Abstract. Email is a core communication mechanism for society. Current email protocols, like SMTP and IMAP, do not provide fundamental security properties like confidentiality, integrity, and authenticity. Users can optionally employ public-key cryptography like PGP on top of email, but the steps for doing so are too complicated for the average user. This paper presents Security-Transparent Email with Automatically-managed Keys (STEAK), a backwards-compatible email system that offers stronger security guarantees than SMTP while retaining most of the usability benefits of webmail. Our main contribution is a key management protocol that performs key generation, distribution, and revocation securely and automatically. Like webmail, users can seamlessly access email from various devices.

For message exchange, unlike conventional email, STEAK uses a pull-based approach where senders host messages for receivers to download. We exploit this to implement an "unsend" feature, as well as economically disincentivize spam. Our prototype implementation and a qualitative usability evaluation show that STEAK requires less workflow changes than using PGP with email. Our preliminary performance evaluation shows that the system is responsive enough for typical use.

1 Introduction

Over the past decades, email has gained wide-spread adoption and is now a core communication service for society. However, some of its original design choices have had a profound impact on email information security and storage, making it a fundamentally insecure communication channel. In this paper, we revisit some of these choices and propose a backwards-compatible email service, called STEAK, that addresses this limitation in the context of webmail-like usability expectations.

Users send a wide variety of information in their email messages, including financial documents, healthcare records, legal documents, etc. Neither traditional email protocols, like SMTP and IMAP, nor contemporary webmail, provide adequate security guarantees. In a secure communication channel, when Alice sends Bob a message, only Bob can read it (message confidentiality), Bob can verify that the message he received was in fact sent by Alice (message authenticity), and that it contains the data she sent (message integrity). Email and webmail provide none of these properties out of the box.

At the same time, users have come to expect certain features on top of traditional email. These include automatic spam filtering, the ability to search and organize messages, and ubiquitous access to their email through webmail from a variety of user devices. We believe that providing stronger security guarantees cannot come at an expense of reduced usability. Our goal is to design an email service that provides these properties (called CIA guarantees in the rest of the paper), while providing the features and user experience of webmail. This is a non-trivial problem and current security approaches require running out-of-band security systems "on top" of email usually by leveraging public-key cryptography. These include S/MIME and PGP, as well as more recent ID-based encryption schemes [4].

The problem with out-of-band security approaches is that they require active involvement in key management. Users must generate, distribute, and revoke public keys with the out-of-band system, and carefully guard their private keys while remaining vigilant for compromises. We believe this is unreasonable because most users do not understand practical information security [7], and want the convenience of webmail despite the security problems it introduces. Even if they understood the security concerns, using public-key cryptography in this manner greatly increases the complexity of basic email tasks.

Our key insight is that email’s store-and-forward approach makes CIA guarantees hard to achieve. Because each email server stores and processes messages, a user must either trust the server completely to not to
Break CIA, or perform end-to-end authenticated encryption outside of the system. The former is unrealistic, but the latter requires users to set up and manage keys out-of-band. To address these challenges in this paper, we present a new email system called Security Transparent Email with Automatically-managed Keys (STEAK).

The contributions of STEAK are 1) an automatic key management system (called AutoKey) and 2) an email exchange protocol (called Secure Message Request Protocol, or SMRP) that allows users to access their email using a web browser on any device. STEAK enables access to email with only a username/password pair and a small amount of additional logic at the email client. To do so, STEAK leverages the user’s own personal cloud storage to host sealed account state, sealed keys, and sealed messages. The user devices (clients), not servers, process messages after decrypting them. The AutoKey system distributes public keys using more non-colluding key repositories than an adversary can compromise. Once keys are distributed, users fetch mail with SMRP by downloading them from the sender’s cloud storage. All the while, STEAK remains transparently backwards-compatible with SMTP, and offers more limited CIA guarantees to non-STEAK users that are still stronger than SMTP.

The remainder of this paper is organized as follows. In section 2, we define our usability requirements, and argue that our strategy is necessary to meet them for providing CIA guarantees. Then, we present the design of STEAK in section 3, with a focus on AutoKey and SMRP, and describe how basic email activities are performed. Afterward, we present our implementation strategies and prototype (section 4), and give a qualitative usability analysis and a preliminary performance evaluation (section 5). We finish with related work (Section 6) and a conclusion (section 7).

2 Motivation

Any secure communication protocol needs to provide three fundamental security guarantees of message confidentiality (only sender and receiver can read the message), message authenticity (receiver can verify that the message came from the correct sender), and message integrity (the message was not tampered with during communication). We call these CIA guarantees. Traditional email protocols like SMTP do not provide them.

CIA guarantees can be achieved by using an out-of-band security protocol like PGP on top of email. However, this significantly complicates the user experience and is not a practical approach for a majority of email users. Most email users are not tech-savvy and over the years they’ve become accustomed to the ease-of-use of web-based email services like Gmail [9]. Providing CIA guarantees while keeping the system as easy to use as webmail is a non-trivial technical problem.

In this paper, we focus on providing end-to-end CIA guarantees with a user experience as close as possible to that of webmail. Specifically, we focus on providing CIA guarantees on the message contents (subject, body, and attachments) and the account state only. Providing security guarantees for metadata, such as when the message was sent and received and what the senders’ and recipients’ addresses are, is the subject of future work.

We first describe our threat model and present possible adversaries that two webmail users Alice and Bob are likely to face. From the threat model, we derive the requirements on key management. We describe the usability requirements of Alice and Bob for various email tasks, and argue that to perform the required key management automatically and securely, a webmail-accessible email service must have at least three separate non-colluding services—one for processing state, one for hosting state, and one for hosting replicas of public keys and anti-tampering metadata.

2.1 Threat Model

Let Alice and Bob be two users that rely on webmail to communicate. They do not understand key management, but they never divulge their passwords. They each use multiple different user devices (endpoints) to access their accounts, and have fixed expectations on how to carry out common tasks in webmail (Table 1).

Let Eve and Mal be technically capable adversaries of Alice and Bob. Eve wants to break message confidentiality by reading as many messages as possible. Mal wants to break integrity and authenticity by altering or forging messages. However, they are computationally bound, they cannot compel users to divulge information, and they want to avoid getting detected.

We assume that it is more cost-effective for Eve and Mal to target email servers and communication channels than (potentially millions of) user devices i.e., attempting to compromise an email server that hosts information for $n$ users is more cost effective than trying to compromise those $n$ users individually. Also, from a practical point of view, most messages do not have much information of interest to Eve and Mal.

Eve’s main method of attack is eavesdropping. With one exception, she can read traffic on every network link and server, as well as every bit of information exchanged between users without being detected. She can compel any server or network link to disclose all of its information to her over any length of time, and can remember an arbitrary amount of data. The only bit of information she cannot observe is Alice’s password when she registers her account.
Additionally, we assume that Eve can make offline copies of any endpoint’s stable storage, but she cannot observe the state of the user devices while they are in use. In other words, we assume that Eve does not have access to decrypted private keys.

Unless otherwise noted, we assume that at any given time, Mal can either alter the behavior of servers, or alter packet flows in communication channels, but not both at once. We assume that if Mal compromises servers, she cannot compromise at least one of the STEAK servers Alice and Bob rely on. Similarly, if Mal compromises communication channels, she cannot alter the packets in at least one of the channels Alice and Bob use. Alice and Bob do not know which servers or channels are uncompromised. Further, Mal cannot cause endpoint devices to misbehave while they are in use.

We will show in subsequent sections that under this threat model, STEAK provides Alice and Bob message and account state CIA. Eve cannot feasibly learn the contents of their conversations, and Mal can only prevent the users from communicating.

2.2 Key Management and Usability

Even though we only care about CIA guarantees on messages and account state, the key management necessary to provide them, given our threat model, affects other necessary tasks like session management, account management, and contact management in addition to basic sending and receiving of email (Table 1). However, Alice expects to use any endpoint to access her email, so endpoints cannot preserve hard state across sessions. Thus, Alice must securely obtain not only Bob’s public key, but also her private key each time she logs in.

<table>
<thead>
<tr>
<th>Task</th>
<th>Key activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log in</td>
<td>Unseal private keys</td>
</tr>
<tr>
<td>Log out</td>
<td>Clear private keys</td>
</tr>
<tr>
<td>Create account</td>
<td>Create private keys, revocation cert; publish pubkeys</td>
</tr>
<tr>
<td></td>
<td>Publish revocation cert</td>
</tr>
<tr>
<td>Delete account</td>
<td>Reseal private keys</td>
</tr>
<tr>
<td>Change password</td>
<td>Regenerate private keys, or recover with security questions</td>
</tr>
<tr>
<td>Reset password</td>
<td></td>
</tr>
<tr>
<td>Add contact</td>
<td>Obtain and verify pubkey; watch for revocation</td>
</tr>
<tr>
<td>Write email</td>
<td>Encrypt and sign</td>
</tr>
<tr>
<td>Read email</td>
<td>Verify and decrypt</td>
</tr>
</tbody>
</table>

Table 1: Common webmail tasks and the requisite key management to perform to gain CIA.

At the same time, Alice and Bob do not know how to use or manage keys. They expect to submit their usernames and passwords to begin a session in their web browsers, and gain CIA guarantees automatically. They also expect to use any device and always see up-to-date information. This means the user endpoints must synchronize hard state across sessions, including the keys, account information, and messages, with one or more common always-on data repositories. Moreover, they must do so in a tamper-evident way to prevent Mal from silently altering state, and in a privacy-preserving way to prevent Eve from reading it.

If Alice and Bob communicate through a single server, Mal can trick their endpoints into learning the wrong public keys if she compromises the server or one of the links. If Alice and Bob are not aware of Mal, they can end up using the single server and fall for the attack. Exchanging public keys must involve leveraging more servers and links than Mal can compromise, and we require at least one source not be compromised. Meaning, there must be at least two sources for both public keys and anti-tampering metadata.

Because Eve can read any server and any network link, the endpoints must necessarily implement end-to-end state encryption between Alice and Bob. This prevents the (computationally bound) servers from processing messages, however, so the endpoints must work together to do so instead, constituting a separate message processing subservice within the email system. Thus we need at least three components in the design of our system: a trusted message processing component at the endpoints, and at least two sources for public keys and anti-tampering metadata repositories.

Using our strategy, we reduce the problem of giving Bob Alice’s public key to helping Bob’s endpoint discover which repositories can serve it. We address this by embedding the names of the repositories directly into Alice’s email address. We realize that embedding this extra information into the email address makes the email address harder to read/remember and we plan to address this issue in future work.

3 Design

As described in our motivation, there are two major aspects to STEAK’s design. The first is the subsystem for securely and automatically managing keys. We refer to this subsystem as AutoKey. The second is the protocol for sending and receiving emails, called Secure Message Request Protocol (SMRP). SMRP relies on AutoKey to ensure the sender and receiver have the appropriate keys in place before communicating.

3.1 Architecture

Our design assumes Alice and Bob both have sets of trusted endpoints that run loosely-synchronized clocks,
For hosting hard state across sessions, we leverage Alice and Bob’s personal cloud storage accounts. Additionally, we assume Alice and Bob’s endpoints can run code in execution and resource contexts outside of the web browser (such as in a VM).

There are four components to STEAK, beyond cloud storage and endpoint devices (Figure 1). The first is the STEAK endpoint code, which runs separately from the browser. It performs key management, session management, caching, cloud storage access, cryptographic operations, and UI proxying. The second is the webmail UI, which differs from a conventional webmail UI only in that it issues its RPCs to the locally-hosted endpoint code instead of a remote webmail server. These two components are intentionally separated for practical reasons, because at the time of this writing there is no way to perform cryptography inside a remotely-served webpage without exposing the user to a multitude of Javascript-based attacks, such as code injection and inter-tab information leakage.

The third is the STEAK server, which helps the endpoints discover when the user has new mail, assists with backwards compatibility. It also serves copies of users’ public keys, certificates, and storage signatures, which are mirrored by one or more metadata repositories (the fourth component). Metadata repositories are not required for correct execution since they host redundant information, but using one or more of these increases the number of servers Mal must compromise to trick Alice and Bob.

The Alice’s endpoint, STEAK server, and metadata repositories work together to implement AutoKey for her. Alice and Bob’s STEAK servers, cloud storage, and endpoints work together to implement SMRP. Figure 1 shows how this manifests in inter-component communications.

In AutoKey, each user is given an account public/private key pair, a storage public/private key pair, and one or more signing public/private key pairs. The private keys are stored encrypted with the user’s password, and only decrypted when the device is running a session. The storage and signing public keys are made available in the form of certificates signed by the account private key. The use for each key pair is listed in Table 2.

### 3.1.1 Storage

STEAK protects the contents of cloud storage from tampering by replicating cryptographic signatures. Each file uploaded to cloud storage is sealed with the user’s storage private key for confidentiality. STEAK organizes the files into a Merkle tree, such that each directory contains the signed hash of all of its children’s signed hashes. Each hash is sealed by the storage private key, making external tampering from Mal evident.

Writing data is similar to two-phase commit. A writer endpoint prepares to write by generating the new root signature, and replicating it to its STEAK server and all metadata repositories. If they all authenticate and accept it, the writer uploads the data to cloud storage, and then broadcasts a signed success message. The write completes when the STEAK server and metadata repositories acknowledge upon successful authentication.

Because reads and writes are serialized by the fact that the user does not use two endpoints simultaneously, the metadata repository only needs to store the last root signature it received. To avoid timing out partway through writing, the endpoint splits large writes into smaller ones and commits them separately.

On read, the endpoint verifies the integrity of cloud storage by fetching the root signatures from these servers and comparing them to the one in cloud storage. Because Mal cannot compromise all servers, and cannot compromise all channels, the reader will learn if Mal has tampered with a server if the signatures do not match. These read and write protocols are executed by AutoKey to store trusted public keys in a tamper-evident way.

Given the threat Mal poses to the system, STEAK employs a fail-fast approach to handling storage faults. This is because unavailability or inconsistencies discovered in publicly-hosted data (such as incorrect signatures) are assumed to be due to external interference from Mal. This is a reasonable trade-off, because it alerts users and cloud storage operators to Mal’s presence. In practice, STEAK servers are deployed in highly-available infrastructure to reduce the risk of transient faults leading to unavailabili-
<table>
<thead>
<tr>
<th>Key Pair Name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account key pair</td>
<td>Certifying and revoking other public keys.</td>
</tr>
<tr>
<td>Storage key pair</td>
<td>Sealing/unsealing and signing cloud-hosted data. Used for message confidentiality and state CIA.</td>
</tr>
<tr>
<td>Signing key pair</td>
<td>Signing email messages and attachments. Used for message integrity and authenticity.</td>
</tr>
</tbody>
</table>

Table 2: Names and usages for a user’s key pairs used in STEAK’s AutoKey subsystem.

Because only the user’s endpoints may write to the user’s cloud storage, STEAK is fundamentally a pull-based architecture. Instead of sending data from Alice’s endpoint to Bob’s email server, Alice will write messages into her cloud storage, and Bob will download them later using SMRP. This is a departure from SMTP, which we exploit to offer an “un-send mail” feature and to economically disincentivize spam mail by requiring the sender to pay for storage.

### 3.1.2 Usability

A user interacts with STEAK through the web browser, using the local endpoint as a web proxy. The UI is stored as files in cloud storage and are signed by the storage private key. This lets the endpoint verify their authenticity and integrity before serving them over the local trusted RPC channel, effectively preventing code injection attacks from Mal.

Before Alice or Bob can use STEAK, they must set up the endpoint code. However, this can be made easy in practice (see section 4). We assume in the following sections that the endpoints all run instances of the endpoint code in local VMs.

### 3.2 Bootstrapping Trust

Before Alice can use STEAK, she needs an account. In creating one, she bootstraps AutoKey by establishing her public key and registration date with each component, populating her cloud storage with initial application state, and granting her endpoint permission to access it. She also establishes security questions and answers to be used to recover her private key, should she forget her password.

To do so, her endpoint replicates her account public key and backs up her revocation certificate to more servers than Mal can compromise. We assume that Mal cannot alter the behavior of the servers during the registration process; if the servers do not behave correctly, the registration fails (Figure 2).

From a usability standpoint, creating an account on the STEAK server is similar to doing so on a webmail server. Alice begins by submitting a desired username and password to the STEAK server through the endpoint, as well as any number of hard-to-guess security questions and answers, and the identifier for an uncompromisable out-of-band channel to confirm the registration (e.g. a phone number or existing email address). The STEAK-specific extra requirements are the authentication tokens needed to access her cloud storage, and her additional metadata repositories.

Once the information is entered, the endpoint uploads the username and password to the STEAK server, which replies with a one-time-use registration URL via the out-of-band channel (we assume that Eve cannot determine the password in this step). Alice navigates to the URL to complete her registration. Once Alice confirms, the STEAK server remembers iterated salted hash of the password and out-of-band channel identifier, so Eve and Mal cannot easily learn either and so Alice can use the channel identifier later to reset her password.

Once Alice’s account is created on the STEAK server, the endpoint sets up her keys and certificates. It generates her public/private key pairs, and signs the storage...
and signing public keys with the account private key. It then generates revocation certificates for the account and storage keys, as well as two public certificates—one for the signing key, and one for the storage key—that contain Alice’s username, the public key, the timestamp, and a nonce for uniqueness (“pubkey certs” in Figure 2). It signs these certificate with the account private key.

It then proceeds to replicate the account public key and certificates to each server, using the STEAK server to vouch for her. To upload them to the STEAK server, it generates an HMAC over them using the password as the shared secret. Once the STEAK server validates and accepts them (remembering the nonce to prevent re-register attacks), the endpoint uploads them to each metadata repository. The endpoint authenticates to each using OpenID protocol and Alice’s password, where the STEAK server acts as the OpenID provider. The metadata repositories do not consider a revocation certificate to be in force until Alice activates it later.

The endpoint finishes by populating Alice’s cloud storage with STEAK account state, using the authentication tokens she supplied at the beginning of the process. It uploads her public key certificates as a world-readable files in cloud storage (for Bob to download later) as well as her revocation certificates and account public key (which are NOT encrypted). It encrypts the private keys and cloud storage credentials using an authenticated symmetric key scheme, deriving symmetric keys from her salted password. It also seals a copy of the storage private key with the answers to her security questions, so Alice can change her password without losing her data. The endpoint uploads the security questions and sealed keys to cloud storage, seals the MACs and salts from the encryption with the password, and erases them from RAM, completing the registration and AutoKey bootstrap process.

### 3.3 Session Management

Once AutoKey is bootstrapped, Alice can use multiple endpoints to access her account transparently. To do so, Alice simply enters her username and password, and the endpoint (via AutoKey) fetches her sealed storage private key, endpoint-specific sealed signing private key, and sealed cloud storage credentials. If the device has never been used before, the endpoint generates a new signing key certificate and revocation certificate for it, seals the private key with the password, and distributes the certificates to the STEAK server and metadata repositories (which authenticate them with the account public key they obtained on registration). The servers do not publish the revocation certificates until Alice signals them to do so.

Alice has the option to designate that the endpoint will be used for only one session (akin to a “This device is public” checkbox in webmail). If so, the signing key will have an expiration date in the very near future, e.g. 30 minutes.

When Alice logs out or her session times out, the endpoint has AutoKey erase any secrets from RAM. As a precaution, it also erases any data it cached data, and will erase cached data automatically if it restarts. Cached data is always sealed with one of Alice’s public keys before being written to local storage, preventing Eve from breaking confidentiality via offline analysis.

### 3.4 Key Distribution

In AutoKey, Bob’s endpoint must learn Alice’s account, storage, and signing public keys before he can communicate with her. Once it has them, AutoKey puts them into his cloud storage, executing the two-phase write protocol so his other endpoints can securely and automatically fetch them later. The challenges are in obtaining them for the first time in such a way that Eve and Mal cannot trick him into learning the wrong ones, and in revoking trust in them without also being tricked.

Bob’s endpoint will trust one of Alice’s public keys only if AutoKey can get unanimous agreement on the key’s value from each server that hosts them, and only if the key is not expired. Similarly, AutoKey will check for an activated revocation certificate if all servers agree on the same key, and the key is different from the currently trusted key. This strategy works under our threat model because without Alice’s password, Mal is not powerful enough to send the wrong key on every channel, or change it to every server. Bob waits for unanimous disagreement before checking for a revocation certificate to avoid Mal tricking him by activating it on one server.

While Bob trusts the keys, AutoKey periodically checks to see if it Alice’s servers have changed them. Because the account public key and storage public key do not change as often as the signing key (see the next section), it will check the account key at most once per session. However, it will check the storage and signing public keys every time Bob sends a message to Alice.

To advertise her keys, Alice’s STEAK address is a well-formed email address that indicates her username, her cloud storage provider, her STEAK server, and any metadata repositories that host her keys. We use the character “ˆ” to separate them—for example, Alice’s STEAK address might be aliceˆcloudstorage.comˆkeybackup.org@mail.net, indicating that her data is hosted on cloudstorage.com, her keys are replicated to keybackup.org, and her STEAK server is mail.net. Each service makes both the keys and certificates available via canonical URLs derived from the username and the service name. Bob’s endpoint has Au-
toKey use TLS-secured channels with authenticated cipher suites (i.e., AES-GCM) to fetch copies from each service, in order to hedge against external tampering from Mal (but not eavesdropping from Eve).

In the event that some of Alice’s services are unreachable, AutoKey continues to trust Bob’s copies of Alice’s public keys even if there is disagreement among the online servers. This is to prevent Mal from weakening the system with denial-of-service attacks. In practice, Alice’s services run on highly-available infrastructure, so unreachability problems are expected to be rare and transient.

### 3.5 Key Revocation

![The STEAK key revocation protocol.](image)

Inevitably, AutoKey will need to revoke Alice’s keys for her. When Alice removes access permission from a trusted endpoint device, AutoKey revokes its signing key. When she changes or resets her password, AutoKey revokes and regenerates her keys by default because the reason for doing so may be due to a password compromise.

Revoking a signing key is a matter of activating its stored revocation certificate (Figure 3). To do so, AutoKey first obtains it by downloading it from each of the servers, using a request signed by the account private key. It also gets a copy from cloud storage. At least one of the servers will reply with the correct revocation certificate, since Mal cannot compromise all of them under our threat model.

AutoKey then re-uploads it in a signed request to the servers, indicating that they should publish it. The servers erase the old public key certificate on successful verification, and AutoKey removes it from cloud storage. Then, when Bob contacts Alice next, his endpoint will detect the key discrepancy, discover the revocation certificate, and stop trusting the old signing key.

If Eve compromises Alice’s password, she will be able to read her incoming messages, and messages sent to others. If Mal compromises Alice’s password, she can do anything with the account. To regain control, Alice resets her password, effectively re-registering her account and re-bootstrapping AutoKey using registration protocol described earlier (but with two small differences described below). As before, we assume that Mal cannot alter the behavior of the servers during re-registration.

Unlike with first-time registration, the STEAK server requires Alice to submit the same out-of-band communication channel as she did before. Also, AutoKey sets a flag while uploading new public key information to ask for a reply containing the old revocation certificate. It also obtains the copy uploaded earlier from cloud storage, and then executes a variant of the signing key revocation protocol to publish the revocation certificates for her account and storage keys. The only differences are that the revocation certificates will be signed by the new account key, and that the servers will additionally verify the request with OpenID to ensure that it came from Alice (the only person who knows the new password).

When Alice logs in next, her endpoint “imports” her old account data by unsealing it with the old storage key and re-sealing it with the new storage key. If Alice remembers her old password, her endpoint decrypts the old storage key automatically. If she does not, her endpoint obtains the copy of the storage key sealed with the answers to her security questions, and prompts her for them to decrypt it.

Deleting an account is similar to revoking the keys. The only difference in the process is that no new keys are generated.

### 3.6 Sending, Receiving, and Un-Sending

Once Alice and Bob have exchanged public keys with AutoKey, they communicate with SMRP, described in this section. In SMRP, Alice sends Bob a message by sealing it with his storage public key, uploading it to her cloud storage, and making it world-readable. Bob’s endpoint downloads and decrypts it, sanitizes it, and serves it to Bob’s browser for him to read. The same principle applies to attachments, which in STEAK are sent as separate files to be downloaded. Alice makes a copy for herself in her “sent” folder, if desired.

If Bob expects email from Alice, he can search Alice’s cloud storage for new messages and fetch them. The UI allows him to query individual contacts for new messages.

The challenge for Bob is to discover when Alice sent him an unsolicited message. To address this in SMRP,
Bob’s STEAK server acts as a notification service for new email. Alice sends Bob’s STEAK server a message metadata record, sealed with his storage public key and signed with her signing private key, that identifies herself as the sender and Bob as the recipient. It serves as a hint to Bob that he has unsolicited mail, and tells him the message recipients, the subject, the human-readable names of the attachments, and the attachment signatures. This information is hidden from the STEAK server to provide end-to-end message CIA, and its small size makes it easy for it to store many of them.

When Bob accesses his email, his endpoint downloads and decrypts the unread message metadata records from the server. Each of them appear in Bob’s UI as unread messages. The UI tells Bob the subject, recipients, and attachment names. Once Bob’s endpoint reads and stores the metadata records, the STEAK server is free to erase them.

When Bob opens an unread message from Alice, his endpoint downloads, decrypts, and verifies the authenticity and integrity of Alice’s message body, using the public keys obtained by AutoKey. Once verified, Bob reads it via the UI. To download an attachment, the endpoint streams it from Alice’s cloud storage, verifies it, decrypts it, and delivers it to the UI as a file download. As a performance optimization (and to assist automatic spam filtering), the endpoint prefetches message bodies as they are discovered.

Mailing lists in SMRP are treated like user accounts. The only difference is that every participant on the list has a sealed copy of the account storage key and a personal signing key.

In SMRP, Alice can “un-send” a message by erasing the body and attachments from cloud storage. This allows her to withdraw messages sent in haste, or sent accidentally to unintended people. Bob will discover that she deleted it (i.e., he will receive a metadata record for a message that does not exist), but this may be preferable to having Bob read the entire message. As such, once Bob obtains a message’s body and attachments, his endpoint replicates them to his cloud storage by default to keep them available even if Alice deletes it later.

### 3.7 Searching, Tagging, and Filtering

Beyond sending and receiving email, webmail users expect automatic spam filtering, the ability to organize messages and conversations, and the ability to search them. In traditional webmail systems, this functionality resides on the server. The challenge in providing it in STEAK is in moving it to the endpoint to preserve confidentiality, but without compromising features.

Tagging messages is a matter of associating a given message with a bag of tags that Bob defines. Bob’s endpoint incrementally builds an index that pairs each tag to a list of message identifiers. To search messages by tag, the endpoint fetches a page of the tag’s index, looks up the messages, and serves them to the UI.

To search messages, the endpoint incrementally builds a search index as Bob reads them, using an off-the-shelf indexer. The indices are updated and synchronized with cloud storage in the background as he opens and reads messages. As a performance optimization, the endpoint lazily downloads the index by chunk they are needed by the indexer. It caches them for the session, and asynchronously replicates modifications.

To filter messages, the endpoint uses a combination of a locally-trained classifier and sender blacklists. Bob has the option of marking messages as spam and marking senders as spammers, in which case the endpoint feeds the message body into its local spam classifier and replicates the classifier’s new model parameters to cloud storage. This allows the classifier to be used across endpoints without revealing its parameters. If Bob marks a contact as a spammer, AutoKey revokes trust in its public key, and the endpoint will ignore future messages from it.

Using only a local spam filter is desirable for confidentiality. The spam classifier model parameters can leak information about the contents of the messages they have classified, so it is important that Bob’s endpoint not share this information directly. This is in contrast to webmail providers today, which learn to classify spam by monitoring many user accounts. However, empirical tests [5] suggest that existing local spam filters can achieve upwards of 98% effectiveness in practice, suggesting that our strategy should not significantly impact usability if the endpoint code ships with a pre-trained classifier.

### 3.8 Backwards Compatibility

Because email is already a widely-used service, STEAK must remain backwards compatible with it to allow Alice and Bob to communicate with Charles, a user who relies on conventional email. To do so, the STEAK server provides an SMTP/SMRP gateway that translates one protocol into the other. The STEAK server has its own cloud storage to hold messages from Charles to serve to Alice via SMRP. Alice’s messages to Charles are automatically routed through SMTP because Charles does not have a well-formed STEAK email address.

The gateway alone does not provide CIA guarantees. If Charles wants a limited degree of CIA, STEAK offers one of two options that require a minimal change in his workflow. First, if Charles uses PGP already, he will send his public key and encrypted messages via SMTP. AutoKey obtains Charles’ public key automatically. While this is less trustworthy than STEAK’s approach, it is better than nothing, because Mal has only one very short
window of time to alter the public key.

If Charles does not use PGP, he must first agree with Alice on a shared secret out-of-band, and assume that Mal does not compromise the STEAK server or the TLS CA server that vouches for its TLS public key. To send her a message securely, he navigates to her STEAK server, enters the secret, and is given a webmail form for composing her a message. When he submits his message, STEAK it available to Alice via SMRP.

When Alice replies via SMRP, the STEAK server sends Charles a URL instead of the message body. When opened, the STEAK server gives him a specially-crafted HTML page that contains Alice’s ciphertext, a decryption algorithm in Javascript, and a secret submission form. When Charles enters the secret, the page decrypts the message for him.

4 Implementation

The individual technologies needed to implement STEAK already exist. For storage, STEAK relies on Syndicate [11] to provide an abstraction layer over storage providers while implementing common consistency, integrity, authenticity, and authorization semantics. For running the endpoint code, STEAK relies on now-ubiquitous virtual machine technology to achieve portability and isolation from the browser and underlying OS.

The two usability challenges to overcome are setting up the endpoint VM and getting access to the user’s cloud storage. To address the former, we deliver endpoint code as an app that the user installs via a device-specific app store. Once installed, it does not have to be accessed directly, and the app store will keep it up-to-date. We believe users are already used to installing software from app stores, and thus we do not believe this extra step poses a significant usability barrier.

Our endpoint’s runtime requirements are flexible. While a full-blown virtual machine is sufficient, the endpoint can also run in an OS container, a user-mode Linux instance, or a Portable Native Client (PNaCl) [1] browser plugin. Our prototype supports all but the last option, which is under development.

We leverage Syndicate to make it easy to access cloud storage, and to simplify our storage layer’s implementation. To set up STEAK, the endpoint and Syndicate execute a protocol similar to OpenID whereby Alice authorizes STEAK to create and access a Syndicate volume. The metadata repository implementation employs a similar strategy.

Regarding deployments, a STEAK server is meant to serve an organization. Because it can read sender email addresses and source IP addresses, it employs techniques similar to those used in SMTP to rate-limit and black/whitelist external malicious users. For SMRP clients, the server additionally verifies the message metadata record signature before accepting the request, and blocks senders who do not have known public keys. The server deals with malicious traffic on the SMTP gateway using conventional techniques.

Regarding spam, since Bob can be expected to check his account frequently (once per day), it is unlikely that his message metadata records on his STEAK server will consume too much space. If space becomes a concern, the STEAK server can opt to remember only the unique set of senders, in which case Bob will scan their cloud storage accounts for the messages at the cost of latency.

To enhance spam filtering, we are investigating the feasibility of privacy-preserving data mining, to allow many users to train a shared spam classifier. One possible solution is a distributed privacy-preserving Naïve Bayes classifier [17].

Key plaque is a known problem with key servers. Metadata repositories prevent it by periodically querying the issuer’s STEAK server for the key, and automatically erasing certificates with built-in expiration dates.

4.1 Prototype

Our prototype is in an early alpha state, but is under active development. The endpoint and server are implemented as daemons in Python, with 6,000 and 1,000 lines of code respectively. The UI is implemented with Google Web Toolkit, and contains about 2,000 lines of Java. We use the PyCrypto package for cryptography, and rely on Syndicate’s Metadata Service as a metadata repository. We are in the process of integrating the logic for handling attachments, mailing lists, searching, and filtering.

The STEAK server and metadata repository will be compatible with multiple PaaS providers, to allow them to horizontally scale to support large organizations. Users will have the option to write public keys and certificates to one or more cryptocurrency blockchains, greatly increasing the number of servers Mal must compromise to change it.

5 Evaluation

Our usability goal has been to preserve the semantics of webmail while offering transparent end-to-end CIA. To evaluate this, we consider the steps and information required to complete common webmail tasks, and compare them to how STEAK and PGP perform them. These include setting up and destroying an account, adding and removing a contact, sending and receiving messages, and changing account passwords.

In evaluating against PGP for usability, we use the user’s guide for PGP 7.0 [15] as a baseline comparison.
We also consider approaches taken by Enigmail [6] and Mailpile [12], which seek to make PGP easier to use for securing email. We show that while STEAK does not offer exact webmail semantics, it offers semantics much closer to webmail than even user-friendly PGP implementations.

At the same time, our performance goal for STEAK is to remain responsive to user actions. Our preliminary evaluation shows that the added latencies our design imposes on synchronous storage operations (including reading and writing messages, keys, and contacts) are both negligible.

5.1 Endpoint Management

Assuming the web browser is already installed, setting up webmail takes no effort from the user. However, because performing encryption in server-chosen Javascript with no validation is unsafe, both PGP and STEAK require an additional software component to be installed on each endpoint. While this can be simplified with the endpoint device’s app store, a PGP user must also create a subkey for each device, and sign it with the master key.

<table>
<thead>
<tr>
<th>Webmail</th>
<th>STEAK and PGP</th>
</tr>
</thead>
<tbody>
<tr>
<td>(None)</td>
<td>1. Install endpoint software. 2. (PGP) generate and sign subkey.</td>
</tr>
</tbody>
</table>

Table 3: Steps to set up an endpoint.

Destroying an account in webmail and STEAK are equivalent—the user navigates to the page in the UI to do so, and enters the username and password. The rest is automated. In PGP, the email account must be destroyed separately from the key. The private keys must also be erased, and the public keys revoked.

5.2 Account Management

Registering an account in STEAK has a similar workflow to registering a webmail account; the only difference is that STEAK users must also submit cloud storage authentication tokens and additional metadata repositories (if desired).

By contrast, a PGP client requires the user to create an email account before configuring it, making the task at least as difficult as webmail. It additionally requires users to generate keys and revocation certificates, and export the public key (e.g. to a key server). While it is possible to integrate this into the account registration process (the approach taken by Mailpile), the user is inevitably involved in the key setup process because she will need to learn how to sign and trust other users’ keys.

<table>
<thead>
<tr>
<th>Webmail and STEAK</th>
<th>PGP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Navigate to provider URL. 2. Submit username, password, out-of-band channel (STEAK: also cloud storage creds and metadata repositories). 3. Activate account with information sent on out-of-band channel.</td>
<td>1. Create email account (steps 1-3 in webmail). 2. Configure PGP client to use the email account. 3. Generate and encrypt keys. 4. Generate revocation cert. 5. Publish pubkey.</td>
</tr>
</tbody>
</table>

Table 4: Steps to create an account

Changing or resetting a password in STEAK is a matter of either submitting the old password or answering the security questions, and waiting for the account state to be re-sealed. This is similar to webmail’s semantics. While some PGP implementations offer a key recovery option using security questions, it must be manually enabled. Moreover, the user must manually change the key password after recovering the key.

<table>
<thead>
<tr>
<th>Webmail and STEAK</th>
<th>PGP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Go to to reset form. 2. Answer security questions. 3. Obtain reset URL out-of-band. 4. Enter new password. 5. (STEAK) Wait for re-sealing.</td>
<td>1. Enable key recovery option. 2. Contact recovery server. 3. Answer security questions. 4. Change key password.</td>
</tr>
</tbody>
</table>

Table 5: Steps to reset a password.

Deleting an account in webmail is as simple as entering one’s password and confirming the request. This is also the case in STEAK, where all public keys are automatically revoked once the request is confirmed. In PGP, however, the user must not only delete the email account, but also any private keys associated with it. The user must erase the public keys from the key servers, and publish revocation certificates for the private keys.

5.3 Contact Management

Adding a contact in webmail is usually a matter of filling out a form, if it is not done automatically. At a minimum
it requires an email address and a name, but other fields are possible.

With PGP, the user must obtain, verify and trust a public key as well. Mailpile attempts to reduce this to scanning QR codes or obtaining them from email headers, but nevertheless the user must make a choice on the trustworthiness of the peers who signed it. Applying TOFU semantics makes it easier to use but less secure than STEAK, because there is only a single source for the key. Moreover, PGP users must periodically re-acquire keys when old ones expire.

<table>
<thead>
<tr>
<th>Webmail and STEAK</th>
<th>PGP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Open new contact.</td>
<td>1. Open new contact.</td>
</tr>
<tr>
<td>2. Add name, email.</td>
<td>2. Add name, email.</td>
</tr>
<tr>
<td>3. Save.</td>
<td>3. Obtain, verify, and trust pubkey.</td>
</tr>
<tr>
<td>4. Save.</td>
<td>4. Save.</td>
</tr>
<tr>
<td>5. Repeat 3-5.</td>
<td>5. Repeat 3-5.</td>
</tr>
</tbody>
</table>

Table 6: Steps to add a contact.

5.4 Accessing, Sending, Receiving

Accessing email from multiple devices is trivial in webmail if the device has a web browser. It is equivalently easy in STEAK and PGP once the endpoint code is installed and set up.

Sending a message in webmail is a matter of opening a window to compose it, selecting the recipients, and sending it. However, PGP implementations require the user to at least click through UI elements to encrypt and sign the message. This is because only the user knows whether or not the recipient will be capable of decrypting the ciphertext. STEAK’s transparent backwards compatibility lets users avoid this, akin to webmail.

<table>
<thead>
<tr>
<th>Webmail and STEAK</th>
<th>PGP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Open compose window.</td>
<td>1. Open compose window.</td>
</tr>
<tr>
<td>2. Select recipients.</td>
<td>2. Select recipients.</td>
</tr>
<tr>
<td>3. Type body, add attachments.</td>
<td>3. Type body, add attachments.</td>
</tr>
<tr>
<td>4. Send message.</td>
<td>4. Encrypt and sign.</td>
</tr>
<tr>
<td>5. Send message.</td>
<td>5. Send message.</td>
</tr>
</tbody>
</table>

Table 7: Sending a message.

Reading a message in webmail is a matter of selecting the unread message. This is also the case in STEAK-to-STEAK communication, and STEAK/SMTP communication where no CIA is needed. Even in STEAK/SMTP communication with limited CIA, submitting a shared password to access a message is not a foreign concept.

PGP implementations can decrypt messages automatically if the key is known. However, the user may be prompted to obtain and trust the key if not.

<table>
<thead>
<tr>
<th>Webmail and STEAK</th>
<th>PGP</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. (STEAK/SMTP): submit CIA secret.</td>
<td>2. If not trusted, obtain, verify, and trust pubkey.</td>
</tr>
</tbody>
</table>

Table 8: Reading a message.

5.5 Performance

Keeping latency low is critical for a smooth user experience. To do so, our implementation reads and writes state asynchronously, and displays a UI element to indicate its status. Nevertheless, we seek to minimize the latency our design introduces on top of cloud storage access.

Writing a record entails both encrypting the record, and replicating storage signatures. To understand how much latency this adds, we used a four-year-old Core i7-620M laptop to encrypt a 26KB message with a 4096-bit RSA key pair 100 times. We employed constant-time encryption, so the particular key would not affect the timings. We chose 26KB because it was the average email message size without attachments in 2013, according to one market research firm [13]. We found that on average, it took only 0.27 seconds to encrypt the message (with a standard deviation of 0.005 seconds). This is almost beneath notice.

When reading or writing a record, our prototype additionally gathers and scatters storage signatures. To measure the latency this can add for a medium-sized organization, we deployed the Syndicate Metadata Service (which served as our metadata repository) on Google AppEngine [10], and used the Princeton VICCI [18] cluster for cloud storage. Both handled signature requests over HTTPS.

We used 300 PlanetLab [16] nodes to simulate the organization by reading and writing signatures simultaneously. Half of them received the signatures in less than 0.25 seconds, and 90 percent in less than 0.5 seconds. Half of them wrote new signatures in less than 1.5 seconds, and 90 percent in less than 2.5 seconds. We believe this is acceptable, since in practice many minutes pass between storage operations.
6 Related Work

STEAK falls into the growing body of work relating to HCI in information security. Barriers to using secure email include lack of technical aptitude [19] [7], and stigmatization of chronic use [8].

There have been several attempts to create usably secure email. The simplest approach is incorporating key management functions into the UIs of existing email clients, as done by Enigmail [6]. More sophisticated variants of this approach perform automatic decryption and verification, and recent variants like Mailpile [12] attempt to streamline key distribution and key fingerprint verification. However, they do not succeed in making the process transparent—the user is still involved in trusting, verifying, generating, distributing, and revoking keys.

A more sophisticated approach is to automate key generation and distribution, as seen in systems like ESCAPE [2] and ePOST [14]. In these systems, a user’s host generates a key and one or more trusted CAs automatically vouch for it, allowing users to bootstrap trust in one another. However, this approach assumes the CAs are never compromised, and requires them to decide when to revoke keys. In contrast, STEAK trusts CAs only for registration and backwards compatibility, and revokes keys when users believe their passwords are compromised.

Another approach is ID-based cryptography, whereby system components derive a user’s public key from her identity [4]. This eliminates the need for PKI and public key distribution. However, revoking unexpired keys is infeasible—either the user must change her identity, or the master private key (and all other public keys) must be regenerated.

We consider STEAK’s architecture to be an incremental improvement on the Internet Mail 2000 [3] proposal. We take advantage of cloud storage independent of the user’s ISP, while also addressing security and ubiquitous access in addition to disincentivizing spam.

7 Conclusion

We have presented STEAK, a backwards-compatible email system that gives users webmail-like access semantics while providing message CIA guarantees. It does so by leveraging a user’s cloud storage for hosting state, and by distributing and revoking keys using more servers and channels than adversaries can compromise. Our prototype is under active development, but is qualitatively easier to use than PGP and does not hinder user-perceived performance.

References