FriendlyMail: Confidential and Verified Emails among Friends*

Atieh Saberi Pirouz, Vladimir Rabotka, Mohammad Mannan
Concordia University, Montreal, Canada

Abstract

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 relationships between a sender and receiver on an online social networking (OSN) site. FriendlyMail can provide integrity, authentication and confidentiality guarantees for user-selected messages among OSN friends. A confidentiality-protected email is encrypted by a randomly-generated key, and the key and hash of the encrypted content are *privately* shared with the receiver via the OSN site. Our implementation consists of a Firefox addon and a Facebook app, and can secure the web-based Gmail service using Facebook as the OSN site; the addon is available at: `https://madiba.encs.concordia.ca/software/friendlymail/`. 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. In contrast to most other solutions, we limit our target user base to existing OSN users, to facilitate ease of adoption. In this paper, the focus of our discussion includes: the design, implementation and security analysis of the proposed solution. We acknowledge that a user study will be required to validate usability-related features of FriendlyMail. We are currently considering a comprehensive user study as separate future work; cf. past such studies of PGP (Whitten and Tygar, USENIX Security 1999), S/MIME (Garfinkel and Miller, SOUPS 2005).

1 Introduction

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 up 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. This incredible paradigm shift took place within the span of only a few decades. Apparently, with the ubiquitous adoption of electronic communications, century-old privacy expectations 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., [26]). **(b)** A wide-spread, common misconception among users is "I've got nothing to hide" [38]. 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 want to dig into their personal lives. So, even if some users understand that their emails are accessible to service providers, they do not deem it necessary to explore privacy-friendly alternatives. **(c)** The inadequacy of existing email security solutions, as analyzed in several past and recent studies [46, 19, 17, 27, 34].

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

*Version: March 14, 2014. Contact: `m.mannan@concordia.ca`

PGP has been available for over two decades now, and the proposed secure email standard S/MIME [35] has been implemented in several email clients, most emails are still sent unencrypted/unprotected. Other proposals also emerged, e.g., STEED [28], Waterhouse [29] and Aquinas [6]. Most solutions require a certain level of technical understanding for proper use (e.g., the idea of public key systems, certification); later work has identified several shortcomings when these tools are used by everyday people (e.g., [46, 19, 41]). The end result, so far, is that the adoption of these techniques remains consistently low.

We propose *FriendlyMail*, a secure email technique designed for everyday users, who are also connected via an online social networking (OSN) service, 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 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 securely (e.g., via an SSL-protected channel) 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. 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. See Fig. 1 for a brief overview.

Obviously, FriendlyMail requires the email and OSN sites to be non-colluding entities (e.g., Gmail and Google+ would be a self-evident bad choice). 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 [46]. Note that, the use of OSN sites for sharing long-term public keys is not new; see e.g., Waterhouse [29] (more details in Section 7). However, we leverage OSN sites as an instantly-accessible, authenticated channel for distributing per-email symmetric keys, largely avoiding the key distribution and key management issues of existing public-key based solutions.

In contrast to other solutions that target *all* email users, but are scarcely adopted in practice (if at all), we limit FriendlyMail to enable secure emails primarily among *friends*, more specifically, OSN friends. Our hope is that FriendlyMail may be more easily adopted due to this targeted user base and transparent key management (in contrast to PGP). FriendlyMail is designed and implemented considering web-based Gmail and Facebook services, to make the solution readily available to a large number of users. There are about 425 million Gmail users [20] (as of June 2012) and 1.19 billion monthly active Facebook users [14] (as of Sept. 30, 2013). However, FriendlyMail can be extended to other email clients/services, and be used with other OSN services.

In this paper, we report primarily on the design, implementation and security analysis of the proposed solution. We discuss challenges in implementing the seemingly straight-forward design into existing popular services. Specifically, we address issues related to multiple levels of OSN relationships, different use-cases of email (one-to-one vs. group emails), unavailability of APIs for Gmail's web interface, the seamless UI integration without exposing user content during email composition (e.g., how to deal with the *auto-save* feature), and risky OSN features that can threaten email privacy in general (e.g., Facebook's friend-search from email contact lists). The discussion of these challenges may help future efforts in designing and implementing practical secure email solutions. While a key step in validating FriendlyMail's adoption incentives and usable key management, a comprehensive user study is being considered as separate future work (cf. PGP [46], S/MIME [19, 17]).

Contributions.

1. Leverage existing user-practices. FriendlyMail takes advantage of existing user practices (i.e., the use of OSNs) 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. Transparent key exchange. 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 OSN friends with the receiver.
3. No server-side changes. No changes to the server-side of the social networking sites or email providers are needed. Users can immediately benefit from FriendlyMail.
4. Gradual adoption. FriendlyMail allows a sender to indicate which emails should have integrity and confidentiality protections. Default encryption of all emails would be more privacy-preserving; however, such a design may not work for many users (due to e.g., not using OSNs).
5. Support for indirect OSN connections. FriendlyMail can provide email integrity and origin authentication between parties with *weak* pre-existing OSN relationships (e.g., Facebook Like). Integrity protection can be enabled without any OSN relationships.
6. Implementation. To evaluate the feasibility of our design, we have implemented FriendlyMail as a Firefox addon and Facebook app, for web-based Gmail using Facebook as the OSN provider. Basic email features between two

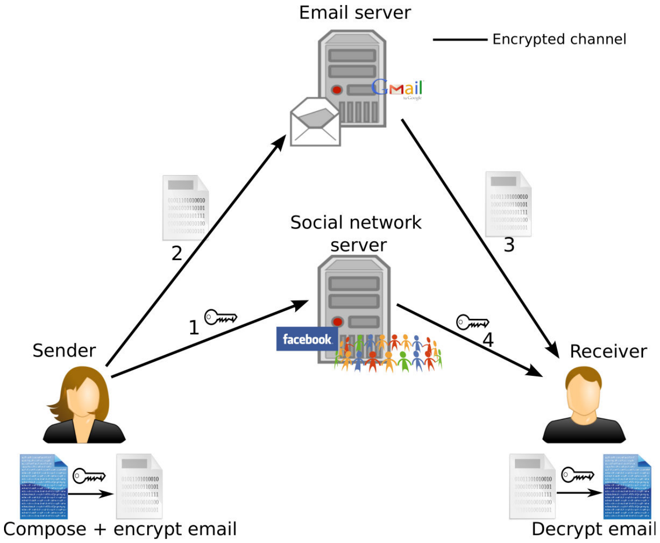


Figure 1: Simplified FriendlyMail steps: (1) a per-message randomly generated key is shared with the recipient via an OSN site; (2) the encrypted email message is sent via the regular email provider; (3) the recipient receives the encrypted email; and (4) the email content is decrypted by the per-message key retrieved from the OSN.

users, e.g., message compose, send, receive, reply, and forward have been implemented; multi-party/group email and attachment support are in-progress. We discuss several implementation issues that highlight complications and challenges of implementing privacy-protection mechanisms on top of real-world web-email services that are inherently designed not to support such efforts.

2 Threat Model and Operational Assumptions

Below we discuss FriendlyMail's trust relationships, threat model, and operational assumptions.

Trust relationships. 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; see Table 1.

Trust relations	Protections provided		
	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.

The trust relationships we assume include the following: (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* personal connections (e.g., Facebook friends-of-friends), where the sender and receiver are related via one or more direct acquaintances; (c) *impersonal* connections with web presences of known physical/online entities, e.g., users connected to a Facebook page (e.g., of a bank, entertainer) through the *Like* feature; and (d) *unconnected*.

We assume that the trust-relationship between a sender and receiver can be determined from their email applications; i.e., OSN connections (or lack thereof) can be queried from email applications. 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 use the same email address or full-name for the OSN account.

Assumptions on OSN and email providers. We use the OSN provider for sender origin authentication, and assume that OSN profiles and connections between users are largely genuine. OSN providers, e.g., Facebook actively tries to discover and remove fake profiles. However, such profiles are still a significant concern (see e.g., [3, 5]). Detecting fake and compromised accounts is also an active research area (e.g., [7, 12]). We also require that the OSN

ID_A, ID_B	Unique email addresses 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

providers can protect the confidentiality and integrity of user posts, e.g., to not expose privately posted keys/hashes to unauthorized parties (but see [10]; also see Frientegrity [16] for a recent proposal on achieving confidentiality and integrity guarantees of user posts on an untrusted OSN provider). Similarly, we assume email providers can protect user emails from unauthorized parties (but see e.g., [43, 1, 23] for recent Gmail security issues). At least, for the sake of their business reputation, email/OSN providers are apparently diligent in fixing known vulnerabilities and maintaining their site security. Note that PGP public key servers also do not always function as intended; known issues include: older version usage, key synchronization problems (see e.g., [8, 36]).

Non-collusion and legal issues. 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. Either of the service providers may cooperate with an adversary, but not both. Note that, for cloud storage/application services, defining legal boundaries may be tricky; see e.g., Hoboken et al. [24], for how U.S. laws (Patriot Act/FISA) can be used to access user data in EU countries. We also exclude recently revealed clandestine government surveillance programs that may force competing IT companies to share user information with government agencies, and efforts to weaken widely-used crypto systems (see e.g., [13, 22, 42]).

Channel and host security. Network connections between users and OSN/email servers must be protected (e.g., via SSL). The user-to-OSN channels must be secure 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. We assume a Dolev-Yao [9, 32] network adversary, with no control over end-user machines. Both the sender and receiver-end machines are assumed to be malware-free; otherwise, malware can simply expose or modify the email content while being composed or read.

Key deletion and other assumptions. To prevent perpetual access to an email's 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.[1] 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. [47]).

As a sender, the adversary may impersonate a friend, 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., SPF/DKIM). A user's email and OSN account credentials must not be compromised; we discuss consequences of such compromises in Section 5. Also, the recipient of a confidential email is trusted not to share/forward the content with unauthorized parties.

3 User Steps and Variants

In this section, 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 email to Bob. Parts of these steps are explained through our prototype for Gmail and Facebook (detailed in Section 4). The primary

[1] `http://www.facebook.com/help/356107851084108/`

FriendlyMail mode assumes a direct trust relationship and is described in Sections 3.1 and 3.2. In Section 3.3, we consider other trust relationships, and also discuss several variants that may address some limitations of the primary mode.

Design overview. FriendlyMail leverages symmetric key cryptography and OSN integration, to enable email confidentiality, origin and content integrity verification. The design is related to the well-established notion of using multiple channels to achieve security goals (e.g., [48, 31]). FriendlyMail employs OSN sites as an additional channel, mainly to automate key management and integrity verification. Secure emails are communicated using email providers as the main channel. Existing pre-authenticated connections among OSN users are leveraged to exchange secrets between email senders and receivers. OSNs also serve as secure channels for sharing hash values, used to provide email content integrity; the hash values are stored at known, integrity-protected locations.

3.1 Confidential Emails

Sending an encrypted email. We require 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 "Protecting emails during compose and read" in Section 4.1. Therefore, we need an additional button inside the email client's interface; see, e.g., *Secret Compose* in Fig. 2. The familiar compose window is displayed with some visual cues (e.g., a green border), indicating that a secure email is being composed; see Fig. 3. FriendlyMail will ask Alice to log into her OSN account, unless she is already logged in. Alice will be able to send an encrypted email to Bob, only if Bob is among Alice's OSN friends. Alice's OSN friends' list can be searched for Bob to check if they are friends, using e.g., ID_B or Bob's real name, and Bob's OSN profile must be verified by Alice for the first protected email. If Bob is not a friend, then Alice has no choice but to discard the email. 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.

When Alice indicates that she has finished the message composition (e.g., by hitting the *Send* button), the FriendlyMail addon performs the actions as listed in Protocol 1 (under "Sender-side actions").

We do not encrypt the email subject line (*Subject*), similar to PGP, see e.g., Symantec PGP Desktop.[2] Users may decide on opening an email based on its *Subject*, or later search emails using keywords from subject lines. However, we integrity-protect *Subject* and other selected header items.

Note that 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. The inclusion of N_m could be redundant based on the encryption mode used (e.g., N_m is not needed for CCM, as it uses non-repeated nonces).

The footer (Ftr) is added to all emails processed by FriendlyMail. It makes the receiver (Bob) aware of the use of FriendlyMail and provides a download link to the FriendlyMail addon. Recall that we do not require Bob to perform any setup steps prior to receiving FriendlyMail-protected emails. The start and end markers (Mrk_s and Mrk_e) are the same for each email, and allow the addon to detect and process encrypted content (example markers: "*Start of protected message*" and "*End of protected message*").

Decrypting a received email. Bob first receives the encrypted version of the email (i.e., C_{fm}); see step 3 in Fig. 1. If the FriendlyMail addon is not installed, Bob can choose to install it from the link provided in the footer. If the addon is installed, Bob is asked to log into his OSN account, unless he is already logged in. Next, the addon automatically verifies the email and retrieves the encryption key to decrypt the email content. To avoid privacy breaches in public places due to automatic decryption, the addon may be configured to ask for confirmation from Bob before decrypting the email. For verification, the addon performs the actions as listed in Protocol 1 (under "Receiver-side actions").

Replying to an 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. The same steps are then followed as for composing and sending an encrypted email. Alice can choose to encrypt her reply to an unencrypted email.

3.2 Integrity-protected Emails

Sending an email with verifiable origin and content integrity. Alice can enable origin and content verification e.g., through a checkbox placed next to the *Send* button (see Fig. 4 and 5 in Appendix A). When Alice indicates that she has completed the message composition (e.g., by hitting the *Send* button), the send event is caught, and

[2] http://www.symantec.com/docs/HOWTO41924

Protocol 1 FriendlyMail addon steps for encrypted emails

Sender-side actions:

1. Generate a nonce N_m and a symmetric key K_m; both parameters are specific to the current email.
2. Encrypt the plaintext email with the appended nonce using K_m. After adding the markers and footer, the message body is set to $C_{fm} = Mrk_s || E_{K_m}(C_m || N_m) || Mrk_e || Ftr$.
3. Use $C_{hm} = Subject || ID_B$ as selected header items for integrity protection. In the case of multiple recipients (discussed more under variant (g) in Section 3.3), ID_B would include all addresses from `To:` and `CC:` email header fields; Bcc'ed addresses are excluded as they are unavailable to recipients.
4. Securely publish the key K_m and hash value $H(C_{fm} || C_{hm})$ 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 Fig. 1.
5. Send the processed email content (C_{fm} with all header parameters) to Bob via the email service provider; see step 2 in Fig. 1. The user-to-OSN/email channels must be protected (e.g., via SSL), as explained under "Channel and host security" in Section 2.

Receiver-side actions:

1. Compute the hash $H(C_{fm} || C_{hm})$ from the received encrypted email.
2. Search for the hash value on Alice's OSN account. If a matching hash value is found, the message is decrypted by K_m as posted under the hash value (step 4 in Fig. 1). A successful decryption verifies Alice's identity and the integrity of the received email (due to the use of authenticated encryption).
3. 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 UI.

Alice is asked to log into her OSN account (if not already logged in). FriendlyMail 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 only interested in providing integrity protection for the email content.

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 5, item (e). The hash value is shared on Alice's OSN account (e.g., Facebook wall), accessible only to Bob. The email content (C_{fm}) remains unencrypted and is sent to Bob through regular email services.

Verifying an email's origin and content integrity. 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). Bob is then asked to log into his OSN account, if not already logged in. The addon then searches Alice's OSN account for: $H(C_{fm} || C_{hm})$; if found, the message content is verified. The message origin (i.e., Alice's identity) is also verified when Alice is identified as Bob's OSN friend. The verification result is presented to Bob through the email UI.

3.3 Variants

We outline several variants below. We have also implemented variants (a) and (b).

(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 Section 5, item (a)). As a mitigation action, advanced users can generate their own encryption keys (or keys derived from high-entropy passwords) and share them via a secure channel, such as in-person meeting or over the phone. A fixed email encryption key for a given sender-receiver pair is then generated from the shared key/password; the encryption key is used for all confidential emails between this pair. However, such key generation, storage and sharing will require active user involvement (and thus degrade usability), at least during the key setup for each pair of users. This variant may be feasible only for few selected contacts.

(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. 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; 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, so that receivers can easily locate the Page. However, the Like relationship is still checked for verification.

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. [44]). In contrast, FriendlyMail automates the verification process, and relies on the existing trust relationship (albeit weak) as established through the Like feature.

(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 browser 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. OSNs provide 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.[3]

(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, if all emails are encrypted by a single key. However, no such feature is needed, if the email is encrypted with a separate key for each recipient, requiring the sender to generate multiple keys and perform multiple encryption operations. For integrity-only protection, when all recipients are not OSN friends of the sender, the hash can be posted publicly.

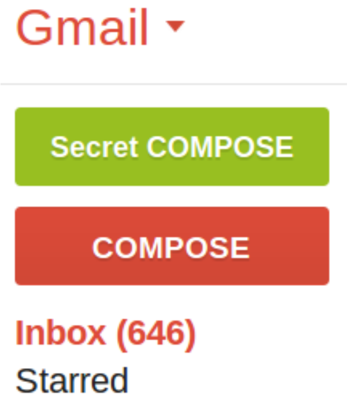
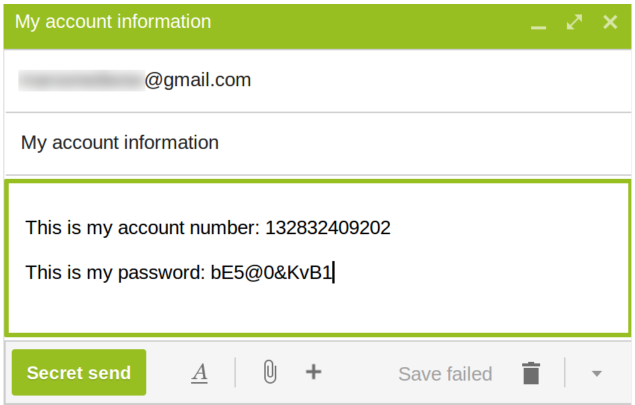


Figure 2: The Secret Compose button



[redacted]@gmail.com

My account information

This is my account number: 132832409202

This is my password: bE5@0&KvB1

Figure 3: Composing a confidential email

[3] https://www.facebook.com/help/220336891328465

4 Implementation

Our implementation for the desktop environment includes a Firefox addon and a Facebook app. To benefit a large number of email users, the prototype is built for the web-based Gmail application, using Facebook as the OSN service. Changes to the Gmail UI are made using the browser addon. 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 (e.g., search by email ID).

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 code, which was subject to few substantial changes during our development and testing over the last several months.

4.1 Firefox addon

Most of FriendlyMail's functionality is provided through a Firefox addon. For cryptographic support (e.g., encryption, hash computation, random key generation), we use the Stanford Javascript Crypto Library (SJCL [40]). We use the latest version of SJCL (Nov. 10 2013 and later commits) to avoid low entropy issues in the SJCL random number generator.[4] 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 [11], and for hash computation, we use SHA-256. The addon supports both plaintext and HTML content in Gmail. A FriendlyMail email has the same format as a regular email, with an additional footer and a link to the addon page. An encrypted email also contains start and end markers that allow the addon to detect and process encrypted content. To convert the binary output of encryption, we use the Base64 encoding function. The output from SHA-256 is formatted into hex before being posted to OSN accounts.

User interface changes. To make its 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. The only addition to the main page of Gmail's UI is a *Secret Compose* button (see Fig. 2). To stand out from the regular compose UI, we make minor changes to the interface, such as adding a green border and replacing the regular *Send* button with *Secure Send*. When the Secure Send button is pressed, we perform all steps needed to secure the email (e.g., encryption and key transmission); we then simulate a click on the actual Send button to trigger Gmail's event listener to send off the encrypted message. For emails created through the regular *Compose* button, integrity verification is enabled through a checkbox next to the Send button (see Fig. 4 in Appendix A).

At the receiver's end, the addon detects a protected email by checking the footer; encrypted blocks are identified by the start and end markers. The addon then automatically verifies, decrypts and displays the plaintext content. The sender's picture and OSN URL are fetched from the OSN account and displayed inside Gmail's UI. Different colors are used to clearly present the status or outcome of FriendlyMail functions. A green border is used to signal a successful FriendlyMail operation (e.g., composition, decryption, and integrity verification); a red border indicates an unsuccessful FriendlyMail outcome (e.g., failed decryption or integrity check); and a yellow border is used when a task requires the user's attention or intervention. 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.

Protecting emails during compose and read. 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. 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 disable draft savings and spell checks by using XPCOM's nsIObserverService[5] to intercept and discard all HTTP requests 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. However, this approach has two limitations. First, users cannot switch between protected and unprotected modes during composition, as otherwise protected content may become accessible to Gmail in the unprotected mode. Therefore, users must decide before composition if an email will be confidential or not. Second,

[4] `https://github.com/bitwiseshiftleft/sjcl/issues/77`

[5] `https://developer.mozilla.org/en-US/docs/XPCOM_Interface_Reference/nsIObserverService`

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. Thus, the HTML email being displayed at the receiver side may be slightly different from the original one, and may produce a different hash value. To overcome this issue, we strip problematic HTML tags, attributes and white spaces at both the sender and 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. Alternatively, the unmodified content could be fetched through Gmail's "Show original" feature.

4.2 Facebook App

To access Facebook's functionality, the Firefox addon uses a simple Facebook app that we developed. When the addon is launched for the first time, the app is also installed (with user permission). We use Facebook Login[6] to authenticate the sending/receiving FriendlyMail users. The familiar login page assures users that they are providing credentials directly to Facebook. An access token is received after a successful login; the token provides secure access to Facebook APIs (albeit time-limited). The token is used to publish/fetch keys and hash values, as feeds to the user's Facebook wall by using the Graph API. The app uses 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 5, item (j).

Optimizing key and hash searches. After calculating the hash of a received email, 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 become 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. This optimization also restricts replays of old emails from a compromised email account; see Section 5, item (f).

4.3 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. Facebook currently does not support searching for email addresses from apps. We gradually build a mapping between email and OSN IDs, and store this mapping locally within the browser. 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 a list of all matching profiles, 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. After Alice's selection, Bob's email address and OSN ID are locally stored and used for future FriendlyMail exchanges. However, the addon verifies if a locally stored Facebook ID is currently a friend of the sender, before publishing keys/hashes.

For variant (a) in Section 3.3 , we allow users to set pair-wise encryption keys that are derived from pre-shared passwords. On each user-end (sender/receiver), we store these passwords on the addon's local storage.

5 Attack Analysis

We consider different attacks on the FriendlyMail proposal under the threat model outlined in Section 2.

(a) 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). 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. [6]). 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 (a Russian language OSN: `vk.com`). However, we believe that such a design can only be useful for

[6] For Login and other Facebook APIs, see: `http://developers.facebook.com/docs/`

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). The shared passwords can be stored within the FriendlyMail addon, which may allow the use of strong passwords. However, sharing of passwords is likely to be a large usability deterrent.

(b) 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.

(c) 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).

(d) OSN notifications. OSN providers might send email notifications for events such as a new post, e.g., Facebook notifications.[7] 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.

(e) 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.

(f) Compromised email accounts. If a sender's email account is compromised, but not the OSN account, several attacks can be considered (besides item (c)). 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 (e). The attacker can also resend a captured FriendlyMail-processed email without even compromising the sender's email account; the email can be sent 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.

(g) Risks from friend finders. Facebook and other OSN sites commonly ask users to log into their email accounts to send automatic invitations to email contacts. Facebook may also ask for a user's email password when resetting her Facebook password, if the user has a Gmail, Hotmail or Yahoo! account listed on her Facebook profile.[8] Even if OSN sites claim that they would not access anything beyond email contacts, or would not store the email password, sharing email passwords must be avoided. Otherwise, confidential emails protected by FriendlyMail may be compromised by OSN sites; users may be warned about this risk by the addon.

(h) 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 (b). Note that, only the OSN provider may learn who the recipients of a protected email are; the sender's OSN friends/non-friends cannot know the receivers' identities.

[7] https://www.facebook.com/help/notifications

[8] http://www.facebook.com/help/136465059824353/

(i) 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.

(j) 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.[9]

(k) Rogue addons and email content. FriendlyMail appends a descriptive footer, including a link to the Firefox addon installation URL (currently hosted on our server, but will be offered through Mozilla addons site in the future). An attacker could craft a FriendlyMail footer (e.g.,"This email has been encrypted with a newer version of FriendlyMail. Click here to update."), which may mislead a victim into installing a malicious version of the addon (e.g., by transparently sending copies of plaintext email to the attacker). Like installing any other software, including Firefox addons and Facebook apps, users must be careful with software sources they trust (albeit difficult for everyday users). Firefox also warns users when they attempt to install addons from unknown sources. Also, FriendlyMail is content-agnostic, i.e., we do not check for malicious email content. Only the confidentiality and integrity of the content, and origin authentication of the sender can be expected from FriendlyMail.

6 Discussion on Usability and Deployment

Below we provide an analytical usability and deployment analysis of FriendlyMail. We have tested the current implementation within our group (8-10 people), mainly to check functionality and UI issues. The addon worked as expected. However, no formal user studies have been performed yet. Usability issues discussed below could serve as a starting point for future user testing. FriendlyMail has also been evaluated and compared with existing solutions using an adapted UDS (usability, deployability, security) framework [4]; the reference has been omitted to meet the anonymity requirement, which can be requested through the Program Chairs.

Deployment issues with email and OSN providers.
(a) Email providers lose access to confidential messages, and thus, cannot directly benefit from content-aware advertising. However, generic ads (or ads based on the subject line alone) can still be served. As only selected messages will be confidentiality-protected, revenues for ad-supported email services are unlikely to suffer.

(b) 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.

(c) OSN providers will also receive more search queries due to FriendlyMail searches for friends (and other supported relationships) and post messages. Queries are 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[10]); however, the number of posted messages from FriendlyMail and regular OSN usage would likely grow over time (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.

Usability issues.
(a) 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.

[9] Cf. Statement from an email security service (`https://www.enlocked.com/Works.html`, accessed: Dec. 7, 2013): "...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."

[10] `https://www.facebook.com/notes/facebook-data-team/anatomy-of-facebook/10150388519243859`

(b) 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.

(c) 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).

(d) 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., [45]). However, keyword searches within server-stored encrypted emails is not supported for now (but cf. [39]). Searches on locally-stored decrypted emails can be easily supported (e.g., by saving the decrypted versions, or the keys).

(e) Users must be comfortable with installing a Firefox addon and a Facebook app as required by our current implementation. Additionally, to prevent certain attacks (e.g., items (c), (d) in Section 5) users must configure Facebook account options (e.g., change the password recovery option from email to SMS). Our solution targets users who already have an email and OSN accounts, and use these on a regular basis. Therefore, we believe that such users already possess the level of technical expertise required by FriendlyMail. Automatic discovery of the recipient's OSN account requires that the same email account is used on the OSN. Otherwise, the user is asked to perform a (one-time) manual account mapping, similar to the existing practice of sending/accepting an OSN friend request.

7 Related Work and Comparison

Numerous proposals for email integrity and confidentiality have been put forward. Below, we discuss a few representative schemes, but exclude enterprise solutions as they are unsuitable for zero-cost mass deployment.

Waterhouse [29] proposes to use OSN sites such as Facebook to distribute long-term public-keys, by posting them on OSN profiles and leveraging existing OSN connections between users. For intuitive identity verification, Waterhouse suggests displaying a sender's OSN photo with each email (cf. Facemail [30]). As the sender's photo alone cannot guarantee trustworthiness of the OSN profile owner, the use of a web-of-trust model is also suggested; only public keys of senders with at least n friends in common with the receiver are accepted. The burden of managing private keys remains on end users. No implementation of this proposal is available (as far as we are aware of).

Stream [18] is a POP and SMTP proxy that sits between the email client and server, automatically encrypting emails when the receiver's public key is available (opportunistic encryption). If no key pairs are found, public/private keys are generated and stored on the fly; the sender's email address is used as an index to locate the associated key pair in the key database. 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. Stream eases the burden of key management but requires users to trust its proxy servers with private keys; such a trust model is particularly unsuitable for webmail providers.

To reduce user involvement, *STEED* [28] proposes significant changes to email providers and Mail User Agents (MUAs). MUAs would automatically generate (public, private) key pairs, or self-signed certificates, each time a new email account is created, and perform opportunistic encryption. Key distribution is done through the DNS server of an email provider, and a trust-upon-first-contact model is proposed, similar to ssh trust-upon-first-use. Users are still responsible for managing their private keys.

Aquinas [6] 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 of exchanged secret messages through steganography, and message integrity using MAC. Keys and encrypted messages are split, and transmitted separately through competitor email providers. A malicious ISP may collect all key/message shares and retrieve the message, when user to email channels are not SSL-protected. To restrict this attack, bogus message shares are also communicated; 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 channel (e.g., phone).

TrustSplit [15] proposes the confidentiality as a service (CaaS) paradigm, and splits the trust between a cloud provider (e.g., Gmail, Dropbox) and the CaaS provider(s). To protect user data, multiple layers of commutative encryption are used, which can be added/removed from user data in an arbitrary order. TrustSplit requires third parties to run CaaS servers; users must also register with these services.

SPEmail [33] uses secret sharing and linguistic steganography to provide confidentiality for webmails. Each message is divided into two shares. After encoding them through a form of text steganography, secret shares are delivered via two different webmail providers. No sender authentication or message integrity can be provided.

Ion et al. [25] propose a system to protect user content from online sharing platforms (e.g., Facebook, Gmail) and unauthorized users through the use of user-level (public, private) key-pairs. The encrypted content is publicly posted to a storage service (e.g., Dropbox), some dummy content is posted on the sharing site, and a hashmap directory (e.g., TinyURL) is used to store the mapping between the dummy and encrypted content. Users must first exchange and verify their public keys, and run a key establishment protocol.

Adams et al. [2] combine social authentication and honest but curios cloud services for private location sharing. The proposed secure location service (SLS) enables users to share location details only with selected friends, without revealing location information to infrastructure providers. An asymmetric key exchange between users is achieved through device pairing (e.g., location-limited visual channels).

Brief comparison. We do not require any new user-level secrets; our key management is completely transparent to users. We use the familiar OSN trust relationships, and avoid (largely-failed) past trust models, e.g., certificate/web-of-trust (as used in e.g., S/MIME and PGP, respectively). Compared to secret-sharing proposals that use multiple channels, we do not require users to create and distribute multiple email accounts to transfer shares. Our key transport leverages already-established OSN relationships. Also, the use of per-email encryption keys in FriendlyMail enables forward secrecy to some extent; e.g., there is no use of long-term encryption keys and the disclosure of an encryption key compromises only the email protected by that particular key.

Drawbacks include: FriendlyMail enables secure email communications mainly between OSN-connected users. The collusion between OSN and email providers could be real a concern. Also, users must explicitly select emails for protection.

8 Concluding Remarks

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 cloud-stored email data (see e.g., [37, 21]), 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. 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). We highly encourage readers to try out our addon at: `https://madiba.encs.concordia.ca/software/friendlymail/`.

References

[1] Acunetix.com. XSS vulnerability injected through Google analytics, executed in IOS's Gmail application. News article (Oct. 16, 2013). `http://www.acunetix.com/blog/web-security-zone/articles/xss-vulnerability-injected-google-analytics-executed-ioss-gmail-application/`.

[2] A. K. Adams, A. J. Lee, and P. S. Center. Combining social authentication and untrusted clouds for private location sharing. In *ACM SACMAT'13*, Amsterdam, Netherlands, June 2013.

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

[4] 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.

[5] 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.

[6] K. Butler, P. Traynor, W. Enck, J. Plasterr, and P. McDaniel. Privacy preserving web-based email. In *ICISS'06*, Kolkata, India, Dec. 2006.

[7] 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.

[8] Cryptome.org. Mining PGP key servers. Mailing list discussion (July 21, 2013). `http://cryptome.org/2013/07/mining-pgp-keyservers.htm`.

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

[10] 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`.

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

[12] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna. COMPA: Detecting compromised accounts on social networks. In *NDSS'13*, San Diego, CA, USA, 2013.

[13] Electronic Frontier Foundation. NSA spying on Americans. Collection of articles by EFF. `https://www.eff.org/nsa-spying`.

[14] Facebook.com. Facebook reports third quarter 2013 results. Online article (Oct. 30, 2013). `http://investor.fb.com/releasedetail.cfm?ReleaseID=802760`.

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

[16] A. J. Feldman, A. Blankstein, M. J. Freedman, and E. W. Felten. Social networking with Frientegrity: Privacy and integrity with an untrusted provider. In *USENIX Security Symposium*, Bellevue, WA, USA, Aug. 2012.

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

[18] S. L. Garfinkel. Enabling email confidentiality through the use of opportunistic encryption. In *Conference on Digital Government Research*, Boston, USA, 2003.

[19] S. L. Garfinkel and R. C. Miller. Johnny 2: A user test of key continuity management with S/MIME and Outlook Express. In *SOUPS'05*, Pittsburgh, PA, USA, July 2005.

[20] Google.com. 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`.

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

[22] M. Green. The many flaws of Dual_EC_DRBG. Blog article (Sept. 18, 2013). `http://blog.cryptographyengineering.com/2013/09/the-many-flaws-of-dualecdrbg.html`.

[23] E. Grosse. Security warnings for suspected state-sponsored attacks. Google online security blog (June 5, 2012). `http://googleonlinesecurity.blogspot.ca/2012/06/security-warnings-for-suspected-state.html`.

[24] 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`.

[25] I. Ion, F. Beato, S. Čapkun, B. Preneel, and M. Langheinrich. For some eyes only: Protecting online information sharing. In *ACM CODASPY'13*, San Antonio, TX, USA, Feb. 2013.

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

[27] A. Kapadia. A case (study) for usability in secure email communication. *IEEE Security & Privacy*, 5(2):80–84, 2007.

[28] W. Koch and M. Brinkmann. STEED – usable end-to-end encryption, Oct. 2011. `http://g10code.com/steed.html`.

[29] 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.

[30] E. Lieberman and R. C. Miller. Facemail: showing faces of recipients to prevent misdirected email. In *SOUPS'07*, Pittsburgh, PA, USA, July 2007.

[31] W. Lou. Spread: Enhancing data confidentiality in mobile ad hoc networks. In *IEEE INFOCOM*, 2004.

[32] 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.

[33] Y. Oren and A. Wool. Perfect privacy for webmail with secret sharing. Technical report, Feb. 2009. `http://www.eng.tau.ac.il/~yash/OrenWool-SPEmail.pdf`.

[34] P. Pennock. Trusting PGP. *USENIX ;login:*, 38(6):33–35, Dec. 2013.

[35] B. Ramsdell and S. Turner. Secure/multipurpose internet mail extensions (S/MIME) version 3.2 message specification, Jan. 2010. RFC 5751 (Standards Track).

[36] D. Ross. PGP: Public key servers. Online article (May 3, 2012). `http://www.rossde.com/PGP/pgp_keyserv.html`.

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

[38] D. J. Solove. ‘I’ve got nothing to hide’ and other misunderstandings of privacy. *San Diego Law Review*, 44:745–772, July 2007.

[39] D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, May 2000.

[40] E. Stark, M. Hamburg, and D. Boneh. Symmetric cryptography in javascript. In *ACSAC’09*, Honolulu, HI, USA, 2009. `http://crypto.stanford.edu/sjcl/`.

[41] R. Stedman, K. Yoshida, and I. Goldberg. A user study of off-the-record messaging. In *SOUPS’08*, Pittsburgh, PA, USA, July 2008.

[42] TheGuardian.com. Revealed: how US and UK spy agencies defeat internet privacy and security. News article (Sept. 6, 2013). `http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security/print`.

[43] TheHackerNews.com. Hacking Gmail accounts with password reset system vulnerability. News article (Nov. 22, 2013). `http://thehackernews.com/2013/11/hacking-Gmail-accounts-password-hack-vulnerability.html`.

[44] P. van Oorschot. Message authentication by integrity with public corroboration. In *NSPW’05*, Lake Arrowhead, CA, USA, Sept. 2005.

[45] S. Whittaker, T. Matthews, J. Cerruti, H. Badenes, and J. Tang. Am I wasting my time organizing email? a study of email refinding. In *CHI’11*, Vancouver, Canada, May 2011.

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

[47] 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, CA, USA, 2010.

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

A Additional Screenshots

Here we provide two additional screenshots of the FriendlyMail addon; see Section 4 for implementation details. These screenshots here may help understand the addon's integration with Gmail's web interface.

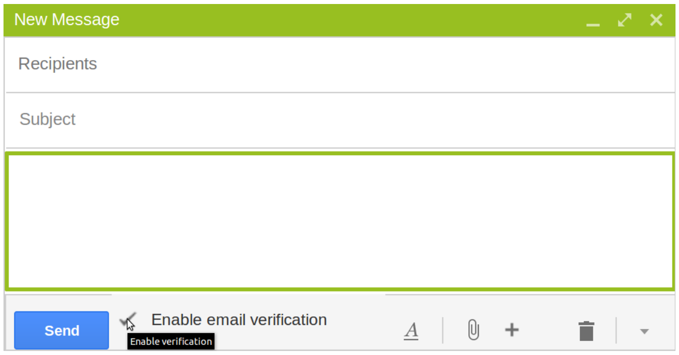


Figure 4: Compose window: email verification option

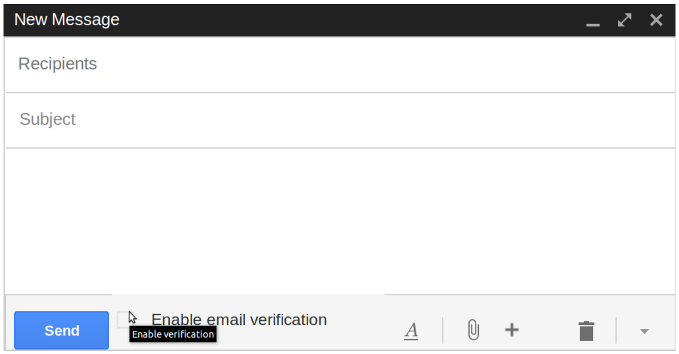


Figure 5: Compose window: email verification enabled (note the color change in the UI)