S/MIME utility. OpenSSL offered an entire S/MIME application, which in other cases may have proved the ideal solution. However, OpenSSL's S/MIME processing is dependent upon its crypto library. In order to use our own underlying cryptographic system, the actual classes and code responsible for processing S/MIME message would have to be separated from the rest of the code base, and calls to their library replaced with calls to ours. Here, SFL's modular design was a major advantage. SFL operates independently of the underlying cryptography. The only necessary change to the library would be the implementation of a Cryptographic Token Interface to communicate with our underlying cryptographic operations.

Additionally, OpenSSL is more than just a tool for processing and creating S/MIME messages. It provides many other services that are not required for the application, and, given the restrictions placed on the application size by the limited memory space of the 4758, it would have been necessary to remove all unnecessary functionality from OpenSSL. With SFL, the library provides only the functionality we need, with the option of plugging in extras, such as Access Control.

3.2 MIME Formatting: Mpack vs. SFL MIME

3.2.1 Mpack

Mpack is a Linux utility operated via the command line [12]. The user inputs a set of parameters, including the recipient or newsgroup, and the file to be encoded. That file is encoded in one or more MIME messages, which are then mailed to recipients, posted to a newsgroup, or written to an output file.

3.2.2 The SFL MIME Library

The SFL MIME library is a set of open source classes included as part of the test code of the S/MIME Freeware Library [6]. Creating a MIME message consists of instantiating a MIME object, filling in data using either the constructors or set methods, and then allowing the SFL MIME library functions to determine content type and produce the message's MIME headers. Once a message object is created, it can be sent to a data stream and written to a file or to the screen.

3.2.3 Selection of SFL MIME

The first cause for choosing SFL MIME over Mpack for our implementation is similar to the first reason that made us choose the SFL over OpenSSL. Mpack, like OpenSSL S/MIME, includes a set of functions not needed by the application we desire to produce; it would have been necessary to tease out only the pieces of the application we actually needed. Additionally, SFL MIME is featured in the test cases for the SFL, not only proving the two can interact, but also providing examples of how to properly use the libraries.

3.3 Hardware: The Secure Coprocessor

Using the IBM 4758 secure coprocessor was a nearly obvious choice. It provides us with a secure place to perform all the necessary cryptographic operations, one of the goals described for our application in Section 1. The 4758 was successfully tested and loaded with a Linux kernel, it is possible to run a C++ application inside the card, and it provides fast hardware support for cryptographic operations [10]. Communication is easily performed over socket connections between the host and card applications. However, the 4758 imposed one major restriction, which spawned difficulties to be discussed later.

Namely, there is an upper limit on the size of the application that can reside on the card of about 1.8M.

4. Design

The design of PKIGate and its place in an overall email system that follows is one of the many possibilities that may have worked given the goals of our project. The design attempts to minimize time and effort costs to users who do not wish to use the new S/MIME functionality, while still providing those with the desire for privacy the means to have it. The section begins with a hi-level description of all the components in the system and follows with design detail for each, where communication between the various parts is explained.

4.1 Components Overview

PKIGate is designed as an add-on to an existing email system and requires that an email server and client already be in place. Instead of being located between the mail server and the outside world like the gateway described in Section 2.2, PKIGate is designed to work in direct conjunction with the mail server. This helps to eliminate some of the time costs to users who do not wish to utilize the system in some cases or at all. Because of this, the normal email server must be reconfigured to communicate over a secure connection with PKIGate in addition to its connection to the outside world. It is also important to note that the application is not designed for use by individual users, but is instead intended for use as part of an institutional mail system with a central mail server and an administrator in charge of account management.

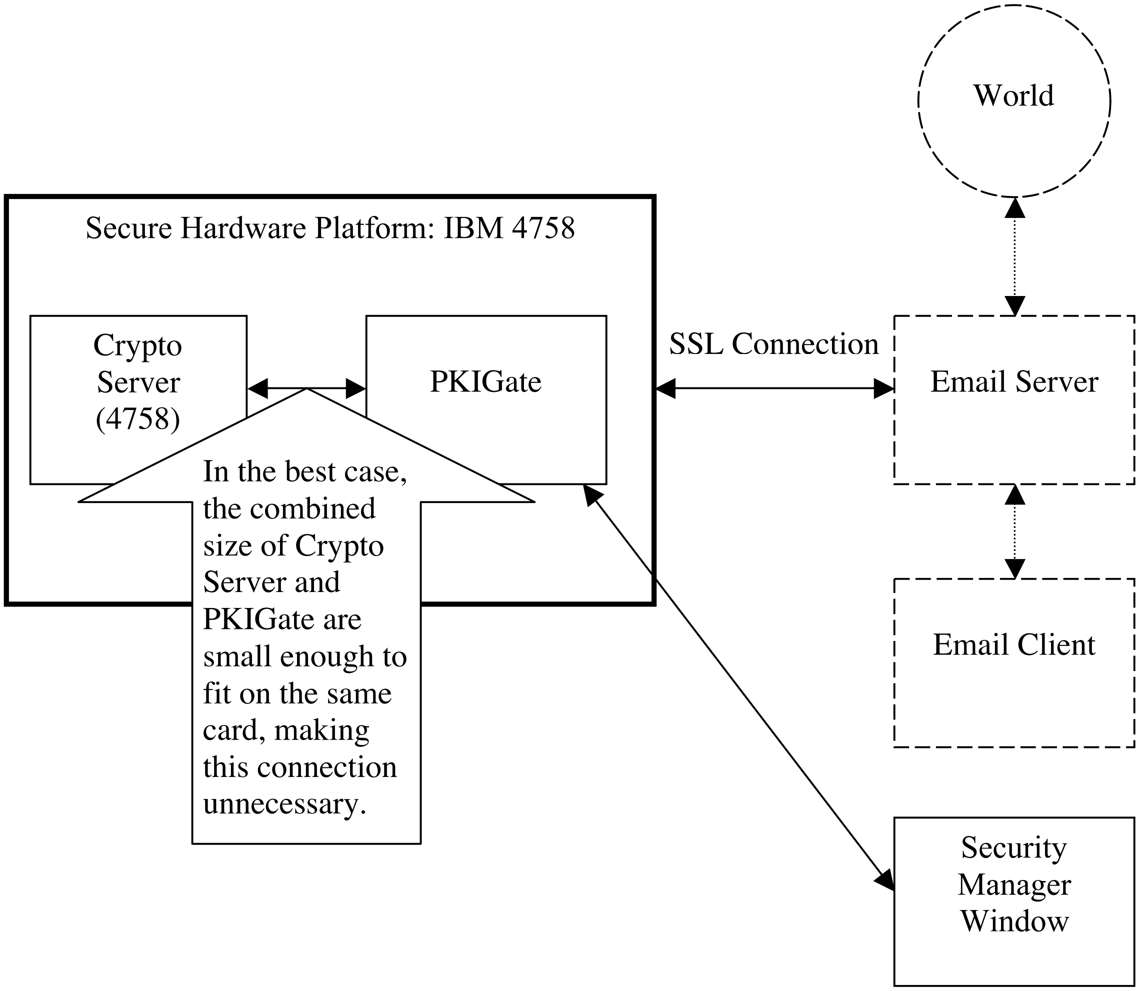
World
Secure Hardware Platform: IBM 4758
Crypto Server (4758)
PKIGate
SSL Connection
Email Server
Email Client
Security Manager Window
In the best case, the combined size of Crypto Server and PKIGate are small enough to fit on the same card, making this connection unnecessary.

Figure 4.1: High-level design of the PKIGate System. Dashed lines represent applications and connections for which we cannot guarantee security.

PKIGate adds three additional components to the existing system: the PKIGate application, Crypto Server, and the Security Manager Window client. PKIGate is the manager of the S/MIME system: it handles requests from the mail server and Security Manager Window for all accounts. The Security Manager Window is a direct secure connection to PKIGate. It allows a user to interact with his/her gateway account, set preferences, update his/her address book, and, in the case of `paranoid` level users, the Security Manager Window provides a direct connection to trusted hardware through which to securely send and view sensitive messages. Crypto Server accesses the 4758 secure coprocessor's built-in cryptography hardware support.

4.2 PKIGate

PKIGate performs the actual functionality to meet the S/MIME standard. The Getronics S/MIME Freeware Library provides S/MIME functionality for the gateway, with cryptographic operations currently offloaded to Crypto Server, and MIME formatting supplied through the SFL MIME Library. Additionally, PKIGate manages all system user accounts. Each user account is kept in an `Account` object, identified by a unique identification (UID) number based on some account setting in the original email system. Eventually, we hope to move PKIGate and the cryptographic functionality onto the same secure hardware device. (For more on this, see Section 6.1.1). For now, the connection between PKIGate and the Crypto Server will be private.

4.2.1 Desired Hardware

In the interest of keeping all text and key pairs secure, our design calls for PKIGate to reside on the 4758 secure coprocessor. Outside applications would communicate with the application through either a private or SSL connection. (Information on difficulties with the implementation follows in section 6.1.1).

4.2.2. Account

An `Account` Object holds all information pertaining to a user. In an unset account, all preferences are set to the defaults (minimum security) and the user's address book is initially empty. Upon login, the user is able to set a SecurityLevel (`secure` or `paranoid`), preferences for his/her account, and his/her address book. (For more information on how these preferences affect the application, see the use cases scenarios in Section 5). An `Account` keeps a list of key pairs and a pointer to the preferred (or default) key pair. The first time during a session a user sends or receives a message, PKIGate checks for an `Account` object for that user. If there is one, it examines the user's preferences to see if any action needs to be taken, and then calls the proper operation using the selected key pair. Otherwise, the message is left unaltered and the system sets a variable stating this user does not have an account, making any future calls to PKIGate during the session unnecessary, and reducing the time costs to user's without accounts.

```
Account

Preferences
SecurityLevel
AddressBook
KeyPairList
PreferredKeyPair
ClientSession
SecurityManagerWindow
```

Figure 4.2: The `Account` Class

4.2.3 ClientSession

ClientSession represents the user’s normal email client account upon login. When a user logs in, the list of `Account` objects is checked by UID. If there is an `Account`, the session is marked to check messages with PKIGate. If a user never logs into the Security Manager, then his/her messages will not enter PKIGate.

4.3 Crypto Server

Crypto Server resides on the 4758 secure coprocessor. It accepts XDR encoded requests from PKIGate containing an operation identifier and all necessary input for the requested operation. (More on these requests follow). Following the ideas in Sean Smith’s paper "Outbound Authentication for Programmable Secure Coprocessors,” the user’s private key is encrypted with the 4758’s public key [16]. Therefore, the secure Crypto Server is the only place where the user’s private key exists unencrypted.

4.4 Connection Between PKIGate and Crypto Server

PKIGate and the Crypto Server communicate through encoded request objects over a private connection (unless their final combined size is such that both applications can be placed on one secure coprocessor, see Section 6.1.1).

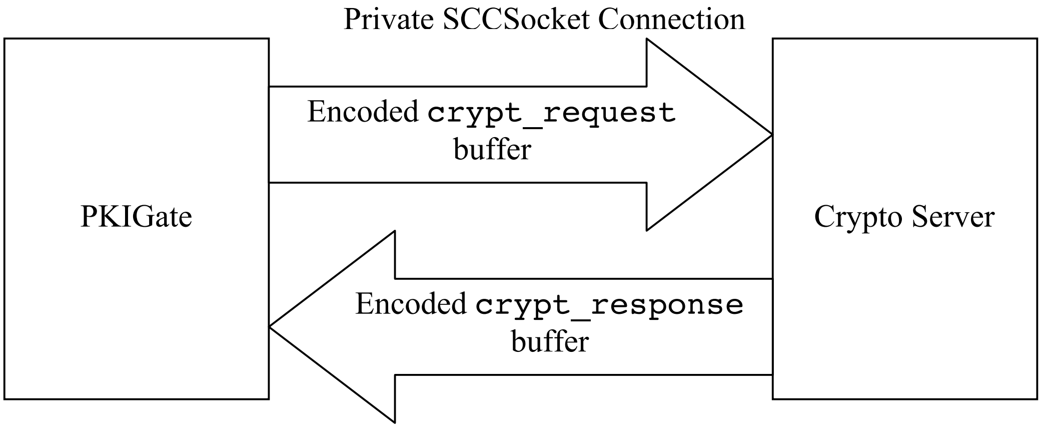
Private SCCSocket Connection
PKIGate
Encoded crypt_request buffer
Crypto Server
Encoded crypt_response buffer

Figure 4.3: Communication between PKIGate and Crypto Server

Our `Libsmcard` library implements the Crypto Token Library interface of the SFL and utilizes the ASN.1 encoding and functionality provided. We added a method, `cardConnect`, to send requests to the Crypto Server and receive and parse responses, and created two C structures for transferring information between components, `crypt_request` and `crypt_response`. Each of the SFL interface cryptographic functions creates a `crypt_request` object (see figure 4.4) specifying the operation and information and input buffers and passes this to `cardConnect`.

```
crypt_request

ENUM operation
Info1
Info2
Info3
Info4
Input
```

Figure 4.4: The `crypt_request` object

The cardConnect function accepts a crypt_request object and an unallocated output buffer. The request is XDR encoded, and sent over a private socket to Crypto Server, where it is parsed. Crypto Server makes the appropriate function call and returns a crypt_response object containing the length of the output buffer and the buffer itself.

crypt_response

```
Length
Output
```

Figure 4.5: The crypt_response object

4.5 Security Manager Window

The Security Manager window communicates directly with PKIGate via an SSL connection. For users at the secure level, the Security Manager window is used only for login and altering account preferences. However, for paranoid level users who are opening or sending encrypted messages, the Security Manager serves as a second secure email client. The Security Manager Window will offer a set of options to the paranoid level user, including setting and altering his/her account preferences. Specifically, it will provide options for creating encrypted or signed and encrypted messages and also for viewing received encrypted messages. (For more details on each of the operations, see Section 5, Use Cases).

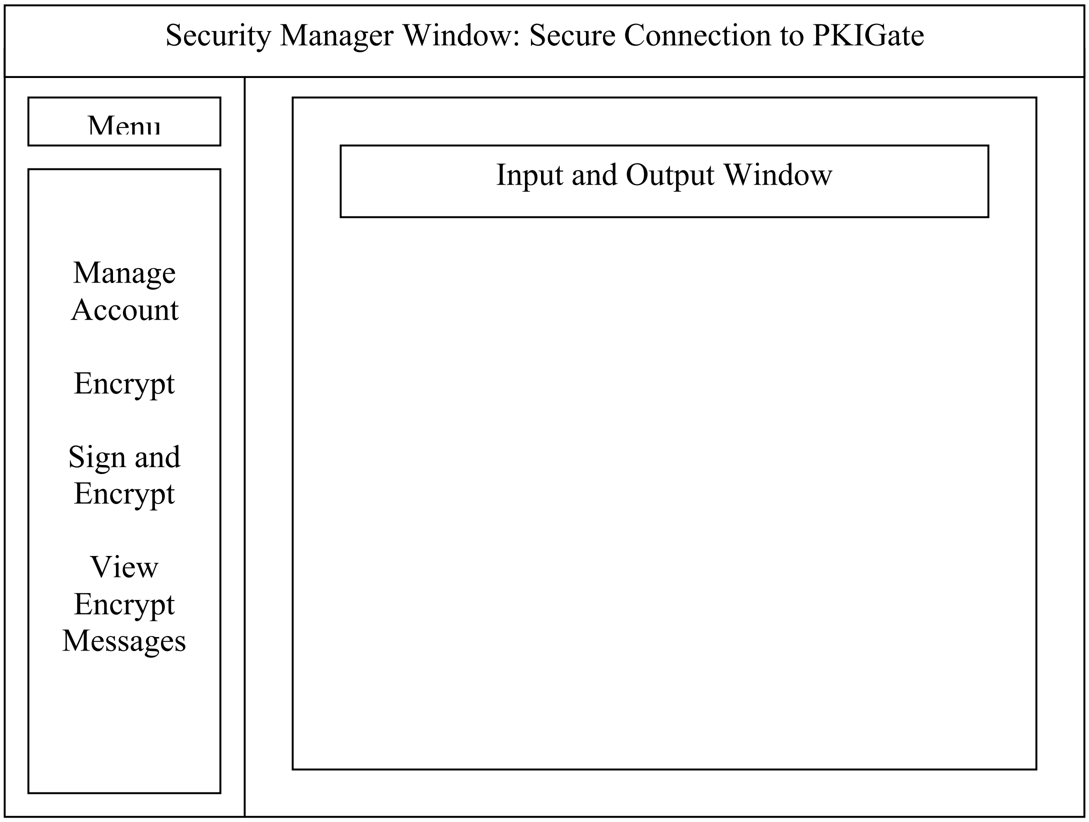

Figure 4.6: Possible layout for the Security Manager Window, Paranoid Level User View

5. Use Cases

The following use case scenarios are the result of conversations between members of the PKI Lab and brief user studies I organized and conducted with undergraduates at Dartmouth. They do not represent the only possible implementation of PKIGate, but simply one of the many that matched the design and restrictions of the system as described in the previous section. In this set of use cases, users are organized into three levels of security, `insecure`, `secure`, and `paranoid`, descriptions of which follow. The different user levels allow minimal to maximum user interaction with PKIGate via the secure client, the Security Manager Window.

We now discuss these use cases as they relate to the different user security levels of PKIGate.

5.1 Launch and Login

Login is necessary for users at either the `secure` or `paranoid` levels. Brainstorming sessions I led earlier this term with a user group led to two suggestions for convenient ways to launch the two-window secure email system. In each of the below cases, the user would set his security level as part of account preferences during his first login to the Security Manager window, and have the option of changing it at any later point in time. The first two cases below refer to logins for web-based email clients, while the third refers to the launch of a desktop client.

The first login scenario requires administration by an entity with knowledge of all email clients operating with the PKIGate application. The administrator would create a website with links to each of the supported clients from amongst which the user would select the link for his email client of choice. The link would open two windows: one with the original email client login screen and the second with the Security Manager Window login screen. After logging in to both applications, the user would be signed in to the system at his set security level.

The second idea is similar to the first, except that instead of a page of links, there would be a secure link for each of the supported email clients. For example, the secure version of “Webblitz” would reside at http://secure.basement.dartmouth.edu/~blitz instead of http://basement.dartmouth.edu/~blitz. At that address, the Security Manager and regular email client login forms would open in separate browser windows, allowing the user to log in to both applications.

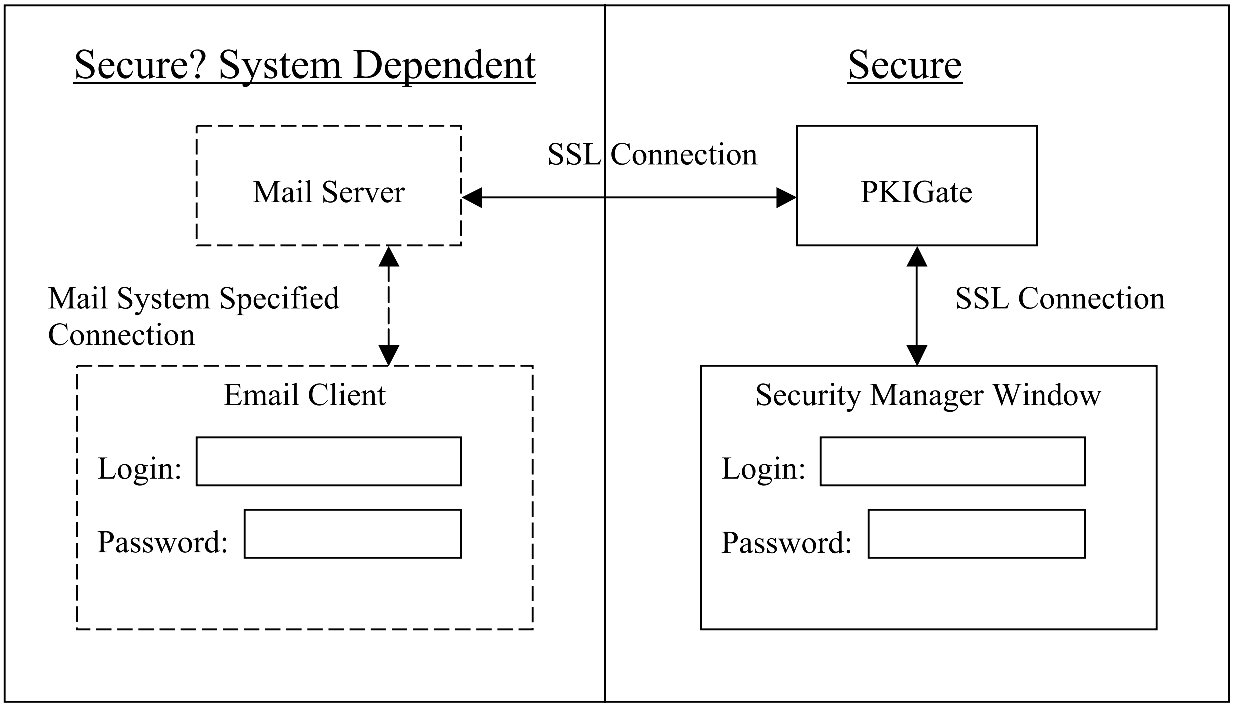

Figure 5.1: PKIGate Secure vs. Insecure Connections. Dashed arrows and boxes represent applications and connections for which we cannot guarantee security.

Since the Security Manager is a Web-based client, there are no clear shortcuts for desktop email clients to launch the system without altering the client. Therefore, it would be necessary for the user to launch a browser window for the Security Manager login screen.

5.2 Insecure

The `insecure` level denotes a session with the user's regular email client. In this case, it seems justified to assume that a user who never has an account created with PKIGate and who never logs into the Security Manager system should not be subject to the time or effort costs involved in using the gate. The Security Manager window need not be opened, nor does the user ever need to log into that part of the system. On the first attempt during a session by the mail server to contact PKIGate with this user's messages, the application will notify the server that the user does not have an account and set a variable to that effect.

5.3 Secure

A `secure` level user has an account with PKIGate, has registered at least one key pair, and is comfortable with allowing the application to operate on her email with little to no interaction. At some point, she has set overall account preferences or preferences in her account address book for PKIGate to follow in performing S/MIME operations; the session operates nearly seamlessly once preferences have been set. The `secure` level user has the ability to protect her outgoing and incoming messages from possible interception by adversaries outside her mail system. She is also able to digitally sign and verify messages, all without changing the client through which she normally sends or receives messages.

5.3.1 Sign

For signing, the user creates an outgoing message in her regular client. The message passes to the regular email server, which then contacts PKIGate through an SSL connection. Assuming the user has an account, PKIGate checks the address book for recipient preferences, and, if none are found, checks the overall account preferences to determine if the message should be signed. If either of the above cases holds true, the user's private key, either default or preference specified, and the message is sent via a private connection to Crypto Server, the only location where the private key is decrypted.

The output of the signing operation is returned to PKIGate where it is formatted by the S/MIME Freeware Library, wrapped in MIME headers, and returned to the mail server for sending. If neither the recipient nor the account preferences are set for signing, the message is returned to the mail server unaltered for sending.

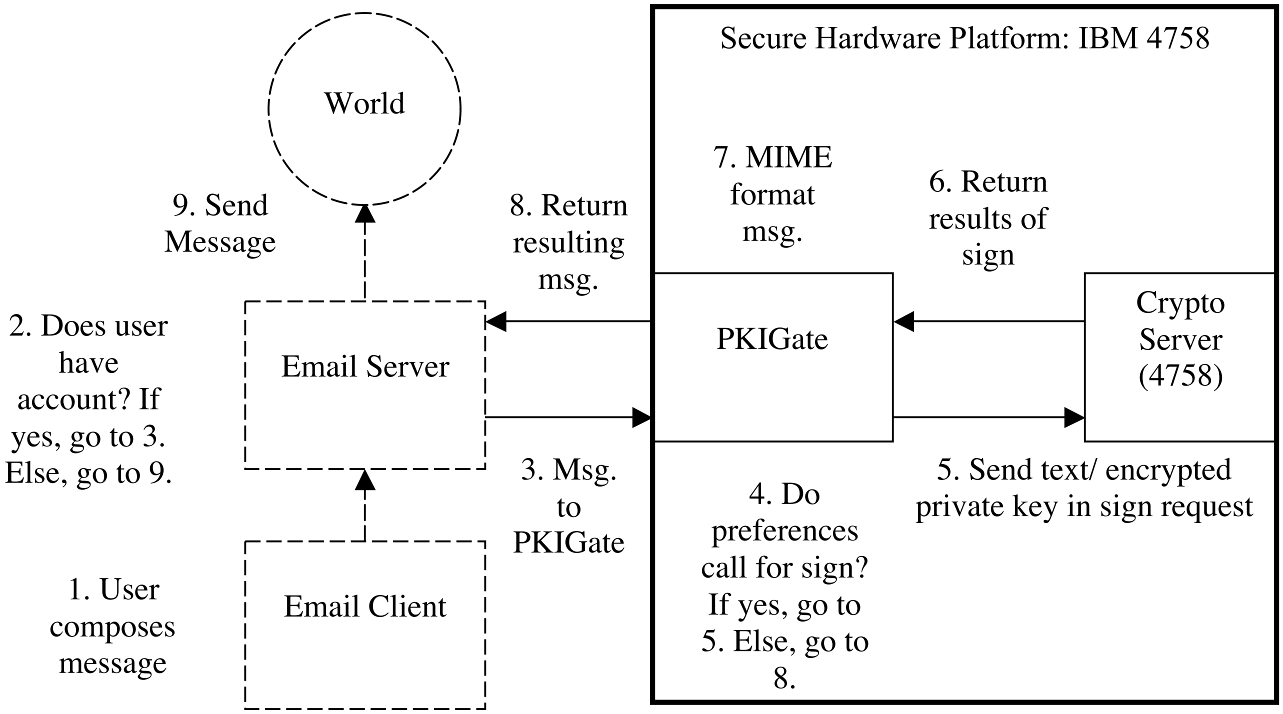
Secure Hardware Platform: IBM 4758
World
9. Send Message
8. Return resulting msg.
7. MIME format msg.
6. Return results of sign
2. Does user have account? If yes, go to 3. Else, go to 9.
Email Server
PKIGate
Crypto Server (4758)
3. Msg. to PKIGate
4. Do preferences call for sign? If yes, go to 5. Else, go to 8.
5. Send text/ encrypted private key in sign request
1. User composes message
Email Client

Figure 5.2: The path of a `Secure` Level Signed Message
Dashed arrows and boxes represent possibly insecure connections or components.

5.3.2 Verify

A user at the `secure` level allows the Security Manager to automatically verify her incoming messages. Assuming an account exists, PKIGate checks incoming email. The message MIME headers are parsed for information on which, if any, operations were performed on the email and, if the message was signed, PKIGate obtains the signer's information and sends a request to Crypto Server on the 4758. Crypto Server verifies the signed content and returns a string representing the success or failure of the operation. After verification, a content-description header stating whether or not verification was successful is added to the message.

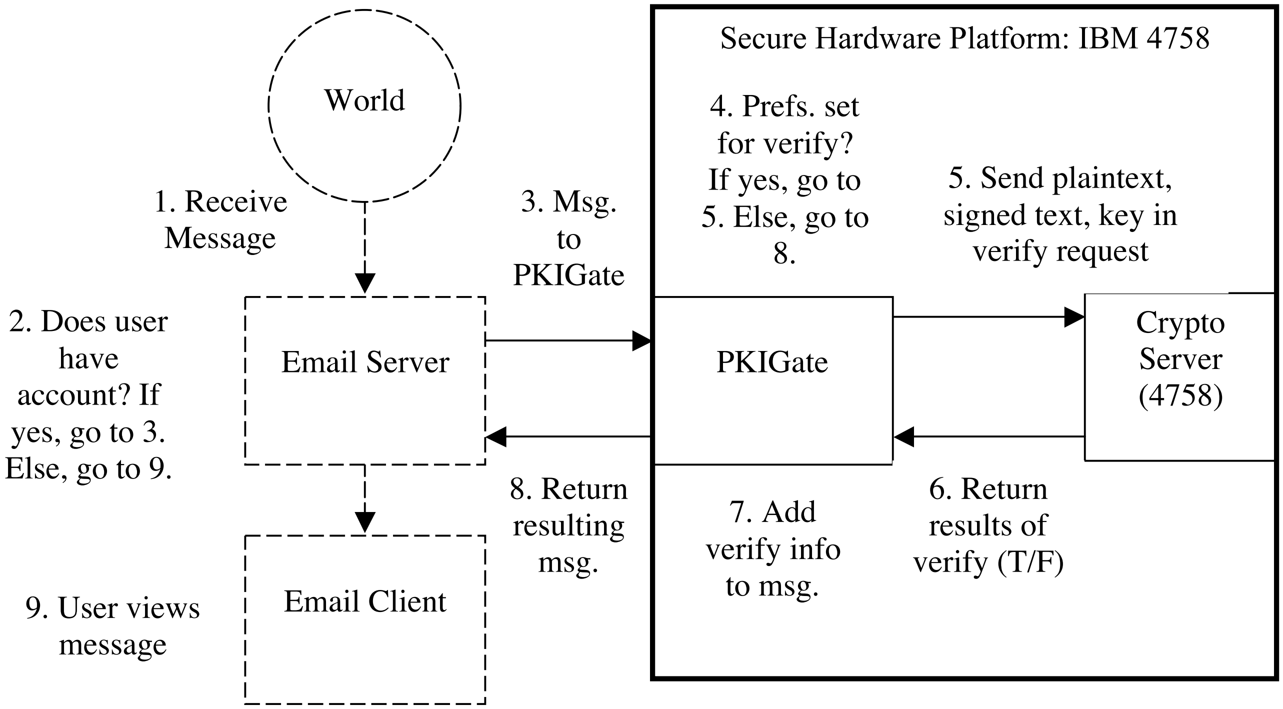
World
1. Receive Message
2. Does user have account? If yes, go to 3. Else, go to 9.
Email Server
Email Client
9. User views message
3. Msg. to PKIGate
8. Return resulting msg.
Secure Hardware Platform: IBM 4758
4. Prefs. set for verify? If yes, go to 5. Else, go to 8.
5. Send plaintext, signed text, key in verify request
PKIGate
Crypto Server (4758)
7. Add verify info to msg.
6. Return results of verify (T/F)

Figure 5.3: The path of a `Secure` Level Message Verification. Dashed arrows and boxes represent possibly insecure connections or components.

5.3.3 Encrypt

Encrypt works in a similar manner to sign. The user creates an email message in her regular client. The mail server forwards the relevant information to PKIGate, which checks the recipient followed by the account preferences and, if warranted, sends a request consisting of a content encryption key (CEK) and the content to Crypto Server. Once returned, PKIGate sends a second request to Crypto Server to encrypt the CEK with the recipient's public key, wraps the key and message in MIME headers, and returns it to the mail server for sending.

5.3.4 Decrypt

A user at the `secure` level allows PKIGate to automatically decrypt her incoming messages. PKIGate parses the incoming message headers for information on what, if any, cryptographic operations were performed. If the message was encrypted, a request containing the user's encrypted private key and the content encryption key (CEK) is sent to the Crypto Server via a private connection. The user's private key can only be decrypted inside the Crypto Server using its host coprocessor's private key. After the Crypto Server returns the decrypted CEK, a second request is issued with the decrypted key and the encrypted content. The Crypto Server returns the decrypted text as part of the `crypt_response` object. After PKIGate adds a content-description header noting the decryption operation performed, the message is returned to the mail server.

5.4 Paranoid

A user at the `paranoid` level has registered for an account with PKIGate and set a preference noting she wishes to have as much control over her email as possible. Like users at the secure level, he has the option of setting account and address book preferences. However, she also has the added advantage of being able to see and verify all of the messages PKIGate sends out using her key pairs.

In exchange for the ability to hide her secure messages from the email administrator, the paranoid level user surrenders the ability to view decrypted messages or compose message to be encrypted with her own email client. Instead, in order to guarantee the security of her messages, the user must type or paste these messages directly in to the Security Manager window. To have a copy of the message, we suggest the user add herself as a recipient. Encrypted messages may only be viewed decrypted in the Security Manager window; they are stored encrypted on the mail server.

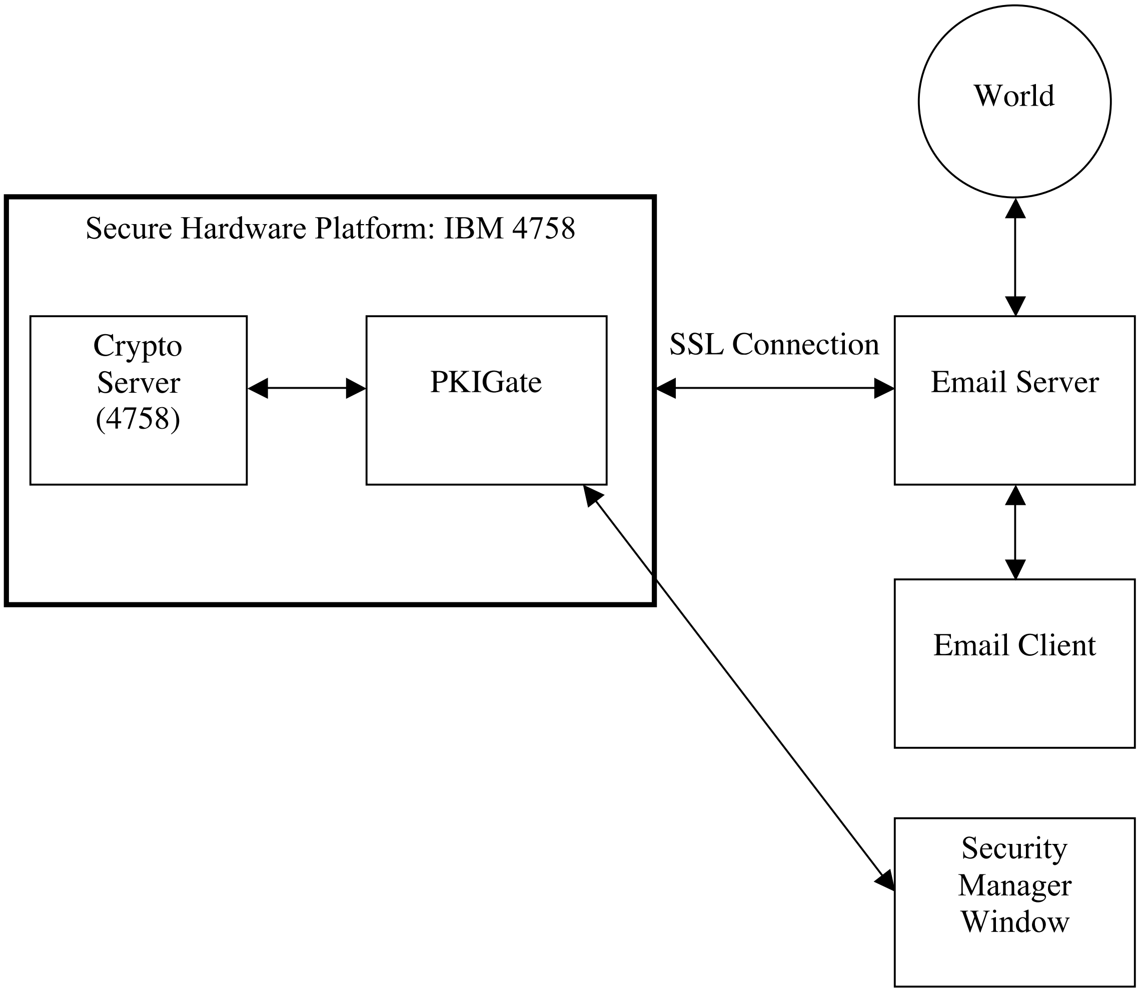

Figure 5.6: The PKIGate system view for a `Paranoid` Level user

We feel the gains for a `paranoid` level user are well worth the additional effort she must put in to send or view encrypted messages. In addition to being able to hide messages from adversaries on the outside of the user's mail system, the paranoid user can also hide messages from the email administrator and any other parties snooping within her mail system. The user does not need to trust anyone involved in the email service and can assume that if her messages are sent to the correct recipient, only the intended reader will be able to view the plaintext message.

5.4.1 Sign

To add a signature, the user creates a message using her regular email client. The mail server contacts PKIGate, and if account or recipient preferences are set to sign the message, the text of the message, the recipient, and identifying information for the key used in signing are displayed in the Security Manager Window. There, the user has the option of making any changes to the content or to the key pair being used before verifying that she really wishes to send it. If the user consents to send, PKIGate issues a request to the Crypto Server containing the user's encrypted private key and the content to be signed. Crypto Server returns the output of the operation to PKIGate, where it is wrapped in MIME headers and sent. Otherwise, the user cancels the entire message.

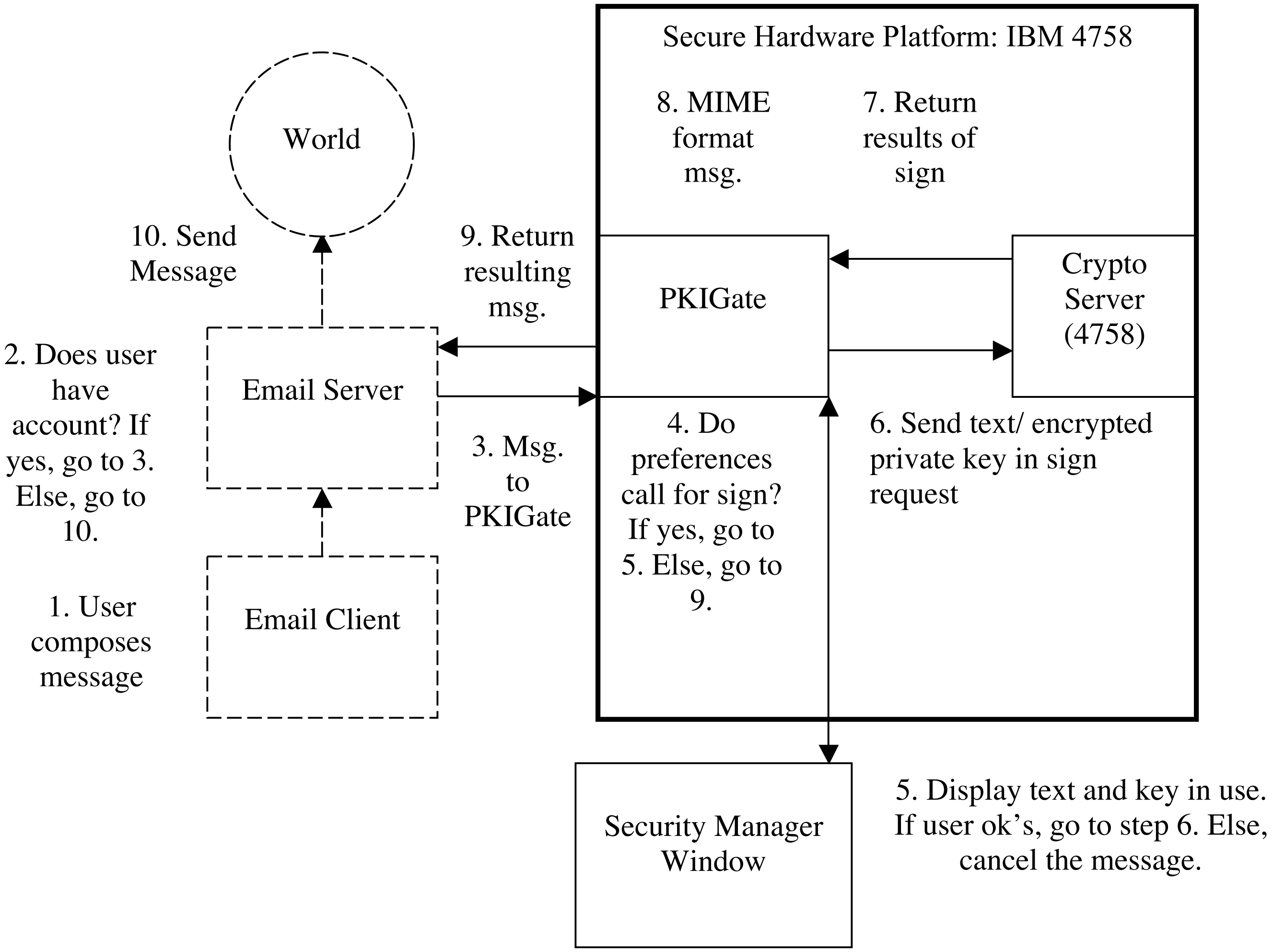
Secure Hardware Platform: IBM 4758
8. MIME format msg.
7. Return results of sign
World
10. Send Message
9. Return resulting msg.
PKIGate
Crypto Server (4758)
2. Does user have account? If yes, go to 3. Else, go to 10.
Email Server
3. Msg. to PKIGate
4. Do preferences call for sign? If yes, go to 5. Else, go to 9.
6. Send text/ encrypted private key in sign request
1. User composes message
Email Client
Security Manager Window
5. Display text and key in use. If user ok's, go to step 6. Else, cancel the message.

Figure 5.7: Signing a message at the `Paranoid` Level
Dashed arrows and boxes represent possibly insecure connections or components.

5.4.2 Verify

A user at the `paranoid` level allows the Security Manager to automatically verify her incoming messages. The mail server shunts the incoming message to PKIGate, and, if the user has an account, MIME headers are parsed for information on which, if any, operations were performed on the email. If the message has been signed, PKIGate obtains the signer's information and sends a request to the Crypto Server on the 4758 over a private connection containing the sender's public key and the signed and plain content for comparison. The Crypto Server verifies the signed content and returns a string representing the success or failure of the operation. After verification, PKIGate adds a content-description header to the message stating whether or not verification was successful.

5.4.3 Encrypt

The encryption case for a user at the paranoid level is difficult to implement both securely and conveniently. A user at this security level does not want the email administrator to have access to the unencrypted content of her messages. However, if a message passes through the client to the regular mail server, it will exist unencrypted on the server. Given the difficulty of this case, especially since at this point the user is not required to change her email client, it makes sense to have the user type or paste her message directly into the Security Manager window. We realize that this is inconvenient, and leave a better solution to future work.

To send an encrypted message, the a user opens the Security Manager window, selects “Encrypt” from the list of tasks, and types or pastes the message directly into the trusted window. The message passes via an SSL connection directly to PKIGate. There, a request containing the content to be encrypted and the content encryption key (CEK), produced previously by Crypto Server, is created and sent to the Crypto Server over a private connection. The server returns the output of the encryption operation to PKIGate, which sends a second request, this time for encryption of the CEK. PKIGate creates a message with the encrypted content and encrypted CEK, wraps it in MIME headers, and returns it to the mail server for sending.

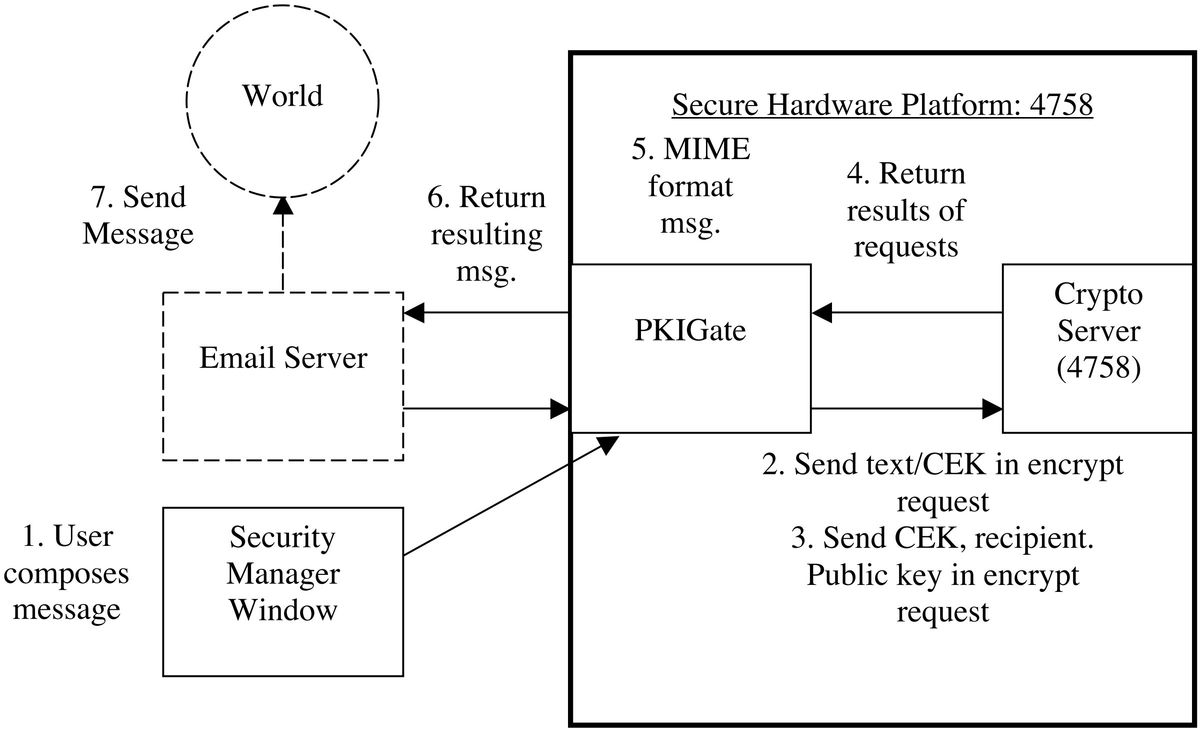

Figure 5.9: Encrypting a message at the `Paranoid` Level. Dashed arrows and boxes represent applications and connections for which we cannot guarantee security.

5.4.4 Decrypt

Decrypting messages is another area in which the `paranoid` user must make sacrifices for her desire for security. The message cannot reside on the mail server unencrypted, therefore it must not be allowed to pass out of PKIGate in its decrypted state. Instead, when a user is not logged in, the messages go to the mail server encrypted. When the user logs in to their regular mail client, the messages will appear encrypted. To decrypt them, the user must log into the Security Manager window, thereby creating a direct secure connection to PKIGate. From there, she is given a list of the encrypted messages residing in her account from which she may select a message to decrypt and view. The message is ONLY decrypted on the screen; it remains unaltered on the mail server.

To decrypt the text, PKIGate sends a request with the encrypted content encryption key (CEK) and the user's encrypted private key to the Crypto Server. The private key is ONLY decrypted inside the Crypto Server by using the host 4758's private key. The server decrypts the CEK and returns it. Another request with the encrypted content and the decrypted content encryption key are sent to the Crypto Server, which decrypts the message and returns the output to PKIGate. The decrypted text is displayed in the Security Manager window. In the case of an attachment, PKIGate provides a link to another browser window where the attachment may be viewed.

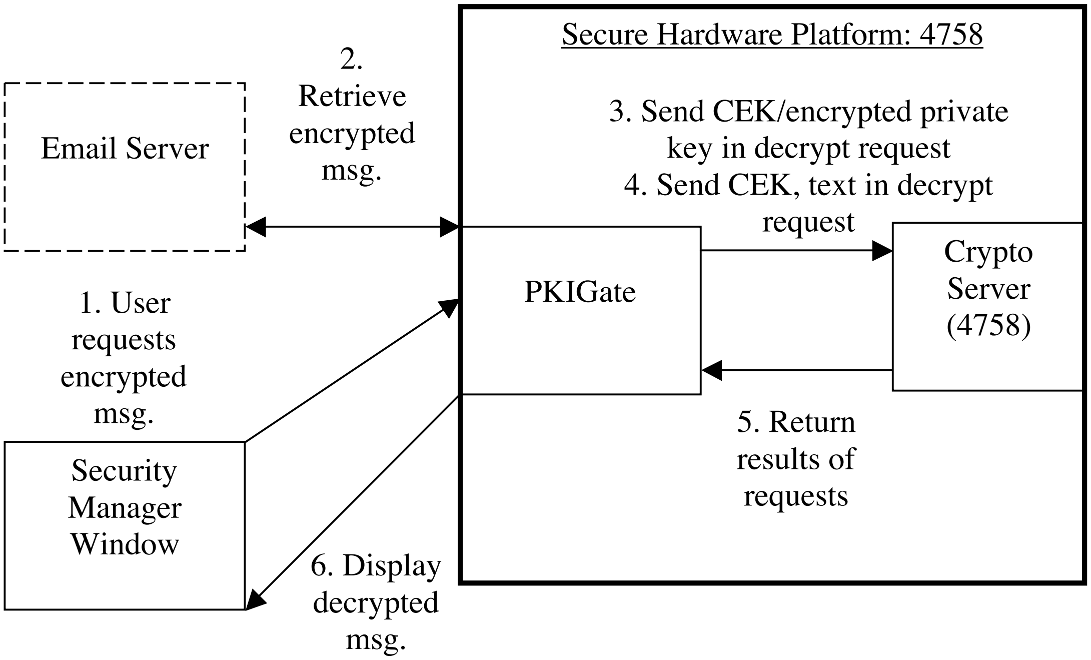
Secure Hardware Platform: 4758
2. Retrieve encrypted msg.
Email Server
3. Send CEK/encrypted private key in decrypt request
4. Send CEK, text in decrypt request
PKIGate
Crypto Server (4758)
1. User requests encrypted msg.
5. Return results of requests
Security Manager Window
6. Display decrypted msg.

Figure 5.9: Decrypting a message at the Paranoid Level. Dashed arrows and boxes represent applications and connections for which we cannot guarantee security.

6. Implementation and Conclusions

This section describes difficulties experienced in the process of implementing PKIGate that prevented us from completing a prototype and suggests future work and user studies in the S/MIME gateway application area. We conclude by evaluating the success of PKIGate with respect to our goals for the project and suggesting a new path to victory in creating a mobile S/MIME solution.

6.1 Implementation Difficulties

The implementation of PKIGate was plagued by several difficulties, preventing us from creating the desired prototype implementation. The most major of those difficulties, a suggested solution, and an outline for applicable future work follow.

6.1.1 PKIGate Size or "So, how big is it?"

The design for PKIGate calls for the S/MIME functionality as well the cryptographic operations to reside on the IBM 4758 secure coprocessor. However, the 4758 imposed a size restriction on the executable of about 1.8M.

To get an accurate size estimate before writing to the card, we coded an application representing the SFL object construction and library calls PKIGate would have to make to offer both authentication and encryption services. We compiled and statically linked the above executable with all the required libraries using G++3.2.2. Since the libraries (including the ESNACC 1.4 distribution) were created with G++3.0 in mind, and the compiler is known to have issues, some of which arose in coding other parts of the system, we were forced to apply patches from a newer release that corrected the issues with our compiler, version 3.2.2. (All steps taken to make the release compile correctly were documented and are available upon request). The Unix strip utility, which removes all unnecessary information from the executable, brought the total size of the executable to 4.8M, larger than the space available on the 4758.

To further reduce the size of the application, we mounted the uClibc (micro-c-libc) file system. Uclibc is a smaller version of the standard C library used in developing embedded Linux systems [18]. The file system contains a version of the GCC distribution and standard libraries that were compiled with the uClibc set of libraries instead of with the standard versions, reducing their size. Compiling with uClibc G++3.2.2 reduced the size to 3.9M, almost a 1M difference, but still not small enough to fit on the card.

Bob Colestock, the S/MIME Freeware Library point of contact, suggested in an email message earlier this term that it would be possible to remove the Cryptographic Token Interface Library component of the S/MIME Freeware Libraries and plug our cryptography directly into the application. We hope this will allow the final reduction in size required to fit PKIGate inside the 4758, and make it possible to combine Crypto Server and the S/MIME application, therefore removing an unnecessary line of communication in the system. It is also possible that IBM will release a 4758 secure coprocessor with more memory space for executables. We leave moving the PKIGate executable to a secure coprocessor for future work.

6.2 Implemented Parts

In spite of time consuming difficulties, portions of PKIGate were implemented. The XDR encoding and communication to be used between PKIGate and Crypto Server has been coded and tested to work over a regular socket connection (thanks to Alex Iliev and his previous work with the technology). Additionally, the `crypt_request` and `crypt_response` structures described in Section 4 are implemented and there is an outline of the PKIGate `Account` class and necessary calls for S/MIME formatting, used in attaining an executable size.

6.3 Future Work

We see several areas for future work in addition to the implementation of a full prototype from the design and the move of PKIGate to a secure coprocessor, as described in Section 6.1.1.

First, as previously stated, we suggest finding a better way to implement a `paranoid` user's encryption and decryption operations. The current system forces the paranoid user to use the Security Manager Window as an email client when sending or viewing encrypted messages and does not provide her with a way to securely save plaintext messages.

The above ties into an idea for another set of user studies that should be conducted before a final implementation of PKIGate is released. As Jon Callas notes in "Improving

Message Security with a Self-Assembling PKI," PKI will only work if people use it, and people will only use it if it is as easy to use as the insecure alternative [3]. We suggest presenting the prototype and alternatives to users who would operate at both the `secure` and `paranoid` levels.

Finally, PKIGate is not a fully operational email client. Eventually, there will come a time when users are willing to switch to a new email client in exchange for a higher level of security. This calls for the design of a new trusted Web-based client. Evan Knop's paper "Secure Public-Key Services for Web-Based Mail" suggests a method by which the 4758 secure coprocessor can serve its own, trusted content to a client, creating a trusted environment for S/MIME email [11].

6.4 Conclusions

The PKIGate design meets the goals we laid out in the introduction for a mobile S/MIME solution at Dartmouth. The system:

1) Gives users a mobile solution. Since a user's key pair is stored on a central server, he/she has access to it for cryptographic purposes anywhere at anytime.
2) Supplies a trusted environment. It utilizes the IBM 4758 secure coprocessor as the location for all cryptographic operations.
3) Allows users to retain their original email client. PKIGate operates in conjunction with the mail server without any changes to the email client.

Given the importance of correct use cases, interfaces, and S/MIME formatting to PKIGate and the difficulty of installing executables on the 4758, I suggest future work follow a different research path than the one we took. It would perhaps make more sense to implement an insecure version of the application on the Blitzmail system or in conjunction with one of the Web-based email clients, and allow users to test the system first. To take the study one step further, the application's underlying cryptography could be replaced with a temporary stand-in set of functions, such as from the Crypto++ library, before moving it to a secure hardware platform.

In conclusion, through this thesis I learned many lessons about the design of a large-scale project. Namely, there will always be more than one good way to design the initial incarnation of any new application and it is vital to conduct background research, talk to groups of "real" users, and synthesize the many ideas presented carefully. A better design will result from better research and from having to justify one's design decisions than from simply deciding on a design path. With this in mind, I suggest to anyone undertaking this project in the future to familiarize themselves not only with the ideas and technologies presented in this paper, but also conduct to their own research of new technologies and with new users.

References

[1] Baycorp Advantage. MailMarshalSecure. `http://www.baycorpid.com/id_services/information_strategic_mailmarshal.asp.`

[2] Biodata IT. CryptoEx Gateway. `http://www.biodata.co.za/gw.htm.`

[3] John Callas. Improving Message Security With a Self-Assembling PKI. In *2nd Annual PKI Research Workshop—Pre-Proceedings*, April 2003.

[4] Dreamhost. What's an email client? What does it do? `https://panel.dreamhost.com/kbase/index.cgi?area=2376.`

[5] Dreamhost. How does email work? `https://panel.dreamhost.com/kbase/index.cgi?area=2403.`

[6] Getronics Government Solutions. *S/MIME Freeware Library Software Design Description*, version 2.1, July 2002.

[7] Getronics Government Solutions. *S/MIME Freeware Library Application Programming Interface*, version 2.1, June 2002.

[8] Getronics Government Solutions. *Cryptographic Token Interface Library Application Programming Interface*, version 2.1, June 2002.

[9] HIPAAdvisory. *What's HIPAA? - A Basic HIPAA Primer.* `http://www.hipaadvisory.com/regs/HIPAAprimer1.htm.`

[10] IBM Research News. IBM Research Demonstrates Linux Running on Secure Cryptographic Coprocessor `http://www.research.ibm.com/resources/news/20010828_mycroft.shtml`

[11] Evan Knop. Secure Public-Key Services for Web-Based Mail: A sketch of a secure environment. *Honors Thesis*, *Dartmouth College*, 2001.

[12] Mpack Manpage: 1995.

[13] Openssl. smime. `http://www.openssl.org/docs/apps/smime.html#.`

[14] RSA Security. S/MIME Frequently Asked Questions. `http://www.rsasecurity.com/standards/smime/faq.html`

[15] Security Softlabs. SMIME Stripper. `http://www.vroyer.org/smimestripper/index.html.`

[16] S.W. Smith. Outbound Authentication for Programmable Secure Coprocessors. In *7th European Symposium on Research in Computer Security.* Springer-Verlag LNCS 2502.

[17] S.W. Smith, E. Palmer, S.H. Weingart. Using a High-Performance, Programmable Secure Coprocessor. In *2nd International Conference on Financial Cryptography.* (Springer-Verlag LNCS.) February 1998.

[18] uClibc. A C Library For Embedded Systems. `http://www.uclibc.org.`

[19] Webopedia. MIME. `http://webopedia.com/TERM/M/MIME.html`

[20] XiCrypt Technologies. IAIK S/MIME Mapper. `http://www.xicrypt.com/modules.php?op=modload&name=IncludePage&file=smime_eng.`

[21] Bennett Yee, J.D Tygar. Secure Coprocessors in Electronic Commerce Applications. In *First USENIX Workshop on Electronic Commerce*. July, 1995

[22] ZipLip. Integrated Email Gateway. `http://www.ziplip.com.`