Secure Email with Fingerprint Recognition

Ahmed Obied

Department of Computer Science
University of Calgary
obieda@cpsc.ucalgary.ca
http://www.cpsc.ucalgary.ca/~obieda

Abstract. *Public key cryptographic techniques have been used to protect email messages via encryption and digital signatures for more than 26 years. Such techniques, however, failed to adopt secure email messaging due to a combination of technical, social, and usability issues. We present a new approach to email security that uses fingerprint recognition and cryptographic hash functions to secure access to email accounts and messages, and to sign and verify email messages. Our approach does not require doing expensive computations to verify a user's signature as opposed to public key cryptographically protected email. We keep the amount of user interaction required to the minimum, and provide email users with security features that include state-of-the-art biometric authentication schemes.*

Key words: Biometrics, Email, Privacy, Security, Authentication, Fingerprint recognition, Signatures, Cryptography, Hash Functions, Public key Cryptography, Spam

Table of Contents

Secure Email with Fingerprint Recognition 1
Ahmed Obied
1 Introduction 3
2 Previous work 4
2.1 Existing Standards and Technologies 5
PEM (Privacy Enhanced Mail): 5
S/MIME (Secure Multipurpose Internet Mail Extensions): 6
PGP (Pretty Good Privacy): 6
2.2 Current Research 7
3 Approach 11
3.1 Components 11
Environment: 11
Mail servers and accounts: 11
Database: 12
Fingerprints: 12
3.2 Design and Implementation 12
Enroller: 12
Login: 13
Inbox: 13
Sending email messages: 13
Verifying email message: 14
3.3 Testing 14
4 Benefits 14
4.1 Secure email access 15
4.2 Prevent email spoofing 15
4.3 Prevent against man-in-the-middle attacks 16
5 Future Work 17
6 Conclusion 17

List of Figures

1 Spoofed email address 4
2 Email messages are vulnerable to interception at many points 7
3 Enigma system configuration 10
4 SEFR's high-level design 15

1 Introduction

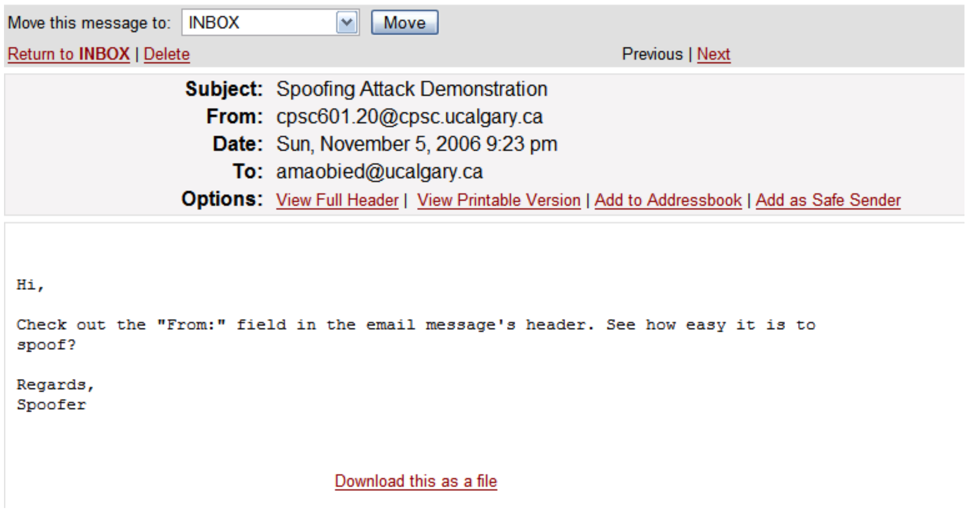
Email changed the way we communicate in today's highly technical world. Its usage increased tremendously in the last few years and millions of users world wide joined this technological revolution that made the world look so small and at our disposal. The widespread use of email caused the number of warnings being made about the dark side our technological revolution to increase and we are becoming uniquely vulnerable to many mysterious and malicious threats. Viruses, worms, and other forms of malicious software started targeting our email inboxes to propagate. Spam and other forms of unsolicited bulk electronic commerce started filling our email inboxes and invading our privacy. Phishing and other forms of fraud attacks have been using email as their primary communication channel to trick users into giving out their credentials. Email could have been a killer application for the Internet if none of the problems mentioned above exist.

Email messages move across the Internet from mail clients to mail servers, from mail servers to other mail servers, and from mail servers to mail clients in the clear. Messages can be intercepted and read by unauthorized or unintended individuals. Furthermore, email messages can be automatically scanned for keywords of interest to an eavesdropper. It has been widely reported that national intelligence agencies are already performing such searches on a large scale [6]. Email can also be surreptitiously modified–even forged (figure 1)–creating the impression that a person made a statement that he did not [8]. The use of secure communication channels can protect email messages indeed. However, deploying such secure channels is not possible in a large-scale environment with distributed management. As a result, the only way to protect Email is via the use of cryptography. Yet even though cryptographic technology is now built into the email program being used by most Internet users, few messages that travel over the Internet are actually secured [9].

Cryptographically protected email technologies such as PGP (Pretty Good Privacy), PEM (Privacy Enhanced Mail), S/MIME (Secure Multipurpose Internet Mail Extensions) have been proposed and developed to assure integrity, privacy or establish authorship of email messages. Encryption algorithms provide two primary functions for email. *Signing* attaches to the email a digital signature which can be used to verify the authenticity of the sender and to detect message tampering. *Sealing* scrambles the content of a message so that it cannot be deciphered by anyone other than the indented recipient [7]. Cryptographically protected email technologies have the potential to solve the problems with today's Internet mail. However, such technologies have a justly deserved reputation of being difficult to use [8]. PGP, PEM, and S/MIME use public-key cryptography which is a form of cryptography that generally allows users to communicate securely without having prior access to a shared secret key. This is done by using a pair of cryptographic keys, designated as public key and private key, which are related mathematically [3]. Maintaining keys, and ensuring the security of the public and private keys of users is a difficult problem indeed. The security of a public-key cryptosystem depends upon the security of the public and private keys of users. The security of public-key cryptosystems can be broken if the private key of a user gets compromised by an adversary or if the public key of a user is altered.

To solve the problems with cryptographically protected email technology and make email security more appealing to users we can simply use biometrics. Humans have used body characteristics such as face, voice, and gait for thousands of years to recognize each other [10] but only recently humans started developing and deploying biometric based systems to authenticate individuals. Biometrics is the science of establishing or determining an identity [10] based on the physical or behavioral traits of an individual such as fingerprints, iris, face, palmprint, hand vein, voice, keystroke, retina, facial thermogram, etc. Biometric systems are essentially pattern recognition systems that read as input biometric data, extract a feature set from such data, and finally compare it with a template set stored in a database. If the extracted feature set from the given input is close to a template set stored in the database then the user is granted access.

In this paper, we present an approach that uses fingerprint recognition to secure access to email accounts and messages, and uses cryptographic hash functions to sign and verify users' messages.


Move this message to: INBOX
Move
Return to INBOX | Delete
Previous | Next
Options: View Full Header | View Printable Version | Add to Addressbook | Add as Safe Sender
Download this as a file

Subject: Spoofing Attack Demonstration
From: cpsc601.20@cpsc.ucalgary.ca
Date: Sun, November 5, 2006 9:23 pm
To: amaobied@ucalgary.ca

```
Hi,

Check out the "From:" field in the email message's header. See how easy it is to
spoof?

Regards,
Spoofer
```

Fig. 1. Spoofed email address

2 Previous work

In the past 26 years, numerous efforts have been made to make secure email possible, if not ubiquitous. In this section we provide an in-depth discussion of the standards, technologies, and algorithms that were proposed and developed. We also provide a discussion on current research related to email security.

2.1 Existing Standards and Technologies

PEM (Privacy Enhanced Mail): In mid-1980, the Internet Activities Board's Privacy Task forced started to develop standards designed to provide end-to-end encryption for email [8]. These standards became known as *PEM* and they defined a way to provide encryption and signature for ASCII email messages based public-key cryptography using the RSA (Rivest Shamir and Adelman) algorithm. Using PEM, if Alice wants to send an encrypted message M to Bob then Alice will have to do the following:

1. Encrypt M using Bob's public key K_b^{public} which must be published in digital certificates as defined by the X.509 CCITT standard. After encrypting M, $C = K_b^{public}(M)$ is obtained.
2. Send C to Bob.

When Bob receives C, Bob will have to do the following:

1. Decrypt C using his private key $K_b^{private}$ which must be stored on his computer. After decrypting C, $M = K_b^{private}(K_b^{public}(M))$ is obtained.

Since Bob is the only one who has access to his private key then he is the only one who can decrypt the message. If Alice wants to send a digitally signed message M to Bob, then Alice will have to do the following:

1. Encrypt M with her private key to obtain $S = K_a^{private}(M)$.
2. Send M and S to Bob.

When Bob receives M and S, then Bob will have to do the following:

1. Decrypt S using Alice's public key $K_a^{private}$. After decrypting S, $M = K_a^{public}(K_a^{private}(M))$ is obtained.
2. Check if the decrypted M is equal to M that was sent.

If the decrypted message is equal to the message sent then Alice must have sent the message since she is the only one who has access to her private key. On the other hand, if the two messages are not equal then the message must have been altered during transmission or forged.

In 1989, there was not a centralized online public key directory so PEM was designed to operate without one. Each signed message included all of the certificates in the Chain needed to verify the message signature so when a message is received, PEM implementations would store those accompanying certificates on the recipient's computer [8].

S/MIME (Secure Multipurpose Internet Mail Extensions): When MIME (Multipurpose Internet Mail Extensions) was introduced, RSA Data Security re-implemented the PEM concept on top of the MIME standard and called it *S/MIME*. MIME defines mechanisms for sending other kinds of information in email, including text in languages other than English using character encodings other than ASCII as well as 8-bit binary content such as files containing images, sounds, movies, and computer programs [1].

Because of single root with a single certification policy proved to be problematical in PEM, S/MIME implementations do not implement a strict hierarchy of certificates, but instead accommodates any number of trusted CA (Certificate Authority) [8]. S/MIME these days is integrated into many email clients like Microsoft Outlook, Netscape Communicator, Lotus Notes, and others. However, S/MIME is not integrated into any web-based mail systems like Gmail, Hotmail, Yahoo, etc. On web-based mail systems, S/MIME digitally signed S/MIME messages appear as ordinary messages with an additional attachment name *smime.p7s*, while S/MIME messages that are sealed with encryption are indecipherable [8].

PGP (Pretty Good Privacy): In 1991, Phil Zimmermann released a program called *PGP* that provides cryptographic privacy and authentication for email. PGP uses public-key cryptography (as in PEM and S/MIME) and includes a system which binds the public key to user identities [3]. If Alice wants to send an encrypted message M to Bob using PGP then PGP does the following at Alice's side (| denotes concatenation):

1. Generates a shared key K_s.
2. Encrypts M using K_s to obtain $C_1 = K_s(M)$.
3. Encrypts M and K_s using Bob's public key K_b^{public} which must be stored in Alice's public key database. After encrypting C_1 and K_s, PGP obtains $C_2 = K_b^{public}(C_1|K_s)$.
4. Sends C_2 to Bob.

When Bob receives C_2, PGP does the following:

1. Decrypts C_2 using Bob's private key $K_b^{private}$ stored on Bob's computer. After decrypting C_2, PGP obtains $C_1|K_s = K_b^{private}(K_b^{public}(C_2))$.
2. Decrypts C_1 using K_s to get $M = K_s(C_1)$.

A similar strategy is (by default) used to detect whether a message has been altered since it was completed, or (also by default) whether it was actually sent by the person/entity claimed to be the sender [3]. Digital signature algorithms (e.g., RSA, DSA) and hash functions are used to create a digital signature for a message in PGP. If Alice wants to send a digitally signed message M to Bob, then PGP does the following:

1. Computes the hash for message M using a one-way hash function (e.g., SHA1, MD5, etc) to obtain $H(M)$.

2. Encrypts $H(M)$ with Alice's private key to obtain $S = K_a^{private}(H(M))$.
3. Sends M and S to Bob.

When Bob receives M and S, PGP does the following:

1. Decrypts S using Alice's public key $K_a^{private}$ stored on Bob's computer. After decrypting S, PGP obtains $H(M) = K_a^{public}(K_a^{private}(H(M)))$.
2. Computes the hash for the sent M and check if its equal to $H(M)$.

If the computed hash value and the hash sent are equal then Alice must have sent the message since she is the only one who has access to her private key. On the other hand, if the computed hash value and the hash sent are not equal then the message must have been altered during transmission or forged. The primary difference between PGP and PEM was the system's approach to certification: whereas PEM specified a centralized PKI (Public Key Infrastructure) with a single root, PGP users can both independently certify keys as belonging to other users, and decide to trust certification statements made by other users [8].

2.2 Current Research

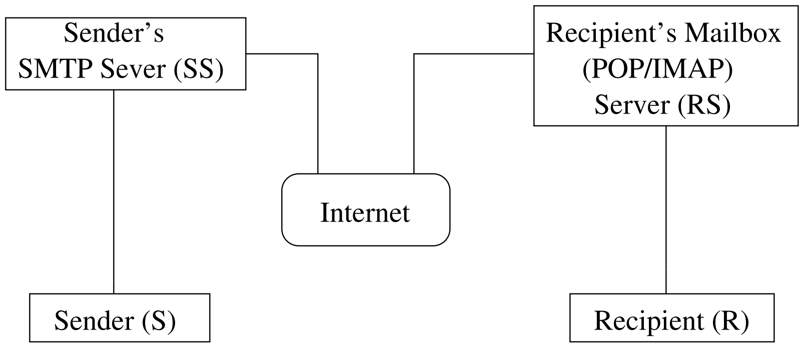


Fig. 2. Email messages are vulnerable to interception at many points

Simon L. Garfinkel [7] from the MIT Laboratory for Computer Science proposed a new approach to email security that employs opportunistic encryption and a security proxy to facilitate the opportunistic exchange of keys and encryption of electronic mail. Garfinkel's approach might offer less security than established email systems that uses certificate authorities. However, the security of his approach is equivalent to today's systems based on PGP, PEM, and S/MIME. Figure 2 shows the path followed by many mail messages on the Internet today. Any message from a given sender S to a recipient R will usually travel through the sender's SMTP (Simple Mail Transfer Protocol) Server SS and then be spooled for final delivery in a mailbox on the recipient's POP (Post Office Protocol) server RS [7]. The use of encryption might not be needed unless there is a risk of:

– Sender spoofing.
– Unauthorized message adulteration.
– Unauthorized message interception.

Garfinkel believes that the reason email encryption is rare, despite the risks, is because of usability issues. If Alice wants to send an encrypted message M to Bob then Alice must go through the following steps:

1. Determine if Bob wants to receive encrypted messages and has the appropriate tools to decrypt them.
2. Determine the email encryption Bob is using.
3. Obtain Bob's public key K_b^{public}.
4. Verify that K_b^{public} belongs to Bob not to some other Internet user with a similar name or email address.
5. Verify that Bob has the corresponding private key $K_b^{private}$.
6. Load the public key K_b^{public} into the email client.
7. Create a public and private key pair $(K_a^{public}, K_a^{private})$ to sign the message if the key pair does not exist already.
8. Compose the email message M.
9. Use $K_a^{private}$ to sign the message to get $C_1 = K_a^{private}(M)$ and encrypt it with K_b^{public} to get $C_2 = K_b^{public}(C_1)$.
10. Send the encrypted and signed message C_2.

When Bob receives the encrypted and signed message C_2 then Bob must decrypt it and verify that the sender is Alice. Bob must go through the following steps:

1. Provide the encrypted and signed message C_2 as input to a suitable decryption program.
2. If the private key $K_b^{private}$ is encrypted then Bob must enter a pass phrase first to decrypt it. The decryption program will use Bob's private key to decrypt C_2 and obtain $C_1 = K_b^{private}(C_1)$.
3. Obtain Alice's public key K_a^{public} and use it to verify the message. Applying Alice's public key to C_1 will give us $M = K_a^{public}(C_1)$.
4. View the decrypted and verified message.

Additional barriers exist that prevent Alice from even systematically signing all outgoing email messages as a matter of course, thanks to the way that programs such Microsoft Outlook and Outlook Express handle signed messages received in PGP or S/MIME format [7]. Outlook, for instance, displays signed messages as a blank message with two attachments. The first attachment is for the signed message and the second attachment is for the signature. Viewing signed message this way can be quite cumbersome and as a result some people who receive messages that are merely signed will ask the sender to stop sending digitally signed messages because they are annoying to the recipient [7].

To overcome the usability problems in the above scenario, Garfinkel designed an email encryption system called *Stream* that uses a zero-click interface. Stream operates as a filter on outgoing email messages through the use of an SMTP proxy, and on incoming email messages through the use of a POP proxy [7]. Stream automatically performs the following actions on each outgoing message M:

1. Determines Alice's email address E_a.
2. If a public and private key pair does not exist for E_a then a key pair $(K_{E_a}^{public}, K_{E_a}^{private})$ is created.
3. Places a copy of the public key $K_{E_a}^{public}$ in M's header.
4. Checks if the recipients R_i where $i = 1, 2, 3, ..., n$ of message M have a public key $K_{R_i}^{public}$. If not then go to step 5, else do the following:
   (a) Takes M's original mail header and encapsulates it within M.
   (b) Adds the key fingerprint for each recipient's encryption key to the encapsulated header.
   (c) Creates a new sanitized mail header for M that contains a single *To:* address and a nondescript *Subject* line.
   (d) Encrypts M for the recipient to obtain $C_i = E_{R_i}^{public}(M)$ and sends the message through the SMTP server.
5. Sends M to R_i.

As you can see in scenario above, Stream provides opportunistic encryption so the email message will be encrypted if it can be encrypted. This behavior mimics the behavior of existing users of encryption: they use it if they can, but if they can't, they send their message anyway [7]. For each incoming message, Stream performs the following actions:

1. Checks if the mail header has a public key $K_{E_a}^{public}$ for Alice. If yes then go to step (a):
   (a) The key is added to the user's public key database.
   (b) If the key is a new key for an existing email address then the user is warned. This is similar to the warning SSH clients generates when an SSH server's public key sent does not exist in the SSH client's public key database.
2. If the message M received is encrypted as C_i then:
   (a) Decrypts $C_i = E_{R_i}^{public}(M)$ using R_i's private key $E_{R_i}^{private}$ to obtain M.
   (b) Un-encapsulates the encapsulated mail headers to obtain the key fingerprint for Alice's public key.
   (c) Verifies that the obtained key fingerprint for Alice's public key matches the one stored in the database.
   (d) If the key fingerprint does not match, a warning is displayed to the recipient.

While most PGP and S/MIME implementations leave messages encrypted except when being viewed, Stream removes cryptographic protection once a message reaches its final destination [7]. This transparent process can make the use of such system more appealing to users. Users can protect the messages they send across the Internet without worrying about generating keys, understanding the use of public and private keys, etc. The only problem with Stream is that it only encrypts the message without signing it. Matching key fingerprint is only effective when digital signature is used. Stream is clearly vulnerable to a man-in-the-middle attack. An Adversary (Mallory) can intercept the messages sent from Alice to Bob; generate her own messages, encrypt them with Bob's public key and send them to Bob as if they were sent from Alice. Since Stream does not use any digital signature schemes, Bob will never know whether a given message is sent from Alice or Mallory.

Ian Brown and C. R. Snow [6] proposed an approach which is similar to the approach proposed by Garfinkel [7]. The authors argue that in some circumstances, the additional functionality can be provided by tapping into the protocol exchanges rather than modifying the applications themselves to represent a more manageable approach to the problem of adding additional facilities to applications [6]. As a result, Brown and Snow created a proxy sitting between a mail client and server that signs and encrypts outgoing mail, and decrypts and verifies incoming mail. This transparent approach can work with standard mail protocols without requiring separate upgrading. The authors called their system *Enigma*, after the World War II German encryption machine. Enigma (shown

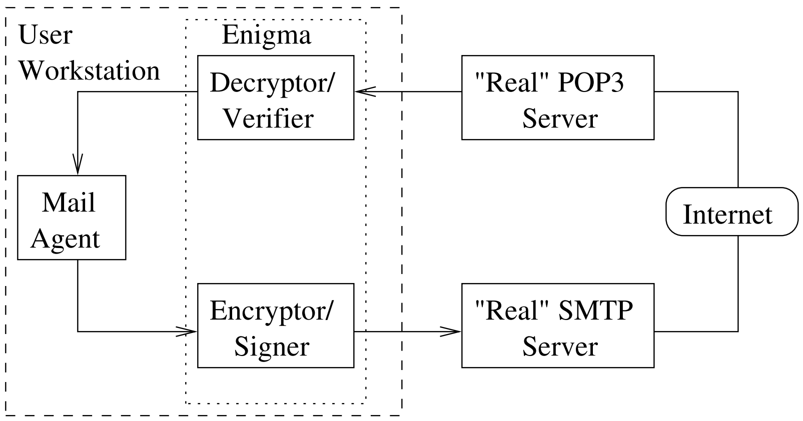


Fig. 3. Enigma system configuration

in figure 3) has two parts: one to act as an SMTP server for outgoing mail, and another to act as a POP3 server for incoming messages [6]. If Alice wants to send a message to Bob and both Alice's and Bob's mail agent is configured to use Engima as a proxy, then the following is done:

1. Alice composes a message using her mail agent.
2. The mail agent connects and forwards the message to the *Encryptor/Signer* part of Enigma which searches for Bob's public key on the local public key database and encrypts the message using Bob's public key. The *Encryptor/Signer* then uses Alice's private key (which must be stored in Engima) to sign the message.

3. Enigma uses SMTP to send the message to the real SMTP server.
4. Bob starts his mail agent, and the mail agent connects to the *Decryptor/Verifier* part of Enigma which connects to the real POP3 server to download Bob's messages. If a message is encrypted and the recipient's private key is available, it will be decrypted [6]. If a message is signed and the signer's public key is available, the signature will be checked [6]. Once decryption and signature verification are done, the *Decryptor/Verifier* forwards the messages to Bob's mail agent.
5. Bob views Alice's message.

3 Approach

After discussing a number of technologies and approaches to email security, we can conclude that the use of cryptographic techniques in Internet mail has proved to be successful only when the process of encrypting/decrypting and signing/verifying email is transparent. Keeping the amount of user interaction to the minimum and providing security functionality for users without having them learn a complex new user interface or algorithm is essential.

We present a new approach to email security that uses fingerprint recognition to authenticate users and provide them with a transparent process of signing and verifying email messages. The idea is to enroll a user fingerprint, associate the fingerprint with a record that is unique to that user, and finally use the user's fingerprint and unique record to authenticate the user, sign the user's email message, and verify other users' email messages. Our approach was implemented as an email client called *SEFR*. A detailed description of our SEFR is described below.

3.1 Components

Environment: SEFR was designed and implemented on a T2400 processor running Windows XP. Visual Studio 2005 was the Integrated Development Environment (IDE) used to write to code, and design the Graphical User Interfaces. The code was entirely written in C and used the Win32 API, OpenSSL and MySQL libraries.

Mail servers and accounts: *Gmail* is a web mail system created by *Google Inc.* We used Gmail's mail servers to send emails and retrieve emails from a user's inbox. Two test accounts were created on Gmail: *amaobied* and *sefr.obied.* To send emails from our email client, we used Gmail's SMTP (Simple Mail Transfer Protocol) server: *smtp.gmail.com* which is listening on port *465.* To download emails to our email client, we used Gmail's POP (Post Office Protocol) server: *pop.gmail.com* which is listening on port *995.* To connect to Gmail's mail servers, you will have to authenticate successfully to it. Furthermore, Gmail's mail servers require email clients to set up a secure tunnel via SSL before attempting to communicate with them. The test accounts that were created allowed us to authenticate successfully to the mail servers. We installed the OpenSSL library on the machine where the email client is running; used it to set up the ciphers, do the SSL handshake, and route all the traffic via a secure tunnel. Gmail's SMTP server required that the username and password of a Gmail account be sent in base 64. We simply used the OpenSSL library to encode the username and password in base 64 before sending them to the SMTP server.

Database: A database was required to store the user's Gmail account information (username and password), the user's fingerprint, and the hash value of emails sent. We used a MySQL server hosted on *dbs2.cpsc.ucalgary.ca*. The database contained two tables: *account* and *email*. The *account* table had the following fields: username (varchar), password (varchar), fingerprint image (large blob), hash of fingerprint image (varchar) and size of fingerprint image (integer). The *email* table had the following fields: email from (varchar), email to (varchar), and email hash (varchar). A MySQL ODBC driver and library have been installed where the email client is running to enable it to communicate with the MySQL server.

Fingerprints: We were not able to use the fingerprint scanner in the Biometric Laboratory (BT lab) since it lacks an API that one can work with to interface with it and acquire the fingerprint images directly via the email client. We overcome this problem by simulating the scanner, so instead of asking the user to present his fingerprint to the scanner, we asked the user to provide the path to the fingerprint image stored on disk.

We acquired 6 fingerprint images using the optical fingerprint scanner in the BT lab. The fingerprint images were used to simulate the availability of a fingerprint scanner.

3.2 Design and Implementation

Enroller: In the registration process, users have to provide the enroller with their Gmail account information (username and password), and the path to their fingerprint image. Once the enroller acquires the information correctly, the enroller stores them in the MySQL database. The Secure Hash Algorithm (SHA-1) is used to hash the password and store the hash value in the database instead of the password itself to enhance security. SHA-1 is also used to compute a hash value of the fingerprint image to be used in the matching process.

Users can use the enroller anytime they want to check if their account information and fingerprint have been already registered or not. In the retrieve process, the enroller takes the username and password, and checks if a fingerprint has already been registered under the given username and password. If yes, then the enroller downloads the fingerprint image and displays it to the user in the enroller's GUI (Graphical User Interface). Figure 6 shows the register feature and figure 7 shows the retrieve feature.

If a user tries to register using an account that already exists then SEFR displays an error message. Also if a user tries to retrieve the fingerprint and either the username or password he provides are invalid then SEFR displays an error message. When a user registers, the path to the fingerprint is checked. If the path is invalid or does not exist then SEFR displays an error message.

Login: Since users tend to forget their passwords or simply use weak passwords that allow an adversary to break into their email accounts, we used a second authentication layer that uses fingerprints. By having 2 layers of authentication (knowledge-based and biometric-based), breaking into an email account becomes very difficult. No one will be able to break into an email account on SEFR unless he has your account username and password, and your fingerprint. Figure 8 shows SEFR's login screen, figure 9 shows an error message when an invalid username and/or password error message are provided, and figure 10 shows that an invalid fingerprint image has been provided.

When a user provides his username and password, SEFR attempts to connect via a secure tunnel to Gmail's POP and SMTP servers using the given username and password. SEFR uses the syntax described in the POP and SMTP RFC (Request For Comment) documents to talk directly with the servers. If any of the servers returns a soft error (e.g., 4xx error) or a hard error (e.g., 5xx error), then SEFR exits immediately. If any of the servers return a message saying that the given username and/or password are invalid, then SEFR aborts the connection and displays an error message to the user.

Inbox: Once the user successfully logs in, SEFR will send commands to Gmail's POP server to download the user's inbox. The commands sent to the server follows the syntax described in the POP RFC document [2]. Figure 11 shows SEFR's inbox GUI.

Sending email messages: When a user, say Alice, who has a Gmail account (*amaobied*) tries to send a message using SEFR, Alice will have to authenticate to SEFR using her fingerprint image which must be stored in the database. The idea of having to authenticate yourself every time you want to send an email message is to provide integrity and authorship of email messages. If everyone on the Internet is using SEFR to send their email messages then we will know for sure that the email messages we receive must have been sent by the person who claims to have sent them. This is due to the fact that forging fingerprints is difficult.

When Alice successfully authenticates to SEFR while trying to send an email message, SEFR strips from the message the following: tabs, new line feeds, carriage returns, spaces, and form feeds. For example a message which has the following form:

Hi,

Welcome to reality.

Becomes:

Hi,Welcometoreality.

After transforming the message, SEFR inserts into the *email* table: the email address of Alice, the email address in the *"To:"* field, and the hash value of the transformed message hashed using SHA-1.

SEFR uses the SMTP commands described in the SMTP RFC document [4] to send a message via Gmail's SMTP server. Figures 12, 13, and 14 show SEFR's send mail features and error checking.

Verifying email message: Suppose Alice has a Gmail account (*amaobied*) and Bob has a Gmail account (*sefr.obied*). If Alice was able to send a message to Bob then Alice must be a legitimate user on SEFR since her fingerprint image is stored in the database, and has been used to invoke the hash function used to sign her message. When Bob successfully authenticates to SEFR, Bob will be able view his inbox. Every email message in the inbox will have a verification status: *Success* or *Failure*. The verification is done automatically by SEFR. To verify a message, SEFR does the following:

1. Downloads the email message from Gmail's POP server using the commands described in the POP RFC document [2].
2. Parses the email message and retrieves the email address in the *"To:"* and *"From:"* fields of the email message.
3. Strips the tabs, spaces, line feeds, form feeds, and carriage returns from the email body.
4. Computes the hash of the stripped message using SHA-1.
5. Checks if the retrieved email address has the hash value associated with it in the database. If yes, then the verification status is set to *Success* else it is set to *Failure*.

3.3 Testing

We tested SEFR's functionality by sending a valid email message and a forged one. We forged an email message by deleting its entry in the database. This means that when SEFR tries to validate the authenticity of the email message, SEFR will not find its entry in the database and hence should display a warning. Figure 15 and 16 shows the valid email and forged email that were used and sent.

4 Benefits

In the previous section, we presented a new approach to email security implemented as an email client called *SEFR* which uses fingerprint recognition and

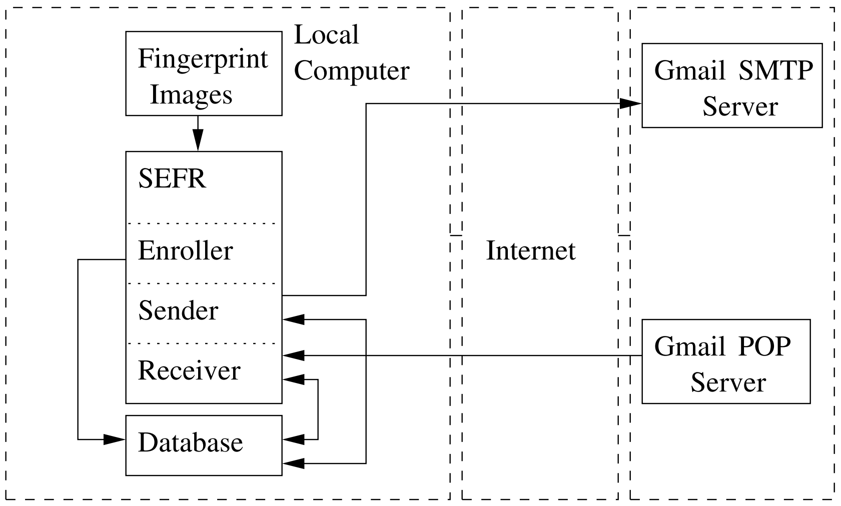


Fig. 4. SEFR's high-level design

cryptographic hash functions. The benefits of such approach are described in the following sections.

4.1 Secure email access

In traditional email systems, you authenticate yourself using a username and a password. If an adversary was able to guess your username and password or steal them, then the adversary will be able to access your email account, view your email messages, and send email messages. SEFR has 2 layers of authentication: knowledge-based (using passwords) and biometric-based (using fingerprints). If an adversary knows your username and password then the adversary will be able to bypass the first authentication layer. However, the adversary will not be able to bypass the second layer since he does not have your fingerprint. Forging fingerprints is not impossible. However, it is known that using biometric-based authentication makes it harder for an adversary to break the authentication system.

4.2 Prevent email spoofing

We demonstrated in figure 1 how emails can be spoofed easily. SEFR does not allow users to send emails unless they successfully authenticate using their fingerprints. Once a user successfully authenticates, a record is added to the database to keep track of who sent what and to whom. Furthermore, SEFR associates that record with the hash value of the sent message. If an adversary tries to send a spoofed email from *cpsc601.20@cpsc.ucalgary.ca* as shown in figure 1, then SEFR will first hash the message body as described before; then SEFR will try to find a record in the database that has *cpsc601.20@cpsc.ucalgary.ca* associated with the computed hash. Since such email address or hash value does not exist then SEFR will display a warning to the user.

4.3 Prevent against man-in-the-middle attacks

To demonstrate how SEFR prevent against man-in-the-middle attacks, consider the following scenario. Suppose that Alice, and Bob are legitimate users in SEFR. If Alice wants to send a signed message to Bob then the following is done:

1. Alice starts SEFR.
2. SEFR asks Alice to provide her fingerprint.
3. Alice provides SEFR with her fingerprint. Since Alice is a legitimate user then SEFR will allow Alice to login and view her inbox.
4. Alice selects the compose message option and starts writing a message to Bob.
5. Alice selects the send message option and SEFR asks Alice to provide her fingerprint.
6. Alice provides SEFR with her fingerprint. Since Alice is again a legitimate user then SEFR will create a hash value of the composed message using SHA-1, stores Alice's email address and hash value in the database, and send the message to Bob's email address via SMTP.
7. Bob starts SEFR.
8. SEFR asks Bob to provide his fingerprint.
9. Bob provides SEFR with his fingerprint. Since Bob is a legitimate user then SEFR will allow Bob to login and view his inbox.
10. SEFR starts downloading Bob's email messages via IMAP.
11. SEFR scans through the email messages and finds Alice's message. Since Alice's message was sent via SEFR then SEFR will create a hash value of the composed message using SHA-1, and check the database to see if the created hash matches the hash values associated with Alice's email address. If there is match then SEFR sets the verification status to *success*. Otherwise, SEFR sets the verification status to *failure*.
12. Bob views Alice's message and checks the verification flag. Since it was signed by Alice then Bob will see the status set to *success*.

What if Mallory was able to intercept SEFR in step 10 above and modifies Alice's message? In step 11 above, when SEFR creates a hash value of the composed message using SHA-1, the created hash value will be different because the message was altered. The property of one-way function is that it is computationally infeasible to find x and y such that $H(x) = H(y)$ so if Mallory modifies one character in the original message then the created hash value will change. When SEFR checks the database to see if the created hash matches the hash values associated with Alice's email address, SEFR will not find it. Therefore, SEFR will set the verification status to *Failure* and Bob will be notified that the message was altered in transit.

5 Future Work

The course in general and this project in particular have inspired us with a number of interesting ideas that one can use to change the way authentication is done these days. Biometric-based authentication has the potential to be the next big thing of the World Wide Web.

We have been involved in developing a couple of open source plug-ins (SSH and POP(s)/IMAP(s)) for a software called pGINA (http://www.pgina.org) that stands for pluggable Graphical Identification and Authentication. PGina replaces Microsoft's GINA DLL (Dynamic Link Library) and loads plug-ins that can use any method of authentication. PGina is used at the University of Calgary, and a number of Universities and organizations around the world. We plan to develop in the future an open source fingerprint plug in to help people learn more about biometric authentication, and inspire them to develop applications based on biometrics.

Another interesting idea which we plan to examine in depth is using biometrics to defeat spam. Spam is unsolicited email sent by a third party which can be offensive, fraudulent (e.g., phishing, scam), and malicious (e.g., carry viruses, worms, spyware, or cause denial of service attacks). Spam consumes computer and network resources, and wastes human time and money. A number of anti-spam approaches have been proposed and developed. However, most of these approaches fail to defeat spam completely or track spammers. To our knowledge, no one has proposed how biometrics can be used to defeat spam. One of the most important things that anti-spam researchers try to do is to know if an email was sent by a real human or an automated software. Some of the proposed methods use Challenge-Response mechanisms. However, such methods are considered annoying since users tend not to like sending a response after a challenge. If one enforces the use of biometrics with emails (as proposed in our approach) then we can know for sure that an email is coming from a human. Hence, be able to identify and classify spam easily.

6 Conclusion

Despite the widespread availability of public key cryptographically protected email, secure messaging is not widely practised. This is due to technical, social, and usability issues. Cryptographically protected email uses traditional authentication schemes that are vulnerable to many attacks. The security of the email system can be broken if an adversary is able to crack the password that protects the private key of a user, or modify the public key of a user.

In this report, we presented a new approach to email security that allows users to secure access to their email accounts, and messages. Furthermore, the presented approach allows users to sign and verify email messages effectively and without the use of public key cryptographic techniques. The presented approach uses fingerprints which provide a better and stronger factor of authentication. Authenticating users based on what they are instead of what they know or what they have provides users with a more effective, convenient, and secure way to access their email account and messages. We also presented a method that uses cryptographic hash functions which is not computationally expensive to compute to provide email users with an effective way to sign and verify email messages.

The presented approach does not only have the potential to protect email accounts and messages but it also have the potential to protect users against spam. Automated spam software will cease to work if SEFR is used since live fingerprints have to be presented before emails are sent.

References

1. RFC 1521, MIMIE, http://www.faqs.org/rfcs/rfc1521.html.
2. RFC 1939, Post Office Protocol, Version 3, http://www.faqs.org/rfcs/rfc1939.html.
3. RFC 1991, PGP Message Exchange Formats, http://www.faqs.org/rfcs/rfc1991.html.
4. RFC 2821, Simple Mail Transfer Protocol, http://www.ietf.org/rfc/rfc2821.txt.
5. H. Berghel. Email–the good, the bad, and the ugly. *Communications of the ACM*, 40(4):11 – 15, 1997.
6. I. Brown and C. R. Snow. A proxy approach to e-mail security. *Software Practice and Experience*, 29(12):1049 – 1060, 1999.
7. S. L. Garfinkel. Enabling email confidentiality through the use of opportunistic encryption. In *Proc. of the 2003 annual national conference on Digital government research*, pages 1 – 4, 2003.
8. S. L. Garfinkel, D. Margrave, J. I. Schiller, E. Nordlander, and R. C. Miller. How to make secure email easier to use. In *Proc. of the SIGCHI conference on Human factors in computing systems*, pages 701 – 710, 2005.
9. P. Gutmann. Why isn't the internet secure yet, dammit. In *Proc. of the AusCERT Asia Pacific Information Technology Security Conference 2004; Computer Security: Are we there yet?*, pages 71 – 79, 2004.
10. A. K. Jain, A. Ross, and S. Prabhakar. An introduction to biometric recognition. In *Proc. of IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Image- and Video-Based Biometrics*, volume 14, pages 4 – 20, 2004.
11. D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar. *Handbook of Fingerprint Recogition*. Springer, 2003.
12. I. Ploeg. Written on the body: biometrics and identity. *ACM SIGCAS Computers and Society*, 29(1):37 – 44, 1999.
13. A. Suglura and Y. Koseki. A user interface using fingerprint recognition: Holding commands and data objects on fingers. In *Proc. of the 11th annual ACM symposium on User Interface software and technology*, pages 71 – 79, 1998.
14. A. Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *Proc. of the 8th USENIX Security Symposium*, 1999.
15. J. A. Zdziarski. *Ending Spam*. No Starch Press, 2005.