# SECURING EMAIL THROUGH ONLINE SOCIAL NETWORKS

Atieh Saberi Pirouz

A thesis

in

The Department

of

Concordia Institute for Information Systems Engineering (CIISE)

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science (Information Systems Security) at

Concordia University

Montréal, Québec, Canada

August 2013

# Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Atieh Saberi Pirouz**

Entitled: **Securing Email Through Online Social Networks**

and submitted in partial fulfillment of the requirements for the degree of

**Master of Applied Science (Information Systems Security)**

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

| | |
|---|---|
| Dr. Benjamin C. M. Fung | Chair |
| Dr. Lingyu Wang | Examiner |
| Dr. Zhenhua Zhu | Examiner |
| Dr. Mohammad Mannan | Supervisor |

Approved _____

Chair of Department or Graduate Program Director

_____ 20 _____ _____

Dr. Christopher Trueman, Dean

Faculty of Engineering and Computer Science

# Abstract

Securing Email Through Online Social Networks

Atieh Saberi Pirouz

Despite being one of the most basic and popular Internet applications, email still largely lacks user-to-user cryptographic protections. From a research perspective, designing privacy preserving techniques for email services is complicated by the requirement of balancing security and ease-of-use needs of everyday users. For example, users cannot be expected to manage long-term keys (e.g., PGP key-pair), or understand crypto primitives.

To enable intuitive email protections for a large number of users, we design FriendlyMail by leveraging existing pre-authenticated relationships between a sender and receiver on an Online Social Networking (OSN) site, so that users can send secure emails without requiring direct key exchange with the receiver in advance. FriendlyMail can provide integrity, authentication and confidentiality guarantees for user-selected messages among OSN friends. FriendlyMail is mainly based on splitting the trust without introducing new trusted third parties. A confidentiality-protected email is encrypted by a randomly-generated key and sent through email service providers, while the key and hash of the encrypted content are privately shared with the receiver via the OSN site as a second secure channel. Our implementation consists of a Firefox addon and a Facebook application, and can secure the web-based Gmail service using Facebook as the OSN site. However, the design can be implemented for preferred email/OSN services as long as the email and OSN providers are non-colluding parties. FriendlyMail is a client-end solution and does not require changes to email or OSN servers.

# Acknowledgments

I wish to thank everyone who helped me to complete this thesis. I would like to express my sincere gratitude to my supervisor Dr. Mohammad Mannan. I will be forever grateful to him for giving me the opportunity to work under his guidance and improve my academic knowledge. The completion of this thesis would have not been possible without his endless support, effort, encouragement and most importantly his enduring patience.

It was a great honor for me to have the best friends and colleagues, who made the research lab a cooperative, warm and friendly environment. I would like to thank Adam Skillen, Suryadipta Majumdar, A. Mert Kara, Lianying Zhao (Viau), Vladimir Rabotka and Xavier de Carné de Carnavalet, who made themselves available by testing my tool, having great discussions and providing support in many ways. Thank you all! I would specially thank Ehsan Khosrowshahi Asl. No words can express how grateful I am for all his kind support.

Last but not least, my deepest thanks go to my beloved parents and brothers for their unconditional love, never-ending support and always encouraging me to pursue my studies.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In this chapter, we discuss our motivation for this thesis and summarize our contributions.

## 1.1 Motivation

Billions of emails are sent everyday, with almost all of them being stored/available in plaintext to one or multiple third parties, e.g., email service providers, ISPs, and wifi hotspot providers. Imagine the outrage that would have erupted if a large paper-mail provider such as the U.S. Postal Service would have opened and kept a scanned copy of every mail/document they processed. Yet, today's email users are apparently finding it acceptable that a few email providers have complete access to their most intimate messages. (The CEOs of Sun Microsystems, Google and Facebook remarked at various times that "privacy is dead"). This is a fantastic development that happened within the span of only a few decades. Apparently, century-old privacy expectations of personal communication have just evaporated into thin air.

We believe the current situation is the result of several factors, including the following. **(a)** Most people are unaware that their emails are not private at all. The email infrastructure (e.g., ISPs, email providers) are more or less transparent to average users. When sending an email, or any message for that matter, users have

the illusion that they are sending the email directly to the recipient. Some experts identify this situation as web service providers (e.g., Facebook) being a transparent man-in-the-middle (see e.g., [46]). **(b)** A wide-spread, common misconception among users is "I've got nothing to hide" [78]. Most users apparently believe their emails or other messages are not very sensitive or interesting to service providers; i.e., users may take precautions not to disclose an email to their family and friends, but they do not believe that large corporations like Google or Facebook would be interested to dig in their personal lives. So, even if some users understand that their emails are accessible to service providers, they do not feel the necessity to explore privacy-friendly alternatives. **(c)** The inadequacy of existing email security solutions.

Without the availability of effective solutions, issues in (a) and (b) cannot possibly be addressed, e.g., just asking people not to send anything sensitive via email is a non-solution. PGP is one of the pioneer solutions enabling adequate security features for emails (e.g., confidentiality, authentication, integrity). Unfortunately, even though PGP has been available for over two decades now, most emails are still sent unencrypted. Other proposals also emerged, e.g., STEED [48], Waterhouse [50] and Aquinas [10]. Most solutions require a certain level of technical understanding for proper use (e.g., the idea of public key systems); later work has identified several shortcomings when these tools are used by everyday people (e.g., [87, 76, 82]). The end result, so far, is that the adoption of these techniques remains consistently low.

## 1.2 Thesis Statement

The primary objective of this thesis is to address security and privacy challenges of email services as faced by everyday Internet users. As part of this goal, I will explore the following research questions:

***Question 1.*** Can we design novel privacy-preserving architectures and techniques for popular email services given that users will continue using these free services?

***Question 2.*** Can we integrate usable security/privacy mechanisms with the functioning of the existing (privacy-unfriendly) systems without introducing a new service?

This dissertation explores new directions in privacy research to enable usable privacy friendly emails by leveraging existing services, and strengthen existing research on online message privacy in general. Another important focus of this research is to increase awareness and understanding of email privacy issues among everyday users.

## 1.3 Summary of Our Proposal

We propose *FriendlyMail*, a secure email technique designed for everyday users. To achieve security goals, FriendlyMail leverages widely popular, frequently-used online social networking (OSN) services, e.g., Facebook. We assume senders and receivers are connected through OSN sites (e.g., as Facebook friends). A sender-side addon creates a per-message symmetric key to encrypt the email content; a cryptographic hash of the encrypted message is also generated. When confidentiality is unwanted and instead, the goals are to authenticate the sender via a second (secure) channel and to verify the integrity of the email content, a hash of the plaintext message is created. The hash/key values are then published on the OSN site, which are instantly accessible *only* to the receiver, e.g., on the sender's Facebook wall, or as a private Twitter message, etc. An addon in the recipient's email client (stand-alone or browser addon) is configured to access the hash and message key; the addon verifies the hash and decrypts the email content (if encrypted). The receiver is assured of the email's integrity from the result of the hash verification; the sender's authenticity is verified implicitly by the OSN site, as the hash/key values are accessible only to the receiver through the pre-existing social relationship. Confidentiality is maintained by the per-message encryption key.

Obviously, FriendlyMail requires the email and OSN sites to be non-colluding entities. We use existing OSN sites as a key transport method to simplify the key

sharing and verification process, which has been identified as an important barrier to PGP's adoption [87]. Our hope is that FriendlyMail's design leads to better adoption rates among regular users than existing public-key based solutions. FriendlyMail is designed and implemented considering web-based Gmail and Facebook services; however, it can be extended to other email clients/services, and be used with other OSN services. Note that there are about 425 million Gmail users [68] (as of June 2012) and 1.11 billion monthly active Facebook users [22] (as of March 2013).

## 1.4 Contributions

Following the above discussion, the main contributions of this dissertation towards securing emails are as follows.

1. FriendlyMail takes advantage of existing user practices (i.e., the use of OSN) to make popular email services more privacy-friendly. We expect this design choice to increase adoption rates, as it can reach a significant portion of web users.

2. No changes to the server-side of the social networking sites or email providers are needed. Therefore, users can immediately benefit from FriendlyMail.

3. To facilitate gradual adoption, FriendlyMail allows a sender to indicate which emails should have integrity and confidentiality protections. Default encryption of all emails would be a more privacy-preserving design choice; however, such a design may not work for many users (due to e.g., not using OSN).

4. Unlike most other solutions, FriendlyMail does not require the receiver of a confidential email to create keying materials (as in public key systems) *before* she can receive such emails. The only pre-requisite for the sender to initiate a confidential email exchange is to be friends with the receiver in a common OSN.

5. In addition to email privacy, FriendlyMail can provide email integrity and origin authentication between parties with *weak* pre-existing OSN relationships (e.g., Facebook Like). When the sender and receiver have no previous contact, integrity is perhaps more important than confidentiality. Integrity protection can be enabled without any OSN relationships. Past work generated some theoretical proposal in this area, e.g., public corroboration [84]. FriendlyMail extends such work so that it can be deployed in reality and used by common email users.

6. To evaluate the feasibility of our design, we have implemented FriendlyMail for the web-based Gmail interface using Facebook as the OSN provider. Basic email features between two users, e.g., message compose, send, receive, reply, and forward have been implemented.

7. FriendlyMail has carefully been integrated with Gmail UI to reduce disruption to regular email use. Some features are still in progress, including multi-party/group email, attachments, etc.

## 1.5 Outline

The rest of this thesis is organized as follows. Chapter 2 reviews related work regarding existing email security standards and proposals. Chapter 3 provides our threat model and assumptions. We propose FriendlyMail email security solution and detail its design in Chapter 4. Implementation details of FriendlyMail and some variants to FriendlyMail are also presented in Chapter 5. In Chapter 6, attacks on FriendlyMail based on our presented threat model are analyzed. We also discuss several usability and deployment issues of FriendlyMail. Chapter 7 discusses some conclusions of this thesis and future research directions.

# Chapter 2

# Background and Related Work

In this chapter we briefly discuss few email security mechanisms. Then we go through several related proposals.

Security for emails using cryptography can be provided in various ways. Signature schemes are used for authentication and integrity verification. Encryption mechanisms such as symmetric, public key encryption are generally used to achieve email confidentiality. Other proposals incorporate steganography to avoid secret sharing and the existence of confidential emails. Considering a network adversary, confidentiality protection of the message body and header can also be provided by encrypting the client-to-mail server and mail server-to-mail server connections, e.g., using TLS.

Numerous proposals for email integrity and confidentiality have been put forward. Some designs are completely client-side. Thus, users are in the control of their email encryption. Installation of additional software or plug-in is also required. Moreover, interoperability is a known issue for client side end-to-end approaches. Other designs employ third party servers, to take the burden of key management away from users (e.g., Enlocked [20]). Therefore, they cannot offer a complete end-to-end encryption, as they have access to the encryption keys and email contents. Encrypted communications between different email providers also do not always guarantee end-to-end confidentiality. TLS for the same service is safe from network attacker, but not

against the email provider itself. TLS protected emails may still be unsafe from network attacker, if either of the sender's or receiver's email service is not TLS-protected. Moreover, there is no way to make sure that every hop on the email's route to the receiver's inbox, has also TLS enabled.

Several email security standards have been introduced. Privacy Enhanced Mail (PEM [53]) and MIME Object Security Services (MOSS [15]) are introduced in 1987 and 1995 respectively, which were never widely deployed or implemented. Currently, PGP/OpenPGP (RFC 4880 [1]) and S/MIME (RFC 2634 [72]) are the most widely used standards, that can provide end-to-end email security. In the following sections we discuss some standards and related proposals, but exclude enterprise solutions as they are unsuitable for zero-cost mass deployment. The evaluation and comparison of discussed schemes are presented in details in Section A.2

## 2.1 PKI-Based Approaches

### 2.1.1 PGP/OpenPGP

OpenPGP(RFC 4880 [1]) is a standard based on PGP, originally developed by Phillip R. Zimmermann in 1991. To provide non-reputation and message authenticity (origin and content integrity verification), it employs hash functions (e.g., MD5, SHA-1) and signature algorithms to digitally sign email bodies and data files. Confidentiality is also provided through the use of a combination of symmetric (e.g., IDEA, CAST) and public key cryptography (e.g., RSA, Diffie-Hellman, Elgamal). Each PGP user generates a key pair (public/private) and must share his/her public key with other PGP users before starting PGP confidential communications. There is no actual key distribution mechanism in PGP; public keys can be exchanged using, e.g., email, personal web pages, meet in person. Each user should also verify the identity and public key (fingerprint) of other users using, e.g., phone or meet in person. Key distribution and verification using above mentioned methods are not appropriate for large communities with unknown users. Thus, public keys can also be shared through

a key server. However, key distribution via key servers still does not guarantee that a key belongs to the intended person, as anyone can create and publish a key on these key servers using any identifier. To deal with this issue, PGP introduced the Web of Trust model, which is discussed below.

The process of encrypting an email using PGP is as follows. Email content is first compressed to save transmission time. It is then encrypted using a one-time-only random session key (symmetric key). The session key is then encrypted using the sender's public key and is sent along with the encrypted email to the sender. The receiver decrypts the session key, using his own private key, to be able to decrypt the received email. To provide message authenticity, the hash of the email content is calculated and digitally signed using the sender's private key and is accompanied with the email content. Using the sender's public key, the receiver verifies the sender's signature on the message digest. If both confidentiality and message authenticity are desired, the signature is first generated, after applying compression on the signature and email content, the result is encrypted as the last step. Different PGP versions use different algorithms during each process (e.g., MD5, SHA-1, IDEA, CAST, RSA, Diffie-Hellman).

**PGP trust model.** Compared to previous email security standards, PGP trust model does not require a centralized PKI with a single root or any trusted third parties as certificate authorities to sign the certificates and enable trust among users. Instead, each user can act as a CA and certify other users' public keys and identities. Therefore, instead of one signature, there can be multiple signatures on PGP certificates. The signer of a certificate can also indicate his/her level of trust on the owner of the certificate to express his/her trust in a user as a voucher for other users' certificate authenticity. This form of decentralized trust model in which users recommend each others' certificates is called Web of Trust. On the plus side, Web of Trust is similar to how trust is established in the real life. Therefore, it may seem closer to the user's mental model compared to other trust models. Although, it still has few downsides. This trust model does not completely match the user's mental model, as

users may certify other users, whom they may not know directly beforehand; and as the circle of communications grows, there is no actual accountability of trustworthiness on users' certificate, as their owners are not in a close circle. Moreover, trust is based on others' recommendations than the user experience with certificates during communications over time [48].

**PGP shortcomings.** In addition to its trust model issues, PGP has several limitations. Key management is considered as a big challenge in PGP and PKI-based solutions in general. Public key cryptography requires the sender to obtain the receiver's public key beforehand, to be able to start any PGP encrypted communication (also it should be done in a secure way to prevent man-in-the-middle attacks). Moreover, there is still no practical secure approach to private key management; users should create a backup of their private key, store it in a safe place and be careful not to loose it, otherwise old encrypted emails cannot be decrypted anymore. Additionally, in case the private key is compromised, the attacker can trivially decrypt all the (old or new) encrypted emails. Therefore, a certificate revocation list (CRL) is required to facilitate the revocation of all compromised keys which also must be shared with all users. Key revocation prevents the attacker from having access to the encrypted emails in the future. However, key revocation brings the same usability problem regarding key distribution and also requires consistent updates.

Other usability problems of PGP software (PGP 5) regarding the user interface have been discussed in detail by Whitten and Tygar [87]. The result of their user study and cognitive walkthrough on PGP 5 also suggests that the desired security level of the PGP software has not achieved due to usability problems. Some users could not figure out the concept of PKI and did not know which key to use to encrypt their email. Those who had knowledge about PKI, still had difficulty to find out how they can get a key for the receiver. Also, one user accidentally sent a sensitive email in the clear, while she thought it had already been encrypted. In another usability study on PGP 9 [75], key verification is considered confusing; users could not figure out that in order to verify a key they need to sign it. Unlike PGP 5, users had so much

difficulty with digital signatures in PGP 9 due to the absence of a cue in the interface, representing signing emails. Transparent encryption is also found problematic when there is no feedback to the user if an email will be encrypted or not. Additionally, when emails are automatically decrypted spoofing attacks may not be noticeable to all users; and users should compare and verify the key presented in the email, with the one existed in PGP.

Unlike S/MIME, major email clients including Microsoft Outlook, IBM Lotus Notes, and Mozilla Thunderbird do not directly support PGP. However, PGP can be enabled by adding an additional add-on or plug-in to the mail client. Compared to other email security standards, PGP has been adopted or improved by some email security tools and other proposals. The GNU Privacy Guard (GnuPG) is a free implementation of OpenPGP. GPG and OpenPGP have been incorporated by several tools. For example, Enigmail [56] is an extension to the Mozilla Thunderbird and Seamonkey. It provides encryption and signing using OpenPGP. To secure emails, one should install OpenPGP in addition to the Enigmail extension. To eliminate the need for installing PGP on user's machine, OpenPGP has been implemented for browsers in JavaScript, so that it can be used in other applications and plug-ins as an open-source library [65]. GMail-crypt [73] and GPG4Browsers [33] are also chrome extensions to the Gmail interface and enable email security using OpenPGP JavaScript library. Other proposals based on PGP are discussed in detail in the following sections.

### 2.1.2 PGP-based Proposals

**Waterhouse.** Waterhouse [50] is proposed to secure existing email clients using PGP. It uses OSN sites such as Facebook to distribute long-term public-keys, by posting them on OSN profiles and to leverage existing OSN connections between users. However, the burden of managing private keys remains on end users; which is still an open problem that negatively impacts PKI-based approaches in general.

For intuitive identity verification, Waterhouse suggests displaying a sender's OSN

photo with each email (cf. Facemail [52]). As the sender's photo alone cannot guarantee trustworthiness of the OSN profile owner (due to the existence of fake account impersonating OSNs' users), the use of a Web of Trust (WoT) model is also suggested; only public keys of senders with at least $n$ friends in common with the receiver are accepted.

All outgoing messages are signed even in the absence of any connection on the OSN. However, to have a confidential communication, senders and receivers must be friends in the targeted OSN site. Their current prototype has been integrated into an open-source, web-based email client. No implementation of this proposal is available (as far as we are aware of).

**Stream.** Stream [36] is a POP and SMTP proxy that sits between the email client and server. To remove all user involvement and elevate the usability, Stream offers no user interface; all emails are encrypted using PGP at the SMTP proxy, when the receiver's public key is available (opportunistic encryption), otherwise they are sent in the clear. Stream [37] also provides users with an option, by which if the subject line contains *mandatory encryption character*, and the recipients' public keys could not be found, the email is sent back to the sender, explaining that the email could not be encrypted for those recipients.

At the sender side, Stream grabs the message right after leaving the sender's inbox and before reaching to the SMTP mail server. It then uses sender's email address as an index to locate the associated key pair in the key database; If no key pairs are found, public/private keys are automatically generated and stored on the fly by the proxy. Stream offers opportunistic key distribution by signing each recipient's public key and adding it to each email's header. At the receiver side, the POP proxy verifies the sender's public key, decrypts and delivers the email. If a new public key is detected for an existing email address, the receiver is notified through an email. The public key is then added to the user's database.

Stream eases the burden of key management but requires users to trust its proxy

servers with private keys that can decrypt any email at any time without users' knowledge; such a trust model is particularly unsuitable for webmail providers. Moreover, opportunistic encryption used in Stream may disclose sensitive email content, in the case when the receiver's public key cannot be found and the email is sent unencrypted, without any feedback to the user. Stream does not come up with any solution addressing man-in-the-middle attacks, as public keys are sent along with the encrypted messages. If a public key is replaced by an attacker and all of the outgoing messages are modified by the attacker, the attack could not be noticeable to the receiver.

**STEED.** STEED [48] employs a set of existing techniques to address usability problems of PGP and S/MIME, including non-user friendly trust models. To reduce the user involvement, it proposes significant changes to email providers and Mail User Agents (MUAs). MUAs would automatically generate public/private keys (or self-signed certificates) each time a new email account is created, and perform opportunistic encryption using GPG. Key/certificate distribution is done through the DNS server of an email provider. Compared to key servers, DNS is decentralized, available and its structure also matches email addresses that can be used to bind the user's identity with a certificate. Incorporating DNS also brings the benefit of the improved security using DNSSEC. This feature provides a secure channel for key/certificate distribution which prevents man-in-the-middle attacks.

Users are still responsible for managing their private keys. They believe there should be a Personal Information Manager (PIM), protecting all the user's sensitive information (e.g., phone numbers, address books, mail account, etc.) in addition to passphrases. Although in the absence of a promising solution, a temporary solution is needed.

STEED's proposed trust model is based on "trust upon first contact" and "persistency of pseudonym" (TUFC/POP). In this model, keys/certificates are accepted in the first contact, and then verified during further communications. Users can verify suspicious certificates at the first contact, through an out-of-band channel, e.g., phones or meet in person. After trust is established, the system helps users to track

any suspicious changes made into trusted certificates.

### 2.1.3 S/MIME

The process toward securing emails in S/MIME (RFC 2634 [72]) is similar to the one used in OpenPGP and they both use digital certificates for key management. S/MIME users obtain a X.509 digital certificate including the user's identity, public key, expiration date and, etc. As S/MIME also uses asymmetric encryption, it has many of PGP limitations. In addition to the PKI-related issues in PGP and S/MIME (e.g., key management), obtaining certificates from well-known CAs with actual identity verification is costly and time consuming. It also requires additional effort and submission of personal information to CAs, which some users may be reluctant to do [76, 51]. Self-signed certificates may seem user-friendly, however they bring their own security risks and usability issues [38].

S/MIME is built into most popular desktop email clients, e.g., Microsoft's Outlook and Mozilla Thunderbird, but is not supported by web-based email clients, e.g., Gmail and Hotmail. However, usability issues of S/MIME specifically regarding its trust model, make it unusable in practice. CoPilot [76] is proposed based on Key continuity management (KCM [42]), to address usability issues of identity certification and automate key management in S/MIME. Instead of concerning about the certificate authenticity of each secure incoming email, the sender's S/MIME certificate is automatically accepted for the first time (as key certification used in SSH) and a new digital ID is generated to be associated with the email address. Copilot then keeps track of further emails sent from the same email address and notifies users about any changes made to the digital ID, associated with each email address, through the use of different background colors and presented information. For example, if a secure email is received from a new address for the first time, it is flagged as yellow. Further digitally signed emails sent from the same address are shown within a green background; CoPilot also counts the number of emails that has been sent with the same email address and digital ID. If an incoming email from the same address has a

different certificate (digital ID), it is shown within a red background and, etc. CoPilot does not guarantee any trustworthiness; it is on users to decide whether to trust a new or changed digital ID or not. Despite all efforts toward having simpler trust model for S/MIME, complexity of public key cryptography concept, and some user interface related usability problems of email clients supporting S/MIME (discussed in a recent work [35]), are still barriers to S/MIME's adoption. Since S/MIME is not broadly used due to the above mentioned problems, we do not discuss further S/MIME related proposals.

### 2.1.4 IBE-based Proposals

Identity-based encryption (IBE) was initially introduced by Adi Shamir [74] in 1984 to reduce the issues of public key management. In 2001, Boneh and Franklin [7] proposed a practical IBE solution with an actual application for email encryption. IBE is based on public key cryptography. However, unlike PGP and S/MIME, public/private keys are not randomly generated by users themselves. Public keys are derived from the arbitrary, unique and publicly available user identifiers, e.g., email address. No certificate or public key exchange is required. Moreover, email communication through IBE is not limited to those who have their own public key in advance; senders can send encrypted emails to the receivers who does not have a public key. A trusted third party key server, called Private Key Generator (PKG), generates its own master public/private keys and publishes its own corresponding master public key. The sender encrypts emails using the receiver's unique information and the PKG's master public key. On the receiver side, the receiver retrieves his own private key from PKG, after authenticating himself to PKG and decrypts the email. Voltage Security [85], Trend micro [83] and FortiMail [34], all offer enterprise email security solutions through IBE. Therefore, we do not discuss them further.

One problem arises from using unique identity information as the public key, in which if the corresponding private key is compromised, there is no substitution to the unique identity information to be able to easily revoke the public key. Therefore,

a time period is also added to the public key. Public keys are then automatically expired and revoked after a specific duration of time. Moreover, one concern about IBE is the ability of PKG to access original content of encrypted emails, as it is in possession of all users' private keys. In contrast to PGP and S/MIME, private keys are generated on demand by PKG and are shared with the user through a secure channel. Therefore, in addition to the security of PKG servers, trust in PKG servers is also of great importance. In the following, we discuss a proposal which addresses the aforementioned problems.

**Lightweight Encryption for Email.** In their previous work, Adida et al. [2] propose using each email domain as a PKG to provide email authentication using IBE. Each email domain distributes its own MPKs (Master Public Keys) through the Domain Name Server (DNS). Each user's secret key is directly delivered to their inbox using email-based authentication. In another proposal [3], Adida et al. extend their approach to provide email encryption using IBE, based on their previous proposals of key distribution through DNS servers.

Their proposal exposes users to two potential privacy issues. First, storing the MSK (Master Secret Key) on a single DNS server, leaves the server vulnerable to malicious attacks by external attackers. By compromising a DNS maintaining the MSK, the attacker can trivially generate all users' private keys from the MSK. To mitigate the problem, Adida et al. perform the key generation distribution among several DNS servers instead of one server. Thus, in order to compromise users' private keys, the attacker should compromise all PKGs to combine all MSK's shares.

Second, an email provider can easily gather all the shared MSKs from all DNS servers, combine them together to generate the final MSKs and obtain all users' private keys. Therefore, Adida et al. further introduce two countermeasures to this problem. Their main approach is to incorporate two separate channels for key sharing: email providers' DNS as and a user preferred channel. Therefore, they put users in charge of generation and distribution of their own key pairs by the means of any online medium, such as personal web pages; and users can be in possession of their

own key pairs which cannot be accessible by email providers. However, this approach suffers from usability issues regarding publishing user generated key shares. It also introduces the same key loss problem as in PGP and S/MIME. Alternatively, two different email providers can be used to send an encrypted email. The way it works is to first encrypt emails using the recipient's public key, published on the first email DNS server, then encrypt the generated ciphertext using the recipient's public key stored on the second email domain. The latter solution, prevents each domain from decrypting the email, as they are not in possession of both private keys at the same time.

## 2.2 Symmetric Encryption and Trust Split Based Proposals

**Aquinas.** Aquinas [10] employs symmetric encryption with per-email keys, and thus avoids several key management issues. An implementation of Aquinas as an open-source Java applet enables confidentiality through AES encryption, deniability using a steganography technique (SNOW) to hide confidential email communications, and message integrity using MAC. Keys and encrypted messages are split, and transmitted separately through competitor email providers; so that a compromised account would not reveal the secret message to the attacker. At the receiver side, all shares are combined to regenerate the original message and keys.

A malicious ISP may collect all key/message shares and retrieve the message, when user to email channels are not SSL-protected. SSL protected channels prevent from revealing the content of transmitted messages, yet cannot hide the source and destination of the packets. Therefore the ISP learns all the involved email providers and colludes with them. To restrict this attack, several solutions have been proposed, including the use of proxies or different email accounts for sending different emails. As another solution, additional email accounts are added to communicate bogus message shares; e.g., 20 out of 40 shares may be used for the actual message. The ISP now

sees all 40 shares, but does not know which ones would construct the real message. The sender and receiver must communicate an initial secret that will be used to determine the shares for the real message; this secret should be established through an out-of-band mechanism (e.g., phone).

A publicly available directory is also suggested to maintain all users' email accounts. This publicly available directory would not reveal any information to the attacker, as the subset of these accounts is still unknown, yet a public directory of users' email accounts endangers users' privacy to some extent. Another concern could be the need of user authentication when a new email account is added to the list, otherwise an adversary can impersonate a user by supplying fake email accounts, and trick other users to send their key and message shares to the fake account.

**TrustSplit.** TrustSplit [30] proposes the confidentiality as a service (CaaS) paradigm, and splits the trust between a cloud provider (e.g., Gmail, Dropbox) and CaaS provider(s) to secure users' data before sharing through the cloud, and manage keys transparently. To benefit from CaaS, users should first register for the service and create a new CaaS account. Users are required to provide an email address and a password, which should be different than the cloud service's password to enable CaaS to authenticate users and bind their newly created CaaS accounts to their existing cloud service ones. To protect user data from CaaS provider, multiple layers of commutative encryption are used, called cLayers, which can be added/removed from user data in an arbitrary order.

The hash value of Alice's data is first computed. Alice's data is then encrypted (AES in counter mode) with a local layer (+cLayerLocalPre), by XORing ($\oplus$) the original data with a key stream of the same size and sent to the CaaS provider. If Alice is authenticated successfully and all the recipients are CaaS users, CaaS provider applies an additional layer of encryption (+cLayerRemote), and returns the result to Alice. The local layer is removed (-cLayerLocalPre) by the client and data encrypted with the CaaS remote cLayer is shared via the cloud provider with the intended recipients.

When the encrypted data is received, Bob adds his local cLayer (+cLocalLayerPost), and then sends the data and the list of recipients to the CaaS provider. CaaS provider removes the remote cLayer (-cLayerRemote) and returns the data to Bob. Now, Bob's local cLayer is removed to reveal the plaintext (-cLayerLocalPost). TrustSplit requires third parties to run CaaS servers; users also must register with these services. The service has been implemented to secure Dropbox, Facebook and Mozilla Thunderbird. The general approach is the same for all, but slightly different in case of Dropbox and email attachments. In these cases, the above mentioned process is done on the symmetric key, used for encrypting the whole data or email attachments instead of the whole data or attachment itself. Their current implementation for email security is an extension to Thunderbird, written using Greasemonkey [41] script to provide encryption and decryption on client side.

**SPEmail.** In another attempt of eliminating usability issues of public key cryptography used in email security, SPEmail [66] uses secret sharing and linguistic steganography to provide confidentiality for webmails. Therefore no key setup, exchange or management is needed. Each message is divided into two shares. After encoding secret shares by applying a form of text steganography based on Huffman coding of Markov graph, secret shares are delivered via two different webmail providers. These providers should be located in countries with different legislations to prevent collusion. SPEmail does not provide sender authentication or message integrity. In addition, no approach for searching among secret shares has been suggested. SPEmail, has currently been implemented as a Greasemonkey script in a Firefox extension, securing Gmail web client. Although the tool is out-dated and cannot be tested.

# Chapter 3

# Threat Model and Assumptions

In this chapter, we first explain FriendlyMail's defined levels of trust relationships on OSNs. We also discuss FriendlyMail's assumptions and threat model, categorized into *in-scope* and *out-of-scope* threats.

## 3.1 Trust Relationships and Protections

FriendlyMail can provide different levels of cryptographic guarantees such as message authentication, integrity and confidentiality depending on user choice and the (existing) trust relationships between the sender and receiver on OSNs. We assume the following trust relationships:

(a) a *direct* OSN connection between a sender and receiver (e.g., Facebook friends), where both parties know each other to some extent (e.g., real-life relations, online-only acquaintances).

(b) *indirect* OSN personal connections (e.g., Facebook friends-of-friends), where the sender and receiver are related via one or more direct acquaintances.

(c) *impersonal* OSN connections with web presences of known physical/online entities, e.g., users connected to a Facebook page (e.g., of a bank, organization, entertainer) possibly through the *Like* feature.

(d) *unconnected*.

Table 1 summarizes FriendlyMail protections for these relationships. Different (existing) OSN trust relationships have different levels of authentication strength. FriendlyMail mainly provides email content confidentiality based on (existing) strong authentication in friends' circle. However, confidentiality protection can be extended to the lower levels of authentication strength (e.g., among friends-of-friends; see Section 4.4 under "User authentication"). Note that, there is a trade-off between providing email content confidentiality based on indirect connections beyond the friends' circle, and increasing the risk of spam or malicious emails. This is due to the existence of fake accounts and the lack of a proper and effective supervision on the users' actual identity on the OSN (see Section 6.1 item 6). However, email content integrity does not require prior direct connections on the OSN between sender and receiver. We also assume trust relationships e.g., OSN connections (or lack thereof) can be determined based on available information in email clients. For example, the receiver's email address or full-name can be searched in the sender's OSN friends' list to verify if they have a direct connection. This verification may require the receiver to be registered with the same email address or full-name for the OSN account.

| Trust relationships | Protections provided | | |
|---|---|---|---|
| | Origin authentication | Integrity | Confidentiality |
| Direct | strong | ✓ | ✓ |
| Indirect | weak | ✓ | ✓ |
| Impersonal | weak | ✓ | |
| Unconnected | | ✓ | |

**Table 1:** FriendlyMail protections for different OSN trust relationships. An empty box indicates the stated protection is not provided.

## 3.2 In-scope Threats

We assume the adversary is capable of performing the following attacks.

1. We assume a Dolev-Yao [17, 60] network adversary. Network connections between users and OSN/email servers are also assumed to be protected (e.g., via SSL).

2. As a sender, the adversary may impersonate a friend, i.e., the adversary is aware of the user's social contacts, or a known company (e.g., the user's bank); i.e., the FROM field can be arbitrary, and we do not assume any other sender verification techniques being used (e.g., Sender Policy Framework (SPF) [55]/DomainKeys Identified Mail (DKIM) [16]).

3. The adversary may compromise the credential of a FriendlyMail user's email account. At the sender side, she may try to abuse FriendlyMail to send malicious emails, in the form of trusted ones, through the compromised email account. She also has access to the email content residing on the inbox of the compromised email account.

4. Email providers are non-malicious but possibly curious, motivated (e.g., financially) or forced by law enforcement to have access to or reveal users' email content.

## 3.3 Out-of-scope Threats

In the following, we briefly go through out-of-scope threats and several assumptions required by FriendlyMail to function properly.

1. With regards to the email content, we assume that the email and OSN providers are non-malicious but curious entities; i.e., they will provide their services in the usual manner, but would prefer to learn the email content (e.g., for advertisements, building elaborate user profiles). Besides, the email and OSN

providers must be separate, non-colluding entities, ideally residing in different legal jurisdictions. Note that, for cloud storage/application services, defining legal boundaries may be tricky; see e.g., Hoboken et al. [45], for how U.S. laws (Patriot Act/FISA) can be used to access user data in EU countries. Either of the service providers may cooperate with an adversary, but not both.

2. To prevent perpetual access to an email content, users can delete keys from the OSN site after an encrypted message has been retrieved, or after a given time period. However, OSN sites may not actually delete any posts for a long period of time; see, e.g., Facebook policy on deleted content [28]. Thus, an encrypted email is not guaranteed to remain confidential forever, assuming currently non-cooperating email and OSN providers may collude at some point in the future. Hence, message *self-destruction* is a non-goal (which is rather difficult to achieve, cf. [88]).

3. We use the OSN provider for sender origin authentication, and assume that OSN profiles and connections between users are largely genuine. For example, Facebook actively attempts to control (prevent or detect) fake profiles by enforcing a policy [26] (regarding, e.g., multiple account creation and providing false personal information); or keeping track of its users actions and behaviours [70] to detect fake accounts. However, such profiles are still a significant concern (see e.g., [6, 9]). Detecting fake accounts is also an active research area (e.g., [11]). Therefore, users are always advised to take precautions to distinguish between fake accounts and genuine ones, when it comes to accepting a friend request on OSNs [71]. We assume that users are aware of this threat and act cautiously when adding friends.

4. FriendlyMail requires that OSNs enforce proper access control mechanisms to maintain their site integrity and take several actions toward making the OSN a secure environment. Access control mechanisms are designed to only allow authorized users to add or delete content to their OSN account. Several security

features are also introduced to detect and decrease account hijacking (e.g., two factor authentication, remote log out, HTTPS support, social reporting in case of Facebook) [25, 69]. Additionally, in our threat model, OSN providers are not malicious in a sense that they do not add or delete any content on behalf of users. Social networking service provider's malicious behaviours have been discussed in [32] and these attacks regarding OSN provider's malicious behaviours are out of FriendlyMail scope. These two assumptions are critical for considering an OSN as a second (secure) channel for sharing keys/hash values and as a means of asserting that the user is authenticated.

5. We require that the OSN providers can protect confidentiality and integrity of user posts, e.g., not to expose privately posted keys/hashes to unauthorized parties (but see [18]).

6. Both the sender and receiver-end machines are assumed to be malware-free; otherwise, malware can simply expose or modify the email content when being composed/read.

7. The user's email and OSN account credentials must not be compromised. By compromising the user's OSN account, the adversary is in the control of the content, stored on the OSN and can add, remove or change them. Also, if she has access to both keys and encrypted emails it is trivial for her to get the original content of the emails. We discuss consequences of such compromises in Section 6.1.

8. The recipient of a confidential email is also trusted not to share the content with unauthorized parties. We exclude such threats in FriendlyMail.

# Chapter 4

# FriendlyMail Design

In this chapter, we describe the overview of FriendlyMail. We present the different modes of FriendlyMail operation and detail FriendlyMail and user steps for sending/receiving confidential and integrity-protected emails. Parts of these steps are explained through our prototype for Gmail and Facebook (detailed in Chapter 5). We also elaborate in detail the reasons of certain choices we made.

## 4.1 Design Overview

In contrast to PKI-based approaches, FriendlyMail opts for a simple solution, using symmetric encryption and integration with OSNs, to provide email confidentiality, origin and content integrity verification. The design is related to the well-established notion of using multiple channels for security (e.g., [89, 54]). FriendlyMail employs OSN sites, as an additional channel, mainly to automate key management and integrity verification process; while secure emails are communicated over email providers as the main channel. It leverages existing pre-authenticated connections among users on the OSN, to address the challenge of exchanging secrets between email senders and receivers. OSNs also serve as secure channels for sharing hash values of email content to provide message integrity, where the hash value is stored on a known, integrity-protected location. In its basic form, FriendlyMail secures email through

| | |
|---|---|
| $ID_A, ID_B$ | Unique user identifier for Alice (sender) and Bob (receiver) respectively. |
| $K_m$ | Per-email, randomly generated symmetric key of adequate length (e.g., 128 bits). |
| $H(\cdot)$ | A cryptographically-secure hash function (e.g., SHA-256). |
| $E_{K_m}(\cdot)$ | An authenticated, symmetric-key based encryption function (e.g., AES in the CCM mode) with key $K_m$. |
| $C_m$ | Content of an email message as compiled by Alice (email body only, excluding email headers). |
| $C_{fm}$ | Content of an email message after being processed by FriendlyMail. |
| $C_{hm}$ | Selected parameters from the email header (e.g., receiver's address, email subject). |
| $N_m$ | Per-email, randomly generated nonce. |
| $x\|\|y$ | $x$ concatenated with $y$. |
| $Mrk_s$, $Mrk_e$ | Marks the start and end of an encrypted email, respectively. |
| $Ftr$ | Footer appended by FriendlyMail; includes URL to FriendlyMail addon. |

**Table 2:** Notation used in FriendlyMail

the use of an OSN and email service provider as two separate channels. To get the original content of an encrypted email, the adversary should have access or intercept both communication channels. For proper functioning of FriendlyMail, we need both channels/services to be available. However, both parties (email and OSN providers) are untrusted with the plaintext email content, and the content will remain secure assuming these parties do not collaborate. Moreover, FriendlyMail is designed as a client-side solution. Therefore, its adoption requires that email applications are extendable (e.g., via browser plugin, or application addon). We would like to avoid introducing a new client for usability/deployment reasons.

The main idea is to store keys and/or hash values of encrypted/integrity-protected emails on the OSN accounts of FriendlyMail senders in email communications. Both

keys and hash values are then made immediately available to the intended receivers. The retrieved keys and hash values can later be used to decrypt encrypted emails and verify their origin and content, or just to validate emails (their origin and content) when just email integrity is desired. Figure 2 provides a brief overview of FriendlyMail.

For confidentiality and integrity purposes, FriendlyMail requires the OSN to present the following characteristics:

(a) The OSN should form connections among users, e.g., friendship connections or following, in which users can authenticate each other's identity before getting connected.

(b) The OSN should authenticate users and provide them with an integrity protected channel, e.g., user's wall in Facebook or direct, private message in Twitter. Therefore, OSNs can be used as a second (secure) channel to provide automatic key authenticity and integrity verification. This authentication is required to prevent from unauthorized access or modification to user's OSN account.

(c) The OSN should enable the sharing of a message between multiple users through a proper access control mechanism, e.g., to maintain the confidentiality of keys posted as a message or wall posts shared between intended users through private messaging or privacy settings.

(d) The OSN should be highly available to provide FriendlyMail with free and scalable servers for storing keys and hashes and should not limit users to have access to their old data residing on the OSN servers.

(e) The OSN should be pervasive, so that it can serve vast majority of senders and receivers of FriendlyMail secure emails.

Any existing OSNs with the above mentioned characteristics could be used as the second channel, and thus, making the FriendlyMail design flexible for greater adoption.

## 4.2 Modes of Operation

FriendlyMail has two primary modes of operation: *Encryption* and *Origin and content verification*. The FriendlyMail primary modes assume a direct trust relationship and are described in Sections 4.2.1, 4.2.2. In Section 5.4, we consider other trust relationships, and also discuss several variants that may address some limitations of the primary mode. Both modes of operations are done on user demand; see Section 4.3. FriendlyMail requires explicit user selection for confidentiality-protected messages, *before* beginning the composition of such a message. Unlike a non-confidential email (e.g., integrity-protected only), this selection cannot be done through a checkbox; see under "Protecting emails during compose and read" in Section 5.2. Therefore, FriendlyMail adds an additional button inside the email client's interface; see, e.g., *Secret COMPOSE* in Figure 1. FriendlyMail also notifies the user about system status through several visual cues along with security statements.

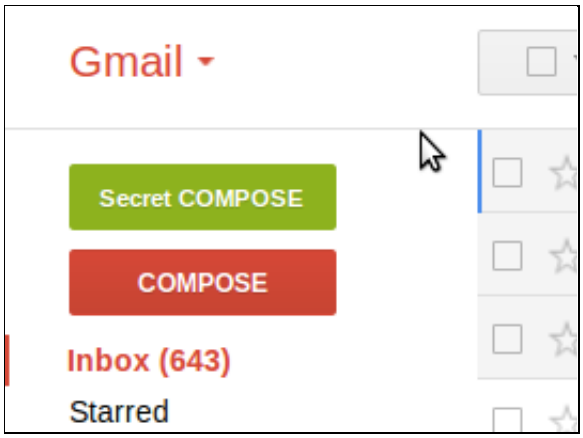



**Figure 1:** Secret compose button

Consider a scenario in which Alice is the sender and Bob is the receiver. Alice installs FriendlyMail (in case of first time usage) and runs her email client. In the following sections, we detail FriendlyMail and user steps for sending/receiving confidential and integrity-protected emails; see Table 2 for notation used. We describe the steps necessary for Alice to send a protected (confidential and/or integrity-protected) email to Bob. Users are also assumed to be logged into their OSN account (for OSN

related steps), otherwise they are asked to login by FriendlyMail. No registration or additional setup is required by the user. Moreover, FriendlyMail does not require that Bob has its plug-in already installed on his machine, so that Alice can send encrypted emails to Bob using FriendlyMail.

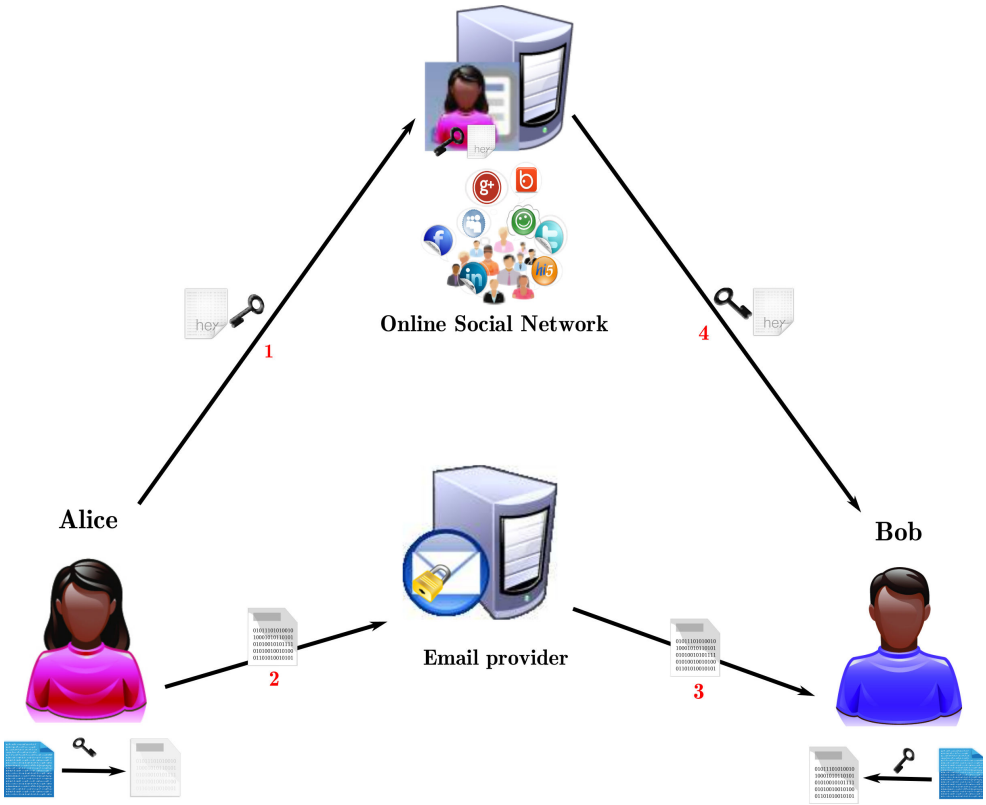



**Figure 2:** Simplified FriendlyMail steps: (1) a per-message randomly generated key and the calculated hash value of the encrypted email are shared with the recipient via an OSN site; (2) the encrypted email message is sent via the regular email provider; (3) the recipient receives the encrypted email; and (4) the email content is decrypted by the per-message key retrieved from the OSN and the integrity of its content is verified by comparing the retrieved hash value from the OSN and the locally calculated one.

### 4.2.1 Confidential Emails

Using the above mentioned scenario, FriendlyMail and users perform the following steps in the encryption mode.

**Sending an encrypted email.**

1. Alice initiates composing a confidential email by clicking on the added "Secret COMPOSE" button in the main page. The familiar compose window is displayed with some visual cues (e.g., a green border), indicating that a secure email is being composed.

2. Alice will be able to send an encrypted email to Bob, only if Bob is among her OSN friends. FriendlyMail searches for $ID_B$ among Alice's OSN friends' list to check if they are friends; see Section 5.2 under "Local storage" for more details on how FriendlyMail matches users' email and OSN accounts. Otherwise, Alice has no choice but to discard the email. FriendlyMail also notifies Alice that Bob is not her friend on the OSN, thus no encrypted email can be sent to Bob. During the message composition, FriendlyMail will block the email client's post/update events that are commonly used to auto-save email drafts. Blocking these events is critical; otherwise, the plaintext content is exposed to the email server. However, to support auto-save, local storage may be used.

3. When Alice indicates that she has finished the message composition (e.g., by hitting the *Secret Send* button), FriendlyMail generates a nonce $N_m$ and a random symmetric key $K_m$; both parameters are specific to the current email. More details on key generation are discussed in Section 4.4 under "Automatic key management".

4. The plaintext email with the appended nonce, $N_m$ , is then encrypted using $K_m$. $N_m$ is used to make each email unique, even if the message content ($C_m$) is the same for different emails. This also ensures uniqueness of the shared hash value $H(C_{fm}||C_{hm})$, which is used as an index during message retrieval on the receiver's side; see Section 4.4 under "Email integrity protection" for more details on $N_m$.

5. After adding the markers and footer, the message body, $C_{fm}$, is set to

$C_{fm} = Mrk_s||E_{K_m}(C_m||N_m)||Mrk_e||Ftr$

For more information about the footer, start and end markers see Section 5 under "FriendlyMail message format". We do not encrypt the email subject line (*Subject*), similar to PGP, see e.g., Symantec PGP Desktop.[1] Users may decide on opening an email based on its *Subject*, or later search their emails using keywords from *Subject*. However, we would like to integrity-protect *Subject* and few other header items; we use $C_{hm} = Subject||ID_B$.

6. FriendlyMail computes the hash value of the message body and selected header parameters, $H(C_{fm}||C_{hm})$;

7. The key $K_m$ and hash value $H(C_{fm}||C_{hm})$ are securely published to Alice's OSN account to be instantly accessible *only* to Bob (e.g., as a post on her Facebook wall or a tweet on her Twitter account); see step 1 in Figure 2.

8. The processed email content ($C_{fm}$ with all header parameters) is sent to Bob via the email service provider; see step 2 in Figure 2. The user-to-OSN/email channels must be protected by other means (e.g., via SSL). The user-to-OSN channels must be protected for obvious reasons (i.e., to protect key and hash values). If user-to-email service channels are not encrypted, the OSN site may break email confidentiality if it can collect the encrypted content.

**Decrypting a received email.**

1. Bob first receives the encrypted version of the email (i.e., $C_{fm}$); see step 3 in Figure 2. If the FriendlyMail addon is not installed, Bob can choose to install it from the link provided in the footer; see Figure 4.

2. FriendlyMail installed on Bob's system detects the encrypted email by finding the start and end markers, ($Mrk_s$ and $Mrk_e$), and automatically attempts to verify the email and retrieve the encryption key to decrypt the email content.

[1] `http://www.symantec.com/docs/HOWTO41924`

To avoid privacy breaches in public places due to automatic decryption, the addon can be configured to ask for confirmation from Bob before initiating the decryption process; see Figure 10.

3. For verification, if Alice is identified as Bob's friend, the addon computes the hash $H(C_{fm}||C_{hm})$ of the received email.

4. It then searches for the calculated $H(C_{fm}||C_{hm})$ on Alice's account. If a matching hash value is found on Alice's OSN account, the message is decrypted by $K_m$ as posted along with the hash value (step 4 in Figure. 2). A matching hash value verifies Alice's identity and the integrity of the received email.

5. The verification result is communicated to the receiver through multiple visual cues: Alice's name and picture are fetched from her OSN profile and presented inside the mail client's interface, along with a link to Alice's OSN profile. If a matching hash is not found, the email's origin and content cannot be verified, and Bob is notified through the email client user interface.

**Replying to an encrypted email.**

When replying to an encrypted email from Alice, Bob must encrypt his message. To be on the safe side, we assume that Alice prefers a confidential response to her encrypted email. To avoid replying to an encrypted email, containing sensitive information, accidentally in the clear by Bob the same steps are then followed as for composing and sending an encrypted email.

**Forwarding an encrypted email.**

FriendlyMail disables forwarding a *decrypted* email to other recipients. When Bob attempts to forward such an email, the original portion sent by Alice remains encrypted under Alice's key used for encrypting the email ($K_m$). Content inserted by Bob into the forwarded email can be encrypted or not, depending on Bob's preference. However, Bob may share Alice's key with other recipients or simply decrypt the encrypted email and resend it in plaintext, or encrypt it using his own key. We

exclude these scenarios and assume that Bob is non-malicious; see Section 3.

### 4.2.2 Integrity-protected Emails

Below we elaborate the FriendlyMail and user steps to make the alteration to the origin and/or content of a sent email detectable to the receiver. When Alice believes that she is not sending any confidential information to Bob, she may decide to inform Bob about any alteration made to her email.

**Sending an email with verifiable origin and content integrity.**

1. Alice enables origin and content verification, e.g., through a checkbox; see Figure 6 and 7.

2. When Alice indicates that she has completed the message composition, as in the encryption mode, FriendlyMail searches for Bob in Alice's friend list on the OSN. If Bob is found, it continues to the next step, otherwise it notifies Alice through visual cues and security statements that Bob is not her friend on the OSN and no integrity verifiable email can be sent to Bob. Therefore the email should be discarded.

3. FriendlyMail grabs the message body, generates a nonce $N_m$, and sets the message body to $C_{fm} = C_m||N_m||Ftr$. Note that, unlike for confidential messages, we do not interfere with the auto-save option here. In this case, we are simply interested to provide integrity protection for the email content.

4. For integrity protection of header parameters, we use $C_{hm} = Subject||ID_B$; then the message hash is calculated as $H(C_{fm}||C_{hm})$. Bob's address is included in the hash calculation to prevent replay attacks as discussed in Section 6.1, item 5.

5. The hash value is shared on Alice's retrieved OSN account (e.g., Facebook wall), accessible only to Bob (e.g., in case of Facebook, through the use of privacy settings).

6. The email content ($C_{fm}$) remains unencrypted and is sent to Bob through regular email services.

7. FriendlyMail communicates to Alice about the success/failure of the operations through visual cues integrated in the user interface and appropriate security statements.

**Verifying an email's origin and content integrity.**

1. When Bob opens Alice's email, a footer indicates that it has been sent through FriendlyMail, and allows Bob to install the FriendlyMail addon (if not already installed).

2. FriendlyMail grabs the message body and calculates the hash of the body content and the selected headers, $H(C_{fm}||C_{hm})$ .

3. It then searches Alice's OSN account for: $H(C_{fm}||C_{hm})$ (in case Alice is already found as Bob's friend on the OSN); if found, the message content is verified. The message origin (i.e., Alice's identity) is also verified as Alice is identified as Bob's OSN friend. If Alice is not found as Bob's friend, FriendlyMail notifies Bob that Alice is not his friend on the OSN, therefore the origin is not verified. Finally if the matching hash cannot be found on Alice's OSN account, the content integrity of the email is not verified. The verification result is presented to Bob through the email UI and through appropriate security statements.

**Replying to/forwarding a regular or verifiable email.**

A regular or FriendlyMail integrity-protected email can be replied or forwarded, in either confidential or integrity-protected mode, based on the user's preference.

## 4.3 Transparency and User Consent

There is a trade-off between security and usability. Users do not generally have an accurate mental model of security concepts and threats. Educating users is not also

very reliable and generally does not result in a desired level of security. [43]. Users also may not be interested in fairly complex security configurations that are put in place to prevent attackers. Moreover, users' interaction with security systems often degrades the security of the system and therefore, user intervention with security related tasks should be avoided [14]. In order to maintain the expected security level of a system, instead of relying on educating users, developers should design their systems with minimal user involvements especially when it comes to make critical security decisions [43, 14]. Therefore, in FriendlyMail we avoid unnecessary user involvements by making most operations automatic by FriendlyMail.

On the other hand, user consent in performing critical actions like encrypting sensitive emails should be taken into account. Email encryption by default might not be always possible especially for client-based email. Compatibility issues of different email encryption approaches as used by different users, prevent from having transparent encryption for all emails. Encryption by default would require a universal standard. Additionally, there is a trade-off between security and convenience in automatic secure systems. Although automation prevents users from accidentally sending sensitive content in the clear leading to sensitive information's disclosure, sending all the emails encrypted, may be annoying to the user. Encrypting all mundane emails can impose usability burden on the receiver [39]. Moreover, providing users with selective email protection, helps them to gradually accustom to, learn and be more conscious of securing their emails. Therefore, we believe users should have the ability to decide which email to encrypt.

Although security tasks should be performed as transparently as possible, still users should be aware of the success or failure of their desired tasks [76, 39]. FriendlyMail notifies users about system states using visual aids along with security statements. Displaying visual cues may not need direct attention of the user and can be easily comprehended (see e.g., Gutmann [43]). Therefore, they are more effective, faster and user-friendly than just security statements. Displaying information and photo of the sender to the receiver may also make the comprehension of security

statements easier [52, 50]. Moreover, displaying OSN personal information and photo of the sender of an email, may be counted as an intuitive proof to the receiver that the user identity is actually verified. We believe that the integration of senders' OSN personal information could be beneficial, specially in cases where weaker authentication is acceptable (e.g., indirect connections). We also believe that using different colors to make secure composition windows stand out, helps users to be more vigilant and avoid accidentally composing sensitive emails in unprotected windows; e.g., users gradually understand that if a green border does not show up in a composition window, their email content would not be protected. Therefore, we employ different border colors conveying the state of the system, and integrate the sender's OSN information with email message viewer in the case of successful email encryption and verification.

## 4.4 Integration with OSNs

The idea of leveraging OSNs stems from the fact that it eliminates the need of further user authentication by FriendlyMail and it simplifies key management. Most email privacy-enhanced tools, lack a simple and intuitive approach for user authentication and key management; for example, there is still no user-friendly approach for binding user identities and certificates in PGP and S/MIME. As discussed in Section 2.1.1, PGP trust model is still rather complex. On the other hand, symmetric key approaches also require users to authenticate each other and share their key in a secure manner. Symmetric key service-based approaches that automate key management (e.g., [29]), are limited to the registered users and cannot easily serve a large population of users. The scalability of such a service in terms of number of connections to the server per encryption/decryption and key storage could also be challenging.

We believe our approach is more feasible as it does not introduce any deployment complexity and future maintenance; OSNs bring the advantage of free, highly available and scalable servers for storing keys and/or hash values; see Section 6.2.1 item

(3). Moreover we believe it has more acceptance among users, as OSNs are already accepted and trusted by a large group of people. According to the global web analytics company, comScore [13, 12], OSNs are getting more widespread among Internet users. Finally as our approach does not require any special characteristic of a specific OSN, several existing OSNs may meet FriendlyMail design requirements and can be used as a second (secure) channel.

**User authentication.** Generally two forms of authentication exist on OSNs. The first authentication is done between the OSN service provider and the user. The OSN service provider checks if the registering user is "human" and not using multiple persona. OSNs also enforce access control mechanisms on the user's account to protect its integrity.

The second authentication is performed in the circle of friends on OSNs. A user receiving a friend request, may verify the identity of the requester by checking available information on his/her OSN account, e.g., profile information, email address, common friends, then confirms his/her identity upon accepting the friendship request. Moreover, there is more chance that users, establishing a friendship connection, have a pre-knowledge about each other or some history together that can help identity verification. This identity verification can also be continuously checked after accepting the friendship request, based on further visible personal information and user activity on the OSN over time.

Additionally, as trust in the OSNs is transitive, the circle of trust and authenticated connections can be extended to the friend-of-friends in a lower trust level. A user may recommend one or more of his/her friends to other friends. Also, receiving a friend request from a person with common friends, is more likely to be accepted due to the trust in friends' judgements.

Based on the above discussion, we define three authentication levels in OSNs. The strongest authentication level is defined among OSNs friends. A weaker authentication exists among friends-of-friends. While the weakest one is among all OSNs users with no direct connection at all.

**Automatic key management.** Secure exchange of secret keys, has always been a deployment barrier for symmetric encryption schemes. Usability issues associated with key distribution in such schemes also force the use of long-term keys. Long-term keys raise a security concern; if the shared secret key is compromised, the original content of all the previously encrypted emails would be revealed.

To address the above mentioned issues, our approach instead, generates a 128-bit random key per message; then automatically distributes the key through an OSN to make it instantly available to the specified receiver. Thus, we avoid the use of long-term keys while taking away the burden of exchanging secret keys from users. Therefore, if a key shared on the OSN is compromised, it only discloses the original content of its corresponding encrypted email while other emails would still remain confidential.

However, in the case where both the OSN and email accounts are compromised, the original content of all previously encrypted emails are exposed to the attackers. One way to mitigate this risk is to delete the keys from the OSN as soon as the sender makes sure the receiver has retrieved the key or after a certain period of time. However, such deletion cannot always guarantee that the key and its corresponding encrypted email are safe, as the adversary can always attack before the key is deleted from the OSN. Also, it may take a long time for keys to be completely deleted from OSNs' servers; see Section 3.3 item 2.

**Email integrity protection.** There are times that users may not need message confidentiality. For the purpose of gradual adoption and flexibility, our approach provides users with selective capabilities: encryption, message authentication and content integrity verification. We use SHA-256 as a hash function to provide message modification and detection. Alternatively, Message Authentication Code (MAC) could also provide integrity and authenticity for emails using a shared secret key. We do not need MAC, since benefiting from the OSNs, we already have a second (secure) channel (the user's OSN account) to distribute the hash values of emails. Moreover, unlike MAC which is sent along with the email content and thus is visible to users,

the process of origin and content integrity verification of our approach is completely transparent to users.

We believe when senders and receivers have no prior contact or direct connection as friends in the OSNs, integrity of their email content is more important than their confidentiality. While hash values can be published through other means, such as a corporate website, FriendlyMail offers some advantages. If the sender publishes the hash values of her emails on a personal/corporate website, the receiver must know the URL beforehand, must trust the website's integrity and the verification process may require careful user-involvement (cf. [84]). In contrast, FriendlyMail automates the verification process, and relies on the existing trust relationship (albeit weak) as established through the Like feature.

To have unique hashes for emails with similar content, a random nonce is added to the content of each email before computing the hash. There are several reasons to have unique hash values. First, if the same email is sent multiple times by the same user, each of them should be verified individually. As we only search among hashes, published during a certain period of time; and OSNs may remove redundant similar posts on the users' wall, similar hash values would not be made visible on the wall and the process of verification may fail. Moreover, as these hash values are also used as the index of encrypted emails to retrieve their corresponding decryption keys, unique hashes (unique indices) are necessary to locate the corresponding key (in cases where customized keys are used and the same key is used for encrypting similar emails). Finally, in the case when hash values are publicly available, nonce can be used as a counter measure to an attack, in which the attacker can compute the hash of several messages and compare them to the one posted on the sender's wall. If a matching hash is found, the attacker could get the content of the communicated emails. In order to prevent this attack, the length of the nonce should be of adequate length (e.g., 128 bits).

# Chapter 5

# Implementation and Variants

To validate the viability of our design, we implement a prototype, called FriendlyMail. Transparent user authentication, key management, encryption/decryption and integrity verification are handled at the client side. It also provides a seamless integration with the email user interface to preserve the existing user experience. However, due to the absence of a secure email standard, an inevitable challenge to client side approaches is to implement the approach for each particular email client. In the following sections we first elaborate our certain implementation choices and challenges. We then discuss our implementation in detail; and finally we outline some variants to FriendlyMail.

## 5.1 Implementation Choices and Challenges

To satisfy our design and implementation goals and requirements, we choose our implementation components accordingly. Since a significant part of the implementation requires changes to the email client user interface and HTML manipulation, FriendlyMail is implemented as a browser extension. Installation of browser extensions is fairly easy. Moreover, a browser addon works on top of any operating system, providing availability and portability to a significant portion of nowadays email users. Although FriendlyMail has been currently implemented for webmails, our design can

be implemented on regular email clients as well (e.g., desktop and mobile clients). To benefit a large number of email users, the prototype is built for the web-based Gmail application [44], using Facebook as the OSN service [44]. For OSN support, we use Facebook APIs, which are relatively stable and easy to use, but sometimes limited in terms of features that would have simplified our implementation; see Section 5.3. Our implementation for the desktop environment includes a Firefox addon and a Facebook app. Our implementation is not restricted to a specific browser and porting FriendlyMail to other browsers should be straight-forward.

The prototype highlights challenges of implementing a rather simple design on top of existing email/OSN services. These challenges also show why real-world implementation is non-trivial, compared to a stand-alone, proof-of-concept implementation. A stand-alone prototype with a specific email client and a custom OSN (e.g., managed by a service run by us) could have reduced our efforts. However, we believe that there is little to no chance of such proof-of- concept implementations being used in practice. Our implementation is complicated by the intricacies of Gmail's client-side implementation, which was subject to few substantial changes during our development and testing over the last 12 months. One major change was due to the introduction of a new user interface for composing an email which forced us to add support for the new UI. No official working JavaScript Gmail API is available, specially for modifying the Gmail user interface. A few years ago Google team provided a Greasemonkey script for Gmail called Gmail-greasemonkey [67], which is currently broken due to the major changes made to Gmail UI and is no longer maintained. There are few other unofficial or user created APIs, which are complex or fragile, due to their dependency on the Gmail client application. Therefore, we prefer to implement Gmail UI and related functionality, rather than relying on unofficial APIs which may not be maintained on time or at all.

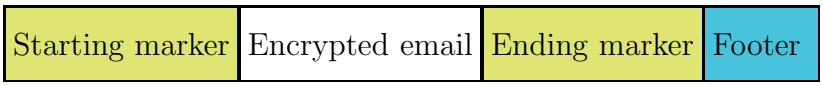
## 5.2 Firefox Addon

FriendlyMail is developed as an extension to Firefox browser in JavaScript and XUL (XML User Interface Language) [58]. Its main task is to modify the Gmail user interface, providing the interaction between the user and FriendlyMail. It employs Firefox XPCOM (Cross-platform Component Object Model) interfaces [57] to perform some browser related functionality provided by Firefox browser. For cryptographic support (e.g., encryption, hash calculation, random key generation), we use the Stanford JavaScript Crypto Library (SJCL [81]). Crypto support through JavaScript APIs is still a work-in-progress; see the current working draft of W3C web crypto API at: www.w3.org/TR/WebCryptoAPI/. For authenticated encryption, we use AES-128 in the CCM mode [19], and for hash calculation, we use SHA-256. In addition to integrity verification purposes, these hash values are used as index of keys/hashes stored on Facebook. To convert the binary output of encryption, we use the Base64 encoding function. The output from SHA-256 is also formatted into hex before being posted to OSN accounts.

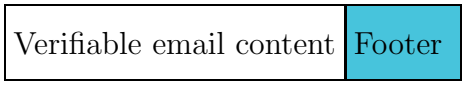
As discussed in Chapter 4, FriendlyMail provides users with two main security features: a) Users may decide to protect their sensitive email content while ensuring the receiver about the originality of the email. b) The content of the email is not of great importance or sensitive, hence only enabling integrity verification would suffice. In the following, we discuss the FriendlyMail message format and changes we made to the Gmail UI. Later we explain the details of implementation of the two mentioned security modes, followed with the challenges we faced during the course of deployment.

**FriendlyMail message format.** Our prototype currently supports emails containing both plaintext and HTML. To make a secure FriendlyMail email (confidential/integrity-protected) distinguishable from other emails, a footer is appended to all emails processed by FriendlyMail. It also makes the receiver aware of the use of FriendlyMail and provides a download link to the FriendlyMail addon. We embed the download link into every FriendlyMail email, in case when the receiver

receives a secure email from a FriendlyMail user for the first time, or the receiver was already a FriendlyMail user but at some point has removed FriendlyMail. In addition to the footer, the start and end markers are appended to the beginning and the end of the ciphertext of an encrypted email and allow the addon to detect encrypted content, and process it accordingly. Figure 9 illustrates the confidential and integrity-protected email format respectively. Figure 4 is the screenshot of a confidential email.




**(a)** Confidential email




**(b)** Integrity-protected email

**Figure 3:** FriendlyMail message format

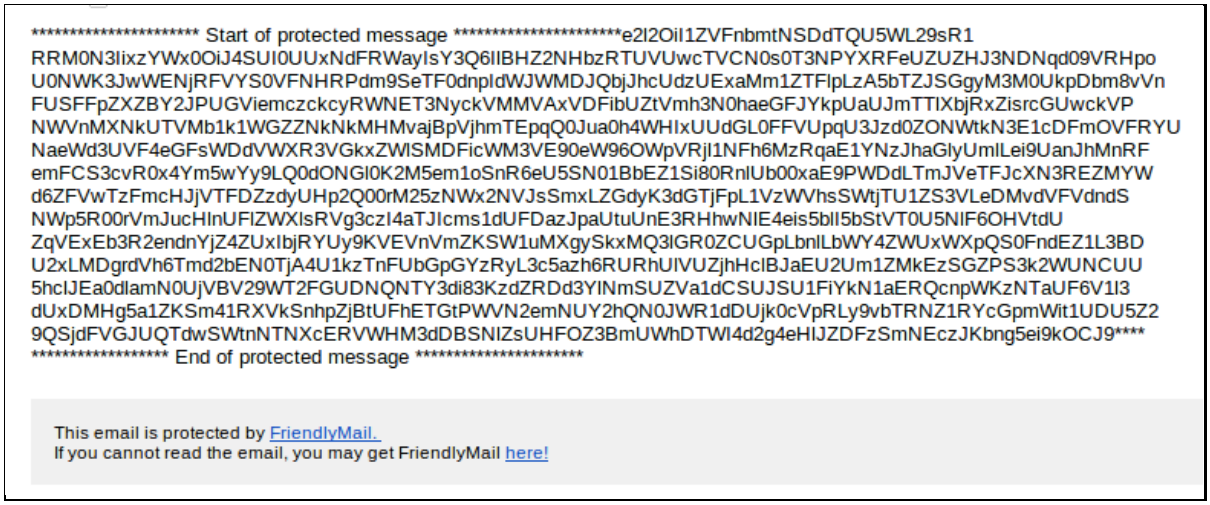



**Figure 4:** FriendlyMail confidential email and footer

**Consistent user interface.** To make FriendlyMail's use simple and effective, we focus on keeping the Gmail UI close to the default interface. We integrate FriendlyMail's functionality into the existing Gmail interface and keep visual modifications

to a minimum. Our few changes to the Gmail UI include adding a few buttons and a check box, e.g., the only addition to the main page of the Gmail's UI is a *Secret Compose* button; see Figure 1.

(a) **Sender side changes.** To make the process of sending an encrypted email automatic, we modify the Gmail UI, by adding our specific *Secret send* button (replaced with the original Gmail *Send* button). As visual aids, we change the color of the added button, the border around the composition area of the composition window to green. We also change the label of the added button to *Secret send*. These changes are intuitive security indicators, by which users can figure out that the current composition window and button are specialized for composing and sending sensitive emails; see Figure 5.

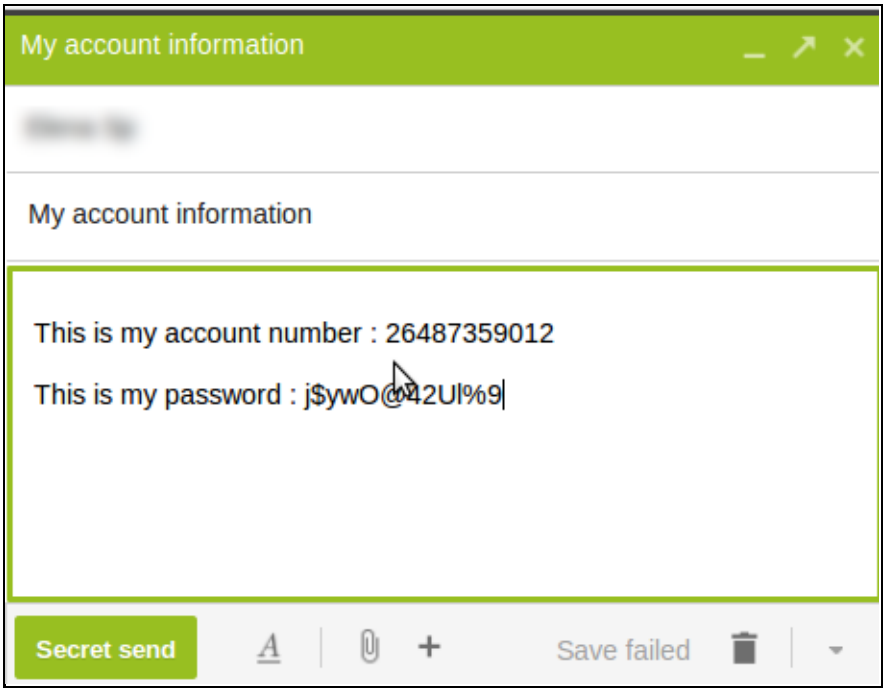



**Figure 5:** Composing an encrypted email

When the user finishes writing her email and clicks on the *Secret send* button to send her email, our prototype immediately grabs the email content, and performs all the steps discussed in Section 4.2.1 to secure the email content in encryption

mode. It then simulates a click on the Gmail original *Send* button to trigger the attached event listener, so that the encrypted email could be automatically sent through Gmail as normal emails. Directly calling Gmail functions, called from the event listener attached to the original send button, or implementing the SMTP protocol to send the encrypted data directly to Gmail servers, are alternatives to the click simulation on the *Send* button.

To enable integrity verification for an email, the user can tick a checkbox, added right next to the Gmail *Send* button in the composition window; see Figure 6. The checkbox is placed close to the send button to be noticeable by the user. To take user consent into consideration, we disable the checkbox by default, as it is the regular work flow of email. Therefore, the user decides when to enable the integrity-protection mode. When the user enables the checkbox, similar to the encryption mode, we change the color of the composition area to green to notify users that integrity verification mode is enabled; see Figure 7. We attach an event listener to the Gmail send button. The event listener is triggered upon clicking the send button by the user. If the integrity mode is enabled, FriendlyMail grabs the email content and carry on all the steps discussed in Section 4.2.2.

(b) **Receiver side changes.** As the user opens and views an email, FriendlyMail checks for encrypted/integrity-protected email by FriendlyMail. If an encrypted email is detected, the sender's identity is verified and the email content is decrypted automatically according to the steps presented in Section 4.2.1. However, if the user enables the option of *do not automatically decrypt*, a security statement in a green border and background along with a decryption button would be shown outside of the message body. The security statement indicates that the email content is encrypted and the user can see the original content by clicking on the decryption button; see Figure 10. If an integrity-protected email is detected, FriendlyMail automatically grabs the content and verifies the sender's identity and integrity of the content.

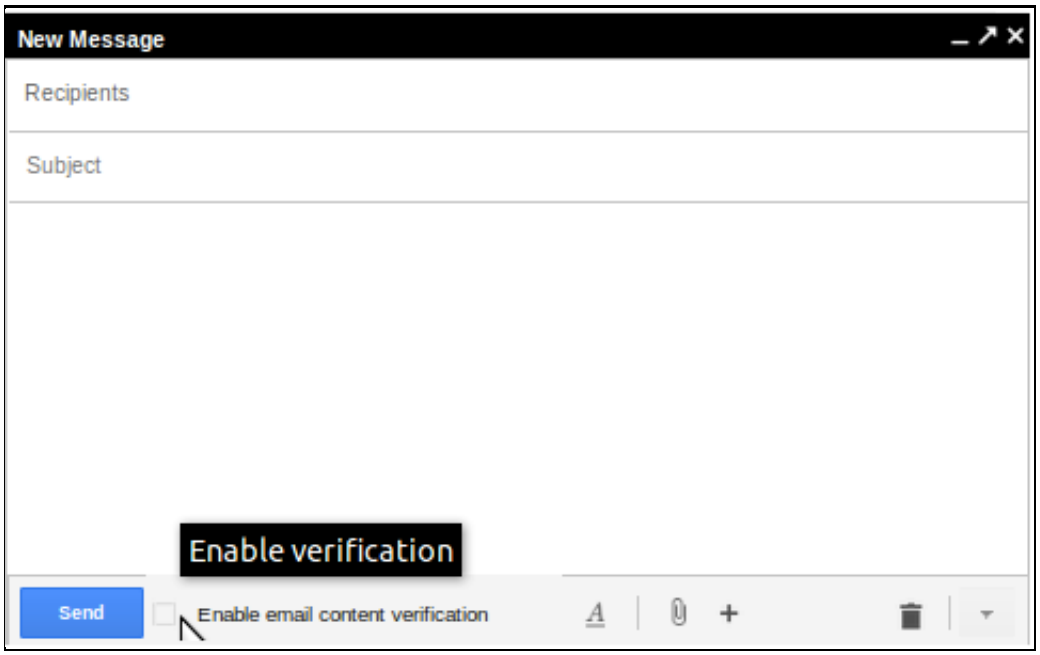



**Figure 6:** Before enabling email verification

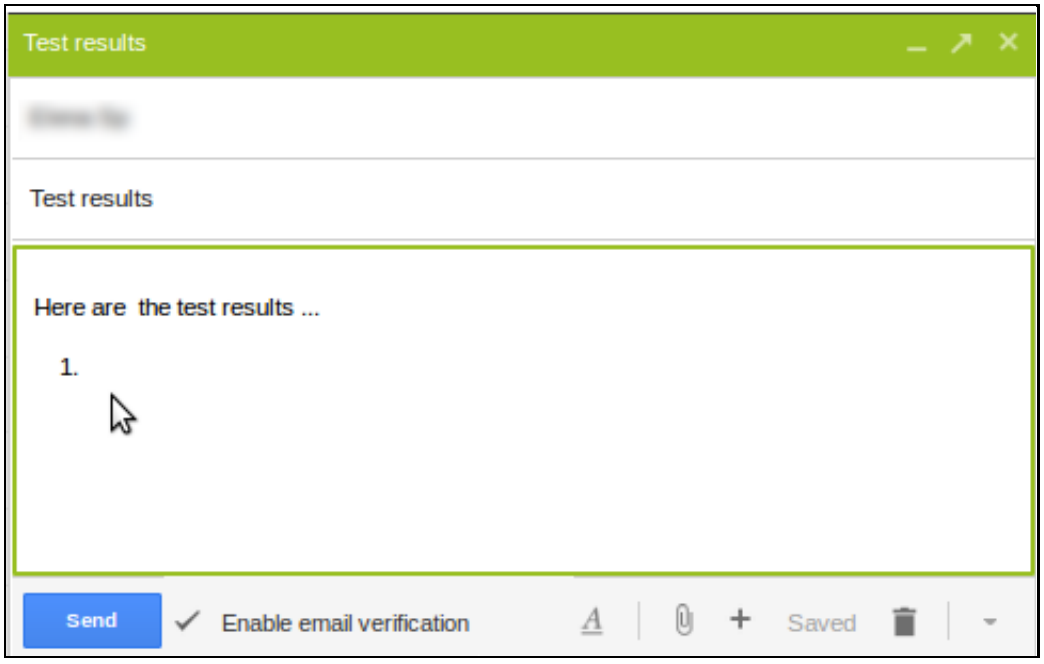



**Figure 7:** After enabling email verification

(c) **Visual cues and notifications.** To notify users about the successful completion or failure of an operation, we integrate several visual cues and security statements

into the Gmail UI. For example, FriendlyMail notifies senders whether or not their email can be protected (in terms of confidentiality and/or origin and content verification). The receiver is also notified if the encrypted email can be decrypted and/or the integrity of email can be verified. We use green color border and background for security statements in case of success and red color border and background in case of failure. Yellow color is used in situations where the current task requires user attention or action; see Figure 9a. As discussed in Section 4.3, after verifying the identity of the sender, we integrate the sender's OSN name, photo and a link to his OSN profile into the Gmail UI; see Figure 9b.

Modifications to the UI, e.g., adding buttons, visual cues and notifications, are also carefully performed to prevent any user confusion or possible attacks (see Chapter 6.1). In the course of development, we designed and implemented several UIs and performed several laboratory user tests among our research group. We consistently changed the UI, according to our tests' results to achieve more effective and usable UI.

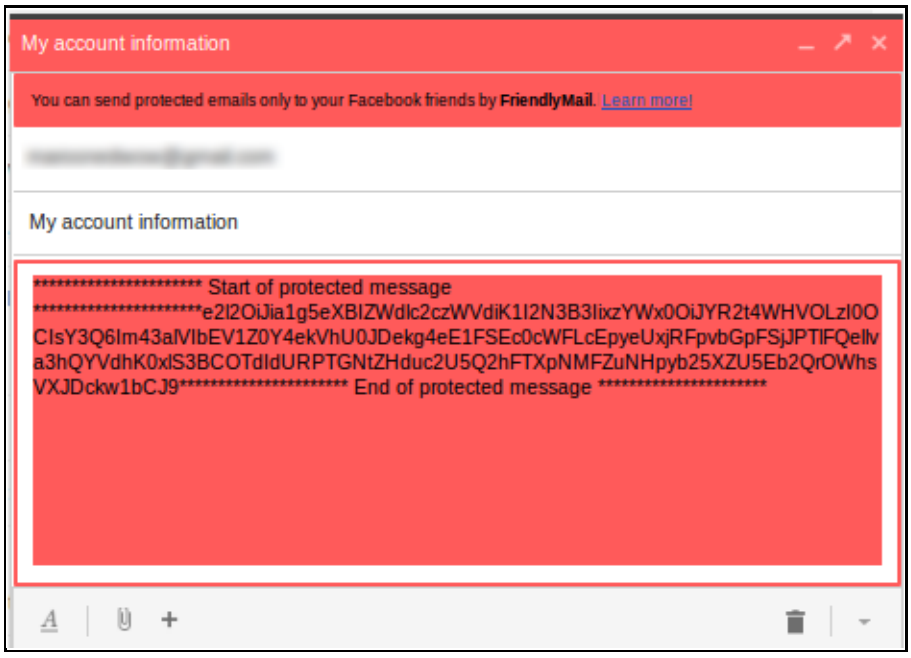
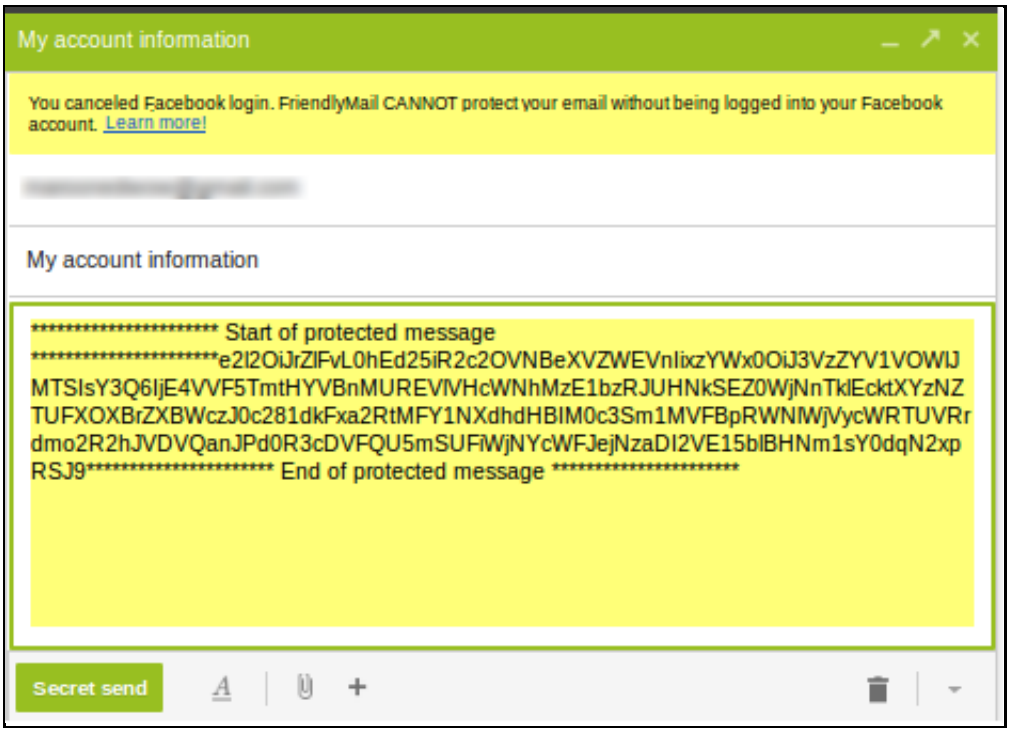
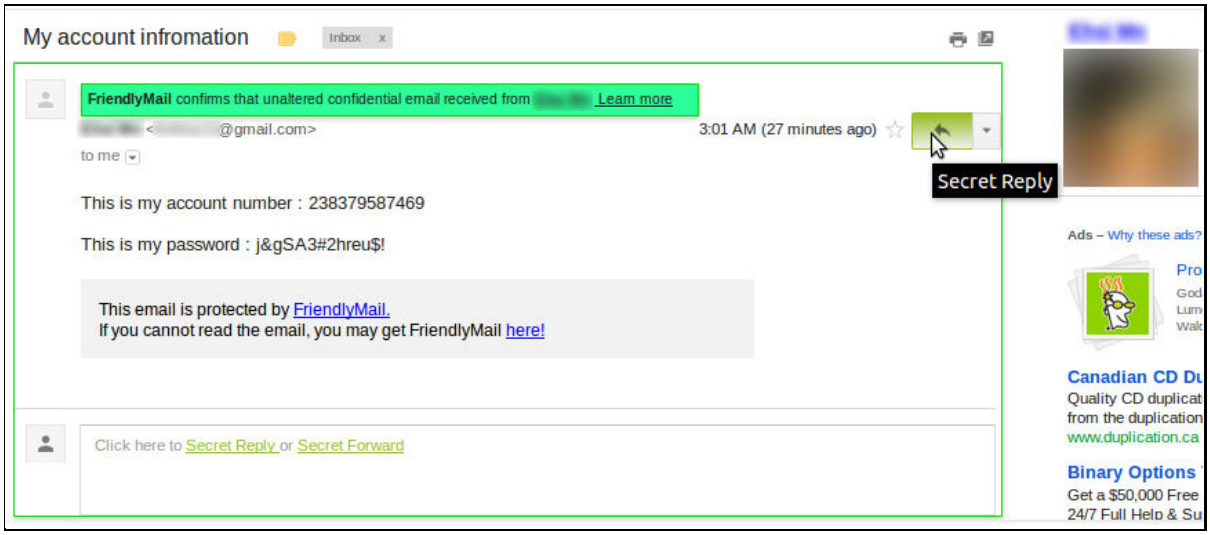



**Figure 8:** Disabled composition UI




**(a)** The modified Gmail UI message composition window with notifications and visual cues




**(b)** The modified Gmail UI message viewer with all integrated receiver's OSN account information and other visual cues.

**Figure 9:** Notifications and visual cues

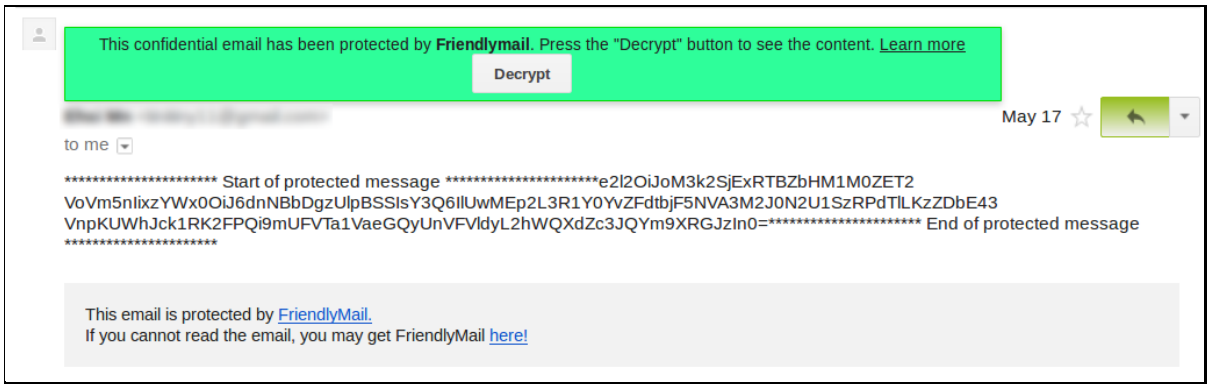



**Figure 10:** Decryption confirmation

**Protecting emails during compose and read.** The webmail interface is controlled by webmail providers. Email providers in general scan, index and save user emails during composition and opening/reading, for reasons including: automatic save, targeted advertising, spam filtering, virus detection, spell checking and search indexing. These operations are performed on the server and/or client ends. Email encryption prevents email providers from having control over email content to some extent; by encrypting emails, email providers can no longer scan email content on the server side. Emails residing on backup servers are also encrypted.

In Gmail's case, emails can be scanned on the client side when a user opens an email (targeted ad) and during composition (saving drafts and spell checks). We display the content of a received FriendlyMail-protected email inside a specialized HTML element, seamlessly overlaid on top of Gmail's email content display area. Thus, decrypted emails are displayed within the locally-trusted browser environment, inaccessible to the Gmail UI. Email content is protected during the composition of a confidential email as follows. We register an observer for HTTP requests (http-on-modify-request notification) using XPCOM's nsIObserverService [59] to intercept HTTP requests and discard requests regarding, e.g., auto saving, that post email content to Gmail servers. This allows the user to benefit from Gmail's rich text editor and formatting options, without leaking any content. FriendlyMail notifies users when drafts are discarded; see Figure 5 at the bottom of the composition window next to the

composition toolbar (save failed). However, this approach has two limitations that may adversely affect usability of email systems to some extent. First, users cannot switch between protected and unprotected modes during composition, as otherwise protected content may become accessible to Gmail in the unprotected mode. Thus, users must decide before composition if an email will be confidential or not. Second, some useful features as offered by Gmail become unavailable, e.g., automatic draft savings and spell checks. Note that, the built-in Firefox spell checker still remains accessible.

**Integrity checks for HTML emails.** Before rendering an HTML email, webmail clients may parse the HTML code through several filters, strip attributes and white spaces, and add new HTML elements. For example, Gmail only supports inline CSS, strips ID and CLASS attributes, and eliminates other attributes for security reasons. Some attribute might also be added by Gmail . Thus, the HTML email being displayed at the receiver side may be slightly different from the original one, and may result in different hash values. To overcome this issue, we strip HTML tags, attributes and white spaces which are not shown at both sides from both the sender/receiver ends before computing the hash. We strip elements that do not interfere with the integrity of the email but are mainly used for rendering HTML, leaving the original content intact. For example, we did not strip $href$ attribute, containing the URL of a link included in an email. The integrity of this URL is important, especially because the URL itself may not be shown to the user and be used to mount phishing attacks. Alternatively, the unmodified content could be fetched from Gmail's Show original link.

**Local storage.** Binding an email address to the corresponding Facebook account is not always straightforward, even when the same email address is used as the Facebook user ID or has been added to the user's account. Facebook currently does not allow applications search user's friends by their email addresses. Facebook applications, that request users' email address permission by the current user access token, can only get the email address of the current user, who is currently logged into Facebook.

However, searching among the current user's friends by their email address, is not allowed.

We gradually build a mapping between email and OSN IDs, and store this mapping locally in a SQLite database under browser's profile folder of the user who is currently using FriendlyMail. For a given email address, this local storage is first searched for the corresponding OSN ID. For the first protected email from Alice to Bob, Bob's OSN ID will not be found in the local storage. The addon will query Alice's Facebook friends' list for Bob, using Bob's name (as found in the email client). If multiple names are found, Alice is shown the profiles of all matching users, and asked to select the intended recipient. When no matching names are found, Alice's entire friends' list is displayed. Alice is also asked for confirming Bob's profile, even when a single match is found; see Figure 11. This local storage also improve the usability of our system, as prompting users to choose the corresponding OSN account, each time they want to send/read a FriendlyMail email, degrades the usability of our system. After Alice's selection, Bob's email address and OSN ID are locally stored and used for future FriendlyMail exchanges. We do not require that each user uses its own browser profile so FriendlyMail can also be used on a shared or public system. Users' email address and Facebook ID are not considered to be sensitive, and the local storage can be deleted after removing FriendlyMail addon from browser. Moreover, in the case that an email address and Facebook ID, found on the database, do not belong to a friend of the current user, FriendlyMail could detect by searching the Facebook id among the current user's OSN friend. However, the addon verifies if a locally stored Facebook ID is currently a friend of the sender, before publishing keys/hashes.

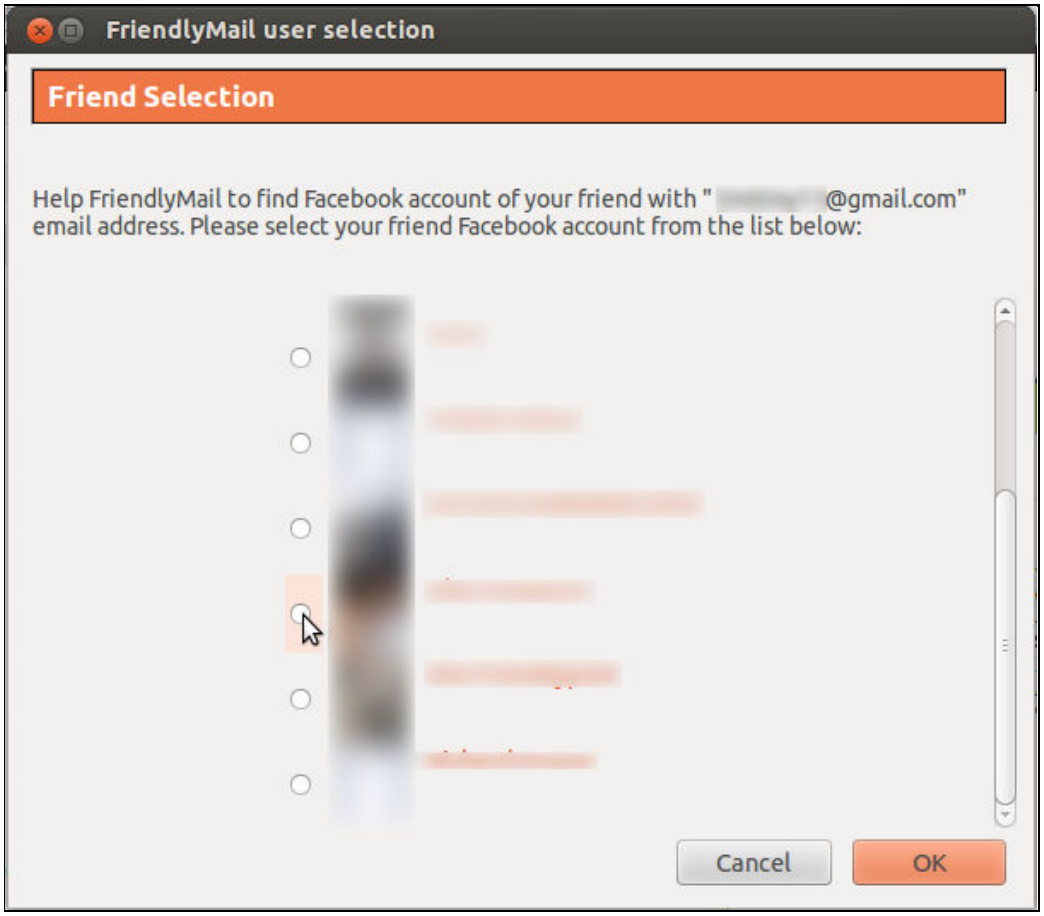



**Figure 11:** Selection/confirmation of friends' Facebook profile

## 5.3 Facebook Application

To interact with Facebook and perform Facebook related functionality on behalf of the user, our Firefox addon uses our registered Facebook application (to leverage Facebook API, an app ID is required). We use Facebook login API [27] to authenticate the sending/receiving FriendlyMail users and get their access token (to retrieve some non-public user information a valid user access token is required). We use the client side version of the login API, in which FriendlyMail requests a login dialog from Facebook to be displayed to the user. The familiar login page assures users that they are providing credentials directly to Facebook; see Figure 12. The login request contains parameters such as FriendlyMail *client_id* and a *redirect_URI* where the access token will be returned to. Another parameter is *scope*, in which all permissions required by the application should be listed. The app requests the following permissions: *read_stream*, *user_likes* and *publish_stream*. During the app's installation, we inform the user about these permissions, and the data that FriendlyMail will be accessing from the user's Facebook profile; see also Section 6.1, item 12.

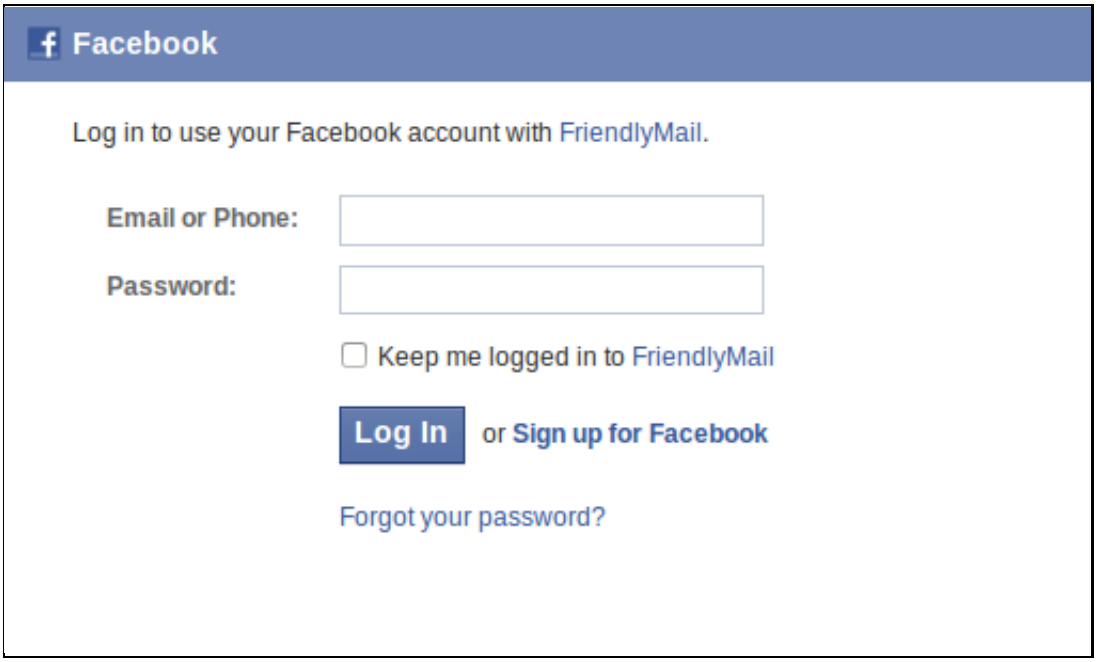



**Figure 12:** Login page

When the FriendlyMail addon is launched for the first time, the Facebook app is also installed (with user permission) and the user is prompted to grant the above mentioned permissions. An access token is received as a part of the redirect-URI in the Facebook response after a successful login. The token provides secure access to Facebook APIs (albeit time-limited). As client side's user access tokens are short-lived, FriendlyMail prompts users to login whenever the access token is expired. The token is used to, e.g., to publish/fetch keys and hash values. Most API calls regarding searching e.g., among friends or hashes/keys are done through Facebook Query Language (FQL)[21], and publishing hashes/keys as feeds to the user's Facebook wall is done through Graph API[24]. All connections between the user client (FriendlyMail Firefox extension) and Facebook servers are also secured using SSL.

**Optimization of searching the keys/hashes of the emails.** To search keys/hashes of FriendlyMail emails through Facebook API, the FriendlyMail addon finds the sender's Facebook account, and searches for the computed hash using the Facebook query language (FQL) API. FQL calls are quite fast, but as more hashes are added to a user's profile, the search time may be noticeable. As an optimization, we currently limit the API query to a specific time interval (e.g., 24 hours), based on when an email is received; the FriendlyMail will retrieve the date and time that a FriendlyMail email is delivered to the receiver and converts it to the UNIX time. It then searches among hashes/keys posted on the sender's wall after the calculated UNIX time. This optimization also restricts replays of old emails from a compromised email account; see Section 6.1, item 7.

## 5.4 Variants to FriendlyMail

We outline several variants below. We have also implemented variants (a) and (b) as customizable options; see Figure 13.

(a) **Custom keys.** One obvious risk for our key transport via OSN sites is that the email and OSN providers may collude to decrypt a confidential email (see

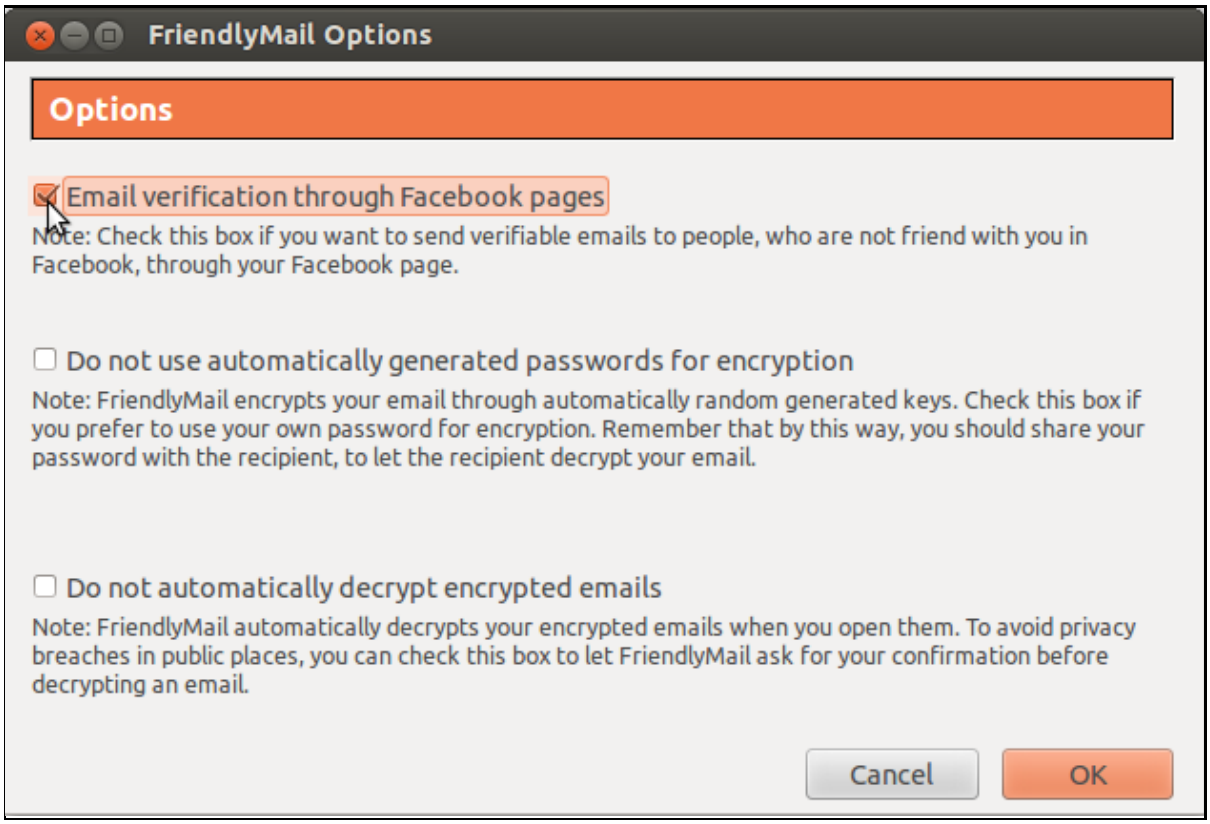



**Figure 13:** FriendlyMail options

Section 6.1, item 1). As a mitigation, advanced users can generate their own encryption keys and share them via a secure channel, such as in-person meeting or over the phone.

(b) **Integrity and origin checks through OSN organizational pages.** Existing impersonal relationships can also be leveraged to provide message and origin authentication. For example, many real-world and Internet-based organizations currently maintain an OSN presence, e.g., via Facebook Pages.[1] Example organizations include: TD Bank, Harvard University, USENIX Association. Many users are connected to these pages through the "Like" relationship. Currently, these organizations cannot send emails with integrity or origin authentication;

[1] `https://www.facebook.com/about/pages`

in contrast, spam and phishing emails are often sent by impersonating such organizations. An integrity-protected email can be sent as follows: the email is sent as usual, and the hash value is posted on the organization's Facebook Page (publicly accessible). Assuming the recipient has already "Liked" the Page, the email is verified as follows: FriendlyMail checks the hash value of the received email on the organization's Page. The verification result and the corresponding page are displayed to the recipient. The organization's Page ID is included in the email (as an additional footer), so that receivers can easily locate the Page; see Figure 14. However, the Like relationship is still checked for verification. Note that, as Facebook pages are public and visible even without having any OSN account, unlike the user profile, Facebook pages can be searched using their email address or their website associated with the page through the API. Therefore, to find a page, if the sender's email address has been added as the Facebook page's admin, it can be directly searched by the email address, eliminating the need for including its ID in the email content.

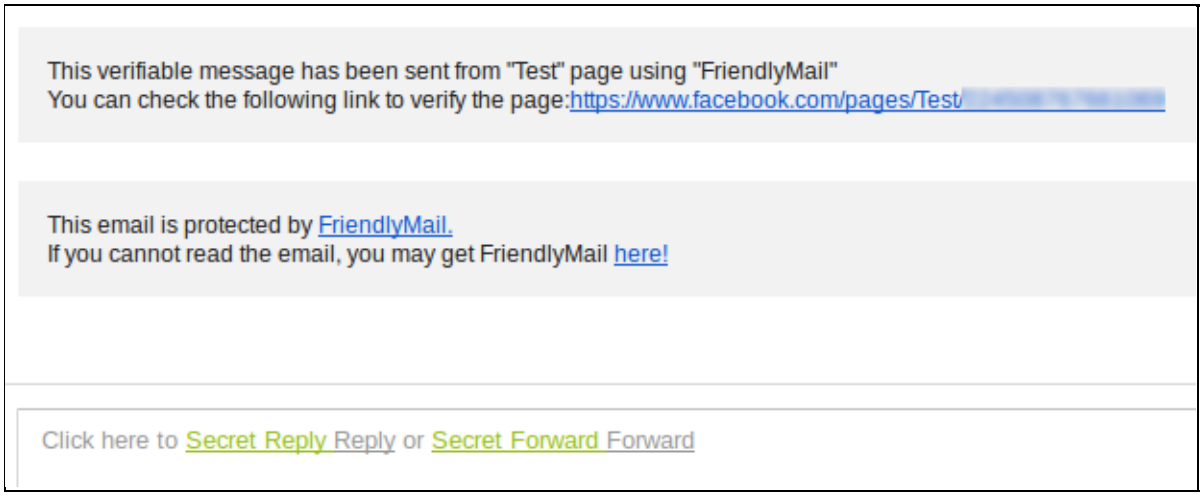



**Figure 14:** Page footer

(c) **Message confidentiality and integrity through indirect OSN relationships.** If two users are connected by mutual friends (e.g., Facebook friends-of-friends), FriendlyMail can provide confidentiality and integrity protections. The

sender can post key/hash values visible only to the receiver. However, this would require that the sender can access her friends' friend lists. The current Facebook Graph API disallows access to friends-of-friends lists by Facebook apps (but can be implemented by JavaScript addons). Direct authentication is not achieved for these indirect trust relationships, as both the sender and receiver must rely on their mutual friend's verification.

(d) **Integrity checks for unconnected users.** If Alice and Bob have no OSN relationships, they can still achieve content integrity protection, but no identity verification. Alice can publicly post the hash of an email, and Bob (or anyone with access to the email content) can verify the content. Alice should include her OSN ID in the email content, so that Bob can easily locate her profile and check the hash value. FriendlyMail notifies both Alice and Bob that only content verification is provided. Bob is also asked to manually verify Alice's identity by reviewing her OSN profile.

(e) **Sharing keys through other channels.** OSN provides an easy way for sharing per-email keys; however, other channels can be used instead. If a user's contact list with phone numbers is available to a FriendlyMail addon (e.g., when used from a smartphone), the keys can be sent via SMSes to the recipient. Contact lists from instant messaging applications may also be used for key transport. The FriendlyMail addon must be able to automate the key extraction from these secondary channels; i.e., users cannot be expected to manually input key materials.

(f) **Group emails.** A group email address can comprise an unlimited number of email users. Group emails may be supported for integrity and origin authentication, if the email group is also represented as an OSN entity (e.g., Facebook Groups). Each group member is connected to the OSN group whose posts are only made available to its members and are not pushed to any user's OSN page; e.g., Facebook Groups with the "Secret" privacy option.[2]

[2] `https://www.facebook.com/help/220336891328465`

(g) **Multiple recipients.** For confidentiality protection, *all* recipients must be directly connected to the sender. The OSN provider must support private message posts to a custom set of users. For integrity-only protection, when all recipients are not OSN friends of the sender, the hash can be posted publicly.

# Chapter 6

# FriendlyMail Analysis

In this chapter, we discuss several attacks on FriendlyMail and provide an analytical usability and deployment analysis of FriendlyMail.

## 6.1 Threat Analysis

We consider different attacks on the FriendlyMail proposal under the threat model outlined in Chapter 3.

1. **Collusion attacks.** Our assumption of non-colluding email and OSN providers may be difficult to satisfy in practice. Beyond usual information sharing between businesses, and company acquisitions, the email and OSN providers' data may be subpoenaed if they are under the same legal jurisdiction (as is the case for Google and Facebook). Note that, the recent incident [5] leaked the fact that the PRISM program used by the National Security Agency (NSA), gets users' private online communications and data, through nine prominent Internet companies, e.g., Microsoft, Yahoo, Facebook, Google, Apple, etc.

   FriendlyMail can of course be extended to support multiple channels for sharing a per-email key or the encrypted email (by dividing them into pieces; cf. [10]). For example, $K_m$ may be divided into two parts (e.g., $K_m = K_1 \oplus K_2$), each

part being shared with the recipient through independent channels; e.g., $K_1$ via Facebook and $K_2$ via VK.[1] However, we believe that such a design can only be useful for a small fraction of users due to usability/maintainability issues (e.g., requiring two OSN accounts with two different services).

Another apparent defence against colluding parties is to share a password between two users, and then post the per-email key to OSN after being encrypted by the password. The OSN provider may try to launch a dictionary attack on the encrypted key, assuming the password is relatively weak. To verify a key, the OSN service would require access to the encrypted email. Thus, colluding parties can still compromise encrypted emails, unless the shared password is *strong* (e.g., with more than 80 bits of entropy).

2. **Information leakage.** With FriendlyMail, the OSN service receives hash/key values for every protected email, including identities of the communicating parties. The OSN provider also learns every time a protected email is accessed (unless the retrieved key/hash values are stored locally). Although the email content remains protected, the OSN provider now has access to the communication patterns of protected emails between two users. To restrict such leakage, the sender's addon may post bogus key/hash values; on the receiver side, the addon may retrieve multiple key/hash values, and then use/ignore these values as appropriate.

3. **Email as account recovery.** Usually, OSN providers rely on the user's email account for password recovery. Thus, if a user's email account is compromised, it is trivial to also compromise the OSN account. Therefore, we recommend that FriendlyMail users be careful with their email accounts (e.g., logging in only from user-owned devices), and use alternative password recovery options (e.g., via SMS).

4. **OSN notifications.** OSN providers might send email notifications for events

[1] A Russian language OSN: `vk.com`.

such as a new post, e.g., Facebook notifications [23]. Notifications for FriendlyMail messages must be disabled; otherwise, keys and hashes are sent to the user's email address, allowing the email provider access to confidential content.

5. **Impersonation attacks.** Assume that the hash of an integrity-protected email is publicly posted (e.g., for unconnected users). If an attacker can intercept the email, he can resend it to a different recipient, impersonating the real sender. To detect this attack, the receiver's address is also hashed, so that the replayed email has a different hash value. As a result, the FriendlyMail addon on the receiver's side will fail to find a corresponding hash on the original sender's OSN account. The receiver is then notified about the hash mismatch.

6. **Impersonation to a friend on OSNs.** The adversary is familiar with FriendlyMail design or is a FriendlyMail user. She intends to abuse FriendlyMail to send malicious emails which look protected and trusted to the targeted user. To do so, she needs to be friend with the receiver on the OSN. Therefore she creates a fake account in the OSN impersonating a friend of the targeted user and tricks the targeted user to accept her friend request. As FriendlyMail relies on existing authenticated connections on the OSN, attack regarding fake accounts cannot be detected.

7. **Compromised email accounts.** If a sender's email account is compromised, but not the OSN account, several attacks can be considered (besides item 3). An attacker will be unable to decrypt encrypted messages without access to the victim's OSN account; note that, we do not consider brute-forcing a random AES 128-bit key. The attacker also cannot send any new protected emails from the compromised account (requires the OSN account access). However, as hashes of previously sent protected emails are available on the sender's OSN account, the attacker could try to resend an old email to the original recipient. Note that, sending such an email to new recipients will not be verified as discussed in item 5. The attacker can also resend a captured FriendlyMail-processed email

without even compromising the sender's email account; the email can be sent from any account by changing the FROM field to the intended sender's address. As the corresponding hash value would already be available on the victim's OSN account, on the receiver's side, FriendlyMail would identify the replayed email as authentic.

This attack could be mitigated by calculating the difference between the time when the email was sent/received, and the time when the corresponding hash value was published. If the difference is above a certain threshold, the receiver is notified. Another detection mechanism would be to verify whether the order in which emails are received corresponds to the order in which the hashes are published.

8. **Exposure of key and hash values.** For obvious reason, the per-email key $K_m$ must be made available only to the receiver. Any other relaxed restriction on $K_m$ may break confidentiality. For example, if the key is published as visible only to the sender's friends' list, any of those friends can access the email content if they have access to the encrypted email. Similarly, for the integrity-only protection, the hash values should preferably be available only to the recipient. Otherwise, the communication patterns of a sender would be exposed, e.g., to OSN friends/non-friends; see also item 2. Note that, only the OSN provider may learn who the recipients are of a protected email, but the sender's OSN friends/non-friends cannot know the receivers' identities.

9. **Compromised OSN accounts.** An attacker who has compromised the victim's (Alice) OSN account has access to all stored key and hash values. The attacker can also monitor the OSN account for new keys as they are posted. If he has access to encrypted emails, the attacker can now read confidential emails. He can also launch impersonation attacks as follows. The attacker creates an email for Bob, impersonating Alice (i.e., FROM = $ID_A$), and publishes the hash on Alice's OSN account (encryption keys can also be posted accordingly).

When Bob checks the email for integrity and origin authentication, the verification process succeeds (Bob is also able to decrypt an encrypted message). The attacker can also delete all stored hash values and keys. This will not affect the decryption and validation of past emails, if local copies of all keys and hashes are kept.

10. **Malicious email content.** An attacker can send malicious URLs (including links to adulterated FriendlyMail addons), and attachments within a protected email. We do not address such attacks, as they are similar to existing phishing emails. Only confidentiality and integrity of the content (malicious or benign), and sender-authentication can be expected from FriendlyMail.

11. **Visual cues and security statements.** All security indicators are placed outside of the email content area, to prevent the forgery of FriendlyMail indicators within the content of a malicious email. The attacker can simply forge a FriendlyMail protected email, add a FriendlyMail security statement conveying a successfully decrypted/verified email in the beginning of the email. She then sends it to another FriendlyMail user, tricking the user that it is a legitimate and verified FriendlyMail email. As the user expects FriendlyMail to be secure by default and everything is automatically controlled by FriendlyMail, if security of FriendlyMail depends on the user's attention, FriendlyMail might fail. Also in the case that an email cannot be protected by FriendlyMail (e.g., the receiver cannot be found among the sender's OSN friends), FriendlyMail disables the composition UI, removes the send button to avoid any sensitive email content to be accidentally sent by the user. It also notifies users through a red border color and a security statement; see Figure 8.

12. **No trusted third-parties.** We introduce no third parties in our design; users are also not required to trust the FriendlyMail developers. Both the FriendlyMail Firefox addon and Facebook app are open-source. We do not run any service for users, or collect *any* information from users. Such a design choice

may help user-acceptance, as a user's email and OSN accounts could be extremely privacy-sensitive. Even though a *trusted* third party could simplify the design and increase ease-of-use, we strongly discourage such practices, especially from a privacy-enhancing tool.[2]

## 6.2 Usability and Deployment

Below we discuss FriendlyMail deployment and usability issues with email and OSN providers. We have tested the current implementation within our research group, mainly to check functionality and UI issues. The addon worked as expected. However, no formal user studies have been performed yet.

### 6.2.1 Deployment Analysis

1. Email providers lose access to confidential messages, and thus, cannot directly benefit from content-aware advertising. However, generic ads can still be served. As only selected messages will be confidentiality-protected, revenues for ad-supported email services are unlikely to suffer.

2. OSN providers may observe only minor changes to the number of posted messages, even if FriendlyMail is largely adopted. Each protected email will result in additional content posted to OSNs; however, the size of each post is relatively small (e.g., tens of bytes). Although the global email volume per day is large, FriendlyMail may be used only for a small fraction of selected messages.

3. OSN providers will also receive more search queries due to FriendlyMail searches for friends (and other supported relationships) and post messages. Queries are

[2]Cf. a statement from the popular Enlocked email security service (`https://www.enlocked.com/Works.html`): "...the only one able to read your secured messages is you!...The systems at Enlocked only have access to your messages for the short time we are encrypting or decrypting them, and then our software instantly removes any copies."

issued when sending and opening each protected email. Server-side costs for these queries would be non-trivial; the average number of friends is expected to be moderate (e.g., 190 in Facebook as of Nov. 21, 2011 [4]); however, the number of posted messages from FriendlyMail and regular OSN usage would likely grow over time (hundreds or thousands, unless old posts are gradually deleted). Note that modern OSN providers are apparently well-equipped to handle such loads, as evident from their support for thousands of OSN-specific apps that make extensive use of such queries.

### 6.2.2 Usability Analysis

1. Assume that a sender, Alice, is already logged into her OSN account; she also identified Bob's OSN profile (the receiver) to the FriendlyMail addon, when she first sent a protected email to Bob. Subsequently, Alice only needs to select *Secret Compose* for confidential emails, and tick a checkbox for integrity-only protection. Bob can open an email as usual; sender authentication, message verification, and decryption are performed automatically. However, to benefit from FriendlyMail protections, Bob must check the UI notification messages as provided by the addon.

2. Users must explicitly select integrity/confidentiality protections for their messages. This allows users to control how their messages are protected. However, users may mistakenly send out sensitive information unprotected. This risk is unavoidable as long as we cannot encrypt all emails by default, which may break email communications in many scenarios (e.g., emails sent to OSN-unconnected receivers).

3. Searching for email content is a useful feature for many users, and could be even more efficient than organizing emails through complex rules (see e.g., [86]). However, keyword search within server-stored encrypted emails is not supported for now (but cf. [79]). Searches on locally-stored decrypted emails can be easily

supported (e.g., by saving the decrypted versions, or the keys).

4. One challenge in our design could be the hash verification race issue, where there is a delay in the stored hash on the OSN to be available to the receiver. However, this should not be a concern now, as OSNs (e.g., Facebook) are very fast. In the presence of such a delay, the receiver may open the received email and try to verify or decrypt it, while the stored hash is not still available on the sender's OSN account. Therefore, the locally computed hash value does not match any hashes on the sender's OSN account and the email cannot be verified or the decryption key cannot be found. To address this issue, one solution could be to check the presence of the stored hash on the sender's OSN account by FriendlyMail, before sending the email to the receiver. As another solution, if FriendlyMail at the sender side could not find the exact matching hash value in the first attempt, it should ask the receiver to check the email again later to make sure if it is not a false alarm due to the latency. However, both solutions adversely affect usability of FriendlyMail to some extent; in the first solution, the sender should wait until the hash is available, then the email can be sent. In the second solution, the receiver needs to check the received email several times.

5. Message posts from FriendlyMail on the OSN site may clutter the sender's message page. However, as is the case for Facebook, the OSN site may support hiding messages from selected apps/users. One limitation of Facebook API is also that Facebook apps cannot automatically publish hidden posts to users' wall. However if users hide them, Facebook apps can search among hidden posts.

6. We depend on the OSN's availability to be able to provide email Confidentiality and email integrity. When the OSN servers are down, users cannot publish/retrieve keys/hash values and encrypt/decrypt/verify their emails. A workaround is to have a locally backup of all keys and hash values on the user's

machine. Also in case of Facebook, using the post method[3] limits Facebook applications to only 25 posts per day to the user's wall post on behalf of the user. Therefore, FriendlyMail users cannot send more than 25 emails per day. The number of posts per day is also subject to change.

[3] www.fbdevwiki.com/wiki/graph:post

# Chapter 7

# Conclusions and Future Work

Few corporations (e.g., Google, Microsoft) are becoming de-facto global communication middlemen, as more users are signing up for web-based email services. Besides these corporations, governments are also trying to access the cloud stored email data (see e.g., [77, 40]), ignoring privacy rights of global citizens. Providing crypto support for email is apparently easy (e.g., several crypto primitives exist to enable authentication, confidentiality and integrity); however, key management is significantly more challenging. We propose and implement FriendlyMail, focusing on key management issues. We automate key generation and transport, by leveraging widely-used OSN services and existing trust-relations among OSN users. Beyond email encryption, such a key transport mechanism may enable privacy protection for additional user-to-user data communication services (e.g., encrypted IMs, file sharing via public cloud). FriendlyMail does not require any server-side modifications, and thus can be immediately deployed. Although it enables secure email communications mainly between OSN-connected users. In contrast, most other solutions target generic adoption, i.e., can support any sender and receiver. The collusion between OSN and email providers could be real a concern. Our current implementation supports the web-based Gmail service for users who are also connected via Facebook (as friends, or through the Like relationship on a Facebook Page). FriendlyMail Firefox addon is available at `http://users.encs.concordia.ca/~mmannan/software.html`.

## 7.1 Future Work

Several points, specially regarding users' perspective, may be considered in the future.

1. Collusion attacks between email and secondary channel providers are still a real problem which requires further investigations.

2. FriendlyMail currently is limited to the basic email features. Further improvements to the implementation are required to add the support for file attachment, searching and archiving emails and key backup for better user acceptance.

3. Another future direction should be toward addressing the fact that how we can make everyday users aware of email privacy issues and influence them using a privacy-friendly solution. What benefits or desired features users would gain (expect to have) by using these solutions, so that they consider using such a privacy-friendly tool.

4. On the trust issue, further studies need to be done to figure out how to convince users to build trust in privacy-enhanced systems; and make them understand that their privacy is maintained and no personal information would leak through the use of such tools. Despite the fact that FriendlyMail is developed as a fully open-sourced tool, still average users do not have enough understanding of the underlying code to verify it by themselves.

# Bibliography


[1] J. Callas and L. Donnerhacke and H. Finney and D. Shaw and R. Thayer. OpenPGP Message Format, November 2007. RFC 4880.

[2] B. Adida, D. Chauand, S. Hohenberger, and R. L. Rivest. Lightweight email signatures (extended abstract). In *Fifth Conference on Security and Cryptography for Networks (SCN'06)*, pages 288–302. Springer Verlag, 2006.

[3] B. Adida, S. Hohenberger, and R. L. Rivest. Lightweight encryption for email. In *USENIX steps to reducing unwanted traffic on the internet workshop (SRUTI'05)*, pages 13–13, Cambridge, MA, jul 2005.

[4] L. Backstrom. Facebook data science. Online article (November 21, 2011). `https://www.facebook.com/notes/facebook-data-team/anatomy-of-facebook/10150388519243859`.

[5] BBC News. Edward Snowden was NSA Prism leak source - Guardian. News article (June 10, 2013). `http://www.bbc.co.uk/news/world-us-canada-22836378`.

[6] BBC News. Facebook has more than 83 million illegitimate accounts. News article (Aug. 2, 2012). `http://www.bbc.co.uk/news/technology-19093078`.

[7] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO'01*, pages 213–229, Santa Barbara, California, USA, 2001.

[8] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *IEEE Symp. on Security and Privacy*, May 2012.

[9] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu. The socialbot network: when bots socialize for fame and money. In *ACSAC'11*, Orlando, FL, USA, 2011.

[10] K. Butler, P. Trayno, W. Enck, J. Plasterr, and P. McDaniel. Privacy preserving web-based email. In *2nd International Conference on Information Systems Security (ICISS'06)*, Kolkata, India, Dec. 2006.

[11] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *USENIX NSDI'12*, San Jose, CA, USA, 2012.

[12] ComScore. It's a social world: Top 10 need-to-knows about social networking and where its headed. Online article (December 21, 2011). `http://www.comscore.com/Insights/Presentations_and_Whitepapers/2011/it_is_a_social_world_top_10_need-to-knows_about_social_networking`.

[13] Comscoredatamine.com. People spent 6.7 billion hours on social networks in October. Online article (January 4, 2012). `http://www.comscoredatamine.com/2012/01/people-spent-6-7-billion-hours-on-social-networks-in-october`.

[14] L. F. Cranor and S. Garfinkel. Guest editors' introduction: Secure or usable? *IEEE Security and Privacy*, 2(5):16–18, Sept. 2004.

[15] S. Crocker, N. Freed, J. Galvin, and S. Murphy. MIME Object Security Services, October 1995. RFC 1848.

[16] D. Crocker and T. Hansen and M. Kucherawy. DomainKeys Identified Mail (DKIM) Signatures, September 2011. RFC 6376.

[17] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, Mar. 1983.

[18] N. Doshi. Facebook applications accidentally leaking access to third parties - updated. Symantec official blog (May 10, 2011. http://www.symantec.com/connect/blogs/facebook-applications-accidentally-leaking-access-third-parties.

[19] M. Dworkin. Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality (NIST SP 800-38C), May 2004.

[20] Enlocked. A plugin for Firefox, iPhone, Android and Outlook. http://www.enlocked.com/.

[21] Facebook. Facebook Query Language (FQL) reference. https://developers.facebook.com/docs/reference/fql.

[22] Facebook. Facebook reports first quarter 2013 results. Online article (May 1, 2013). http://investor.fb.com/releasedetail.cfm?ReleaseID=761090.

[23] Facebook. Notifications. https://www.facebook.com/help/notifications.

[24] Facebook. Publishing. https://developers.facebook.com/docs/reference/api/publishing/.

[25] Facebook. Safety center, tools. Online document. http://www.facebook.com/safety/tools.

[26] Facebook. Statement of rights and responsibilities. http://www.facebook.com/legal/terms.

[27] Facebook. The login flow for web (without JavaScript SDK). https://developers.facebook.com/docs/facebook-login/login-flow-for-web-no-jssdk/#confirm.

[28] Facebook. What happens to content (posts, pictures, etc) that I delete from Facebook? http://www.facebook.com/help/356107851084108/.

[29] S. Fahl, M. Harbach, T. Muders, and M. Smith. Confidentiality as a service - usable security for the cloud. In *TRUSTCOM'12*, Liverpool, UK, June 2012.

[30] S. Fahl, M. Harbach, T. Muders, and M. Smith. TrustSplit: Usable confidentiality for social network messaging. In *(Hypertext'12)*, Milwaukee, WI, USA, June 2012.

[31] S. Fahl, M. Harbach, T. Muders, M. Smith, and U. Sander. Helping Johnny 2.0 to encrypt his Facebook conversations. In *Symposium on Usable Privacy and Security (SOUPS'12)*, pages 11:1–11:17, Washington, D.C., 2012.

[32] A. J. Feldman, A. Blankstein, M. J. Freedman, and E. W. Felten. Social networking with frientegrity: privacy and integrity with an untrusted provider. In *Security'12*, Bellevue, WA, 2012.

[33] Felix. GPG4Browsers. Open source project. http://gpg4browsers.recurity.com/.

[34] Fortinet. Fortimail identity based encryption. White paper. http://www.fortinet.com/sites/default/files/whitepapers/fortimail-identity-based-encryption.pdf.

[35] A. Fry, S. Chiasson, and A. Somayaji. Not sealed but delivered: The (un)usability of s/mime today. In *Annual Symposium on Information Assurance and Secure Knowledge Management (ASIA'12)*, Albany, NY, June 2012.

[36] S. L. Garfinkel. Enabling email confidentiality through the use of opportunistic encryption. In *Conference on Digital Government Research (dg.o'03)*, Boston, USA, may 2003.

[37] S. L. Garfinkel. *Design principles and patterns for computer systems that are simultaneously secure and usable*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2005.

[38] S. L. Garfinkel, D. Margrave, J. I. Schiller, E. Nordlander, and R. C. Miller. How to make secure email easier to use. In *Conference on Human Factors in Computing Systems (CHI'05)*, pages 701–710, Portland, Oregon, USA, 2005.

[39] S. Gaw, E. W. Felten, and P. Fernandez-Kelly. Secrecy, flagging, and paranoia: adoption criteria in encrypted email. In *Conference on Human Factors in Computing Systems (CHI'06)*, pages 591–600, Montreal, Quebec, Canada, 2006.

[40] Google. Transparency report: What it takes for governments to access personal information. Google official blog (January 23, 2013). `http://googleblog.blogspot.co.uk/2013/01/transparency-report-what-it-takes-for.html`.

[41] Greasemonkey. The weblog about Greasemonkey. `http://www.greasespot.net/`.

[42] P. Gutmann. Why isn't the internet secure yet, dammit. In *Asia Pacific Information Technology Security Conference (AusCERT'04)*, Gold Coast, Australia, May 2004.

[43] P. Gutmann, editor. *Engineering security*. Book draft, Apr. 2013. `http://www.cs.auckland.ac.nz/~pgut001/pubs/book.pdf`.

[44] K. Hampton, L. S. Goulet, L. Rainie, and K. Purcell. Social networking sites and our lives. Online report (June 16, 2011). `http://www.pewinternet.org/Reports/2011/Technology-and-social-networks/Summary.aspx`.

[45] J. V. Hoboken, A. Arnbak, and N. V. Eijk. Cloud computing in higher education and research institutions and the USA Patriot Act, Nov. 2012. `http://ssrn.com/abstract=2181534`.

[46] ITWorld.com. Facebook's 'man in the middle' attack on our data. News article (February 5, 2012). http://www.itworld.com/it-managementstrategy/247344/facebooks-man-middle-attack-our-data.

[47] R. Kainda, I. Flechais, and A. Roscoe. Security and usability: Analysis and evaluation. In *International Conference on Availability, Reliability, and Security (ARES'10 )*, pages 275–282, 2010.

[48] W. Koch and M. Brinkmann. STEED – usable end-to-end encryption, Oct. 2011. Whitepaper and open-source project: http://g10code.com/steed.html.

[49] A. Lambert and S. Bezek. Waterhouse: secure e-mail for human beings. Online document. http://social.cs.uiuc.edu/class/cs465/assignments/evaluation%20and%20walkthrough/lambert-bezek.pdf.

[50] A. P. Lambert, S. M. Bezek, and K. G. Karahalios. Waterhouse: enabling secure e-mail with social networking. In *CHI'09*, Boston, MA, USA, Apr. 2009.

[51] A. Levi and Çetin Kaya Koç. Inside risks: Risks in email security. *Commun. ACM*, 44(8):112, 2001.

[52] E. Lieberman and R. C. Miller. Facemail: showing faces of recipients to prevent misdirected email. In *symposium on Usable privacy and security (SOUPS'07)*, pages 122–131, Pittsburgh, PA, USA, 2007.

[53] J. Linn. Privacy enhancement for Internet electronic mail: Part I: Message encipherment and authentication procedures, February 1987. RFC 989.

[54] W. Lou. Spread: Enhancing data confidentiality in mobile ad hoc networks. In *Proceedings IEEE INFOCOM*, pages 2404–2413, 2004.

[55] M. Wong and W. Schlitt. Sender Policy Framework (SPF) for authorizing use of domains in e-mail, version 1, April 2006. RFC 4408.

[56] Mozdev.org. Enigmail: A simple interface for OpenPGP email security. Open-source project: http://enigmail.mozdev.org/home/index.php.html.

[57] Mozilla. Cross-platform component object model. https://developer.mozilla.org/en-US/docs/XUL/Tutorial/XPCOM_Interfaces.

[58] Mozilla. Mozilla's XML-based language for building user interfaces. https://developer.mozilla.org/en/docs/XUL.

[59] Mozilla. nsIObserverService. https://developer.mozilla.org/en-US/docs/XPCOM_Interface_Reference/nsIObserverService.

[60] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[61] J. Nielsen. Finding usability problems through heuristic evaluation. In *Conference on Human Factors in Computing Systems (CHI'92)*, pages 373–380, Monterey, CA, USA, 1992.

[62] J. Nielsen. *Usability engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[63] J. Nielsen and R. Molich. Heuristic evaluation of user interfaces. In *Conference on Human Factors in Computing Systems (CHI'90)*, pages 249–256, Seattle, WA, USA, 1990.

[64] J. Nielsen and V. L. Phillips. Estimating the relative usability of two interfaces: heuristic, formal, and empirical methods compared. In *Conference on Human Factors in Computing Systems (INTERACT'93 and CHI'93)*, pages 214–221, Amsterdam, The Netherlands, Apr. 1993.

[65] Openpgpjs.org. OpenPGP JavaScript implementation. Open source project. http://openpgpjs.org.

[66] Y. Oren and A. Wool. Perfect privacy for webmail with secret sharing., 2009. Technical report, School of Electrical Engineering, Tel-Aviv University, Israel, http://www.eng.tau.ac.il/~yos/spemail/.

[67] M. Parparita. Gmail greasemonkey scripts. https://github.com/mihaip/gmail-greasemonkey.

[68] S. Pichai. Chrome & Apps @ Google I/O: Your web, everywhere. Google official blog (June 28, 2012). http://googleblog.blogspot.co.at/2012/06/chrome-apps-google-io-your-web.html.

[69] E. Protalinski. Facebook announces two safety improvements with White House. News article (March 10, 2011). http://www.zdnet.com/blog/facebook/facebook-announces-two-safety-improvements-with-white-house/741.

[70] E. Protalinski. Facebook immune system checks 25 billion actions every day. News article (October 27, 2011 ). http://www.zdnet.com/blog/facebook/facebook-immune-system-checks-25-billion-actions-every-day/4895.

[71] E. Protalinski. How to spot a fake Facebook profile (infographic). News article (February 4, 2012). http://www.zdnet.com/blog/facebook/how-to-spot-a-fake-facebook-profile-infographic/8580.

[72] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification, 2004. RFC 3851.

[73] Sean. GMail-crypt. Open source project. https://prometheusx.net/introducing-gmail-crypt/.

[74] A. Shamir. Identity-based cryptosystems and signature schemes. In *International Cryptology Conference (CRYPTO'84) on Advances in cryptology*, pages 47–53, Santa Barbara, California, USA, 1985.

[75] S. Sheng, L. Broderick, C. A. Koranda, and J. J. Hyland. Why Johnny still can't encrypt: evaluating the usability of email encryption software. In *Symposium On Usable Privacy and Security (SOUPS'06)*, Pittsburgh, PA, USA, July 2006.

[76] Simson L. Garfinkel and Robert C. Miller. Johnny 2: A user test of key continuity management with S/MIME and Outlook Express. In *Symposium on Usable Privacy and Security (SOUPS'05)*, Pittsburgh, PA, USA, July 2005.

[77] Slate.com. FBI pursuing real-time Gmail spying powers as "top priority" for 2013. News article (March 26, 2013). `http://www.slate.com/blogs/future_tense/2013/03/26/andrew_weissmann_fbi_wants_real_time_gmail_dropbox_spying_power.html`.

[78] D. J. Solove. 'i've got nothing to hide' and other misunderstandings of privacy. *San Diego Law Review*, 44:745–772, July 2007. Available at SSRN: `http://ssrn.com/abstract=998565`.

[79] D. X. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *IEEE Symposium on Security and Privacy*, May 2000.

[80] Spemail. spemail, protecting privacy in webmail with secret sharing. Online document. `https://code.google.com/p/spemail/`.

[81] E. Stark, M. Hamburg, and D. Boneh. Symmetric cryptography in javascript. In *Annual Computer Security Applications Conference (ACSAC'09)*, Honolulu, HI, USA, 2009. `http://crypto.stanford.edu/sjcl/`.

[82] R. Stedman, K. Yoshida, and I. Goldberg. A user study of off-the-record messaging. In *Symposium On Usable Privacy and Security (SOUPS'08)*, Pittsburgh, PA, USA, July 2008.

[83] Trend Micro. Email encryption client, secure end-to-end email delivery. Online document. `http://www.trendmicro.co.uk/media/ds/email-encryption-client-datasheet-en.pdf`.

[84] P. van Oorschot. Message authentication by integrity with public corroboration. In *New Security Paradigms Workshop (NSPW'05)*, Lake Arrowhead, CA, USA, Sept. 2005.

[85] Voltage Security. Voltage SecureMail. Online document. `http://www.voltage.com/wp-content/uploads/Voltage_DS_SecureMail.pdf`.

[86] S. Whittaker, T. Matthews, J. Cerruti, H. Badenes, and J. Tang. Am I wasting my time organizing email? A study of email refinding. In *Conference on Human Factors in Computing Systems (CHI'11)*, Vancouver, Canada, May 2011.

[87] A. Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *USENIX Security Symposium*, Washington, D.C, USA, 1999.

[88] S. Wolchok, O. S. Hofmann, N. Heninge, E. W. Felten, J. A. Halderman, C. J. Rossbach, B. Waters, and E. Witchel. Defeating vanish with low-cost sybil attacks against large DHTs. In *NDSS'10*, San Diego, USA, 2010.

[89] F. L. Wong and F. Stajano. Multichannel security protocols. *IEEE Pervasive Computing*, 6(4):31–39, Oct. 2007.

# Appendices

# Appendix A

# Evaluation

To provide a more concrete understanding of tools and proposals discussed in Chapter 2 and a comparative analysis, we adapt the recently-proposed UDS (Usability, Deployability, Security) framework by Bonneau et al. [8]. We redefine each UDS features of the framework, to fit it into our case of email security. We opt for usability inspection methods of Nielsen's heuristic evaluations [62, 64, 61], mostly because this form of informal evaluation is useful in situations where conducting real user studies is time consuming and costly. We extend this evaluation to deployability and security aspects of secure email systems, based on the defined sets of heuristics, to form our UDS framework. Note that, the framework and comparison, is still a work in progress. It does not cover the evaluation and comparison of all previously mentioned tools and proposals. Also, as evaluation of each tool is a time consuming process which requires several careful tests, our evaluations might not be completely accurate. Additionally, some proposals do not have an available tool for test. In these cases, we perform analytical comparison instead.

## A.1 Evaluation Framework

We define three sets of features: usability, deployability and security. Our usability features are mostly based on five properties of security, presented by Whitten

et al. [87], and the design and user interface usability principals of Gutmann [43], Nielsen [62, 63] and Kainda et al. [47]. We define security features mostly based on email security threat model and some properties of Whitten et al. and Kainda et al. Our deployability features are more concerned about obtaining, installation, and configuration of the software, which are mostly based on the two properties of: *unmotivated property* and *barn door property* as stated by Whitten et al. Our usability features are more related to the user interaction and communication with the software, more specifically its user interface and the user's tasks. These features and their definitions are as follows.

### A.1.1 Usability Features

**U1** *Matching-Users'-Mental-Model*

Users' security-related tasks and the UI presenting those tasks should be easily conceivable, without requiring any knowledge or training about the security concepts. Thus, the user can intuitively find out what task should be done to achieve a specific goal.

Also, these tasks and their related UI, should not be complex to cause users abandon the scheme. Using familiar metaphors, matching users' mental model, to represent the tasks along with familiar words/expressions, greatly helps users to follow them.

As an example, based on the user test result by Whitten et al. [87], PGP 5.0 as a PKI-based approach, does not match users' mental model, as it is not obvious to all users that to send an encrypted email to another party, the public key of the recipient should be obtained in advance, or why they must have two keys, instead of one, to have their emails protected.

**U2** *Transparent-Security-Tasks*

As security of email is not the primary goal of email users, security tasks should be as transparent as possible, and users should not be exposed to details of

security tasks (e.g., like the use of SSL in browsers). Thus, most security-related tasks should be done automatically, without involving the user into the technical details.

**U3** *User-Control*

Email encryption may not be always necessary, convenient or possible. This feature has both usability and security effect and is considered as a both usability and security-related feature. In order to avoid making users completely abandon the scheme, or accidentally disclose their email content, users should decide when to secure their emails and should be informed about the success and failure of the security related tasks.

**U4** *Effortless-Key-Management*

The scheme should employ an appropriate mechanism to reduce key management efforts. Symmetric key encryption mechanisms and PGP, e.g., require initial key establishment using a secure channel like phone or meeting in person; therefore these schemes that do not offer this feature.

The scheme should also impose no burden on users to store, verify, and revoke keys. Schemes, not providing this benefit include: PKI-based ones, which have no satisfactory solution to manage private keys, or any other scheme that requires users to verify their keys manually.

We define a scheme as *Partial-Effortless-Key-Management*, in case that it fails to provide one or more of automatic key generation, distribution, storage, verification, and revocation.

**U5** *Easy-Recovery-From-Loss*

Users should be able to easily access their past encrypted emails, in case of key loss/disclosure. This can be achieved through automatic email or key backup through the scheme.

**U6** *Efficient-To-Use*

The security-related tasks, should be completed in an acceptable amount of time and effort. The number of button clicks or the time needed to complete the whole task, should be minimal. These tasks should also be reliable to avoid user frustration.

**U7** *Consistent-User-Experience*

The scheme should be well integrated with the existing email work flow. The scheme should not prevent users from performing their regular tasks e.g., replying, forwarding emails, multiple recipients or attachments. All required tasks should be performed through familiar existing email client rather than a particular user interface provided by the scheme or external website/service. Moreover, the scheme should be designed and deployed for most platforms (e.g., web, desktop, mobile client) supported by any email provider.

A scheme offering *Quasi-Consistent-User-Experience* benefit, preserves user experience of regular email work flow, but might not support some of the existing email system features, such as searching by content. Moreover its design is not restricted to a specific client or provider, yet has not been implemented for all clients.

**U8** *Effective-Feedback-To-User*

The scheme should notify users about its status, and the failure or the success of any action it takes on the user's behalf, using appropriate feedback in a timely fashion. Error feedback should also explain both the cause of the problem and possible solutions to users. Moreover, scheme status or error feedback should be presented to users using simple yet effective messages. The scheme should communicate with users through familiar words/expressions, including less technically-heavy terms. These expressions should be consistent across the whole system to avoid user confusion. Metaphors, symbols and visual cues can

also be used to make it easier for users to figure out the system states or error feedback.

### A.1.2 Deployability Features

**D1** *No-Server-Side-Changes*

The scheme should not expect any cooperation or support from email service providers. It should not require any changes to be made into email infrastructure, email providers' MUAs (Mail User Agents) and policies or implementation of new security systems by email providers. Email providers, especially free services, have no incentives to make any changes or implement new security mechanisms in their systems. The scheme also may not introduce additional server-side components or employ any server, as server maintenance requires some efforts while the scalability of these servers could be problematic which can affect the availability of the scheme.

**D2** *Portable*

Users are not limited to their primary computer and can use the scheme from any other machines. Schemes that require special software to be installed with administrative rights, or store keys or other sensitive information required in their work flow on users' machine, do not support mobility.

**D3** *Flexible*

The scheme should be designed and implemented in a way that can satisfy both technical and non-technical users. It should provide some options to more experienced users to be able to customize the scheme to fit into their needs. These options should also be configured to their safe default modes, to let inexperienced users utilize the scheme, without making any confusion or security issue for users.

**D4** *No-Prior-Setup-For-Users*

Any prior setup that causes burden or inconvenience to the sender or receiver, or act as an obstacle to email encryption, degrades the deployability. For example, secret keys or public keys that must be shared with the receiver, prevents the sender from sending encrypted emails before the sharing of keys is done. Users do not also need to install, configure, setup or register for additional software or services, to get the scheme to work.

**D5** *Free-And-Open-Sourced*

The details of the scheme and its source code are freely available to be used for any purposes or adapted by anyone. Its source code should also be well-documented, distributable and modifiable or derivable to be used in other projects.

**D6** *Available-Implementation*

The scheme has been implemented and used by people as an actual email security solution rather than a simple research prototype or proposal. The implementation should also be reliable and up-to-date.

### A.1.3 Security Features

**S1** *Confidentiality*

The scheme employs appropriate methods such as encryption or steganography to make the email entirely or partially (e.g., body, subject-line and attachments) confidential. This protection should also be end-to-end; i.e., it is done at the user client than the email server, gateway or a trusted third party, to remain encrypted in transit and deliver encrypted, directly to the user's email client. For example, a trusted third party who automatically generates and maintains keys while is in the possession of encrypted emails, has the ability to decrypt and read or even modify emails at any time without users' notice or consent. It should also avoid content leakage to the email provider, e.g., while being

written, or kept in user inbox.

A scheme rated as *Partial-Confidentiality*, does not provide end-to-end protection at the message level such as using SSL in webmail, opportunistic encryption through gateways, etc. We also rate schemes that rely on non-colluding third parties as *Quasi-Confidentiality*.

**S2** *Authentication*

There should be a possible and intuitive method to verify keys and authenticate an email sender and receiver mutually, to prevent attackers from impersonating the sender or receiver of an email.

**S3** *Content-Modification-Detection*

The scheme employs a method to verify email content integrity, with or without confidentiality.

**S4** *Non-Repudiation*

A scheme offers *Non-Repudiation* if there is a cryptographic proof to verify that an email is sent from the one who claims to be the sender. In another word, in the presence of such a proof, the sender cannot deny that a particular email has been sent by her. Non-repudiation is generally achieved by digital signatures, when, e.g., a certificate issued by a CA can be counted as a proof, as it binds the identity of the sender to the key.

**S5** *Critical-Error-Prevention*

The user interface should be carefully designed and implemented in a way that initially prevents users from making any dangerous errors leading to critical security exposures. This can be done by leaving no choice to make decision by users in critical conditions and making all critical processes automatic. It should not also rely on the direct user attention to the details of each task, to achieve the security goals of the system. Moreover, if any error occurs during

the process, an appropriate and non-confusing feedback, describing the current process status, should be presented to the user. A problematic case regarding this feature could be any scheme not informing users properly about encryption failure and sends it anyways with no feedback, or a scheme that decrypts an email automatically before reaching to the receiver's inbox, leaving the scheme vulnerable to the phishing attacks.

**S6** *Robust-Against-Secret-Key-Exposure*

Compromised secret keys, used for encryption, should have limited effects on the security of the system. In other words, disclosure of a key may limit the exposure of future or previous past email communications . For example, in the case of disclosure of a key with per-message keys, the confidentiality of all email content are protected, other than the one encrypted by the compromised key. This feature makes the process of interception of all emails difficult and costly for attackers.

## A.2 Evaluation and Comparison

In this section, we evaluate FriendlyMail and some previously discussed proposals/ tools in Chapter 2, using the UDS features presented in Section A.1. The summary of evaluations and comparisons are presented in Figure 3.

We modify the original UDS framework, by defining five levels of rating: *Full-benefit*, *Quasi-benefit*, *Partial-benefit*, *Not-Applicable-benefit* and *No-benefit*. A scheme is rated to provide *partial-benefit*, if it offers the feature partially, between *no-benefit* and *full-benefit*. *Quasi*, refers to the case that a scheme almost offers the feature, e.g., in case where no scheme can completely satisfy a particular feature. Finally, we rate a scheme as *Not-Applicable* (N/A) if a feature is not discussed, implemented or cannot be tested. As our user interface usability ratings are not based on usability studies on everyday email users, we rely on our own tests through the use of the tools (if available), and on our understanding of the schemes and the provided

specifications.

### A.2.1 UDS Evaluation of FriendlyMail

FriendlyMail apparently offers *Matching-Users'-Mental-Model*, *Transparent-Security-Tasks*, and *Efficient-To-Use*. The underlying concept of symmetric encryption is quite easy to understand compared to PKI. Steps to be followed by a FriendlyMail user includes: logging into OSN accounts, selecting the sender/receiver's OSN account from a provided list of OSN friends, and deciding when to send a secure email (different modes of operation, e.g., confidentiality vs. integrity-only). It also integrates the receiver's OSN photo and basic information, as an intuitive way of identity verification. All security-related tasks are represented via a minimal and intuitive user interface, implemented and explained through the use of visual cues and familiar expressions, while technical security tasks are completely transparent to users. Moreover, automatic encryption/decryption, integrity check and loading email content are reasonably fast and imperceptible, which would not frustrate the user; see Section 5.2. Providing users with different options for securing email and several feedback on the state of the performed tasks also satisfies the benefit of *User-Control*. It is *Key-Management-Effortless*; see Section 4.4 under "Automatic key management". Key revocation is not applicable. Keys/hash values are optionally and automatically backed up locally, in the case when a key/hash value is deleted from the OSN, satisfying *Easy-Recovery-From-Loss*. FriendlyMail secures regular email tasks e.g., reply, forward and multiple recipients. Also the transparency of technical tasks, minimal UI and user involvement, preserve the same user experience as with the regular email work flow. Yet, it provides *Quasi-Consistent-User-Experience*, as still no promising solution has been designed or implemented for email content searching, backup, etc. It also provides *Effective-Feedback-To-User*; see Section 5.2 under "Visual cues and security statements".

The deployment of FriendlyMail is fairly easy. It satisfies *No-Prior-Setup-For-User*, since it requires no prior setup or installation of additional software except

the plug-in. The sender can also send an encrypted email, even if the receiver does not have the shared key or the plug-in installed on his system in advance. However, FriendlyMail requires both the sender and receiver to be connected in a common OSN, we believe it does not prevent users from using our tool, as we are leveraging existing services rather than asking users to register for a completely new one. In addition, our integrity check mechanism does not require such existing OSN relationships. Moreover, no specific or complex configuration is needed for a normal user. It is *Flexible*, as it provides more technical users with an advanced option of using their own keys. It is *Free-And-Open-Sourced* and offers *Partial-Available-Implementation*, as it is still an academic prototype, but publicly available. FriendlyMail satisfies *Quasi-No-Server-Side-Changes*, as it depends on the OSN services, but does not need additional components. Current implementation of FriendlyMail through Facebook also limits its availability in countries where accessing Facebook is blocked. However, this limitation can be addressed to some extent by extending the implementation via several OSNs.

FriendlyMail offers security features of *Quasi-Confidentiality*, as in certain cases collusion attack cannot be avoided. Non repudiation is not a goal for FriendlyMail. Therefore, we rate it as *Not-Applicable-Non-Repudiation*. It offers *Content-Modification-Detection*, *Critical-Error-Prevention*. However, it provides *Quasi-authentication*, as it relies on the existing pre-authenticated connections among friends on Facebook. This still leaves FriendlyMail vulnerable to impersonation attack, due to the existence of fake accounts. Per-message key approach offers *Robust-To-Secret-Key-Exposure*. See Section 6.1 for details.

### A.2.2 UDS Evaluation of Other Proposals/Tools

1. **Enigmail**

   Enigmail uses OpenPGP, therefore is rated as not offering *Matching-Users'-Mental-Model*, *Transparent-Security-Tasks*; as it is already discussed in Section 2.1.1, the perception of the public/private key concept is difficult for users.

Trust in Web of Trust is another issue, which is not intuitive for average users, specially when it comes to decide about the trustworthiness of others, who are not in the user's close circle. Users should generate a pair of public/private key, a revocation certificate and a passphrase. Users also should manually store their revocation certificate and passphrase in a safe place and distribute their public keys and verify other users' public keys. Moreover, as most key management tasks are on users, it does not offer *Effortless-Key-Management*. It satisfies *User-Control*, as users can choose when to sign or encrypt their emails and will be notified about the failure or success of the task. It does not offer *Easy-Recovery-From-Loss*, as users should create a revocation certificate, store it in a secure place to be later used to revoke their public key, in case when their secret key is lost or compromised. Moreover, if the keys are lost, past encrypted email cannot be decrypted anymore. However, it is *Efficient-To-Use*; signing and encryption/decryption are completely automatic and users should just send their email as sending a regular email and pay attention to some security notifications. Although it keeps asking users to enter their secret key for several times to decrypt and verify an email, even if it is configured to remember the secret key for that session, which could be annoying to users. It also offers *Quasi-Consistent-User-Experience*; it is completely integrated into the UI and supports all the basic email tasks, except searching email content. Finally it provides *Effective-Feedback-To-User*, using different colors, several visual cues, along with security statements.

Enigmail is rated as *No-Server-Side-Changes*, as Enigmail by itself does not require additional servers, although for key distribution in large scale, key servers are largely used. It is not also *Portable*, as users should carry their private key from one machine to another. It is *Flexible*, as advanced options are presented to expert users and the default settings are all configured to the safe options. However, it does not offer *No-Prior-Setup-For-Users*, as users need to have PGP installed on their machine in addition to the Enigmail extension and also

obtain the recipient's public key before encrypting an email for that recipient. They also have to perform several initial configurations that may not be easy or convenient for all users, e.g., key generation may take few minutes and users should backup their key, create revocation certificates and passphrases, etc. It is *Free-And-Open-Sourced* and has *Available-Implementation*.

It provides *Confidentiality* using OpenPGP, *Authentication* through certificates and Web of Trust, *Content-Modification-Detection* and *Non-Repudiation* through digital signatures. Visual cues representing encryption may not be sufficient, as the icon enabling encryption is not quite noticeable, and no other cues such as color have been used, to make sure that users will notice when their email will be encrypted. Thus, it might make users to send a sensitive email without noticing that encryption is not enabled, and cannot be rated as *Critical-Error-Prevention*. Moreover, disclosure of the private key exposes all previously encrypted emails; however, timely key revocation prevents the disclosure of future encrypted emails. Thus, it is rated as *Partial-Robust-Against-Secret-Key-Exposure*.

2. **Waterhouse**

As the tool is not available for test, we rate the scheme's usability features based on their presented mock up and cognitive walkthrough methods [49, 50]. It is *Partial-Matching-Users'-Mental-Model*, by avoiding technical terms, and simplifying trust model. Trust is formed among pre-authenticated friend connections on Facebook and displaying Facebook pictures of sender/receiver acts as a proof of trust to users. Yet, it uses the same notion of public/private keys which is not familiar to average users. It provides *Partial-Transparent-Security-Tasks*, as all security tasks, e.g., signing, encryption/decryption and public key distribution are automatic and hidden from users, except private key storage and backup. Therefore it also gives *Partial-Effortless-Key-Management*, but not *Easy-Recovery-From-Loss*, same as PGP. It secures emails by default

among Facebook friends. However, users have the option to cancel any protection for their emails. Therefore, it gives *User-Control*. Since encryption and decryption are transparent and its user interface is minimal and tasks are completely integrated with email client, it is considered to be *Efficient-To-Use* and it is rated as *Quasi-Consistent-User-Experience*, mainly due to the searching email content issue. Also as we have not tested the tool we suppose they have secured other basic email tasks as well. It also gives *Effective-Feedback-To-Users* through several visual cues and security statements.

Same as FriendlyMail it is rated as *Quasi-No-Server-Changes* and *Portable*, as it depends on the availability of Facebook service and, users should carry their private keys to be able to switch between different machines. It is not rated as *Not-Applicable-Flexible*, as no customization is discussed. It does not offer *No-Prior-Setup-For-User*, as the tool should be installed on both sides to share public keys before having any secure communication. Its tool and source code are not available, not to the public or for test, making it non-*Free-And-Open-Sourced* and non-*Available-Implementation*.

From security perspective, it resembles PGP security to some extent. It provides *Confidentiality*, *Content-Modification-Detection* and *Non-Repudiation*, through encryption and signing using PGP. As FriendlyMail, their approach provides *Quasi-Authentication*, as it trusts users' pre-authenticated connections. Based on their own heuristic evaluation [49], again as tool is not available for the test, we rate it as *Critical-Error-Prevention*. It is *Partial-Robust-To-Secret-Key-Exposure*, as if a private key is compromised, it is revoked and new key pairs are generated and distributed to limit the damage. Although the process is not automatic, and it is on users to revoke their compromised key. This may lead to the exposure of larger number of encrypted emails, where the compromised key is not revoked immediately after disclosure.

3. **TrustSplit**

TrustSplit's simple design (using symmetric key), incorporation of different colors as visual cues, and familiar users' tasks including: an initial registration for an online service and entering a password for user authentication (see [31]) satisfies *Matching-Users'-Mental-Model*. It is rated as *Transparent-Security-Tasks*; it encrypts all emails by default (if possible), and asks users to decide whether to send it in plaintext or do not send it at all, in case where encryption is not possible. Thus, it offers *User-Control*. It provides *Partial-Effortless-Key-Management*, as encryption/decryption and key management are transparently handled. However, the scheme still requires users to create, remember or store an extra password of their CaaS account (which should also be different than their email account's password). It offers *Easy-Recovery-From-Loss* using their CaaS service. It is *Efficient-To-Use* and *Quasi-Consistent-User-Experience*, as it is well integrated into the email client UI and main security tasks are transparent; users should just enter their password few times (each session), click a button for sending secure emails and notice security indicators, while the whole encryption/decryption are done in a relatively acceptable time [30]. However, searching email content is not addressed. It offers *Effective-Feedback-To-Users* using red/green message borders to inform users about the success/failure of the tasks or locks as the symbol of a secure email.

It is not *No-Server-Side-Changes*, as it employs CaaS service. It is not *Flexible* but *Portable* assuming the user can memorize the CaaS password and use it from any machine. It does not offer *No-Priory-Setup-For-User*, as it requires the sender and receiver to register to the CaaS service before using the tool. It is *Free-And-Open-Sourced*, but not *Available-Implementation* for email client.

It provides *Quasi-Confidentiality* and *Content-Modification-Detection* of email body and attachments. It does not offer *Authentication*, as verification through email (Email-Based Authentication and Identification) is not authentication of real identity. As its tool is not available for testing (for email client), we rate it as *Non-Applicable-Critical-Error-Prevention*. Although if an email cannot

be encrypted, it asks user if they prefer to send it unencrypted, which may accidentally lead to disclosure of sensitive content by the user. It also satisfies *Robust-To-Secret-Key-Exposure*, because of using random keys per message. Finally, it does not offer *Non-Repudiation*, as it does not use digital signatures.

4. **Aquinas**

   Steganography and secret sharing concepts are quite simple. The security-related user's tasks, e.g., providing email accounts information and exchanging a secret are also somewhat intuitive. However, due to the requirement of multiple accounts, Aquinas offers *Partial-Matching-Users'-Mental-Model*. However, it is rated as non-*Transparent-Security-Task* and *Partial-Effortless-Key-Management*, as users should decide which covertext to use for steganography; the sender should choose and share a secret with the receiver through a secure channel and store it securely. These tasks are non-trivial specially for non-technical users. It offers *User-Control*, as encryption is not done automatically. It is non-*Easy-Recovery-From-Loss*, as there is no automatic key recovery. It is not *Efficient-To-Use*; users should do several additional tasks other that just composing an email; they should choose a covertext and input email accounts used for key and email transmission. The initial configuration is also non-trivial, if users want to use the scheme on different machines. It does not offer *Consistent-User-Experience*; it is not integrated into an existing email client. We also rate it as *Not-Applicable-Efficient-Feedback-To-User*.

   Aquinas is *No-Server-Side-Changes* and *Portable*; it is a standalone Java applet that can be accessed through the Internet and possibly be used on every machine. It is *Flexible* as users can set the number of email accounts they want to use to secure their email. Basic Aquinas does not offer *No-Prior-Setup-For-Users*; if bogus accounts are used, initial secret sharing is necessary. Also users should add their email accounts into a public database to make them available to the other users. It is *Free-And-Open-Sourced* and *Available-Implementation*.

It offers *Confidentiality* and *Content-Modification-Detection* using (Message Authentication Code) MAC. It does not offer *Authentication* as it is on users to authenticate each other, specially when they add their email accounts to a public database to make it available to other users, there is no way to authenticate and bind email addresses with users identity. It does not also offer *Non-Repudiation* as it does not employ digital signatures and the secret is shared between both sender and receiver, thus a particular email can be sent by both of them. It does not offer *Critical-Error-Prevention* as some security-related tasks rely on users' choice and knowledge, e.g., covertexts should be carefully chosen. It is *Robust-Against-Secret-Key-Exposure*; as per-message keys are used for encryption, and different subset of email accounts can be used for key or email shares transmission.

5. **SPEmail**

   The main motivations of this approach are eliminating the need for any encryption and key management through secret sharing. We rate SPEmail as *Partial-Matching-Users'-Mental-Model* and *Transparent-Security-Task*, mainly because the concept of secret sharing and hiding data through stegonagraphy is relatively intuitive; however it requires multiple accounts. No security tasks are also on users. There is no key management, thus it is both *Effortless-Key-Management* and *Easy-Recovery-From-Loss*. It satisfies *User-Control*; users can secure their email when it is desired and they are notified in case of failure. It is not considered as *Efficient-To-Use*, as all the steps of sending a secure email are automatically performed by the tool. However, the user should provide two email addresses and their corresponding passwords into the FROM field and two email addresses into the TO field. Moreover, the second email address of every recipient could not be trivially obtained or known. SPEmail is well integrated into the email UI; but does not support searching email content, attachments or multiple recipients. Thus, it offers *Partial-Consistent-User-Experience*. We

also rate it as *Not-Applicable-Efficient-Feedback-To-User*, as we could not test the tool.

It is *No-Server-Side-Changes* and *Portable*; but not *Flexible*; as no customization is provided. Its current implementation for Gmail, requires several initial setups, such as installing Greasemonkey Firefox addon, Java Virtual Machine (JVM), accessing a webmail account specifically dedicated to work with the tool. Therefore, it is not *No-Prior-Setup-For-User*. It is *Free-And-Open-Sourced*, while currently there is no reliable working *Available-Implementation*.

It offers *Quasi-Confidentiality* except for the attachments (although auto savings drafts does not seem to work properly), and it requires non-colluding email providers; but does not provide *Content-Modification-Detection*. However, authors suggest an approach, by sending a hashed random number, bound to each secured communication along with a timestamp, to provide integrity verification. It does not offer *Authentication* and *Non-Repudiation* as well. It is rated as *Not-Applicable-Critical-Error-Prevention*, as the tool is not available for test. Finally as no secret key is needed, it is considered as *Robust-To-Secret-Key-Exposure*.

| | | Usability | | | | | | | | Deployability | | | | | | Security | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Reference* | *U1:Matching-Users'-Mental-Model* | *U2:Transparent-Security-Tasks* | *U3:User-Control* | *U4:Effortless-Key-Management* | *U5:Easy-Recovery-From-Loss* | *U6:Efficient-To-Use* | *U7:Consistent-User-experience* | *U8:Effective-Feedback-To-User* | *D1:No-Server-Side-Changes* | *D2:Portable* | *D3:Flexible* | *D4:No-Prior-Setup-For-User* | *D5:Free-And-Open-Sourced* | *D6:Available-Implementation* | *S1:Confidentiality* | *S2:Authentication* | *S3:Content-Modification-Detection* | *S4:Non-Repudiation* | *S5:Critical-Error-Prevention* | *S6:Robust-To-Secret-Key-Exposure* |
| FriendlyMail | | ● | ● | ● | ● | ● | ● | ◐ | ● | ◐ | ● | ● | ● | ● | ○ | ◐ | ◐ | ● | – | ● | ● |
| Enigmail | [56] | | | ● | | | ● | ◐ | ● | ● | | ● | | ● | ● | ● | ● | ● | ● | | ○ |
| Waterhouse | [50] | ○ | ○ | ● | ○ | | ● | ◐ | ● | ◐ | | – | | | | ● | ◐ | ● | ● | ● | ○ |
| TrustSplit | [30, 31] | ● | ● | ● | ○ | ● | ● | ◐ | ● | | ● | | | ● | | ◐ | | ● | | – | ● |
| Aquinas | [10] | ○ | | ● | ○ | | | | – | ● | ● | ● | | ● | ● | ● | | ● | | | ● |
| SPEmail | [66, 80] | ○ | ● | ● | ● | ● | | ◐ | – | ● | ● | | | ● | | ◐ | | | | – | ● |

**Table 3:** UDS evaluation of schemes. Key: ● (offers the benefit); ◐ (almost offers the benefit); ○ (offers partial benefit); – (benefit is not applicable); blank (benefit is not offered). Note that, this table is an approximate analytical comparison. For more details, refer to the detailed discussion and analysis of each scheme, provided in Section A.2.