2009-05-15

EASEmail: Easy Accessible Secure Email

Ryan B. Segeberg
Brigham Young University - Provo

EASEMAIL: EASY ACCESSIBLE SECURE EMAIL

by

Ryan B. Segeberg

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Brigham Young University

August 2009

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Ryan B. Segeberg

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Kent E. Seamons, Chair
Date	Quinn Snell
Date	Dan Olsen

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Ryan B. Segeberg in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	Kent E. Seamons Chair, Graduate Committee

Accepted for the Department

Date	Kent E. Seamons Graduate Coordinator

Accepted for the College

Date	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

EASEMAIL: EASY ACCESSIBLE SECURE EMAIL

Ryan B. Segeberg

Department of Computer Science

Master of Science

Traditional email encryption methods are difficult to set up, as they require senders to obtain a message recipient's public key before a secure communication can be sent. Easy Accessible Secure Email (EASEmail) addresses the key establishment and exchange issues of encrypted email by using a lightweight symmetric key server. Users can send a secure email without establishing or exchanging keys with the recipient in advance. With usability as its primary goal, EASEmail strives to bring usable secure email communication to the masses.

ACKNOWLEDGMENTS

I would like to thank Tim van der Horst for the vital role he played. His prior work on Blind Escrowed Email (Beemail) and 3SKE (both unpublished) were the seeds that germinated into EASEmail. His experience and knowledge were a great resource, many a wrong path was avoided because of his good advice. He was a well of ideas and a sounding board for design decisions during the development process. Were it not for him, EASEmail may not have ever existed.

I would also like to thank Dr. Seamons for his efforts in bringing EASEmail to fruition. The many drafts he read and the advice he gave was invaluable in finishing and refining this paper.

Truely I stand on the shoulders of giants, as the work I was able to accomplish here would not have been possible if not for what so many before me have contributed.

Finally, I'd like to thank my wife and children who have been patient and supportive during the countless hours I have spent on this work.

This research was supported by funding from the National Science Foundation under grant no. CCR-0325951, prime cooperative agreement no. IIS-0331707, and The Regents of the University of California.

Contents

Chapter 1

Introduction

Email communication is ubiquitous. It is used every day for the bulk of mundane exchanges as well as for mission critical business applications. Email traffic on the Internet is typically transmitted in the clear, and, as such, personal information sent via email does not carry the same assurance as regular postal mail does. Regular post is usually sent in an envelope, which obscures the contents of the message and provides some level of tamper resistance. Email is more akin to post cards, whose contents can be read by any passive observer.

A number of email encryption standards have been introduced over the years. Two popular standards are Pretty Good Privacy (PGP) and Secure/Multipurpose Internet Mail Extensions (S/MIME). These asymmetric encryption solutions suffer from usability problems. Senders must first obtain the recipient's public key before secure email can be sent. This basic bootstrapping problem removes the send-and-forget nature of email where any sender can send to any recipient without previous communication. In a survey conducted by Garfinkel et al. [7], the majority of respondents either did not know how to send secure email, or did not even know what secure email was. As a result confidential information is often sent in the clear out of lack of choice, or not sent at all.

A relatively new technology called Identity-based Encryption (IBE) addresses these problems. With IBE a recipient's public key is simply their identifier (e.g., email address) combined with some publicly known values that belong to a key server.

This eliminates the bootstrapping process and allows users to send email without any previous communication. Recipients who do not yet have their private key can simply contact the key server to obtain it after the message is received.

The underlying reason why most existing encrypted email solutions are difficult to use is due to the complexity of the underlying technology. We introduce Easy Accessible Secure Email (EASEmail), which is designed from the ground up with usability as its primary goal. EASEmail relies on a simple, lightweight key server that facilitates key establishment and exchange between authenticated users without user-specific management or pre-established shared secrets. EASEmail is similar in concept to IBE, but leverages symmetric key cryptography.

1.1 Motivating Scenarios

We present three motivating scenarios for EASEmail:

No established secrets Tim really needs a job. A job posting informs him to fill out an application which he should then email, along with his resume, to HR manager Bob. Tim knows that he should protect his personal information, and does not like the idea of sending an application and his resume in a plaintext email. Since Tim does not know Bob, he has no idea if Bob has already established a public/private key pair for either of the two well established email encryption methods (PGP or S/MIME), and even if Bob does have a key pair Tim does not have Bob's public key. The only information Tim has about Bob is Bob's email address. Tim needs a way to send a secure message without having to first contact Bob.

Key Escrow Alice owns a small restaurant chain and needs to send confidential business information to the managers. Neither she nor the managers are very tech savvy. She has tried using PGP, but a laptop crash caused her to lose her private key, leaving her unable to decrypt her messages. Alice needs a simple, lightweight email encryption mechanism that includes key escrow so she can send the occasional encrypted message to her store managers without the burden of personally managing and backing up keys.

Portability Charlie uses web-based email exclusively. He is security minded and would like to send and receive secure email even on the go. He wants an encrypted email solution that will work with his existing web-mail provider's interface, but will still keep his messages confidential, even from his email provider. He could use PGP or S/MIME, but he does not want to carry a private key, and none of his contacts presently use encrypted email. He frequently uses public computers which forbid the installation of software so he needs a solution that does not require client modification to work.

Chapter 2

Related Work

2.1 Pretty Good Privacy

Pretty Good Privacy (PGP) [18] was developed by Phillip Zimmermann in 1991 and uses asymmetric encryption. Users generate their own public/private key pairs and upload their public key to a key server. These keys can be signed by other users to create a "web of trust." Once created, the keys can be used to sign and encrypt messages. Before an encrypted message can be sent the sender must obtain the recipient's public key. This is best done by the sender and recipient exchanging keys out-of-band where each entity verifies the other's identity and public key fingerprint (e.g., a key signing party), after which encrypted messages can be sent. Public keys can also be obtained directly from a key server. This, however, offers no assurance that a key belongs to the intended recipient since anyone can create a key using any identifier.

PGP has several limitations. The loss of a user's private key renders them unable to decrypt any messages (new and old). Compromise of the private key means an attacker can trivially decrypt messages encrypted with the corresponding public key. This necessitates key revocation lists to prevent senders from encrypting messages with compromised keys.

Most major email clients including Microsoft Outlook, IBM Lotus Notes, and Mozilla Thunderbird do not support PGP but instead require an add-on or plug-in.

2.2 Secure/Multipurpose Internet Mail Extensions

Secure/Multipurpose Internet Mail Extensions (S/MIME) [20] was originally developed by RSA Data Security, Inc. Like PGP it uses asymmetric encryption. Instead of a web of trust, users obtain a certificate for their public key from a trusted Certificate Authority (CA). Ideally the CA verifies the user's identity before certifying a key. This certificate is signed by the CA, creating a trust hierarchy that enables it to be trusted by others.

Because S/MIME also uses asymmetric encryption, it has many of the same weaknesses as PGP. Users still need to exchange public keys before encrypted messages can be sent, requiring senders to contact recipients who do not yet have a public key. Obtaining a certificate for a public key requires divulging personal information (for identification purposes) which some users are unwilling to do. Loss or compromise of the private key has the same implications as with PGP and necessitates the CA maintaining a key revocation list.

Almost all major email clients support S/MIME natively including Microsoft Outlook, IBM's Lotus Notes, and Mozilla Thunderbird.

2.3 Identity Based Encryption

The concept of Identity Based Encryption (IBE) was first introduced by Adi Shamir [19] in 1984. Later Boneh and Franklin presented a full IBE scheme using the Weil Pairing on elliptic curves [2].

IBE is asymmetric encryption based, but instead of using randomly generated public/private key pairs, it uses a user's identifier as the public key. A key server called a Private Key Generator (PKG), with its own public/private key pair, generates the corresponding private keys for identifiers. One obvious problem with this scheme is cases where a user's private key is compromised. It is one thing to revoke a randomly generated public key; it is much more difficult to revoke one's identity. To solve this problem short lived keys are used. A complete identity-based public key is based on the user's identifier, public values of the key server, and a time period (such as the current day). Thus the user has a unique public/private key pair for each time period.

Key loss is not an issue since the PKG manages keys. However, since the PKG has the ability to generate any private key, it also has the ability to decrypt any message encrypted with the associated IBE public key. This is in contrast to PGP and S/MIME where only the owner of the private key can decrypt messages encrypted with the associated public key (presuming their private key has not been compromised).

Key compromise is still a problem. This problem is mitigated by the short life time of the key, a compromised key only allows for decryption of a small subset of the messages a user has received. These short lived keys eliminate the need for a key revocation system.

IBE overcomes many of the problems of PGP and S/MIME. Users no longer need to exchange public keys before they can send encrypted messages. Also, a user with a public/private key pair can send messages to a user who does not. If the recipient wants to put forth the required effort they can obtain their private key from the PKG and decrypt the message.

Voltage Security offers a variety of solutions that leverage IBE.

2.4 Lightweight Encryption for Email

Adida et al. [1] present a scheme to improve on a few of the short comings in IBE. To prevent the PKG from being able to decrypt user's messages, they propose a two key system. Users generate their own uncertified public/private key pair. They publish their public key on a website or key server. The PKG publishes its public key as a DNS entry. To compute a recipient's public key a sender simply combines the two keys. To decrypt messages a recipient needs both their self generated private key and the private key associated with their identifier. This way the PKG does not have the ability to decrypt messages, since it does not know the user's self generated private key.

With standard IBE the PKG is a single point of failure. With the private key of the PKG an attacker can generate the private key for any user. The paper proposes that the private key be split and stored on an arbitrary number of servers. The paper includes algorithms for splitting and combining keys so they never have to be physically combined before being used. This requires an attacker to compromise all the PKGs.

The difficultly of publishing one's public key is a major usability issue with this solution. This also has the same key loss problem as PGP and S/MIME, loss of the self generated private key renders messages unreadable.

2.5 Identity Based Message Key Distribution

Khurana and Basney [14] present an alternative to IBE called Identity Based - Message Key Distribution (IB-MKD). IB-MKD utilizes a key distribution center (KDC) that is very similar to the PKG in IBE, except the KDC does not generate any keys; it simply decrypts keys for recipients.

Messages are encrypted by the sender with a randomly generated key. This key, along with the identifier of the recipient and optionally some policy information, is encrypted with the public key of the KDC. This public key is available as a DNS entry. When a message is received the recipient extracts this encrypted information and contacts the KDC. After authenticating the recipient (e.g., with a password) and enforcing the included policy (if any) the KDC decrypts the information with its private key and returns the encryption key to the recipient.

A key strength of this system is also a weakness: each message is encrypted with a unique, unrelated, and randomly generated key. This requires a recipient to interact with the KDC each time a message is received. While KDC interactions should be brief, the need for the KDC to always be available presents a real limitation, whereas with IBE once a user has obtained her private key she does not need to interact with the PKG for the rest of the time period. It is not possible to optimize message decryption since users cannot pre-fetch keys for future time periods.

2.6 Opportunistic Encryption

Garfinkel et al. [6] introduce Stream, a simple opportunistic email encryption scheme. Stream is a proxy filter that automatically encrypts and decrypts messages for email users. The proxy generates and manages public/private key pairs for all users. Messages are sent with the public key in the message header. When a user receives a new public key in a message the proxy adds the key and the associated email address to a database. Users are sent warning messages for situations where the public key in a message and the message sender's email address do not match the entry in the database. Messages are encrypted by the proxy for recipients whose public keys are available, and sent in the clear for all others. Upon receipt, messages are decrypted by the proxy before being delivered to the recipient.

Stream takes the burden of key management away from the user. It does not require the installation of client software. Users behave mostly like they do today. Because the encryption is opportunistic, users may not be aware when their messages are and are not encrypted. It may be the case that some recipients of a particular message receive it encrypted, while others receive it in plaintext. This could lead to users accidentally sending data that is supposed to be secure in the clear. Also, because the messages are encrypted by the proxy, the proxy has the ability to read messages.

2.7 Web-based Providers

A number of web-based encrypted email services have surfaced over the years. The following are the most notable ones.

2.7.1 Hushmail

Hushmail [10] is a web service that uses PGP for email encryption. During registration a public/private key pair is generated and stored on the Hushmail servers. The private key is encrypted with the user's passphrase. Users can choose to sign and/or encrypt their messages, when they do so their private key is retrieved from the Hushmail servers. Encrypted email can be sent to other users of Hushmail and those who have uploaded their public keys to the Hushmail key server. Encrypted email can be sent to non-Hushmail users by supplying a security question and answer that the recipient is able to answer.

While Hushmail attempts to solve the bootstrapping problem of PGP, it still suffers from a different kind of bootstrapping problem. Users still cannot send encrypted email to a recipient they do not know. Sending email to a non-Hushmail recipient is possible, but requires establishing a secret between the sender and recipient. In situations where the sender does not know the recipient (as in the first motivating scenario) this is not feasible. Further, since the mail server and the key server are the same entity, the email provider has the ability to decrypt messages at any time.

2.7.2 freenigma

freenigma [5] is a web browser extension that works with Gmail, Yahoo, and Hotmail. Its goal is to bring usability to encrypted email, abstracting the details away from the user as much as possible. At sign-up a PGP public/private key pair is generated for the user and stored on the freenigma servers. When the user sends encrypted email the extension fetches their keys from the server.

freenigma works like a social network, users invite people to be on their buddy list, and then they can send encrypted email to them. Emails conform to the PGP standard so non-freenigma users who use PGP should be able to read the messages. Users compose their messages using the normal web-interface of the web-mail provider.

While freenigma tries to simplify the bootstrapping process, it is still there. Recipients have to be added to the sender's buddy list before secure mail can be sent. This process requires the recipient to join the freenigma service and establish their own key pair. Also, since messages are composed using the email provider's interface, JavaScript could be used to get the plaintext message before it is encrypted.

Chapter 3

EASEmail

To overcome the usability problems of traditional email solutions we propose Easy Accessible Secure Email (EASEmail). EASEmail leverages symmetric encryption and has several characteristics of IBE: a simplified mechanism for obtaining keys for recipients based on their identity and reliance upon a trusted third party (i.e., key escrow).

3.1 Design Goals

The design goals of EASEmail are usability, mobility, and security.

3.1.1 Usability

Usability is the primary goal in the design of EASEmail. Key establishment and exchange is difficult with some email encryption solutions and is the reason they are hard to use. When making design decisions we leverage existing user experience by mimicking activities users are already accustomed to doing. Ideally, very little user training should be needed before she is able to properly use EASEmail.

One of the major drawbacks of most email encryption solutions is the need to obtain the recipient's public key before a secure message can be sent. The goal for EASEmail is to be able to send an encrypted message to any valid email address with very little bootstrapping. Recipients can then decide if they want to obtain the software needed for decryption.

3.1.2 Mobility

Encrypted email has the potential of limiting users so they can only send and receive secure messages at their own computer, either because they must constantly be in possession of their encryption keys, or they need special software for message encryption and decryption. Many email providers, including corporations, provide POP and IMAP as well as web-based email access. Users can typically access their email account at any time, from any place. A goal for EASEmail is to allow users to access their secure messages from any computer that has access to the Internet, even when they do not have the ability to install software on the system.

3.1.3 Security

Regular insecure email has low security, but is successfully used by a wide range of users each day. When used correctly, solutions such as S/MIME and PGP offer the highest level of security because users create and control their own keys. This increased level of security suffers from usability problems as has already been discussed.

Our goal with EASEmail is to strike a balance between security and usability. The system should be as easy to use as regular email, thus users should not be burdened with key management. Returning to the postcard analogy, the message is placed into a sealed envelope, protecting it from passive observation and raising the bar for active attackers.

3.2 EASEmail Overview

This section provides an overview of EASEmail message encryption and decryption. The details on how and why each step is performed are discussed in the subsequent sections.

3.2.1 Encryption

1. Get the sender key from the KDC

2. For each recipient, derive the sender-recipient key
3. Compute the MAC of the plaintext for each recipient
4. Generate the random message key
5. Encrypt the message key for each recipient with the sender-recipient key
6. Associate the encrypted message key with the hash of the recipient's identifier
7. Encrypt the message with the message key
8. Compute the MAC of the ciphertext for each recipient

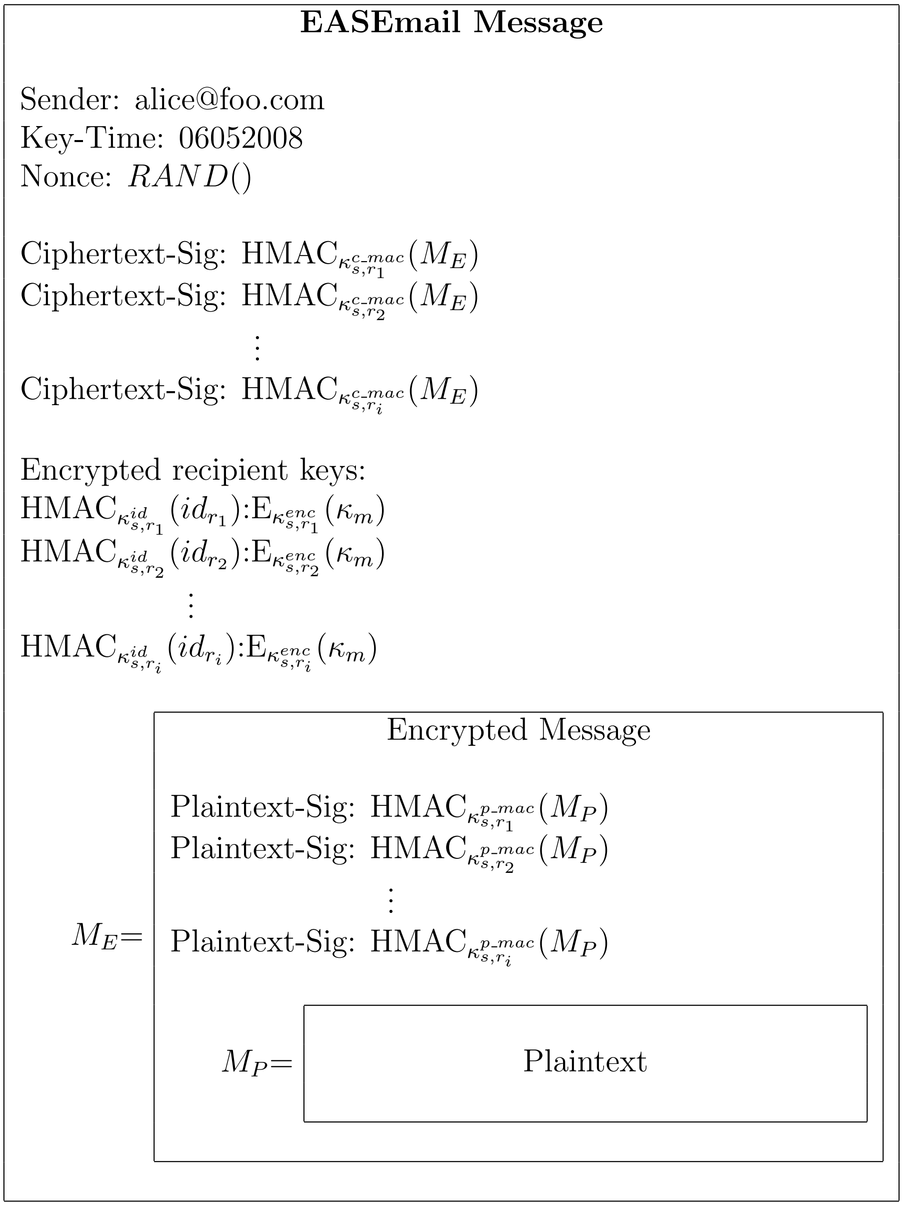
3.2.2 Decryption

1. Get the sender-recipient key from the KDC
2. Verify the ciphertext MAC
3. Look up the encrypted message key associated with the hashed identifier
4. Decrypt the message key with the sender-recipient key
5. Decrypt the ciphertext with the message key to recover the plaintext message
6. Verify the plaintext MAC

Figure 3.1 shows a diagram of an encrypted EASEmail message. A more detailed outline of the message packaging steps can be found in Appendix A.

3.3 Key Establishment and Exchange

EASEmail has a simple mechanism for user authentication and key distribution to senders and recipients. For key establishment and exchange we adopt Source-specific Symmetric Key Exchange (3SKE), and for authentication we leverage Simple Authentication for the Web (SAW) [22].



EASEmail Message

Sender: alice@foo.com
Key-Time: 06052008
Nonce: $RAND()$

Ciphertext-Sig: $\mathrm{HMAC}_{\kappa_{s,r_1}^{c\text{-}mac}}(M_E)$
Ciphertext-Sig: $\mathrm{HMAC}_{\kappa_{s,r_2}^{c\text{-}mac}}(M_E)$
$\vdots$
Ciphertext-Sig: $\mathrm{HMAC}_{\kappa_{s,r_i}^{c\text{-}mac}}(M_E)$

Encrypted recipient keys:
$\mathrm{HMAC}_{\kappa_{s,r_1}^{id}}(id_{r_1})$:$\mathrm{E}_{\kappa_{s,r_1}^{enc}}(\kappa_m)$
$\mathrm{HMAC}_{\kappa_{s,r_2}^{id}}(id_{r_2})$:$\mathrm{E}_{\kappa_{s,r_2}^{enc}}(\kappa_m)$
$\vdots$
$\mathrm{HMAC}_{\kappa_{s,r_i}^{id}}(id_{r_i})$:$\mathrm{E}_{\kappa_{s,r_i}^{enc}}(\kappa_m)$

$M_E=$

Encrypted Message

Plaintext-Sig: $\mathrm{HMAC}_{\kappa_{s,r_1}^{p\text{-}mac}}(M_P)$
Plaintext-Sig: $\mathrm{HMAC}_{\kappa_{s,r_2}^{p\text{-}mac}}(M_P)$
$\vdots$
Plaintext-Sig: $\mathrm{HMAC}_{\kappa_{s,r_i}^{p\text{-}mac}}(M_P)$

$M_P=$

Plaintext

Figure 3.1: EASEmail message contents (attachment)

KDC	Key Distribution Center
$\mathrm{KDF}_x()$	Key Derivation Function (using key x)
RAND()	Random number generator
κ_{kdc}	KDC master key
κ_m	message encryption key
κ_s	sender's derivation key
κ_{s,r_x}	sender-recipient specific key
κ_{s,r_x}^{enc}	encryption key (derived from κ_{s,r_x})
κ_{s,r_x}^{id}	identifier key (derived from κ_{s,r_x})
κ_{s,r_x}^{mac}	MAC key (derived from κ_{s,r_x})
$\kappa_{s,r_x}^{p\_mac}$	plaintext MAC key (derived from κ_{s,r_x})
$\kappa_{s,r_x}^{c\_mac}$	ciphertext MAC key (derived from κ_{s,r_x})
id_s	sender identifier
id_{r_x}	recipient identifier
$id_{r_x}^{dgst}$	hashed recipient identifier
M_E	encrypted message
M_P	plaintext message
mac_P	plaintext MAC
mac_C	ciphertext MAC
τ	time value

Table 3.1: Variable names and definitions

3.3.1 Source-Specific Symmetric Key Exchange

Source-specific Symmetric Key Exchange (3SKE)[1] is a lightweight symmetric key exchange protocol. This protocol includes a Key Distribution Center (KDC) whose responsibility it is to distribute keys to authenticated users. 3SKE is lightweight because it is stateless. The KDC only has to store its own private master key. It does not have to store user-specific keys or transaction information.

The KDC distributes keys to authenticated users based on a master key κ_{kdc}. This key is used in a Key Derivation Function (KDF) [12, 13] (e.g., HMAC [16] can be used as a KDF) to derive all other keys that the KDC distributes. To help mitigate compromised keys and avoid the hassles of key revocation, derived keys are short-lived, ranging from days to weeks to months, depending on the needs of the users of the KDC.

[1] 3SKE is unpublished work by Tim van der Horst.

Senders authenticate with the KDC to obtain their sender key κ_s. Each sender key is valid within a time period τ, as specified by the KDC. Sender keys are not used for message encryption, rather they are used to derive encryption keys. Senders derive sender-recipient keys κ_{s,r_x} offline using a KDF (see Figure 3.2), thus senders only have to interact with the KDC once per time period. A recipient obtains κ_{s,r_x} by authenticating with the KDC, which derives it using the same process the sender did (see Figure 3.3). The recipient never receives κ_s, the KDC derives this, then derives κ_{s,r_x}. Once κ_{s,r_x} has been delivered to the recipient both keys are discarded by the KDC.

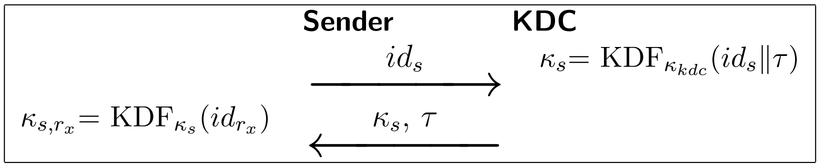

Sender
KDC
id_s
κ_s= KDF$_{\kappa_{kdc}}(id_s \| \tau)$
κ_{s,r_x}= KDF$_{\kappa_s}(id_{r_x})$
κ_s, τ

Figure 3.2: 3SKE Sender key exchange

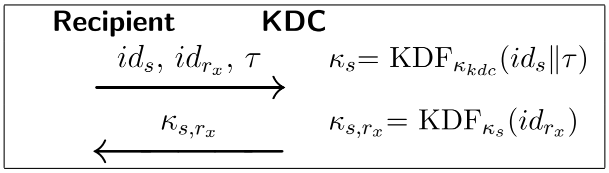

Recipient
KDC
id_s, id_{r_x}, τ
κ_s= KDF$_{\kappa_{kdc}}(id_s \| \tau)$
κ_{s,r_x}
κ_{s,r_x}= KDF$_{\kappa_s}(id_{r_x})$

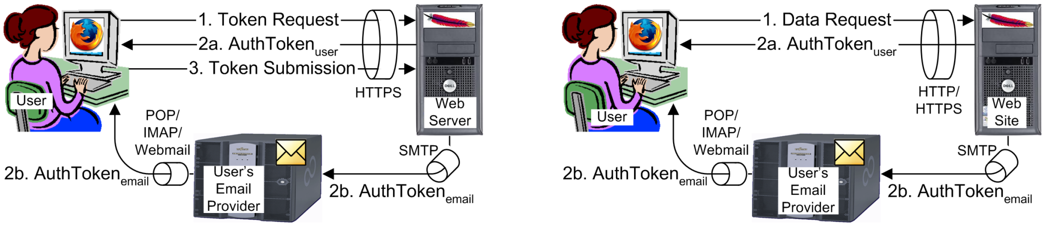
Figure 3.3: 3SKE Recipient key exchange

The 3SKE protocol does not include a mechanism for authentication, it assumes this has already been done. Any system that effectively authenticates users before keys are distributed will work. We elected to use Simple Authentication for the Web (SAW) due to its ease of use and natural fit to a secure email system.

3.3.2 Simple Authentication for the Web

Simple Authentication for the Web (SAW) [22] is an email-based authentication mechanism that uses a two token process to authenticate users. In traditional SAW (see Figure 3.4(a)), users submit their identifier to a website. The site then generates a token and splits it into two tokens. One token ($AuthToken_{user}$) is returned to the user as a browser cookie. The other token ($AuthToken_{email}$) is emailed to the email address provided. The user then retrieves the email and returns both tokens to the server. Her ability to do this proves she controls the email address. Software can automate this entire process, reducing the burden on the user.

One round SAW (see Figure 3.4(b)) is a variation on traditional SAW. It can be used in situations where a single response from the server, instead of a persistent connection, is needed. Instead of generating a random authentication token and throwing it away once the authentication is complete, the complete token is the server's response. One half ($AuthToken_{user}$) is returned directly to the user in the open session. The other half ($AuthToken_{email}$) is emailed to the address provided in the request. The two halves are combined to obtain the complete server response.



(a) Traditional SAW protocol

(b) One round SAW protocol

Figure 3.4: SAW protocols

3.3.3 Combining 3SKE and SAW for use with EASEmail

These two protocols integrate with each other seamlessly. Senders and recipients contact the KDC to request a key. In their request they specify their email address, which is their identifier. The KDC computes the correct key and uses one-round SAW for authentication by returning the first half of the key in the active session and sending the other half to the email address specified in the request. The user then checks her email, combines the two halves, and has the key she needs to complete the encryption or decryption of her message. This process is automated by the client software.

To speed up encryption/decryption and reduce KDC interactions, keys can be cached locally and requested in bulk. Further, recipients need not wait until a message is received to request the key. Recipients can pre-fetch keys to streamline the decryption process once a secure message is received.

3.4 Sending an EASEmail Message

Once the sender key has been obtained the message is packaged for transmission on the Internet. At a high level, the body of the message and any attachments are combined into a single MIME multipart message that is authenticated, encrypted, and then authenticated again to produce a ciphertext message. This ciphertext message, which is a MIME message that is contained in a single text file, consists of the encrypted message and the information needed for decryption as shown in Figure 3.1. It is sent as an attachment in a regular email to the recipient(s) specified by the sender. The KDC does not store or transmit the message. It is sent directly from the sender to the recipient(s) via their email providers, just as regular email is sent today.

Users unfamiliar with EASEmail will not know what to do if they were to receive a seemingly garbled message in their inbox. Thus, plaintext instructions are included with the message to help the recipient figure out what it is, and how to get the software to decrypt it.

The body of the message is replaced with instructions that aid the user in decrypting the message. These instructions contain a brief explanation to the recipient informing them that the attached file is an encrypted message. A link to the KDC is included, allowing the recipient to click the link and obtain the software needed to decrypt the message.

3.4.1 Message Encryption

The Cryptographic Message Syntax (CMS) [9] is used for message encryption. To improve performance, a message is encrypted only once for all recipients with a randomly generated message specific symmetric key κ_m. This key is then encrypted for each recipient with a key derived from the corresponding sender-recipient key κ^{enc}_{s,r_x} and a nonce η. The nonce forces identical messages encrypted with the same key to appear random.

Since the same encrypted attachment can be sent to multiple recipients, including blind carbon copy (BCC) recipients, each recipient needs to identify the correct encrypted message key. Encrypted message keys are associated with their corresponding identifiers by a key identifier. This approach potentially leaks information about the recipients of a message. If the plaintext identifiers of the message's recipients are used, the identity of BCC recipients will be revealed. Instead, the recipients' identifiers are hashed using an HMAC and a key derived from the sender-recipient key $\kappa^{id}_{s,r_x}$and η. A recipient can easily compute their hashed identifier, yet will be unable to determine the identities of the other recipients of the message.

Even though the key identifiers do not reveal who BCC recipients are, they do not hide the presence of the BCC recipients altogether. Two simple solutions to hiding the presence of BCC recipients involve either specifying a set number of message recipients (where fake recipients are inserted to make up the difference between the number of actual recipients and the number specified), or including a random number of fake recipients with each message.

Since adding fake recipients represents additional overhead in terms of message size, the number of fake recipients needs to be limited. The number of fake recipients is not mandated by the KDC and depends largely on the email habits of a particular sender. Individual senders can change this number according to their expected email usage. Present implementations have a set number of recipients of 20, which could be increased to 50 without affecting message size significantly. Even in situations where the presence of BCC recipients is leaked, their identities are still private because they have been hashed.

3.4.2 Message Authentication

Message authentication is accomplished by generating a Message Authentication Code (MAC) for each recipient of the message and setting it in a header of the message. A fake MAC has to be added for each fake recipient that is inserted during the encryption phase to disguise the presence of BCC recipients.

The authentication key κ_{s,r_x}^{mac} is derived from the sender-recipient key and η. To verify a message, a recipient simply uses her key and the message she received to generate a MAC, and compares the MAC she generated to the list of MACs in the message. If her generated MAC matches one of the message MACs, she accept the message.

The finer points of encrypt-then-authenticate and authenticate-then-encrypt have been discussed in security literature [4, 15]. For EASEmail we authenticate each message twice. First a MAC of the plaintext is computed for each recipient and the resulting MACs are set as headers in the message. Next the plaintext along with the MACs are encrypted, finally a MAC of the ciphertext is computed for each recipient and the resulting MACs are set as headers on the ciphertext.

The ciphertext authentication facilitates detection of modifications to the message while it was in transit. Invalid or altered messages can easily be detected and discarded without wasting further resources decrypting them. The plaintext authentication ensures that the correct key was used to decrypt the message and that the decrypted plaintext matches what was originally encrypted.

3.4.3 Subject Line Encryption

Leaving the subject line as plaintext can potentially leak sensitive information about the content of the message. Some users may assume that any and all text, including the subject, is encrypted and thus they can enter sensitive information in the subject. It is not unusual for account numbers, invoice numbers, etc., to be found in the subject line of an email. Some email messages consist of only a subject line with no text in the body.

There is value to leaving the subject line as plaintext, especially since EASEmail messages can be sent to recipients with whom the sender has never had contact. Even though plaintext instructions in the body of a message can help a recipient figure out how to decrypt the message, they may not motivate the user to actually follow the steps. With the pervasiveness of spam, many users unfamiliar with EASEmail and encrypted email may regard legitimate secure messages as junk. Conversely, a plaintext subject line with special meaning to the recipient may provide the motivation needed to decrypt the message. From our first motivating scenario, if Tim encrypts a message and in the subject line states the job to which he is applying, HR manager Bob will be more likely to take the time to decrypt the message.

For these reasons the subject line remains in plaintext and only the message body and attachments are encrypted. The same decision was made for both PGP and S/MIME to not encrypt the subject lines of encrypted email.

3.5 Spam

EASEmail lends itself very naturally to eliminating unsolicited email messages, commonly referred to as spam. A common practice among senders of spam (spammers) is to forge the message sender's address to make the message appear to come from a source known to, or trusted by, the recipient. Since EASEmail requires the sender to prove ownership of an email address before an encryption key can be obtained, spammers will only be able to send messages from accounts they actually control. If a spammer attempts to forge the sender address the authentication of the ciphertext MAC will fail and the message will be discarded. Even in situations where spammers do control a valid email account, the overhead involved in encrypting millions of messages will be too costly for them.

Alternatively, an email provider can use EASEmail to authenticate senders and easily filter out spam messages that contain forged sender addresses. In addition to authenticating the ciphertext and plaintext for the intended recipients as discussed previously, senders can also authenticate just the ciphertext for the recipient's email provider. In the same way a key is generated for a recipient of a message, another key can be obtained for the email provider itself. This enables email providers to authenticate messages before they are delivered to the recipient, without the email provider learning the content of the message (e.g., for Gmail, the ciphertext is authenticated for a well known identifier such as gmail@gmail.com, Gmail then retrieves the key and checks the MAC before delivering the message to the recipient's inbox).

For maximum benefit, these approaches require that all messages are EASEmail messages. In these cases EASEmail can be very effective in eliminating unsolicited messages.

Chapter 4

Implementation

A number of goals were considered important during the design and development of EASEmail. First, the email clients need to be easy to use. Instead of creating an entirely new email client for users to learn, existing email clients were extended in order to leverage the user's previous experience. A key aspect of this goal is to make sure the changes to the host program introduced by an extension are consistent with the existing interface and experience.

Since it is not practical to produce an extension for every email program, a zero footprint Java applet is also available. The applet is used in conjunction with the user's regular email program and performs encryption and decryption operations. This allows user's to simply copy and paste data into a normal email message and send it using their regular email program.

Second, it is essential to maintain client interoperability. While this goal may seem obvious, the Java applet presents constraints that need to be considered. The Java applet is not integrated with an email client, prohibiting the complete automation of the secure message transmission process (the actual encryption and decryption operations are still automated). In particular, usability will be adversely affected if users are required to format messages a particular way, ensure adherence to naming conventions on files, or other such details.

A valid EASEmail message is simply a normal email message with an attachment. This attachment contains the entire encrypted message, including encrypted attachments. This ensures that sending and receiving encrypted messages is easy even with the Java applet. Users only need to download the EASEmail attachment and hand it off to the Java applet. All other aspects of the message do not affect the validity of a secure message.

Finally, users may accidentally insecurely send messages that are intended to be secure. Efforts to warn of or prevent such actions would likely be ignored or become annoying. Instead the problem must be solved through an intuitive user interface that effectively communicates the status of the message during composition.

The EASEmail implementation consists of five components: a KDC, a Java library, a Java applet, an extension for the Mozilla Thunderbird email client, and an extension for the Mozilla Firefox web browser for use with Gmail. Figure 4.1 shows how these components interact.

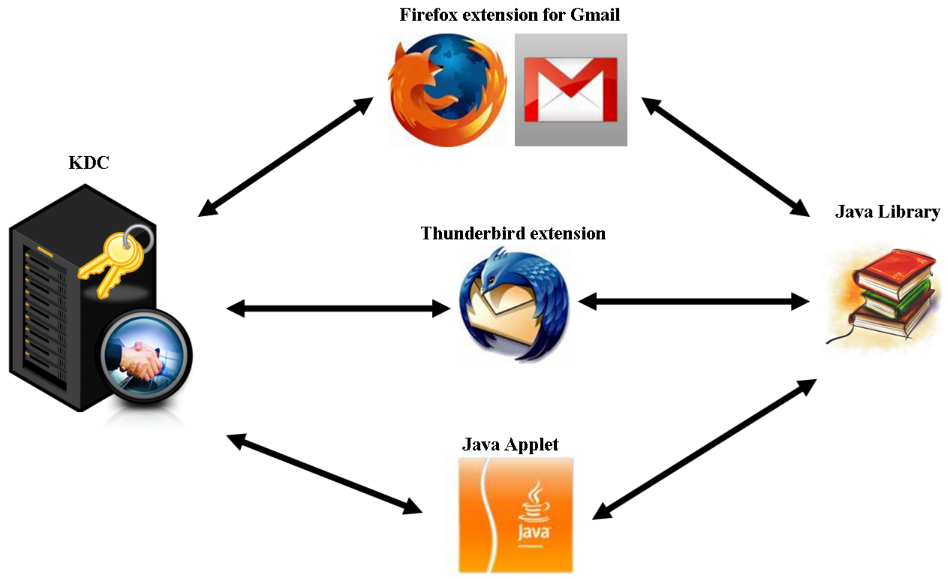


Figure 4.1: EASEmail components and how they interact

4.1 KDC

The KDC is a Java servlet that runs on a Tomcat web server. It responds to client key requests and serves web pages to browsers. Browsers and EASEmail clients use the same URL to access the KDC, which automatically differentiates between normal web browser traffic and key requests.

For browser traffic, users are directed to a website that allows them to learn about EASEmail and download the available client software. Users are also able to access the Java applet by authenticating with their email address using SAW.

EASEmail clients communicate with the KDC in XML. The Java applet receives plaintext key responses from the KDC over HTTPS (users have already been authenticated when they access the Java applet). The Thunderbird and Firefox extensions receive encrypted key responses from the KDC over HTTPS that require a one-round SAW authentication to decrypt.

The KDC master key κ_{kdc} is stored in a local keystore on the local disk. For more secure deployments it can also be stored in tamper resistant cryptographic hardware [11, 21].

4.2 Java Library

All three clients access a shared library for encrypting and decrypting messages. This library leverages the Bouncy Castle S/MIME and CMS Java Crypto APIs [3] for message encryption and decryption. The Java applet accesses the Java library directly, whereas the Thunderbird and Firefox extensions do not. Since Thunderbird and Firefox do not provide effective mechanisms to call Java code from JavaScript, remote procedure calls are used instead. Firefox does have a technology called LiveConnect that allows Java code to be called from JavaScript, but this approach proved problematic and LiveConnect is scheduled to be removed from future versions of FireFox.

In order to enable remote procedure calls into the library, a lightweight frontend was built to allow the library to run as a local service. The service is started any time a call into the library is needed. The service opens a port and communicates with the client using XML over HTTP.

4.3 Firefox Extension for Gmail

The Firefox browser extension is specifically designed to work with Gmail. It was implemented using JavaScript and accesses the local service.

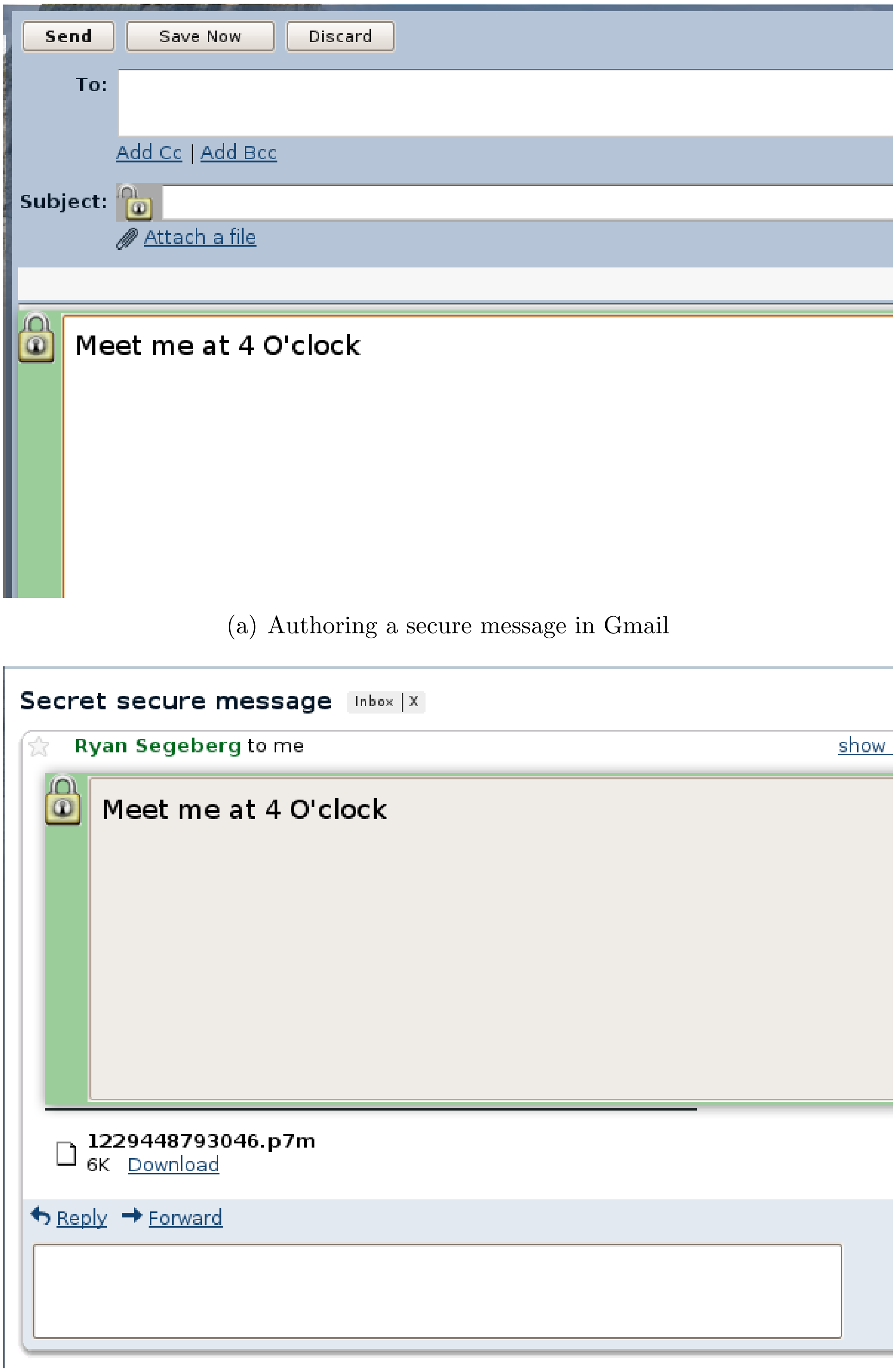
4.3.1 Challenges

The Firefox client presents some unique challenges. First, the plaintext of secure messages must not be viewable by the email provider. Second an effective mechanism that works well in web environments must be used to help ensure users do not accidentally send secure messages insecurely. Finally, the extension must integrate tightly with the web interface, creating a seamless experience that preserves the existing interface and leverages the user's existing experience.

Avoid authoring on the page Web-mail interfaces are controlled by the email provider, allowing providers and potentially man-in-the-middle attackers (for situations where HTTPS is not used) to access email plaintext even during composition. The trust model is two fold: the web browser operates in the trusted local computer environment while the web interface operates in the untrusted web environment.

Email providers access messages during composition in limited ways. Automatically saved drafts of the plaintext are common, and can possibly exist beyond the duration of message composition (e.g., a server backup). Additionally, some email providers scan message text in order to serve targeted ads.

Since the web-mail interface is not trusted, composing and displaying secure messages on the web page needs to be layered. This is accomplished by using an overlay that is placed on top of the usual composition or reading areas. This way messages are never entered on the web page itself but are instead authored or displayed in the trusted browser (see Figures 4.2(a) and 4.2(b)). This provides a seamless interface where the user composes and reads her messages in the same way she does without the extension, yet the plaintext never leaves her local computer.



Meet me at 4 O'clock

(a) Authoring a secure message in Gmail

Secret secure message

Ryan Segeberg to me

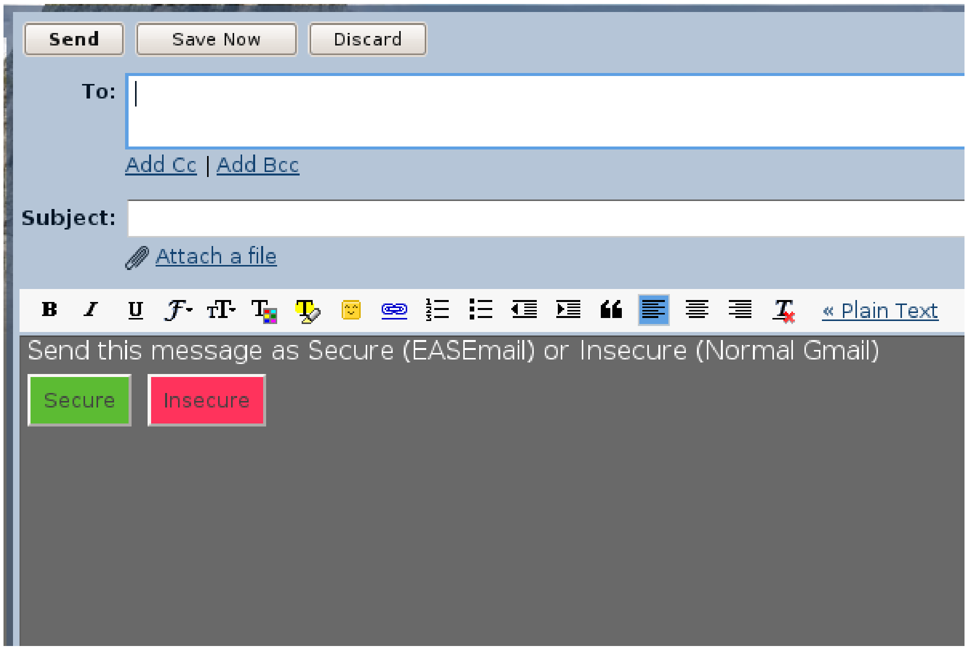
Meet me at 4 O'clock

1229448793046.p7m
6K Download

(b) Reading a secure message in Gmail

Figure 4.2: Gmail overlay interface

When to decide to encrypt For the Thunderbird extension the user is able to toggle security on and off at any time during message composition. However the use of overlays complicates this approach. Insecure compositions do not use the overlay; the regular Gmail composition interface is used instead. If the user composes her message in the regular interface, then later decides she wants to secure it, the message may have already been compromised. Instead the user is required to specify whether she wishes to compose the message securely or insecurely before she begins composing her message (see Figure 4.3) so that secure messages remain secure the entire time they are being composed.



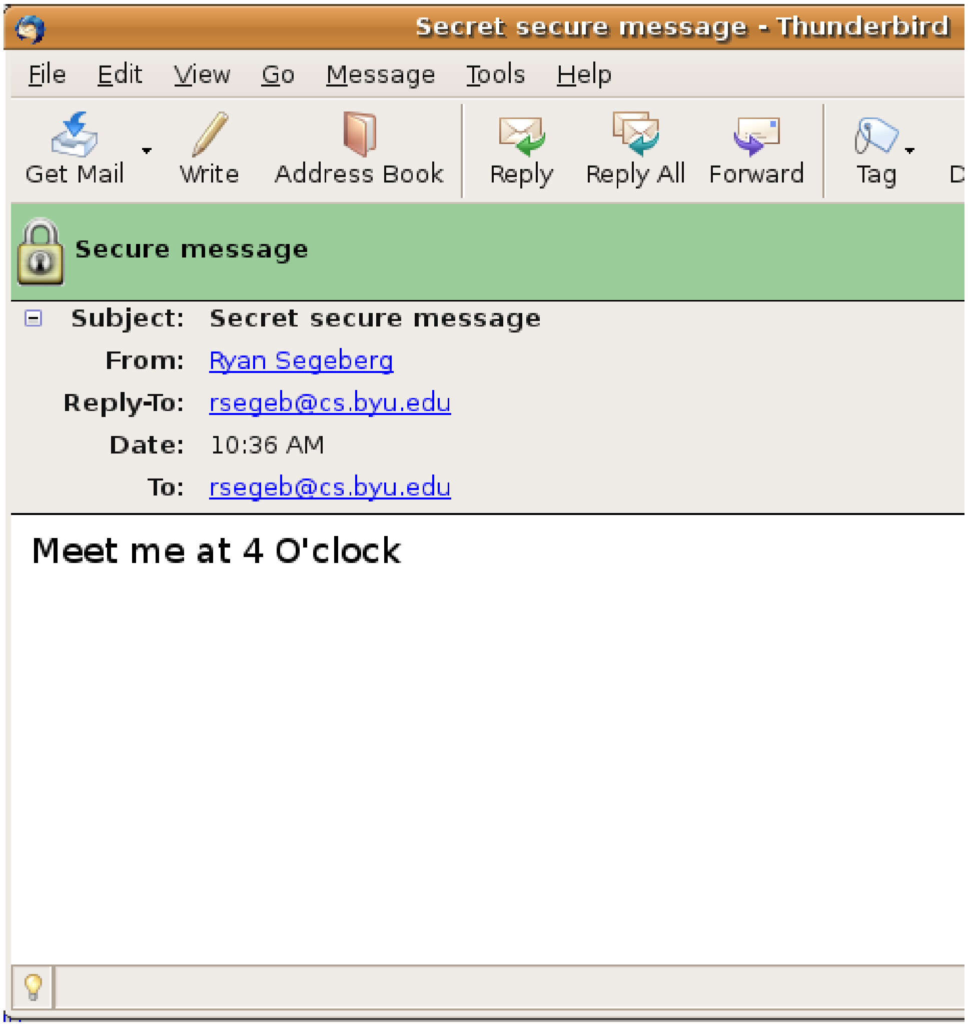
Send this message as Secure (EASEmail) or Insecure (Normal Gmail)

Figure 4.3: Secure or Insecure prompt

Consistent interface The extension integrates tightly with the regular Gmail interface, providing a seamless and natural interface so the user experience remains largely unchanged. As the user navigates web pages the extension automatically detects when a Gmail composition or reading page is loaded. When these pages are detected the extension automatically displays the appropriate prompts for composing a message or opens the correct overlays for reading a message.

4.4 Thunderbird Extension

The Mozilla Thunderbird extension was written in JavaScript and accesses the local service. When a new message arrives in the inbox the message is scanned for attachments. If an EASEmail attachment is found it is decrypted with the sender-recipient key which is requested from the KDC if it is not already cached locally. When a user views a message that arrived encrypted she is notified that the message is secure (see Figure 4.4). EASEmail messages are stored decrypted on the user's local computer, but remain encrypted on the email server.



Secure message

Subject: Secret secure message

From: Ryan Segeberg

Reply-To: rsegeb@cs.byu.edu

Date: 10:36 AM

To: rsegeb@cs.byu.edu

Meet me at 4 O'clock

Figure 4.4: Reading a secure message in Thunderbird

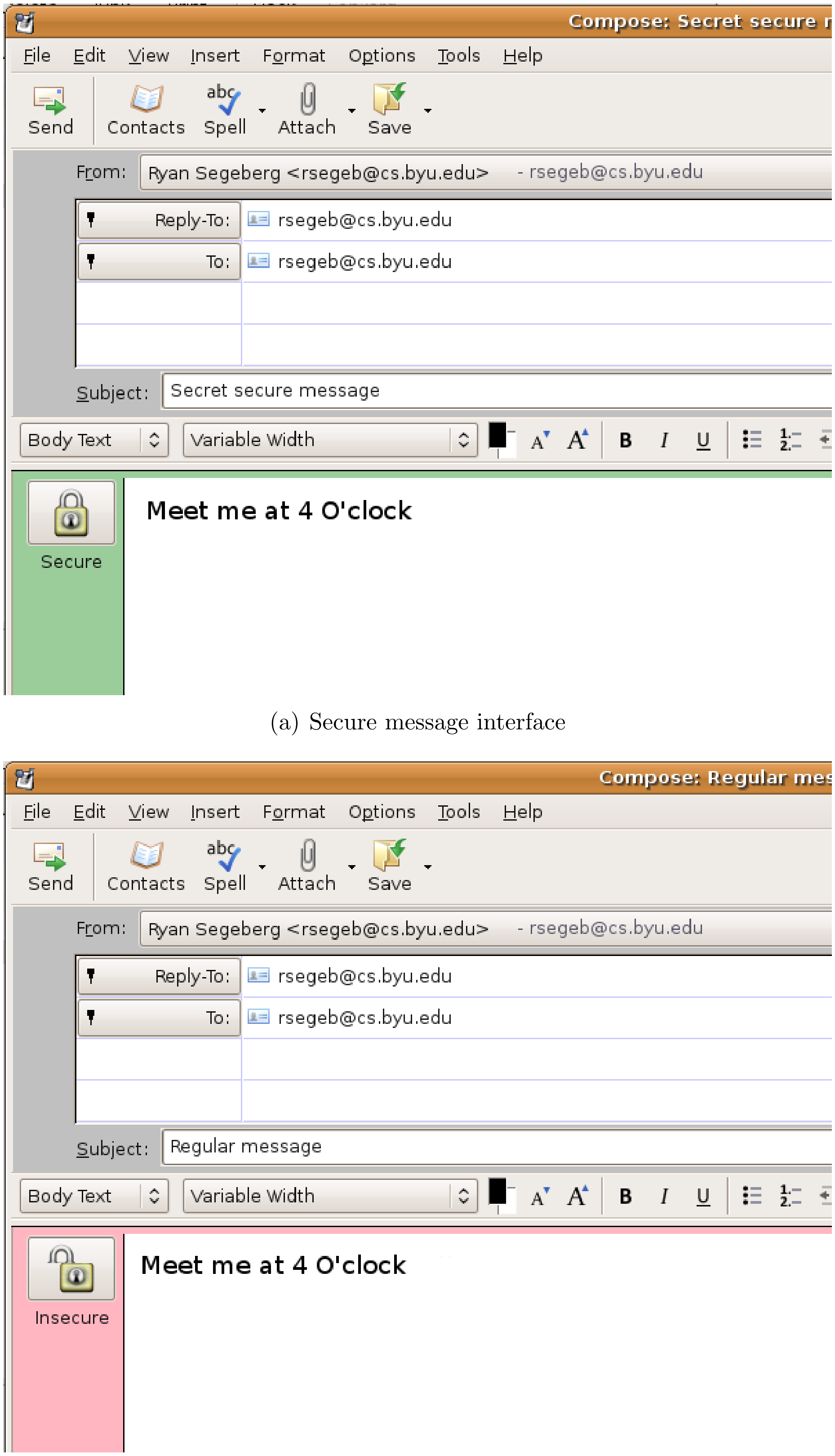
Authoring a secure message in Thunderbird is almost the same as authoring a traditional message. The message composition window is modified to allow the user to toggle whether or not to secure the message. To help minimize the chance users will accidentally send messages insecurely a colored border serves as a visual cue to indicate whether or not a message is secure. By default messages are not secure. Users can toggle message security on or off at any time during message composition (see Figures 4.5(a) and 4.5(b)).

4.5 Java Applet

The zero-footprint Java applet is intended for users who either cannot install one of the extensions on the computer they are using or do not have an email application for which an extension has been developed. The applet performs encryption and decryption operations and requires a regular email client or web-mail interface for message transmission. The user is required to transfer data to/from the applet manually. EASEmail messages are contained entirely within one physical file in order to simplify this process.

The Java applet is hosted by the KDC. To access the client, a user navigates to the applet login page and authenticates using a traditional SAW authentication by submitting her email address. The first half of the SAW token is returned as a cookie in the user's browser and the other half is sent to her email address. The user can click the link included in the email response to complete the authentication or download the SAW toolbar to automate the process.

The applet includes two tabs, one for authoring secure messages (see Figure 4.6(a)) and one for reading secure messages (see Figure 4.6(b)). To author a message the user selects the create secure message tab and fills in the body of the message, attaching files if desired. A list of message recipients is also required so they can be used as identifiers during message encryption.



From: Ryan Segeberg <rsegeb@cs.byu.edu> - rsegeb@cs.byu.edu

Reply-To: rsegeb@cs.byu.edu

To: rsegeb@cs.byu.edu

Subject: Secret secure message

Meet me at 4 O'clock

(a) Secure message interface

From: Ryan Segeberg <rsegeb@cs.byu.edu> - rsegeb@cs.byu.edu

Reply-To: rsegeb@cs.byu.edu

To: rsegeb@cs.byu.edu

Subject: Regular message

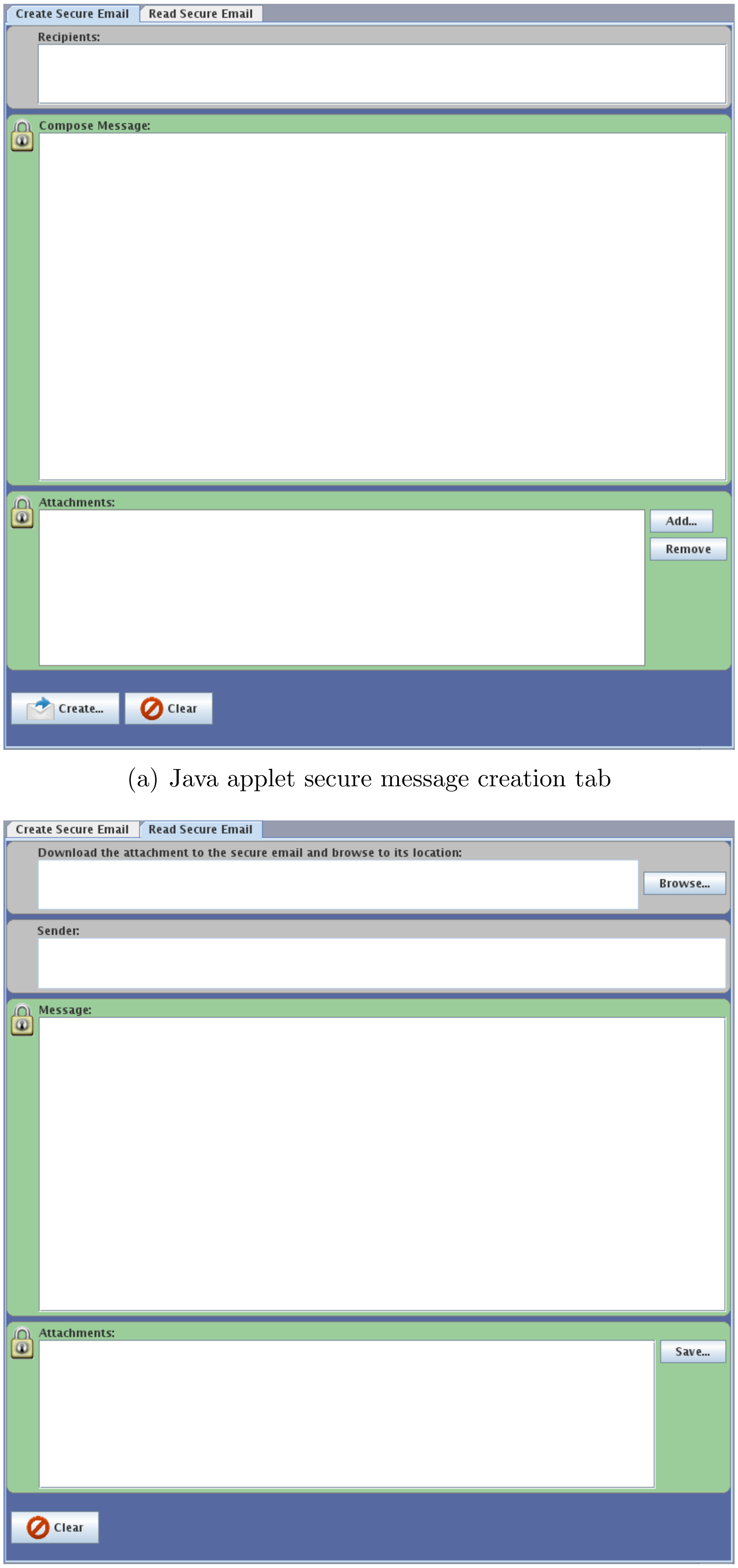
Meet me at 4 O'clock

(b) Insecure message interface

Figure 4.5: Thunderbird message composition

Once a sender has authored a secure message she is presented with a dialog from which she can copy and paste the recipients' addresses and the decryption instructions (see Section 3.4) into her email client or web-mail interface (see Figure 4.7). This dialog also contains the location of the file containing the secure EASEmail message generated by the applet so the user can easily attach it to an email message.

A recipient uses the decryption tab to read downloaded EASEmail attachments. She does so by browsing to a local copy of the attachment. If the user is a valid recipient of the message the sender-recipient key is automatically retrieved and the plaintext of the message is displayed along with any attachments. The plaintext message is stored in a local variable in the applet and is garbage collected when the applet is closed unless the user manually copies and pastes it into another program.

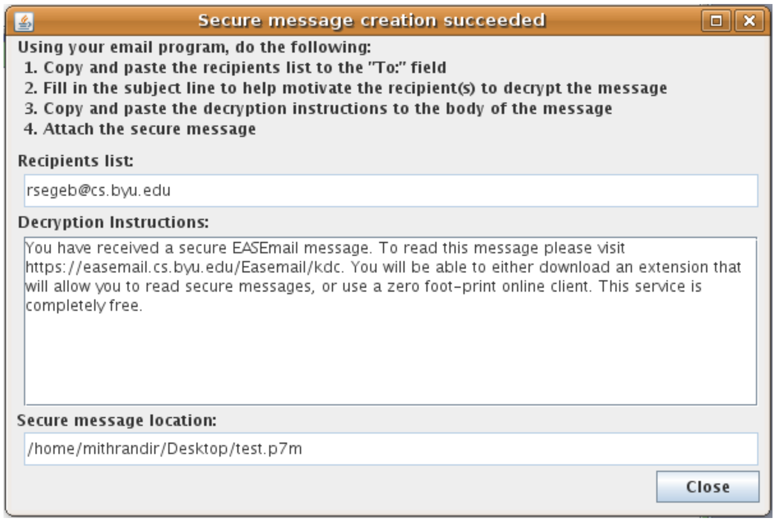


Download the attachment to the secure email and browse to its location:

(a) Java applet secure message creation tab

(b) Java applet secure message decryption tab

Figure 4.6: Java applet interface



Using your email program, do the following:

1. Copy and paste the recipients list to the "To:" field
2. Fill in the subject line to help motivate the recipient(s) to decrypt the message
3. Copy and paste the decryption instructions to the body of the message
4. Attach the secure message

Recipients list:

rsegeb@cs.byu.edu

Decryption Instructions:

You have received a secure EASEmail message. To read this message please visit https://easemail.cs.byu.edu/Easemail/kdc. You will be able to either download an extension that will allow you to read secure messages, or use a zero foot-print online client. This service is completely free.

Secure message location:

/home/mithrandir/Desktop/test.p7m

Figure 4.7: Java applet secure message creation information screen

Chapter 5

Threat Analysis

This section explores the security strengths and weaknesses of EASEmail. First we present the threat model used for this analysis.

5.1 Threat Model

The following is a list of possible attacks and the capabilities of the adversaries that may conduct these attacks.

5.1.1 Attacks

Six major attacks are possible with any email system: observation, traffic analysis, impersonation, replay, message modification, and denial of service.

5.1.2 Attackers

Two different attack models are considered: passive attacks and active attacks. Passive attacks are represented by Eve. Eve is capable of performing either of the first two attacks: observation or traffic analysis. Her goal is to observe email traffic and extract data of interest such as social security numbers, credit card numbers, names, addresses, etc. Eve is limited strictly to observing messages while in transit or as they reside on the email provider's server (in the case of an insider).

Mallory is an active attacker who is capable of all six attacks. She not only wants to extract useful data from normal message transmissions but also wants to mislead users in an effort to get them to divulge personal information that she can use for personal gain, either by leading them to a counterfeit website (e.g., phishing) or taking control of their computer (e.g., malware or spyware). She is also capable of initiating connections with open services such as the KDC or a user's email provider.

5.2 Threats

5.2.1 Eavesdropping

Eve can eavesdrop on EASEmail messages as they flow from the sender to the recipient. The only information Eve can gather is the address of the sender, the recipient's address, and the KDC used. Eve cannot read the actual contents of the message. She can also determine the timing and size of the messages. The use of the nonce with each message prevents Eve from detecting instances when the same message is sent multiple times. In these cases the messages will be roughly the same size, but will look completely different.

5.2.2 Denial of Service

As with any email system, Mallory can intercept EASEmail messages and prevent their delivery. There will be no indication to the recipient that a message was not received. While the KDC could determine that the key needed to decrypt the message was never retrieved (through logging), this capability is not presently in the system. Instead the sender has to detect that the message was never delivered (perhaps through a follow up email) and resend the original message manually.

5.2.3 Message modification

Mallory is capable of modifying EASEmail messages while in transit. She can remove parts, add parts, or simply fiddle with bits. However, any change to the actual message content would be detected by the recipient during the ciphertext authentication check. Once this check fails, the recipient can inform the sender.

A more effective modification attack would involve altering the decryption instructions. Since these instructions are not included in the message authentication they can be changed just as a regular email message can be. The main changes that can be made are to either remove the instructions altogether (so the user does not know what to do with the message) or modify them to point to a different KDC or website.

Being directed to a malicious website should not be a problem for users familiar with the system. The only information they are used to giving the KDC is their email address (which Mallory already has). Since there is no account creation process to use the system, there is no password to phish or other information to harvest. Users who are new to the system should already be wary about the information they provide it. So while the site could request personal information, users should be careful about what they disclose.

5.2.4 Replay

Mallory can save EASEmail messages she observes and replay them at a later time. As with observation, she will not know the contents of the message. A recipient could potentially detect a replay attack by keeping track of the nonces used. When a duplicate nonce is detected then the user would know the message was replayed. This feature is not presently implemented in the clients.

5.2.5 Impersonation to other users

In addition to replaying messages, Mallory can also resend observed EASEmail messages with the sender address modified. This attack is thwarted by EASEmail since the incorrect keys will be requested and the MAC checks will fail, causing the message to be discarded.

5.2.6 Impersonation to the KDC

The weak link is the use of SAW for authentication. $AuthToken_{user}$ is delivered in the session with the KDC. Mallory can start a session with the KDC and obtain $AuthToken_{user}$ for any pair of senders and recipients. $AuthToken_{email}$ is delivered via the requester's email address. It is not possible for Mallory to request a key for a pair of email addresses then request that $AuthToken_{email}$ be sent to an arbitrary email address. $AuthToken_{email}$ is always sent to the address specified in the request. Thus the only ways to obtain $AuthToken_{email}$ are to either eavesdrop on the legitimate user's email traffic, or compromise their email credentials. Since email providers have the ability to read users' email messages, they can impersonate the user at the KDC and receive both pieces of the key, allowing them to arbitrarily decrypt messages.

This is the envelope level security discussed previously. Given the right circumstances Mallory can still decrypt messages, but Eve will not be able to read any messages. In high confidentiality situations a stronger authentication mechanism can be used to solve this problem.

5.2.7 Surreptitious Forwarding

With asymmetric encryption, messages are typically authenticated then encrypted. This method provides assurances that the sender authored the message, but does not provide any guarantees as to who encrypted it (because everyone has access to the sender's public key), leaving the message vulnerable to surreptitious forwarding. Message signing only needs to be done once and each recipient can verify the signature using the sender's public key. This signature proves that Alice sent the message and that the plaintext has remained unchanged.

Authenticating a message symmetrically is a bit different from its asymmetric counterpart. Typically a symmetric key is shared between two parties. When the first party (Bob) receives a message he knows the other party (Alice) was the one who encrypted it because only he and Alice have the encryption key. In this scenario surreptitious forwarding is not possible since the parties with access to the encryption key are well known.

This is complicated when optimizations are added to deal with messages that have many recipients. In these cases, messages are first encrypted with a unique message key. This message key is then encrypted with each sender-recipient key. This way actual message encryption is only done once. This creates a problem however because now no one has any guarantees as to who encrypted the message, since more than two parties possess the key. For example, if Bob receives a message from Alice, and knows that Charlie is also a recipient, he can modify the message, re-encrypt it (leaving the rest of the message intact) and send it to Charlie. Now Charlie has two messages that are very different and both look like they are from Alice. In reality he does not know who encrypted it.

Message authentication solves this problem. By including a MAC digest for each recipient based on the sender-recipient key, Bob no longer has the ability to resend modified versions of the message without being detected. The authentication essentially restores the two party scenario that the multiple recipients optimization removed.

5.2.8 Compromised Keys

As discussed previously it is possible for Mallory to impersonate a user and obtain their keys. This section outlines the damage that Mallory can do with the compromised keys. We start with the most damaging compromise, and end with the least damaging.

Compromise of the KDC master key The worst case scenario is the compromise of the KDC master key. Compromising the KDC master key should be hard, and can only be done by compromising the KDC itself. Never at any time is the master key, or pieces of it, sent over the network.

Since it is used to derive all other keys, its compromise means Mallory has the ability to decrypt any message ever encrypted with a key from the KDC. To prevent such loss the key can reside in cryptographic hardware [11, 21], giving the key greater tamper resistance. Another solution would be to involve several independent KDCs. Each KDC could contribute to the final key, which would then be combined by the client using a key combination algorithm (e.g., XOR). This has the added benefit of Mallory needing to compromise several KDCs, but adds the overhead of having to contact all the KDCs before a key can be obtained. While this operation can be done in parallel, it requires all the KDCs to be available at all times. The complexity of compromising the KDC master key needs to be balanced with the confidentiality requirements of the users of the KDC.

Compromise of sender key If Mallory somehow obtains a sender key she can derive any sender-recipient keys she desires. This means she can impersonate the sender for the remainder of the time period associated with the key. Since keys are sender-recipient specific a compromised sender key does not mean Mallory can decrypt received messages. She is only able to send encrypted email; she has obtained no information that helps her decrypt received messages, including responses to sent messages.

Compromise of recipient key If Mallory obtains a sender-recipient key she has only obtained the ability to decrypt messages from one sender to a particular recipient for the remainder of the time period associated with the key. She can decrypt messages for the sender/recipient pair associated with the key, and no others.

Chapter 6

User Study

The user study consisted of students from the Computer Security class at BYU. This group was determined to be particularly appropriate for a user study because they had completed an assignment that required them to obtain, sign, and exchange PGP keys, and an assignment that required them to send and receive a message secured with S/MIME. With this experience with existing secure email solutions, they seemed ideal for evaluating the usability of EASEmail.

For the study participants received an email introducing the study which also included a survey to complete once they finished the tasks outlined in the email (see Appendix B for the complete text of the email and survey). In the email, participants were instructed to wait for an encrypted EASEmail message. Once the message was received they were directed to the EASEmail website where they could download and install one of the clients, or access the Java applet. Their task was to read the encrypted message and author and send a secure response. Once they completed this task (or at least attempted to) they were asked to fill out the survey and submit it via email (see Appendix B for the unedited survey responses).

6.1 Results

A total of 15 users signed up to participate in the study and 12 submitted a completed survey. Of the 12 that completed the survey, 9 or 75% successfully sent a secured response email. All three who did not send a secure response reported problems with decryption. One of these users (U_3) used an auto forwarding address to their regular email account. EASEmail does not currently support this since the software requests decryption keys based on the address of the account the message was received at instead of where it was sent to. The other two were both Mac users. The clients were tested on a Mac system running OS X, so it is a mystery as to why both Mac users were unable to successfully use the extension.

6.1.1 Multiple Choice Results

Of those who were able to decrypt and encrypt successfully, 8 out of 9 (88.9%) agreed that decrypting and encrypting messages was easy, and that the user interface was effective in communicating when they were authoring or reading secure messages.

Of all the study participants, only 3 of 12 (25%) disagreed that decrypting was easy, 2 of 12 (16.7%) disagreed that encrypting was easy, and 1 of 12 (8.33%) disagreed that the user interface was clear.

6.1.2 Short Answer Results

A number of concerns were raised in the short answer portion of the surveys:

- Encryption and decryption operations in Thunderbird in Windows causes a command window to open briefly, which startled the user. This was known prior to the study.
- For the Thunderbird client, the size of the secure toggle button was too large, taking up too much screen space. The user suggested moving the toggle button to the toolbar instead.
- Several users mentioned security concerns with the use of SAW, and problems with receiving SAW response emails (either taking too long, or not being cleaned up properly).

- A few users mentioned that some documentation would have been helpful. Some brief installation instructions were made available for the Thunderbird client, otherwise documentation was purposefully omitted because the goal was to integrate so closely with the host client's interface that a user familiar with it would find the changes made by the extension intuitive.
- Some expressed concerns with the prompt in Gmail that forces the user to decide whether or not to author a secure message for each message. Many users view the sending of secure messages as an infrequent task and do not want the extension to intrude on their normal work flow.
- The use of Java and a backend service seems to be the cause of some general stability and ease of use issues.

6.1.3 Solutions

In response to the concerns raised in the user study, the following solutions can be considered:

- The current placement of the secure toggle button was selected so it would be obvious to users. While its placement was based on user evaluation, additional evaluation can be used to achieve a more optimal placement.
- SAW can be replaced with an alternate authentication mechanism. A few possibilities include:
    - Users can be required to establish an account with the KDC and supply their credentials to their EASEmail client. The client can connect to the KDC directly over HTTPS and receives a direct response.
    - An alternate distributed authentication mechanism such as OpenID [17] or sSRP [8] can be used so maintain the lightweight nature of the KDC. This

still allows direct connections for fast key retrieval, but does not require the user to establish an account with yet another password.

  - SAW can be used once during client installation to establish a secret between the KDC and the client. This secret can be derived using κ_{kdc}, the user's identifier, and a salt value (so it can be changed if the secret is compromised). The client provides the secret for authentication each time it needs to obtain a key. The KDC does not have to store the secret since it can be derived at any time. This way the user does not have to create an account and KDC interactions will be fast.

- An introductory tutorial can be developed to help users who are unfamiliar with secure email understand how it works and the changes the extension makes to their experience.

- The Gmail extension can take a similar approach as the Thunderbird extension, allowing the user to toggle security on and off. This would require using the overlay all the time, even for insecure messages, so users would not expose the plaintext of their messages. For this situation the implementation of a rich text editor would likely be desirable.

- Instead of using Java, the extension can rely on C/C++ code accessed using XPCOM. This would increase the speed and deployability of the extension.

Chapter 7

Conclusions and Future Work

Using 3SKE and SAW, we designed and implemented a simple yet effective authenticated key exchange protocol. We built a lightweight KDC that does not require user specific management or the storage of user specific long term secrets. The server facilitates the establishment and exchange of secrets needed for users to exchange secure email messages without communicating prior to the exchange.

We designed and implemented three different clients for use with the EASEmail system. The Thunderbird and Firefox clients integrated tightly with their host applications, creating a seamless interface. Users were able to easily decrypt secure messages and author secure responses without having to manage keys.

We designed and implemented a solution to compose and receive secure email messages using a web interface in a way that prevents the email provider from being able to view the plaintext. Never at any time is the plaintext of a secure message available for the email provider to view. This was done in a way that integrates tightly with the email provider's existing interface so the changes are intuitive and natural for the user.

We conducted a user study to evaluate the effectiveness of EASEmail. While a few issues did surface during the study, users generally liked it and felt it was easy to use. With additional development effort to refine the user interface and improve the efficiency of the system we are confident that these issues will be resolved.

A number of improvements to the EASEmail system have been identified. Some of these possible improvements were outlined in Section 6.1.3. Additionally, add-ons for popular email clients (e.g., Outlook) and online email providers (e.g., Yahoo and Hotmail) need to be developed to cater to a larger user base.

We plan to continue to develop and improve EASEmail into a solution anyone can use. After improving the existing clients it is our intent to host a KDC for public use and to make the clients available generally.

EASEmail has the potential to be a widely used secure email solution. The underlying technology makes it easy to use and secure. It addresses the key establishment and exchange problem and allows users to simply focus on the content of their messages instead of the security of them.

Bibliography

[1] Ben Adida, Susan Hohenberger, and Ronald L. Rivest. Lightweight encryption for email. In *SRUTI'05: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet Workshop*, July 2005.

[2] Dan Boneh and Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, August 2001.

[3] Bouncy Castle Crypto APIs. http://www.bouncycastle.org/java.html.

[4] Don Davis. Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, June 2001.

[5] freenigma. http://www.freenigma.com/.

[6] Simson L. Garfinkel. Enabling email confidentiality through the use of opportunistic encryption. In *dg.o '03: Proceedings of the 2003 Annual National Conference on Digital Government Research*, May 2003.

[7] Simson L. Garfinkel, David Margrave, Jeffrey I. Schiller, Erik Nordlander, and Robert C. Miller. How to make secure email easier to use. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, April 2005.

[8] Andrew Harding, Timothy W. van der Horst, and Kent E. Seamons. Wireless authentication using remote passwords. In *WiSec '08: Proceedings of the first ACM conference on Wireless network security*, April 2008.

[9] Russell Housley. Cryptographic Message Syntax. http://tools.ietf.org/html/rfc3852, July 2004.

[10] Hushmail. https://www.hushmail.com/.

[11] IBM PCI-X Cryptographic Coprocessor. http://www.ibm.com/security/cryptocards/pcixcc/overview.shtml.

[12] IEEE Std 1363-2000: IEEE Standard Specifications for Public-Key Cryptography.

[13] IEEE Std 1363a-2004: IEEE Standard Specifications for Public-Key Cryptography - Amendment 1.

[14] Himanshu Khurana and Jim Basney. On the Risks of IBE. In *IWAP '06: Proceedings of the International Workshop on Applied PKC*, November 2006.

[15] Hugo Krawczyk. The order of encryption and authentication for protecting communications (Or: how secure is SSL?). Cryptology ePrint Archive, Report 2001/045, 2001. http://eprint.iacr.org/.

[16] NIST. The Keyed-Hash Message Authentication Code (HMAC). FIPS Pub 198a.

[17] OpenID. http://openid.net/.

[18] OpenPGP. http://tools.ietf.org/wg/openpgp/.

[19] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, August 1985.

[20] S/MIME. http://tools.ietf.org/wg/smime/.

[21] Sun Crypto Accelerator 6000 PCIe Card. http://www.sun.com/products/networking/sslaccel/suncryptoaccel6000/index.xml.

[22] Timothy W. van der Horst and Kent E. Seamons. Simple Authentication for the Web. In *SecureComm '07: Proceedings of the 3rd International Conference on Security and Privacy in Communication Networks*, September 2007.

Appendix A

EASEmail Message Details

A.1 Encryption

1. Get sender key κ_s if not already cached:
   (a) Request κ_s from KDC, obtain half directly from KDC
   (b) Retrieve second half of κ_s from email
   (c) Combine both halves to get final κ_s
2. Generate nonce $\eta = RAND()$
3. For each recipient derive $\kappa_{s,r_x} = \text{HMAC}_{\kappa_s}(id_{r_x})$
4. Sign plaintext for each recipient:
   (a) Derive $\kappa_{s,r_x}^{p\_mac} = \text{HMAC}_{\kappa_{s,r_x}}(Const_{Plain}, \eta)$
   (b) Compute $mac_P = \text{HMAC}_{\kappa_{s,r_x}^{p\_mac}}(M_P)$
   (c) Set mac_P as a header in the message
5. Insert fake plaintext signatures:
   (a) Generate R_{max}-R_{act} random numbers (same length as HMAC output)
   (b) Set random numbers as fake plaintext signatures in the message
6. Generate $\kappa_m = RAND()$
7. Encrypt κ_m for each recipient:
   (a) Derive $\kappa_{s,r_x}^{enc} = \text{HMAC}_{\kappa_{s,r_x}}(Const_{Encrypt}, \eta)$
   (b) Derive $\kappa_{s,r_x}^{id} = \text{HMAC}_{\kappa_{s,r_x}}(Const_{ID}, \eta)$
   (c) Compute $id_{r_x}^{dgst} = \text{HMAC}_{\kappa_{s,r_x}^{id}}(id_{r_x})$
   (d) Encrypt κ_m with κ_{s,r_x}^{enc} $(\text{E}_{\kappa_{s,r_x}^{enc}}(\kappa_m))$
   (e) Set key identifier in message with encrypted $\kappa_m$associated with $id_{r_x}^{dgst}$

8. Insert fake key identifiers:

   (a) Generate R_{max}-R_{act} random numbers for keys (same length as κ_s)

   (b) Generate R_{max}-R_{act} random numbers for identifiers (same length as $id_{r_x}^{dgst}$)

   (c) Encrypt κ_m with each generated key

   (d) Set the key identifiers in the message with each encrypted $\kappa_m$associated with the generated identifiers

9. Encrypt M_P with κ_m

10. Sign ciphertext for each recipient:

    (a) Derive $\kappa_{s,r_x}^{c\_mac} = \mathrm{HMAC}_{\kappa_{s,r_x}}(Const_{Cipher}, \eta)$

    (b) Compute $mac_C = \mathrm{HMAC}_{\kappa_{s,r_x}^{c\_mac}}(M_E)$

    (c) Set mac_C as a header in the message

11. Insert fake ciphertext signatures:

    (a) Generate R_{max}-R_{act} random numbers (same length as HMAC output)

    (b) Set random numbers as fake ciphertext signatures in the message

12. Set id_s, τ, and η as headers in the message

A.2 Decryption

1. Extract id_s, τ, and η from message headers

2. Get sender-recipient key κ_s if not already cached:

   (a) Request κ_{s,r_x} from KDC, obtain half directly from KDC

   (b) Retrieve second half of κ_s from email

   (c) Combine both halves to get final κ_s

3. Derive $\kappa_{s,r_x}^{c\_mac} = \mathrm{HMAC}_{\kappa_{s,r_x}}(Const_{Cipher}, \eta)$

4. Compute $mac_C = \mathrm{HMAC}_{\kappa_{s,r_x}^{c\_mac}}(M_E)$

5. Verify mac_C matches one of the ciphertext signature headers in the message

6. Derive $\kappa_{s,r_x}^{id} = \mathrm{HMAC}_{\kappa_{s,r_x}}(Const_{ID}, \eta)$

7. Compute $id_{r_x}^{dgst} = \mathrm{HMAC}_{\kappa_{s,r_x}^{id}}(id_{r_x})$

8. Derive $\kappa_{s,r_x}^{enc} = \mathrm{HMAC}_{\kappa_{s,r_x}}(Const_{Encrypt}, \eta)$

9. Find the key identifier using $id_{r_x}^{dgst}$ and decrypt the encrypted key with κ_{s,r_x}^{enc} to get κ_m
10. Decrypt M_E with κ_m
11. Derive $\kappa_{s,r_x}^{p\_mac} = \mathrm{HMAC}_{\kappa_{s,r_x}}(Const_{Plain}, \eta)$
12. Compute $mac_P = \mathrm{HMAC}_{\kappa_{s,r_x}^{p\_mac}}(M_P)$
13. Verify mac_P matches one of the ciphertext signature headers in the message

Appendix B

User Study Details

B.1 Introduction email

Secure Email (Part 3)

Overview In previous assignments you have been exposed to both S/MIME and PGP. Now you will have the opportunity to try out a new secure email solution.

Part 3: EASEmail You will participate in a brief user study of Easy Secure Accessible Email (EASEmail). To do this you will need the Java 5 JRE or higher installed on your system. You will be required to use one of three different clients (2 of which require installation on your system) to complete this assignment. If you already have a Gmail account, and you use Firefox, it is preferable for you to download and install the Firefox extension for Gmail. More details on this will be provided during the assignment.

What to do:

1. Wait to receive an encrypted EASEmail message. Follow the plaintext instructions included with the secure email to decrypt it
2. Send an encrypted response answering the question in the secure email (the answer isn't as important as the encrypted response)
3. Send a plaintext email to rsegeb@cs.byu.edu answering the survey questions below. Please include CS465 in the subject. You must complete this step to receive credit

Notes:

- You will need a password when accessing the EASEmail website. The username is easemail, the password is password. The password exists for intellectual property protection purposes only and would not be present in a deployed system. Please ignore its presence in your evaluation.
- You MUST have Java 5 or higher installed on your system

- Attachments are not supported in the Firefox extension for Gmail
- The Thunderbird extension does not support IMAP

B.2 Survey

Please answer the following questions. Carefully consider your answers, don't just fly through them.

System information:

1. Which operating system did you use? Include as much detail as possible (e.g., Windows XP SP3, Ubuntu Linux 8.10, etc)
2. Which browser did you use (include version)? (e.g., I.E. 7, Firefox 3.0.3, etc)
3. Which EASEmail client did you use? (e.g., Thunderbird, Firefox, Java applet)
4. Which version of the Java JRE do you have?
5. Which email provider did you use?
6. How do you usually access your email? (e.g., web-based, client, etc) If you use a client, which one(s)?

Experience feedback: Respond to each of the following with one of 5 answers: A. Strongly Disagree B. Disagree C. Neither agree or disagree D. Agree E. Strongly Agree

7. The decryption instructions included with the initial secure email were clear.
8. Downloading and installing the client (or accessing the online client) was easy.
9. Decrypting the email was easy to do.
10. Authoring a secure email was easy to do.
11. The user interface gave me clear feedback so I knew when I was reading or authoring a secure message, and when I wasn't.

Short answer questions:

12. If you used the Firefox extension for Gmail, how long have you been a Gmail user?
13. How much time (minus answering these questions and waiting for the initial secure email) did you spend on this assignment?

14. Describe your overall experience, what went well, what didn't? Did you run into any problems or bugs? Please be specific.
15. What are your initial impressions of the system?
16. Would you use this system as a practical secure email solution? Why or why not?
17. What parts, if any, of the system or user interface were confusing?
18. How might this system be improved?

Thank you

B.3 Responses

This section contains the unedited responses of the user study participants. They are organized by user for each question, so U_2 in question 1 is the same user as U_2 in question 2, etc.

Question 1 (OS)

U_1 Windows XP Service Pack 3

U_2 Windows XP sp3

U_3 Ubuntu 8.04

U_4 Windows Vista

U_5 Windows XP SP3 - A computer in the CAEDM Computer Lab

U_6 I used windows XP. I tried on Linux and had problems running the applet. I am not sure why.

U_7 Windows Vista sp1

U_8 Mac OS X 10.5.5

U_9 Windows XP SP3

U_{10} Windows Vista 64 Business

U_{11} XP-sp3

U_{12} OS X 10.5.5

Question 2 (Browser)

U$_1$ I used Firefox 3.0.4 to download the Easemail Thunderbird extension.

U$_2$ Firefox 3.0.4

U$_3$ Gecko/2008111317 Ubuntu/8.04 (hardy) Firefox/3.0.4

U$_4$ Firefox

U$_5$ Firefox 3.0.1

U$_6$ Fire Fox, the version in the cs windows labs.

U$_7$ firefox 3.04 (updated today)

U$_8$ Firefox 3.0.4

U$_9$ Firefox 3.0.4

U$_{10}$ Firefox 3.04

U$_{11}$ Firefox 3.0.4

U$_{12}$ Firefox 3.0.4

Question 3 (client used)

U$_1$ The Thunderbird extension

U$_2$ Firefox

U$_3$ Firefox

U$_4$ Firefox

U$_5$ Firefox with Gmail

U$_6$ Fire fox plugin

U$_7$ Firefox plugin

U$_8$ Firefox

U$_9$ Firefox

U$_{10}$ Thunderbird

U$_{11}$ Firefox

U$_{12}$ Firefox

Question 4 (Java version)

U_1 I'm running the Java SE Runtime Environment build 1.6.0_10-b33. That'd make it Java 6 or something... I don't pay much attention to Java, so I'm not exactly sure how they're doing it nowadays.

U_2 Java 1.6.0_10-rc

U_3 Java 1.6.0_06

U_4 1.6.0_07

U_5 Java Version 6 Update 5

U_6 6, I think.

U_7 1.6.011 beta 3 (updated recently for javaFX)

U_8 I'm not sure, and not sure how to find out

U_9 JRE 6.10

U_{10} 1.6_10

U_{11} Java 5

U_{12} 1.6.0_07

Question 5 (email provider)

U_1 I use my personal web server's POP server.

U_2 byu's premium email

U_3 Gmail

U_4 gmail

U_5 Gmail

U_6 gmail

U_7 gmail

U_8 gmail

U_9 GMail

U_{10} gmail

U_{11} gmail

U_{12} Gmail

Question 6 (how email accessed)

U_1 I usually use Thunderbird for all of my e-mail

U_2 client, outlook

U_3 Web Based

U_4 web based

U_5 Web-based

U_6 umm, I go to gmail.com. I am not sure which category that is in.

U_7 web-based

U_8 web-based

U_9 Evolution usually. (I tried doing the web based GMail in Linux but I didn't have the Java JRE working properly.)

U_{10} 1st web-based then Thunderbird.

U_{11} web-based(firefox)

U_{12} Web-based client. Sometimes I also use Apple Mail.

Question 7 (clear instructions)

U_1 Strongly Agree

U_2 B (You need to re-word your instructions explaining everything)

U_3 A - I couldn't get it decrypted.

U_4 disagree

U_5 D

U_6 D

U_7 E

U_8 D

U_9 E

U_{10} B

U_{11} E

U_{12} E

Question 8 (installing client is easy)

U_1 Neither agree nor disagree.

U_2 C I personally found it a pain to have to wait for email and then connect, but if the client handles that for me I guess it wouldn’t be as much as an issue.

U_3 D

U_4 agree (i wish i didn’t have to restart firefox...)

U_5 E

U_6 D

U_7 E

U_8 E

U_9 E

U_{10} E

U_{11} E

U_{12} D

Question 9 (decryption is easy)

U_1 Strongly Agree.

U_2 B Yes, once I realized I had to stick in the email It was sent to and not my gmail account. I’m not sure why gmail was mentioned so many times especially after the fact that we had given our emails to you guys.

U_3 A

U_4 agree

U_5 E

U_6 E

U_7 E

U_8 A (it didn’t work)

U_9 E

U_{10} D

U_{11} E

U_{12} N/A [counted as neutral]

Question 10 (encryption is easy)

U_1 Strongly agree

U_2 E

U_3 A

U_4 disagree. easemail timed out on me as it was negotiating keys or something. i had to try to send the message twice.

U_5 E

U_6 E

U_7 E - switching tabs re-initiated the the process, except it didn't recognize that there was now a draft: which had to be deleted before it would let me "reply"

U_8 C (never got a chance to do it)

U_9 E

U_{10} E

U_{11} E

U_{12} N/A [counted as neutral]

Question 11 (clear UI)

U_1 Strongly agree, but I think the indicator takes up too much space (especially because of the colored background wrapped around the text area). A toolbar button indicator would serve better, I think.

U_2 E

U_3 A

U_4 neutral. i'm getting tons of spam in my mail. how do i get rid of this easemail now?????

U_5 E

U_6 E

U_7 D/E Yes, but wasn't clear if replying or not

U_8 C

U_9 D

U_{10} D

U_{11} D

U_{12} N/A [counted as neutral]

Questions 7-11 Summary

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
7	1	3	0	3	5
8	0	0	2	4	6
9	2	1	1	2	6
10	1	1	2	0	8
11	1	0	3	3	5

Question 12 (Gmail experience)

U_1 blank

U_2 N/A

U_3 I have used gmail for 2+ years.

U_4 quite awhile... 4 - 5 years

U_5 I've been a Gmail user since August of 2004 (4 years).

U_6 2 years

U_7 3 years

U_8 3 years

U_9 2 years

U_{10} 5+ years

U_{11} one year.

U_{12} 4 years

Question 13 (time spent)

U_1 Two or three minutes is all.

U_2 15 minutes.

U_3 30+ minutes

U_4 20 minutes.

U_5 It took about 10 minutes. Most of that time was opening a different version of Firefox because the default version on the lab machine was 2.0.0.16.

U_6 15 mins

U_7 10-15 minutes

U_8 20 minutes

U_9 10 minutes

U_{10} approx 1 hr

U_{11} 15 mins

U_{12} 15 minutes

Question 14 (overall experience)

U_1 The command window popups when I first started Thunderbird and when sending an e-mail were startling and required me to unblock Java’s port in Windows’ firewall. If the popups could be eliminated and the port unblocked automatically somehow, that would make it a lot nicer to use. When I sent the message it made Thunderbird think I had received a new e-mail, so I’ve got a phantom ’new e-mail’ icon in my task tray right now.

U_2 I already mentioned my beefs above. After I realized how the system worked it went smoothly. I would suggest a flash tutorial or something for future new users.

U_3 Installing everything was fine. Then the plug-in would pop up every few seconds and not really do anything other than say it was checking for an EASEMail message. Even when I opened your encrypted email message it checked....but didn’t do anything. The part that I think is the worst is that there are no instructions that I could find anywhere that say what I might want to try next.

U_4 not too well. i dno’t know how to get rid of easemail when i’m done with it. i get lots of spam in my mailbox that i don’t want from easemail and it didnt’ encrypt my message right on the first try – it timed out.

U_5 The overall experience was pretty smooth. The only two issues I encountered were cosmetic: the security overlay was a little off-center and if I scrolled up to the original message when I was replying the overlay would shrink to a small rectangle that would remain on the bottom of the screen even though I was scrolled completely past it.

U_6 I was very confused by the website. It took me a while to realize that I had been given a password, so I didn't know how to log in.

U_7 Was good up until the point where I swiched tabs... Upon return the process started over. Going into the drafts didn't work either... had to delete the draft to start over.

U_8 It was taking forever do decrypt. I tried several times, and had to restart firefox completely. It was very frustrating, especially after the speed and ease of the other steps

U_9 I had Java issues in Linux...kind of out of your control. I don't know how to encrypt a message that is not a reply.

U_{10} The first time I did the lab, the secure email site crashed - the Java servlet loaded, but did not display. When refreshed the URL, I saw the Tomcat admin webpage. I tried it again that evening and was successful. I downloaded and installed the Firefox AND Thunderbird plugins, but they both did not work. The message was not ”automatically” taken care of as the e-mail stated.

U_{11} Good experience. Gmail experience was no different than before.

U_{12} Neither encryption nor decryption completed successfully; the progress bar got stuck halfway through decryption, and the request for an encryption key timed out. I looked in the Console, and found the following error: 12/11/08 8:50:05 PM [0x0-0xcd0cd0].org.mozilla.firefox Unable to access jarfile /Users/bjhomer/Library/Application. It looks like the Firefox plugin is looking for a jar and has a wrong path.

Question 15 (initial impressions)

U_1 I don't feel one way or the other about it at this point.

U_2 I don't like the idea of needing multiple emails to get one email, but it seems secure. It does worry me that if someone were to get into my email they would have access to all of my encrypted emails since the system sends an authentication token to your email. If the site requires some sort of password I guess it might be more secure, granted if someone has access to my email, they might already have access to my password.

U_3 I think it needs to go through a few more iterations.

U_4 if you guys get the bugs worked out, it could work better...

U_5 The system definitely makes secure email significantly easier to send and receive. The Gmail interface is still a little rough around the edges but works well (which is the most important part).

U_6 I liked it a lot.

U_7 I liked it

U_8 I think it's a good idea. Although, I would like to see an option made that would allow me to disable the prompt for a secure/insecure email. (I know what it looks like because I went to the demo. Lucky thing too, or I'd have no idea what this thing was supposed to do).

U_9 Encrypted and decrypting was easy. It does all the complicated 'key exchanges' automatically. no need to exchange keys or certificates

U_{10} A bit raw since I had to do all manually without the assistance of the plugin.

U_{11} I think it is great. This way I can actually start sending encrypted email, since the current technologies are a pain in the butt to send encrypted mail.

U_{12} My impression from using the system on my own computer is that it could be cool, but needs some more debugging. I also saw your presentation at the CS Demo Day, though, and I think that if I could install it, it would be very easy to use.

Question 16 (use again?)

U_1 Personally, I might use it after the quirks have been worked out of the Thunderbird extension and after I've learned what it's doing behind the scenes (to satisfy myself that it is, in fact, secure). I would also want to be assured that it wouldn't interact poorly with my other secure e-mail methods (certificates and PGP).

U_2 Doubtful, unless it was integrated into outlook and didn't use java. Java has the ugliest user interfaces. In all honesty if you don't make a nice integration into outlook you are kind of acting like the business aspect of the world isn't important, but be careful that's where real power lies in trying to set new precedents.

U_3 No. Because it doesn't work.

U_4 yeah, if you guys get the bugs worked out and it doesn't spam me all the time. right now, i just want to take it off.

U_5 I think I would use it. My only hesitation would be putting the security of my messages in a third-party's hands (since, if I understand correctly, the encryption keys are stored on the EASEmail server).

U_6 It is easy to use, but I don't know how secure it is.

U_7 If the interface looked better and had a menu bar with options (reply, reload, discard, etc...)

U_8 Yeah, sure. If I didn't have to choose secure/insecure every single time I wanted to send an email. I would definitely miss the draft and search features. I use those a lot, and that would deter me from using secure email unless I absolutely had to. Which is unlikely.

U_9 It's pretty good. Kind of a weird setup because it relies on a Firefox plugin and then Java. I had to update my Java (I'm not sure if I needed to or if the plugin installer thing just wasn't working right the first time).

U_{10} If the plugins worked, yes.

U_{11} Definitely. See previous answer.

U_{12} I find the searchability of Gmail's web interface to be highly useful, so I probably wouldn't use it for casual use. If I were using an offline client or had trade secrets, I'd be likely to use something like this.

Question 17 (UI confusing?)

U_1 I don't think anything was particularly confusing, though the command window popups surprised me.

U_2 N/A

U_3 The fact that I didn't ever see a user interface.

U_4 finding the initial firefox link was confusing. i didn't know that I had to download something for firefox. i thought there was going to be a java applet that would run on a web page and i just work through taht applet. i wasn't looking for a download, but all the links i was seeing were download links. make it more clear that you're giong to download somethign for all of the options and make it clear where those downloads are.

U_5 I didn't find anything to be particularly confusing. The Firefox / Gmail plug-in installed without any probably and functioned pretty seamlessly.

U_6 Everything was clear, except the website login

U_7 Not confusing, just lacking / not very appealing

U_8 I think it's fine. (again, saw the demo. you guys made it look easy

U_9 I don't see how to encrypt a composed message.

U_{10} The dialog that appears after you create the return secure e-mail suggests that the secure e-mail was sent. I wasn't sure so I responded to the original email with the encrypted file with my encrypted response.

U_{11} Nothing. They were are clear, visible, and user friendly.

U_{12} I found the user interface to be quite good.

Question 18 (impovements?)

U_1 I think I've mentioned the two ways I would improve the system - make the command window popups go away, and move the secure/insecure indicator take up a *lot* less space (preferably moving it to the toolbar of the message composition window).

U_2 Outlook add in

U_3 Sorry I have been so harsh. I feel that I am a pretty confident computer user and this addon just flat out didn't work.

U_4 see above.

U_5 From my brief experiences with the system, the only thing that could use improvement is the fit and finish of the Gmail integration (I didn't try Thunderbird). It's usable as is but between little delays when the overlay is placed on top of the compose window and slight offsets it doesn't feel polished yet. Again, it doesn't really hurt usability but could cause frustration to users.

U_6 Maybe have some flashing boxes on the website, it would be cool.

U_7 Worked well outside of small bugs mentioned

U_8 Speed up the time it takes to check for encrypted mail and try to decrypt it.

U_9 A imple manual

U_{10} a) The plugins should work as advertised. b) Always tell the user what will happen next if they follow the directions so that they know if they did something wrong.

U_{11} Maybe instead of placing the secure compose box on top of the gmail page, have gmail adopt it as a standard in their actual page.

U_{12} Just fix the bugs, I think.

Replied securely

U_1	Yes
U_2	Yes
U_3	No
U_4	Yes
U_5	Yes
U_6	Yes
U_7	Yes
U_8	No
U_9	Yes
U_{10}	Yes
U_{11}	Yes
U_{12}	No