Easy Email Encryption with Easy Key Management

CUCS-004-18

John S. Koh
Columbia University
New York, NY 10027
koh@cs.columbia.edu

Steven M. Bellovin
Columbia University
New York, NY 10027
smb@cs.columbia.edu

Jason Nieh
Columbia University
New York, NY 10027
nieh@cs.columbia.edu

Abstract

Email privacy is of crucial importance. Existing email encryption approaches are comprehensive but seldom used due to their complexity and inconvenience. We take a new approach to simplify email encryption and improve its usability by implementing receiver-controlled encryption: newly received messages are transparently downloaded and encrypted to a locally-generated key; the original message is then replaced. To avoid the problem of users having to move a single private key between devices, we implement per-device key pairs: only public keys need be synchronized to a single device. Compromising an email account or server only provides access to encrypted emails. We have implemented this scheme for both Android and as a standalone daemon; we show that it works with both PGP and S/MIME, is compatible with widely used mail clients and email services including Gmail and Yahoo! Mail, has acceptable overhead, and that users consider it intuitive and easy to use.

CCS Concepts •**Security and privacy → Key management; Public key encryption; Usability in security and privacy;** Web application security;

Keywords IMAP, email, applied cryptography, PGP, S/MIME, key management

1 Introduction

Email accounts and servers are an attractive target for adversaries. They contain troves of valuable private information dating to years back, yet are easy to compromise. Some prominent examples include: the phishing attack on Hillary Clinton's top campaign advisor John Podesta [15], the 2016 email hack of one of Vladimir Putin's top aides [38], the email leaks of former Vice President candidate Sarah Palin and CIA Director John Brennan [11, 32], and other similar cases [18]. These attacks targeted high profile individuals and organizations to leak their emails and damage their reputations. In the Podesta leaks, attackers perpetrated a spear-phishing attack to obtain John Podesta's Gmail login credentials, access his emails, and leak them to WikiLeaks. Sarah Palin was subjected to a simple password recovery and reset attack which granted the attacker full access to her personal email account on the Yahoo! Mail website. John Brennan's AOL web email account was compromised via social engineering. Adversaries also sometimes seize entire email servers such as in the cases of cock.li and TorMail [24, 34], or compromise them, such as in the Sony Pictures email leaks [36].

The common thread is that a compromise exposes the *entire history* of affected users' emails after even a single breach. With the explosive growth in cloud storage, it is easy to keep gigabytes of old emails at no cost. The introduction of Gmail with its massive storage capacity—up to 15 GB for free, or 30 TB for paid options [37]—opened up the possibility of keeping email forever. Consequently, users often email themselves to use their inbox as backup storage for important information, thereby exacerbating the cost of a compromise.

Existing email encryption methods are effective but rarely used and even more rarely used correctly. Examples include Pretty Good Privacy (PGP) [4], and Secure/Multipurpose Internet Mail Extensions (S/MIME). Neither are widely used as they are too complicated for most users; both senders and recipients must comprehend public key cryptography and its infrastructure. The current paradigm places too much of a burden on senders who must correctly encrypt emails and manage keys [30, 35] so even technical users rarely encrypt their email. The problem is that existing approaches seek absolute security via end-to-end encryption at the expense of usability, creating a chasm between protecting all email communications via PGP or S/MIME, and using no email protection at all.

We introduce an approach to encrypted email that addresses the gaping void between unusable but absolute email security, and usable but no email security. We change the problem from sending encrypted emails to *storing* encrypted emails since it is a user's *history* of emails that is most tantalizing to attackers. Our goal is to mitigate the kinds of attacks often publicized in the news where email account credentials are compromised or entire servers are seized. The attackers have access to emails stored on servers but do not have access to individual devices. Most of the attacks are either simplistic phishing attacks for email account credentials or server breaches that include innocent users in the collateral damage. All these compromised emails would have been protected had they been encrypted prior to any breach using a key inaccessible to the email service provider. We therefore seek a client-side encrypted email solution that can ensure that a user's emails stored prior to a compromise remain safe. Furthermore, such a defense must be usable for average people on a daily basis, and compatible with email correspondents who do not use encrypted email.

We present Easy Email Encryption (E3) as the first step to filling this void. E3 provides a client-side encrypt-on-receipt mechanism that makes it easy for users as they do not need to rely on public key infrastructure (PKI) or coordinate with recipients. E3 protects all emails received prior to any email account or server compromise for all of the emails' lifetime, under similar threat modelassumptions made for more complex schemes such as PGP and S/MIME; for ease of discussion we hereafter refer to PGP and S/MIME email as end-to-end encrypted email.

E3 is designed to be compatible with existing IMAP servers and IMAP clients to ease the adoption process. An E3 client downloads messages from an IMAP server, encrypts them in a standard format, and uploads the encrypted versions. The original cleartext emails are then deleted from the server. No changes to any IMAP servers are necessary. Users require only a single E3 client program to perform the encryption. Existing mail clients do not need to be modified and can be used as-is alongside a separate E3 background app or add-on. If desired, existing mail clients can be retrofitted with E3 instead of relying on a separate app or on an add-on.

Users are free to use their existing, unmodified mail clients to read E3-encrypted email as long as they support standard encrypted email formats. The vast majority of email clients support encrypted emails either natively or via add-ons. Other than the added security benefits of encryption, all email functionality looks and feels the same as a typical email client, including spam filtering and having robust client-side search capability.

Key management, including key recovery, is simplified by a scheme we call *per-device key (PDK)* management which provides significant benefits for the common email use case of having two or more devices for accessing email, e.g. desktop and mobile device mail clients. Users with multiple devices leverage PDK with no reliance on external services. Users who truly only use a single device still benefit from PDK's key configuration and management capabilities, but rely on free and reliable cloud storage for recovery. E3 as a whole is a usable solution for encrypted email that protects a user's history of emails while also providing a simple platform-independent key management scheme.

We implemented E3 in multiple environments. First, we created two versions on Android for S/MIME and PGP to show that existing mail clients can be retrofitted with E3 even on mobile devices, and E3 works with any standard encrypted email format. Second, we implemented a daemon-like Python client that allows users to use existing mail clients. Finally, we implemented an extension for the Google Chrome web browser. We tested that the Android and Python prototypes work with popular email services, including Gmail, Yahoo! Mail, and AOL Mail. We also measured the performance of E3 on Android. Our results show that while E3 imposes a one-time cost for email encryption, the total overhead is quite reasonable from a user perspective. Finally, we present the results of a user study for E3 that show that users consider it simple, intuitive, and flexible.

2 Threat Model

The purpose of E3 is to protect all emails stored *prior* to any email account or server compromise, with no software or protocol changes except for installing E3 itself on a recipient's devices. The primary risk we defend against is to stored mail on the IMAP server. If the account or server is compromised, all unencrypted mail is available to the attacker.

We thus guard against *future* compromise of the user's IMAP account or server. We assume that the IMAP account and server are initially secure, and that at some later time, one or both are compromised. We therefore assume that email services are honest; the threat is external entities trying to access email account data. If email service providers are not honest, e.g. keeping separate copies of received emails, then the platform is fundamentally insecure which is out of scope. However, a server attack may occur after the server is discarded by physically compromising the server's disks [13]—few organizations erase old disks before disposal. We assume the enemy is sophisticated but not at the level of an intelligence agency, i.e., the enemy cannot break TLS.

We do not attempt to protect against compromise of the user's devices or mail clients. If those are compromised, the private keys used by E3 are available to the attacker no matter when the encryption takes place. Standard end-to-end encrypted email makes the same assumption.

3 Usage Model

E3 works with any IMAP email service. To get started, all a user does is install an E3 client. E3 clients appear to the user as either a separate one-time use app or as a new feature, such as an add-on, in their mail client of choice. This depends on the user's platform, operating system, and mail client. Regardless of the exact implementation, the initial setup acts like any normal mail client, including asking for email account credentials. By default, the client will be configured to operate in active mode, meaning that the client will encrypt all email on receipt. The user at this point continues using whatever email client he wants and email will work exactly as before including sending and receiving email, except that email is transparently encrypted on receipt. Clients that support encrypted emails identify them as such with visual indicators to help avoid the potentially confusing issue of being *too* transparent [30] as it should be obvious to users whether an email was encrypted or not.

Users may configure multiple devices to use E3. As mentioned above, E3 initially defaults to running in active mode to encrypt emails, but it can also run in passive mode, letting users read E3 encrypted email without performing encryption. Users require only a single active client per email account, so if a user's mailbox already has E3 emails, any additional E3 client defaults to passive mode (which can be

toggled by users at will between active and passive mode). Users who use multiple devices to access their email participate in a simple, brief, and platform-independent verification process for each new device with a passive E3 client.

Suppose a user wants to add a new tablet with E3. He installs a passive E3 client on the tablet which his active client will detect. The active client will then prompt the user to confirm the newly added device by matching a displayed verification phrase with the one shown on his tablet during the E3 setup. The user must select the correct verification phrase from a selection of three different verification phrases displayed on the active client. Additionally, if the user does not confirm the new device within a limited window of time, the request will be canceled and the user will need to restart the E3 setup process on his tablet. Once the user successfully adds his new tablet device, the active client will begin to re-encrypt all the user's emails so that the tablet can read them.

Users rarely add new devices or mail clients to their ecosystem, so re-encrypting is an uncommon cost. Adding a new device generally happens in the following situations: (1) the user obtained a new device to replace an existing one, or (2) the user obtained a new device of a completely different type. If a replacement, then in many cases the old device's data will be cloned to the new device so that nothing more needs to be done. In the second case, a new device type means the user will need to add it as a new E3 client; however, any future devices of the same type will fall under case (1) and will not need to be added as a new E3 client. This is not a common occurrence.

The active client lets the user cancel and undo adding a new device, such as in the case where the user accidentally accepts one. When the active client has just added a new device, it displays a notice to the user that all his emails are being re-encrypted. The notice gives the user the option to cancel the process and return the emails to their original state (or to dismiss the notice). If the user cancels the process, the active client rolls back the work it thus far completed. Similar logic is applied if the user wishes to revoke a device from his E3 ecosystem. The user can select the device by name in the active client and delete it. The client then re-encrypts all email to exclude the device.

A user may occasionally lose a device. The optimal PDK setup assumes that a user configures multiple E3 clients. For example, many users will often have at least two mail clients: a desktop/laptop client and a mobile client. Therefore, if a user loses one device, he can still access his E3 encrypted email on his remaining device(s). PDK with only one device requires a different solution for recovery if that device is lost. For users with a single device configuration, PDK provides the user with a randomly generated recovery password which the user must save by printing it out or recording it somewhere safe. Users do not require a recovery password if they use multiple devices unless a user fears he may lose all of his devices simultaneously. The recovery password can be used with a PDK client on a new device to regain access to the user's emails.

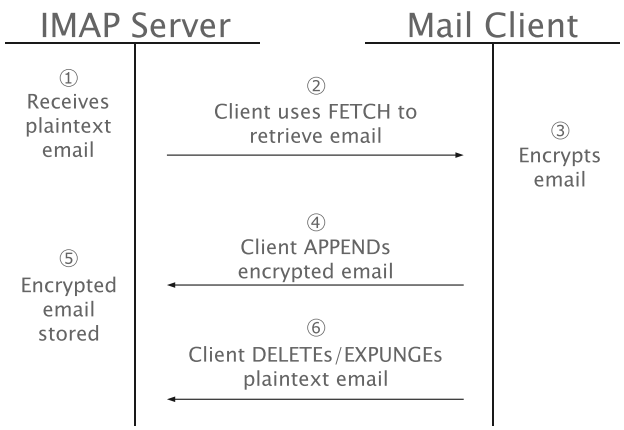

Figure 1. The required communications between an active E3 mail client and an IMAP server for encrypting email.

While most mail clients support encrypted email, one exception is web browser clients such as the Gmail website and other similar webmail services. Browser add-ons for encrypted mail exist (and we have also written one), and it is not unreasonable to expect native browser support if the demand is great enough. For now, users can install web browser extensions that integrate with webmail services to decrypt E3-encrypted email.

4 Architecture

Figure 1 presents a high-level view of E3. An active E3 mail client downloads an email, encrypts it in either PGP or S/MIME format using a self-generated keypair or X.509 certificate, and uploads the encrypted version while deleting the original. For ease of discussion we refer to PGP keys and X.509 certificates as keypairs consisting of public and private keys. E3 builds on existing protocols and encrypted email formats, simplifying its implementation and deployment. E3 leverages Internet Message Access Protocol (IMAP) [7]. S/MIME implementations rely on X.509 certificates and the S/MIME standard as documented in RFC 5280 [6] and RFC 5751 [25]. PGP implementations follow the OpenPGP standard in RFC 4880 [4]. We also address search capability, and key management for multiple devices.

4.1 E3 Modes

Active mode only needs to be enabled on one E3 client. Any other client can operate in passive mode. The active client encrypts received emails and thus is more suited for an always-connected device. It scans the user's mailbox for new public keys labeled as E3 keys. These public keys are securely verified with minimal user interaction by leveraging reliable transparent (to the user) heuristics and temporal proximity: the active client, after detecting a new key in

the mailbox, asks the user if he very recently added a key and to confirm the correct verification phrase among a list of three phrases, two incorrect and one correct. If the user chooses the right phrase, then the active client begins using the new key, otherwise it discards the key. We discuss the key verification system further in Section 4.6. An important feature of encrypted email formats is that a single email can be encrypted to multiple public keys so that any one of the paired private keys can be used to decrypt the entire message. Thus, emails are encrypted using every verified and available public key, including the active client's own key. When a new key is added, the active client re-encrypts emails to old keys as well as the new key.

Passive mode clients only decrypt emails. Importantly, this means that an E3 passive client can be implemented as a one-time use app or add-on which configures a user's existing, unmodified mail client with an E3 private key. A passive client's setup is similar to active mode. The passive client generates a keypair and uploads the public key to his mailbox. Uploaded keys are attached to emails with clear labeling. The keys are discovered by the active client and verified by the user with minimal interaction.

4.2 Keypairs without Web of Trust or PKI

Normally, public keys need to be signed by a trustworthy entity or nobody will trust it. This forms the basis of PGP webs of trust and X.509 public key infrastructure (PKI). We hereafter refer to this general concept as PKI for convenience.

In E3, public keys are never shared with others. They are self-generated and self-signed, and require no PKI. This significantly improves E3's usability over end-to-end secure email solutions which rely on PKI and sender-recipient coordination. Previous work [35] has shown that users find it confusing to correctly obtain and use public keys. In contrast, an E3 user needs only self-signed keys, and any public key exchanges among his devices are automated.

4.3 IMAP Support and Compatibility

Consider common email operations. A mail client downloads a message using the IMAP FETCH command. To delete it, the client uses the IMAP STORE command to mark it with the `\Deleted` flag. IMAP EXPUNGE then purges email marked for deletion. The user may compose and upload an email using IMAP APPEND. These four IMAP commands, FETCH, APPEND, STORE with `\Deleted` flag, and EXPUNGE, play a key role in E3. We henceforth use DELETE as shorthand for the STORE with `\Deleted` flag command.

Figure 1 shows how these four IMAP commands can encrypt email on receipt with existing IMAP servers. E3 can be summarized as downloading a message (FETCH), encrypting it, uploading the ciphertext (APPEND), and deleting the cleartext (DELETE and EXPUNGE). Finally, the client ensures correctness by synchronizing with the server.

This series of commands can be generalized to any IMAP message. It does not matter what mailbox or folder the message is in. The same process can even be applied to sent emails as technically, even "sent" messages are just appended to the IMAP server. All these IMAP commands can be done in the background, decoupling them from the critical path of reading email. Additionally, this series of commands is only relevant for E3 clients in active mode. Passive E3 clients only download and decrypt encrypted mail as needed.

E3 requires multiple round-trip times (RTTs) with the server because IMAP does not support message replacement, but optimizations may be possible in the future. The proposed REPLACE command [3] would be preferred over E3's use of APPEND, DELETE, and EXPUNGE commands. But, this RFC extension is not yet supported. The REPLACE command would eliminate the multiple RTTs associated with DELETE and EXPUNGE, which may be relatively large for small emails, but negligible for large emails. Another optimization would be to use IMAP pipelining, but not all IMAP servers support it, and REPLACE would obviate the need for it.

E3 is compatible with TLS [21] (or STARTTLS) which encrypts all communications with the IMAP server. Although eavesdropping is not the primary security focus, E3 with TLS protects against attackers who could otherwise capture cleartext emails when they are first downloaded by the client.

Users only need a single E3 client in active mode to encrypt emails which by design avoids race conditions among multiple active clients. This follows the standard practice of discouraging automatic message modifications on multiple IMAP clients, which is a reasonable approach since IMAP lacks atomicity guarantees. However, users may accidentally enable active mode on multiple clients. E3 thus uses approaches similar to existing IMAP clients in dealing with race conditions, such as multiple clients trying to encrypt the same message.

Currently, the blessed way of achieving pseudo-atomicity when modifying IMAP messages is to use the IMAP CONDSTORE extension [1]. CONDSTORE is supported by major IMAP email services and open source servers, including Gmail and Dovecot. This extension requires servers to maintain a last-modified sequence (mod-sequence) number on messages which is returned to the client. An active E3 client which wishes to encrypt a message can take advantage of this by adding a flag (either `\Flagged` or `\E3Encrypting` depending on custom flag support) to a message using a special UNCHANGEDSINCE modifier to the normal IMAP STORE command so that the STORE flag command will only succeed if the message has been unchanged; this will also update the mod-sequence value of the message, so any other active clients who try to issue the same command for the given message will fail since it already has been modified.

The flag and updated mod-sequence value act like a lock, thereby alerting other active clients that this message is

slated to be encrypted. Then, the client with the lock can issue normal IMAP commands without racing others. One potential issue is the active client with the lock may crash before it completes its work, thereby leaving a dangling lock. There are many ways of addressing this problem. One method is to use a heuristic based on a message's received timestamp. An active client may periodically scan the user's mailbox for messages with the `\E3Encrypting` flag that have not been encrypted, and based on the timestamp heuristic, determine if too much time has passed since the message was received. For example, if the message has not been encrypted for three hours since it was received but has the `\E3Encrypting` flag, then the active client may remove and re-add the flag to obtain the lock on the message again and encrypt the message as normal.

If CONDSTORE is not available, an alternative is to make a best-effort using IMAP custom flags and custom IMAP folders. The strategy, like with CONDSTORE, is to mark a message with a custom flag (keyword) entitled `\E3Encrypting`, and to move it into an IMAP folder named `E3-Temp`. Then, any active E3 client that sees the E3 flag on a message in the special temporary folder should not encrypt it. This does not rule out race conditions entirely, but will certainly shrink the window that it could occur within.

4.4 Ciphertext Format

E3 uses the widely supported OpenPGP message or S/MIME Enveloped-Data formats depending on client preference. E3-encrypted S/MIME messages can be read using any existing unmodified mail client that supports S/MIME, including Apple Mail, Mozilla Thunderbird, and Microsoft Outlook, assuming at least one E3 private key is available in the local keychain. The same holds for PGP. Since these formats only encrypt the body text, all of the original headers are maintained except for the Content-* headers, which are updated to ones appropriate for encrypted emails. Since the Received timestamp header is unchanged, mail clients can display messages in their original order. E3 also adds a custom header, X-E3-ENCRYPTED, to distinguish E3 emails from other encrypted emails. This is useful for IMAP servers which do not support custom flags or keywords.

E3 normally does not re-encrypt emails that are already encrypted when received, i.e., when receiving email from a sender using end-to-end encryption. However, there are situations where re-encrypting emails is useful such as when a crypto algorithm or key size is no longer secure. In this case, E3 can easily re-encrypt existing encrypted email to a newer crypto standard.

E3's encryption does not interfere with spam filters. Spam filters can be on servers or on clients. When they are on the server, such as with Gmail, the mail service can filter spam emails before they are encrypted. For client-side spam filters, the user's mail client will detect spam messages and move or delete them. However, since the client performs the filtering, it can apply the filter before encryption, or decrypt E3-encrypted messages to scan them for spam.

4.5 Search Capability

Searching is straightforward: index and store the decrypted content of messages locally. This is compatible with existing mail client local search, and provides full, fast local searching. Storing messages locally is a common practice among modern mail clients, and examples can be seen in Gmail on Android, Mail on iOS, and Mozilla Thunderbird and Apple Mail on desktops. While message content is stored in the clear locally, we do not worsen security over the norm—many mail clients already do this. An option for the more security-conscious is to apply full disk encryption in conjunction with device-wide security features. An alternative would be to store ciphertext content locally, but this does not provide any real security benefits since the key is also available locally, and would also interfere with local searching.

A limitation of encrypted email schemes is that they do not support searching the body content of encrypted emails on unmodified email servers. (Headers, including the `Subject:` and other metadata fields, are searchable.) If the server can be modified, E3 can be used in conjunction with systems such as SSARES which allows for searchable encrypted email [2]. For unmodified servers, the IMAP SEARCH command cannot be used, so clients that search both locally and on IMAP servers will only return results for local search and remote metadata matches. On the other hand, IMAP search is significantly slower than local search and is based on naive string matching which yields low quality results. Thus, users often will not wait for IMAP server search results in practice since local search queries are nearly instant. Furthermore, many email clients such as K-9 Mail only perform local search unless remote search is specifically enabled, which would provide the exact same search capability with or without E3.

4.6 Key Management, Migration, and Recovery

E3 eliminates manual public key exchanges. This simplifies the problem of key management by removing half of it. What remains is the problem of private keys. Traditional security best practices advise users to never transport private keys because doing so is insecure. This advice is almost never followed in practice because users often access email from multiple devices, all of which need the same private key when using common secure email usage models.

E3 returns to the traditional security advice of never transporting private keys. In contrast to most secure email schemes which assume a user has a single private key, E3 asserts that a user should have a unique private key for every device. We refer to this as a per-device key (PDK) scheme as depicted in Figure 2. With PDK, a user does not need to move any private keys among his devices. Instead, each of his passive clients automatically makes available its *public key* to the device in charge of encrypting emails, i.e., his selected active

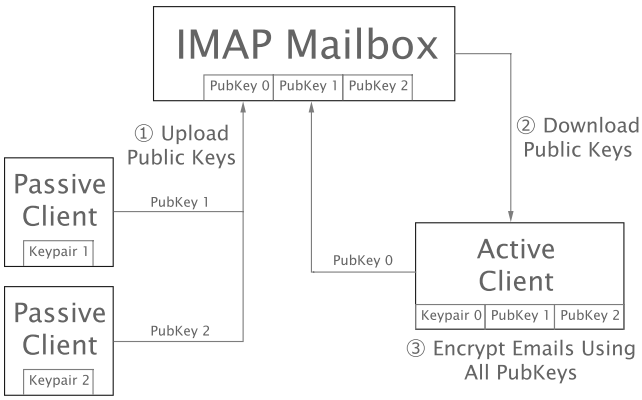

Figure 2. The per-device key (PDK) architecture.

E3 client. The active E3 client can then encrypt the user's emails using the public keys from all of his devices.

The PDK scheme has several advantages over using a single private key:

1. Private keys never move and never leave a device.
2. A private key can be "revoked" by re-encrypting emails to all public keys except for the lost one.
3. Private key recovery, normally mitigated with private key backups, is reduced to data backup which is easier and inherent to the design. As long as one device is available, emails can be decrypted. If all devices and backups are lost, then emails cannot be decrypted.
4. Public keys can be automatically distributed using the email account.
5. All keypairs are self-generated and self-signed so users can freely add new devices as needed.

The active E3 client detects the user's public keys uploaded to his mailbox and uses them to encrypt emails. Passive E3 clients only upload their E3 public keys to the user's mailbox and perform no encryption.

Public keys in the user's mailbox cannot be blindly trusted. The active client must securely confirm whether a new public key really belongs to the user. The active client periodically scans for new keys, and as a first heuristic, ensures that the sender of the key email (i.e., the "From:" field) matches the address of the account owner; any emails containing keys from other senders are not accepted. However, this heuristic alone is not enough to verify the key as an attacker may gain access to the email account and upload a malicious key with the correct sender address, or simply just spoof it. We therefore augment this check by requiring temporal proximity and matching a verification phrase. Temporal proximity means the user has a limited window of time to accept and verify a newly detected public key. Any keys which are not accepted within the time window are discarded by the active client.

Temporal proximity relies on verified and signed timestamps. A passive client uploads its public key along with a signed timestamp obtained from services such as Roughtime [14], and active clients verify if the timestamp is within the allowed time window and trustworthy. For example, an active client configured to only allow public keys uploaded within the last 60 seconds will reject any uploaded public keys with a verified timestamp that is older than 60 seconds. The timestamp is verifiable since it is signed using the Roughtime service's certificate. As an additional measure, the active client will rate limit the number of requests to add a new key. For example, the client will only consider at most three key requests in a period of five minutes. Any more than that are suppressed, and a warning is shown to the user that unusual behavior has been detected.

In addition to temporal proximity, E3 also uses a verification phrase that is easy for humans to recognize and match. When a passive client uploads its key, it adds a randomly generated verification phrase to the key email which is prominently displayed on the passive client. This verification phrase is also shown alongside two randomly generated incorrect verification phrases on the active client when it detects the uploaded key. The user must select the correct verification phrase in order to accept the newly added key. This multiple choice confirmation is useful to reduce the chances of a user accidentally accepting a key.

The verification phrase is generated at random from a curated list such as the PGP Word List [19]. This string is added to the email containing the uploaded public key and displayed on the passive client. The active client displays the verification phrase from the email along with two incorrect phrases, and asks the user to choose the one matching the one in the passive client. As shown in SafeSlinger [10], this technique is effective and usable for quickly authenticating identities even with only three words. Users who speak other languages use word lists in their language. Another option is to rely on a recognizable but randomly selected or generated image instead of a word-based string. Further research is needed to better understand what kinds of strings or images real users can correctly recall and verify while making minimal errors.

The active client must re-encrypt all emails to include new public keys. A user may accidentally accept a key he did not intend to accept, or simply may wish to cancel the process. The active client therefore displays a semi-persistent notice to the user indicating that re-encryption is taking place, but can be stopped and reversed at any time. If the user stops and reverses the re-encryption process, the active client re-processes the emails it re-encrypted, and re-encrypts them again to the original set of keys.

The logic for key revocation is nearly the same. In the exceptional case where a user wishes to revoke one of his E3 keys, he deletes the key by device name from the active client's list of keys. The active client then re-encrypts all emails to every key excluding the revoked one.

As a side note, advanced users may prefer the traditional fingerprint-based string matching or QR code scanning methods of verifying public keys. E3 supports these as well, but only as an advanced option that is not enabled by default and therefore not used by normal E3 users.

PDK achieves a streamlined key verification process where, for every newly added key, the user must only agree to an automatic prompt on the active client, and ensure that the verification phrase matches the one he recognizes. This is in contrast to key verification for end-to-end secure email which often relies on confusing public key fingerprint matching, QR code fingerprint scanning which is unavailable for devices without cameras, and understanding of PKI. E3 key verification is possible with these standard techniques but unnecessary given the unique environment in which E3 operates. Another issue with existing end-to-end email is verifying the public key of every new email correspondent. In E3, adding a new key is a rare occurrence and only happens when configuring a new mail client such as when getting a new device.

PDK's recovery mechanism inherent to the multi-device design is available to the majority of users who access email using two or more devices. However, there may be users who truly only ever access email with a single device. For these users, PDK key recovery uses the traditional method of encrypting the user's private key with a password—a recovery key—and then storing the encrypted private key on a backup device or in cloud storage. If stored in cloud storage, the provider should be different from the email service provider. For example, E3 clients configured for Google's Gmail service might store the private key on Dropbox but not Google Drive. The key retrieval system adheres to best practices for secure credentials exchange as seen in the design of Securely Available Credentials (SACRED) [16].

PDK is also compatible with re-encrypting emails to future-proof them. Algorithms age, so ciphers and key sizes that are secure today may not be in the future. PDK implementations can simply download a user's new keys and purge keys from the system that are no longer usable.

One avenue for future work is the problem of reading email on public computers. In this case, users access their confidential data on a fundamentally untrusted device which cannot be trusted with private keys. This is a concern for all encrypted email schemes, not just E3. Even though solutions are technically possible, they are insecure due to the high risk of unwrapping private keys in an untrusted environment.

5 Security Analysis

E3 does not intend to be an end-to-end, maximum security solution, but a strict improvement over the norm that is easy to use and deploy. We sacrifice a small amount of security to gain tremendous usability over existing secure email models. We henceforth show that E3 provides tangible security benefits compared to no email encryption, and compare its security with traditional end-to-end secure email.

E3 protects all emails for all of their lifetime as long as they are encrypted *before* any email account or server compromise. Standard end-to-end encryption does the same, but E3 does so without the complexity of public key exchanges and PKI.

Like end-to-end email, E3 protects sent and received mail assuming all correspondents use E3. Senders can encrypt their sent email copies as stored on the sender's IMAP server. Unlike end-to-end, which *require* that both the sender and receiver use it, E3 provides useful protection even if only one side uses it. If the sender uses it, his emails that are encrypted before an attack are protected from compromise of his email account or server. The same holds for the receiver without loss of generality. In other words, E3 provides better protection than end-to-end email for communications in which one party does not use email encryption because end-to-end encryption cannot be used and would therefore provide no protection at all.

If not all email correspondents use E3, it is possible for an attacker to compromise the emails of any correspondent not using E3 to expose email communications with one that uses E3. Regardless, this property actually confers a benefit to E3. E3 can be incrementally deployed since not all correspondents require it. E3 also exhibits network effects: it provides better security as more users use it.

Unlike end-to-end email, E3 requires additional measures to protect against eavesdropping. Fortunately, these measures are completely transparent to users: E3 uses TLS or STARTTLS so there is no threat of eavesdropping if TLS is secure. Furthermore, TLS and STARTTLS are supported and encouraged by practically all major mail services.

Email may or may not be protected in transit between SMTP (not IMAP) servers. SMTP server links are increasingly protected by TLS; if not, the problem is out of scope. Services such as Gmail flag email that arrive via unprotected SMTP connections. That said, such backbone links are very hard to tap. The risk, then, is minor.

After an email account or server is compromised, E3 cannot protect newly arriving emails. This is a limitation compared to end-to-end encryption which protects new emails sent by other end-to-end users. Nevertheless, we argue that end-to-end encryption rarely sees actual use among users and therefore provides no practical security for the majority of the population. In contrast, E3's ease of use makes it much more likely to be adopted while providing a strict security benefit. In a mailbox with just a few thousand messages, compromise of new emails comprises a minuscule percentage of total emails. New emails are still important, but it is clear that encrypting the majority of emails is better than encrypting none.

Email account compromise can happen in many ways, primarily through credential or key compromise. That, in turn, often happens because of user error, especially in cases

of (spear-)phishing. While devices do have OS-level security features to help combat phishing, E3 by design also provides a strong defense. E3 does not try to protect private keys stored locally since the device is assumed to be secure, so users are never requested to manage their private keys. Thus, there is no password for attackers to phish, and users have no knowledge of where the private key is stored. This latter intrinsic property of E3 also makes it difficult for an attacker to trick a user into providing his private key—since the user does not know where it is!

One major obstacle in other secure email schemes is ensuring availability of the private key on all devices. There is no standard for secure, usable key transport and the market is fragmented, so it is difficult to do a comparison with E3. We have designed PDK as a secure and usable scheme that leverages users' tendency to access email on multiple devices, and the inherent support for multiple recipients in encrypted email formats.

An attacker may try to trick a user into accepting and authenticating a malicious public key by sending a fake E3 key email to the user. If the user were to accept it, all emails would be encrypted using the malicious key, allowing the attacker to decrypt the user's email if the account is ever compromised. Therefore, PDK is only as strong as the key authentication system used in conjunction with it. The first line of defense is to ensure that uploaded keys came from the user's own email address. Keys attached to email from other addresses are rejected. However, an attacker may have access to the email account allowing him to upload keys originating from a valid address, or simply just spoof the sender field. We therefore rely on temporal proximity such that an attacker would need to strike literally minutes or even seconds before the user generates a new key. Otherwise, the uploaded key would be rejected for being too old if encountered by the target at a later time. This is similar to time-based one-time password schemes as seen in two-factor authentication, e.g., RSA security tokens and Google Authenticator.

An attacker without access to the mailbox would need to also correctly guess the randomly generated verification phrase. An attacker with access to the mailbox could wait for the user to upload a new key and then duplicate the key email but attach his malicious key instead. This would allow the attacker to construct a key email with the correct verification phrase. However, this requires immediate temporal proximity, i.e., as soon as the user uploads a new key. Furthermore, the attacker will likely wait a *very* long time before the user adds a new key since adding a new device is a rare occurrence. By this time, most modern email services will have warned the user of a suspicious account access.

An adversary may then resort to a denial-of-service attack and send many fake keys to a user in hopes the user will make a mistake and accidentally verify a malicious key. To address this, the active client rate limits requests to add new keys, and shows a warning to the user. As a final measure, the client also immediately discards keys and any on-going confirmation prompts from any key emails with duplicated verification phrases.

These checks alone suffice to exclude most attacks. On top of these key verification checks unique to E3, we *optionally* support traditional methods for verifying public keys including fingerprint string matching and QR code-based fingerprint verification. However, these methods are only be available to advanced users and are not enabled by default.

E3 considers compromised servers and devices as out of scope. E3 cannot protect against an IMAP server that is dishonest from the start. E3 also does not protect against compromise of the user's devices or mail clients, but neither does end-to-end secure email. Similarly, if a user's device is stolen, E3 may not protect his email. However, many devices are password-protected with data encrypted in local storage, and have remote wipe functionality. In all cases, E3 provides a strict security benefit, and makes security no worse than the current common practice of no email encryption.

6 Implementation

We implemented four prototypes to verify E3 and PDK's applicability in common use cases. The prototypes consist of: K-9 S/MIME, K-9 PGP (via OpenKeychain), a Python separate client, and a Google Chrome extension.

The K-9 implementations require different dependencies thus motivating the separate prototypes. The vanilla K-9 Mail client purposely includes no crypto libraries, instead preferring to offload crypto operations to separate crypto provider applications. However, no such crypto provider app for S/MIME currently exists. Our K-9 S/MIME implementation therefore includes the Spongycastle [26] crypto library and performs all key generation and management on its own. K-9 S/MIME represents a worst case scenario where nearly all functionality is implemented from scratch.

The K-9 PGP (OpenKeychain) implementation contains far fewer code changes compared to K-9 S/MIME. This is due to the OpenKeychain app which is both a keychain and crypto provider app. K-9 offloads all PGP crypto operations, and key storage and verification to OpenKeychain which exposes an external cross-application API, so it was not necessary to add a crypto library to K-9. We modified both K-9 Mail and OpenKeychain to support E3, in particular adding an API call to OpenKeychain (OpenPGP-API) for storing E3 keys, and allowing OpenKeychain to verify and recognize emails which have been self-signed by the email recipient as opposed to the standard PGP use case where it verifies signatures based on the email sender.

K-9 PGP also adds optional metadata to existing OpenKeychain API calls to avoid redundant code within K-9 Mail. For example, OpenKeychain exposes an API call to request key material to which we added optional metadata including the key's PGP fingerprint and QR code bitmap data. Figure 4 depicts how this extra metadata is used by K-9 to generate the

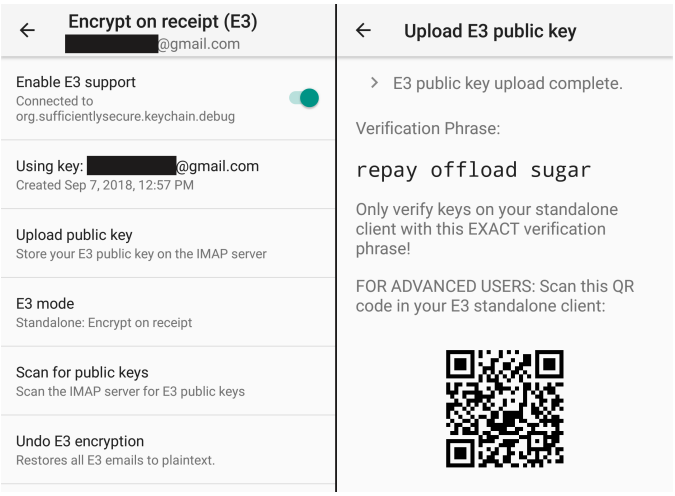

Figure 3. E3 settings and key upload screens.

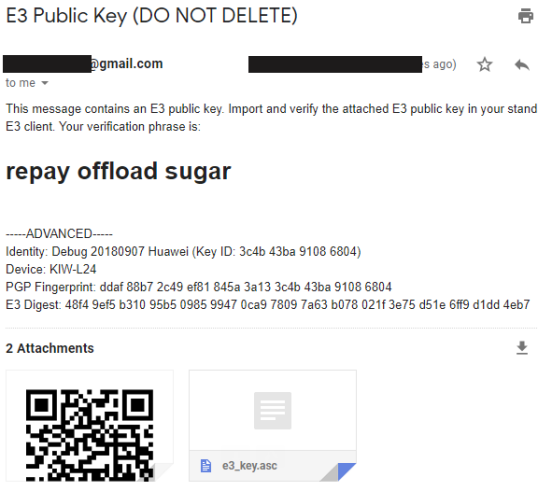

Figure 4. An E3 public key uploaded to a user's email.

	Gmail	Yahoo	Outlook	AOL	Yandex	Dovecot
E3	○	○	○	○	○	○
CONDSTORE	○					○
REPLACE						

Figure 5. Tested servers and their compatibility with E3.

email containing an E3 client's uploaded public key. Note the prominent three word verification phrase also in Figure 3.

The Python E3 daemon generates an E3 keypair and uploads it to the user's mailbox, along with supporting E3 active mode. It currently does not automatically add the key for use with existing mail clients; this is a work in progress. However, we sketch out what the solution would look like on different platforms. On macOS and iOS, we can leverage the system Keychain which the Apple Mail and iOS Mail clients already integrate with. The Python app can add its E3 keypair to the Keychain with an ACL tailored for the targeted mail clients [8]. Android has a KeyChain API [9] for storing system-wide keypairs and can be used in a manner similar to Apple's Keychain. However, Android clients that do not rely on the KeyChain API will require modifications; for example, OpenKeychain must be modified to allow an app to add an E3 private key to it, then existing PGP clients can seamlessly use the key. On Windows, the E3 client can generate a PKCS12 key file to import into Windows' certificate store which is used by the Outlook mail client. For clients such as Mozilla Thunderbird that do not rely on the certificate store, users can install an E3 add-on.

We also wrote a Google Chrome extension to interface with the Gmail website and add basic support for reading E3 encrypted emails and key management. This extension was a proof of concept to show that reading E3 email on website clients is possible and practical.

7 Experimental Results

We used our K-9 S/MIME prototype for testing. While K-9 is roughly 150,000 lines of code, E3 only added about 2,500 lines excluding crypto libraries. This suggests E3 comparatively represents a modest amount of complexity. We verify that E3 works with existing IMAP services, measure its performance overhead, and evaluate its usability with real users.

7.1 Compatibility and Interoperability

To verify that E3 is compatible with existing IMAP and S/MIME systems, we tested our prototype on several of the most popular commercial and open-source email servers. Figure 5 shows the results of our compatibility testing. E3 worked seamlessly with all IMAP email services tested. We also checked for IMAP CONDSTORE and REPLACE support with the former enabling better IMAP atomicity, and the latter enabling better performance. We also verified that unmodified S/MIME mail clients, including Apple Mail, and Thunderbird, could be used to read E3-encrypted email.

7.2 Performance

We measured E3's performance on mobile devices because of the popularity of mobile email and to provide a conservative measure as they are resource constrained. We used a Huawei Honor 5X (8-core Cortex-A53 with 2 GB RAM) smartphone running Android 6.0.1. We compare the performance of our modified E3 K-9 Mail client against the standard K-9 Mail client. Both versions were instrumented to obtain measurements. The E3 K-9 client used OpenSSL 1.1.0b, and the S/MIME emails used Cryptographic Message Syntax (CMS) with 128-bit AES CBC for compatibility reasons. All experiments were conducted using Gmail accounts populated with the same email content, and a WiFi connection to a small business fiber optic network. We chose to use a real email service with a typical Internet connection to better understand performance with real limitations, such as asymmetrical download/upload speeds to the Gmail service. To

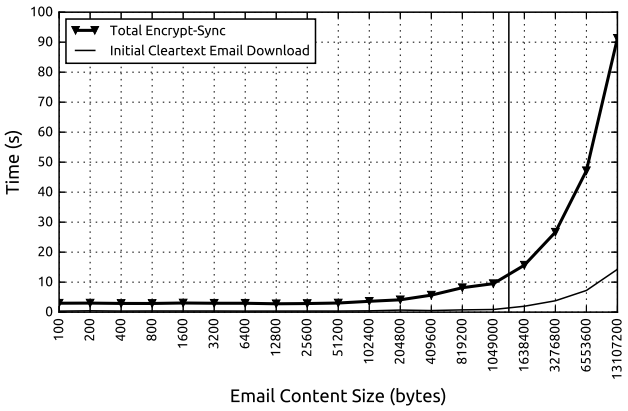

Figure 6. Time spent for the one-time "encrypt/synchronize" compared to the cleartext download. Points right of the line are emails with a JPG.

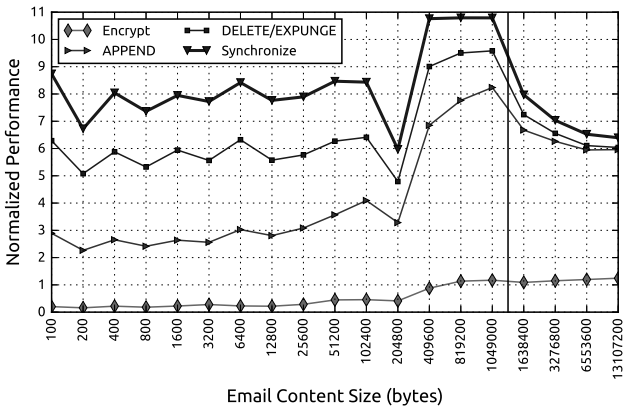

Figure 7. Normalized time spent for the one-time "encrypt/synchronize" relative to the cleartext download (not pictured). Points right of the line are JPG emails.

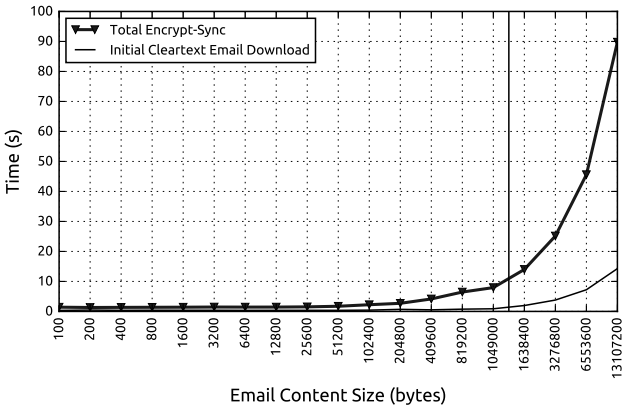

Figure 8. Expected time for the one-time "encrypt/synchronize" with REPLACE compared to the cleartext download. Points right of the line are JPG emails.

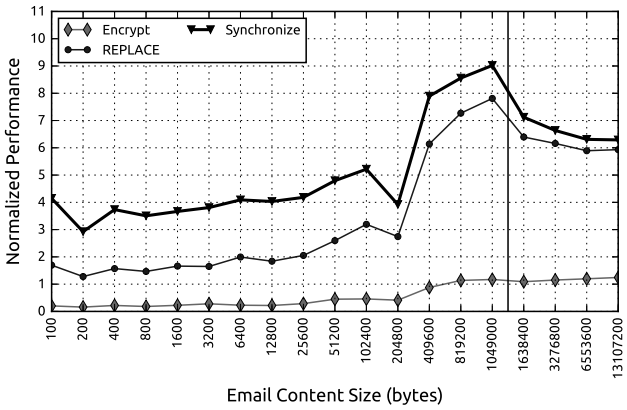

Figure 9. Normalized expected time for "encrypt/synchronize" with REPLACE relative to the cleartext download (not pictured). Points right of the line are JPG emails.

account for variability, each measurement was repeated 30 times, the three lowest and highest outliers were discarded, and an average was taken over the remaining measurements.

We considered email operations where E3 imposes additional work over a standard email client. We did not measure searching as it has no overhead compared to a standard mail client. We measured receiving a new cleartext email in which E3 downloads, encrypts, and replaces it at the server with the encrypted version, followed by a quick synchronize.

We used a range of email content sizes from 100 B to 12.5 MB. 12.5 MB is the maximum because when encrypted, it increases in size to about 24.7 MB due to limitations of the MIME format. Popular services such as Gmail enforce a 25 MB size limit. Emails of size 100 B to 1 MB were two-part MIME messages with a `plain/text` and `html/text` part. Larger emails were two-part MIME messages with a one byte `plain/text` part, and an attached JPG file.

7.2.1 Encrypt and Synchronize

Figure 6 shows the time it takes to encrypt email and replace it on the server, and synchronize the client and server. The plot labeled "Total Encrypt-Sync" includes: Encryption, APPEND, DELETE and EXPUNGE, and Synchronize. Figure 6 also shows the time to initially synchronize and download the original cleartext email. This is strictly not part of E3, but provides a basis to show the relative cost of E3 compared to a standard client. The time to download the cleartext email was the same for both E3 and unmodified K-9.

Before discussing the results, we highlight two important points. First, the overhead of encrypt/synchronize is a one-time cost. Once a message is encrypted and uploaded, it does not need to be processed again. Second, operations run in the background so the user is unaffected.

Figure 6 depicts the encrypt/synchronize time in seconds for each email size. Although the encrypt/sync time is 6× to 11× the time to synchronize cleartext emails, the overhead is not visible to users as it is processed in background threads.

Figure 7 shows the same encrypt/sync measurements as Figure 6, but normalized to the cost of downloading the original cleartext email. This shows a breakdown of the relative cost of each part labeled: Encrypt, which encrypts the message; APPEND, which uploads the encrypted message; DELETE/EXPUNGE, which deletes and expunges the cleartext message from the server; and Synchronize, which verifies client-server consistency. The components are stacked so that each line is cumulative and the area between lines is the overhead for the component. For example, the total normalized overhead for 1600 B emails is 8× the initial cleartext email download, comprising of Synchronize (25%), DELETE/-EXPUNGE (40%), APPEND (30%), and Encrypt (5%).

Encrypting is brief and generally takes no more time than downloading cleartext email. The cost is constant for emails smaller than 102,400 B, then grows linearly in proportion to size. This suggests that for small emails, encryption is dominated by initialization which includes generating the IV and encrypting the AES key. Once size grows beyond a critical mass, encryption time increases as well.

For small emails, the primary overhead is DELETE/EXPUNGE's multiple RTTs which are significant relative to a short APPEND time. To mitigate this overhead, clients can issue a single DELETE and EXPUNGE for batches of emails. For larger emails, APPEND (upload) dominates for two reasons. First, uploading to Gmail was slower than downloading which magnifies the APPEND overhead. Second, the Gmail server supports Deflate/Gzip compression, and the cleartext compresses well. In contrast, ciphertexts are indistinguishable from random bits so they cannot be compressed. Thus, E3 APPENDs the full message size. However, the effects are lost for content that is incompressible. This is the case for the emails larger than 1 MB since they contained a single JPG (incompressible) image; they consequently exhibit less overhead compared to the text emails.

The remaining overhead is due to Synchronize, which appears substantial for small messages. This involves verifying client-server consistency, updating the UI to show progress, and processing any pending commands. This constant overhead—less than a quarter of a second—is magnified for smaller emails, but becomes negligible for larger ones.

Currently, there is no way to replace a message on an IMAP server in a single operation. The proposed IMAP REPLACE extension [3] would eliminate the DELETE/EXPUNGE, so REPLACE's overhead will resemble APPEND alone. We approximate this by taking Figure 6 and removing DELETE/EXPUNGE. This leaves Encrypt and APPEND as visible in Figure 8. Normalized performance can be seen in Figure 9. Like Figure 7, Figure 9 is stacked so that each line is cumulative and the area between lines is the overhead for the component. The reduction in the time for the worst case—small emails—is almost half.

7.3 Usability

After its initial configuration, E3 by default works transparently to the user. The user thus does nothing different from using a regular mail client. As a result, E3's usability is the same as a regular mail client for everyday email usage.

E3 can also be implemented to encrypt on demand, a less transparent way of using email encryption. We do not recommend this for real implementations since this imposes more work on the user. However, since it is possible and also a worst-case implementation for usability, we opted to study it. We ran an IRB-approved user study with twelve participants who were asked to use three different email solutions on a Nexus 7 Android (Qualcomm Snapdragon S4 Pro 8064 with 2 GB RAM) tablet running Android 5.0.1. We compared the usability of an unmodified K-9 client, our E3 K-9 S/MIME client, and an unmodified K-9 client with PGP provided via the OpenKeychain Android app. While E3 does not require entry of an encryption password, we configured the E3 K-9 client to prompt for a password which protects the private key. Although this is more complicated than how E3 is configured in practice, we wanted a conservative environment for comparison. Each user was provided with a Gmail account with the same email contents.

All participants had experience with mobile device mail clients, comprising of nine tech-savvy computer science graduate students at a well-known research university and three non-tech savvy adults between the ages of 20 to 60 that were either unemployed or working in blue-collar occupations. The nine graduate students included several PhD students in systems and security.

The participants volunteered in 60 minute sessions in which they role-played as a tax accountant using email to request a client's tax forms. The 60 minutes comprised three 20 minute sessions, each devoted to using: K-9, E3 K-9 S/MIME, and vanilla K-9 with PGP (not to be confused with our E3 K-9 PGP implementation). We instructed users to send and receive emails three times for each email solution within 20 minutes regardless of whether they succeeded or not. The 20 minute constraint was used to limit the study length. Users were free to use any resource, including the Internet, other than the study coordinator. To mitigate the effects of short-term memory on survey results, we randomized the order of the email clients. To avoid priming participants for favorable responses, we explained our research purpose only after the surveys had been completed.

After participants completed their tasks or reached the 20 minute limit per client, they completed the System Usability Scale (SUS) [22], an industry-standard questionnaire also used in many similar studies [27–30], for the system they had just used. At the end of the study, participants completed eight additional survey questions specific to our research, and a final free-form question requesting any comments. To

Soln.	Count	Mean	Std. Dev	Min.	Q1	Median	Q3	Max
K-9	12	82.12	11.67	65	72.50	82.50	90.00	100
E3	12	81.73	10.82	60	72.50	82.50	90	97.50
PGP	12	34.81	23.09	2.50	18.13	30.50	38.75	47.50

Figure 10. System Usability Scale summarized scores.

ensure that participants actually understood each email solution they used, the study coordinator explained each system prior to completing the eight additional survey questions.

The summarized SUS scores are presented in Figure 10. The results were favorable for both K-9 and E3, but PGP received remarkably low ratings which on average were about half of the K-9 and E3 scores. The SUS scores for K-9 and E3 were very similar despite our "naggy" configuration prompting for the encryption password. We conclude that E3 is at least nearly as intuitive to use as K-9 even in the conservative worst case implementation.

We also directly asked users to compare the three email solutions which we summarize in Figure 11. Responses are on a scale of 1 (Strongly Disagree) to 5 (Strongly Agree). In questions 11-14, users rated both K-9 and E3 as significantly easier to use than PGP. Interestingly, K-9 with E3 had an equal average easiness rating with vanilla K-9. This could be attributed to small inconsistencies in users' perceptions due to the randomized order of the study.

In questions 15-17, participants almost unanimously agreed that E3 provides worthwhile protection. In contrast, a large majority viewed PGP as not worthwhile. They rated PGP so low because of how confusing and unusable it was. Free responses from users included referring to PGP as "garbage," referring to the key exchange as "wildly impractical," and one noting that he has "never actually seen [PGP] used." Most users failed to complete the tasks with PGP in the 20 minute limit, even after searching the Internet. The only participants who succeeded with PGP were the six graduate students who specialized in security; the other three computer science graduate students did not succeed with PGP. In contrast, all of the participants succeeded with both K-9 and E3 without seeking any help online.

We draw these conclusions: (1) K-9 and E3 are roughly as *intuitive* as the other, and E3 is easy to understand. (2) E3 is easy to use even for non-tech savvy users, and even with a naggy implementation. (3) E3 is much more usable and intuitive than PGP. (4) PGP is too unwieldy to actually be used. Overall, our user study results were very positive in favor of E3, but further studies with more users and a wider range of activities would be illuminating.

#	Question (1 = Strongly Disagree, 5 = Strongly Agree)	Mean	Std.	Min.	Med.	Max
11	I found it easy to use unmodified K-9 w/o encryption.	4.38	0.96	2	5	5
12	I found it easy to use K-9 w/ E3 encryption.	4.38	0.65	3	4	5
13	I found it easy to use PGP encryption.	2.08	0.95	1	2	4
14	I thought that K-9 w/ E3 was easier to use than PGP.	4.77	0.44	4	5	5
15	The extra security with K-9 w/ E3 encryption is worth the extra steps compared to unmodified K-9 w/o encryption.	4.23	0.73	3	4	5
16	The extra security with PGP encryption is worth the extra steps compared to unmodified K-9 w/o encryption.	2.69	1.44	1	2	5
17	The extra security with PGP is worth the extra steps compared to K-9 w/ E3 encryption.	1.92	0.95	1	2	3

Figure 11. Summarized scores for added survey questions. (Questions are abbreviated for spacing reasons.)

8 Related Work

The seminal "Why Johnny Can't Encrypt" paper illuminated the confusing process of encrypting email and showed how inaccessible PGP is to average users [35]. They found that correctly sending encrypted email is outstandingly difficult.

Ruoti et al. designed Pwm, usable secure mail that integrates with major email providers while hiding the security details [30]. They found that transparency creates issues because users mistakenly think they are encrypting messages when they are not. This suggests that sending secure email is the pain point. In later work, Ruoti et al. observe no significant differences between automatic and manual encryption [28], but this is a separate issue from sending encrypted email. Furthermore, Pwm's key management relies on a third-party identity-based encryption server.

There are approaches to encrypting email on receipt that rely on modifications that are too technical for average end users. Most examples involve modifying one's Mail Transfer Agent (MTA) or Mail Delivery Agent (MDA) to encrypt emails before delivering them to the client [5, 17]. These works also provide no solution for key management.

STREAM is an approach for opportunistic email encryption [12]. It uses an SMTP and POP proxy for outgoing and incoming messages but does not consider IMAP.

STEED mail clients perform end-to-end, transparent encryption [20]. STEED is compatible with PGP and S/MIME, and furthermore, uses trust upon first contact (TUFC). STEED requires a special client-server protocol for encryption and thus a new IMAP extension. This creates deployment issues so STEED is not amenable to incremental deployment.

Lavabit, now defunct, was a commercial service for secure IMAP email that protected emails in storage. It generated keys from users' passwords, and stored users' keys on Lavabit servers alongside a set of master private keys. These master keys could decrypt all emails, thus making them vulnerable to compromises and subpoenas. Furthermore, Lavabit's encryption was only available for Lavabit users.

Posteo is a paid email service which encrypts its servers' hard drives and provides the option to encrypt emails, including metadata, on receipt [23]. Posteo's approach is server-side and requires modifying IMAP servers. Also, the option to encrypt metadata breaks standard formats.

Tutanota provides a service for sending encrypted emails to non-Tutanota users [33]. Tutanota does not use standard encrypted formats. If a Tutanota user emails a non-Tutanota user, then the decryption password must be conveyed via a secondary channel. Moreover, the email only contains a link to the Tutanota website which the recipient must click on then input the password to decrypt the message. Exchanging passwords and clicking on a link sent via email to input the password can be seen as extremely suspicious by users who are trained to be wary of phishing.

Autocrypt [31] proposes a decentralized and incrementally deployable system for distributing public keys. Key exchanges are transparent as public keys are transmitted via email headers. Although the goal is to improve usability, the methods differ from E3 as Autocrypt focuses on end-to-end, sender-recipient encryption.

9 Conclusions

We have described and implemented a system called Easy Email Encryption (E3) for using existing unmodified IMAP servers and unmodified IMAP mail clients with a simple E3 client to *encrypt* and *store* email. E3 uses an encrypt-on-receipt model, thus eliminating the need for senders to understand and use encryption. Keypairs are self-generated and self-signed, so they do not require complex PKI and sender-recipient coordination. E3's per-device key (PDK) scheme vastly simplifies the problem of private key handling.

Although E3 does not achieve full end-to-end security, it is usable *because* of this. E3 benefits from network effects: as E3 use increases, security increases. More importantly, encrypted emails are protected for their entire lifetime.

We provide a strong defense for emails protected by E3 before compromises of IMAP servers that requires no server or client changes which eases adoption and implementation, thus stimulating the growth of network effects. We encourage any further work in refining existing infrastructure or E3 to support email security while keeping usability in mind as a first-class citizen.

References

[1] D. Cridland A. Melnikov. 2014. *IMAP Extensions: Quick Flag Changes Resynchronization (CONDSTORE) and Quick Mailbox Resynchronization (QRESYNC).* Technical Report. IETF. 1–52 pages. https://tools.ietf.org/html/rfc7162

[2] Adam J Aviv, Michael E Locasto, Shaya Potter, and Angelos D Keromytis. 2007. SSARES: Secure searchable automated remote email storage. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual.* IEEE, 129–139.

[3] S. Brandt. 2016. *IMAP REPLACE Extension.* RFC Internet-Draft. IETF. 1–10 pages. https://tools.ietf.org/html/draft-brandt-imap-replace-02

[4] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. 2007. *OpenPGP Message Format.* RFC 4880. http://www.rfc-editor.org/rfc/rfc4880.txt

[5] Mike Cardwell. 2011. Automatically Encrypting all Incoming Email. https://www.grepular.com/Automatically_Encrypting_all_Incoming_Email. (2011).

[6] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, Russ Housley, and William Polk. 2008. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 5280. http://www.rfc-editor.org/rfc/rfc5280.txt

[7] M. Crispin. 2003. *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1.* RFC 3501. RFC Editor. 1–108 pages. https://www.rfc-editor.org/rfc/rfc3501.txt

[8] Apple Developer. 2018. SecACLCreateWithSimpleContents - Security — Apple Developer Documentation. https://developer.apple.com/documentation/security/1402295-secaclcreatewithsimplecontents?language=objc. (2018).

[9] Google Developers. 2018. KeyChain — Android Developers. https://developer.android.com/reference/android/security/KeyChain. (2018).

[10] Michael Farb, Yue-Hsun Lin, Tiffany Hyun-Jin Kim, Jonathan McCune, and Adrian Perrig. 2013. SafeSlinger: Easy-to-use and Secure Public-key Exchange. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking (MobiCom '13).* ACM, New York, NY, USA, 417–428. https://doi.org/10.1145/2500423.2500428

[11] Lorenzo Franceschi-Bicchierai. 2015. Teen Hackers: A '5-Year-Old' Could Have Hacked into CIA Director's Emails. VICE Motherboard. (October 2015). https://motherboard.vice.com/read/teen-hackers-a-5-year-old-could-have-hacked-into-cia-directors-emails

[12] Simson L. Garfinkel. 2003. Enabling Email Confidentiality Through the Use of Opportunistic Encryption. In *Proceedings of the 2003 Annual National Conference on Digital Government Research (dg.o '03).* Digital Government Society of North America, 1–4. http://dl.acm.org/citation.cfm?id=1123196.1123245

[13] Simson L. Garfinkel and A. Shelat. 2003. Remembrance of Data Passed: A Study of Disk Sanitization Practices. *IEEE Security & Privacy* 1, 1 (January–February 2003), 17–27. https://doi.org/10.1109/MSECP.2003.1176992

[14] Google. 2018. roughtime - Git at Google. https://roughtime.googlesource.com/roughtime. (2018).

[15] Tal Kopan Gregory Krieg. 2016. Is this the email that hacked John Podesta's account? CNN Politics. (30 October 2016). http://www.cnn.com/2016/10/28/politics/phishing-email-hack-john-podesta-hillary-clinton-wikileaks/

[16] D. Gustafson, M. Just, and M. Nystrom. 2004. *Securely Available Credentials (SACRED)—Credential Server Framework.* RFC 3760. http://www.rfc-editor.org/rfc/rfc3760.txt

[17] Björn Jacke. 2018. How to automatically PGP/MIME encrypt incoming mail via procmail. https://www.j3e.de/pgp-mime-encrypt-in-procmail.html. (2018).

[18] Raphael Satter Jeff Donn, Desmond Butler. 2018. Russian hackers hunt hi-tech secrets, exploiting US weakness. Associated Press. (7 February 2018). https://apnews.com/cc616fa229da4d59b230d88cd52dda51

[19] Patrick Juola and Philip Zimmermann. 1996. Whole-Word Phonetic Distances and the PGPfone Alphabet. In *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP '96*, Vol. 1. 98–101 vol.1. https://doi.org/10.1109/ICSLP.1996.607046

[20] Werner Koch and Marcus Brinkmann. 2011. *STEED–Usable End-to-End Encryption.* Technical Report.

[21] C. Newman. 1999. *Using TLS with IMAP, POP3 and ACAP.* RFC 2595. RFC Editor. 1–15 pages. https://www.rfc-editor.org/rfc/rfc2595.txt

[22] U.S. Department of Health & Human Services. 2015. System Usability Scale (SUS). (2015). http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

[23] Posteo. 2018. Email green, secure, simple and ad-free - posteo.de - Features. (2018). https://posteo.de/en/site/features#featuresprivacy

[24] Kevin Poulsen. 2014. If You Used This Secure Webmail Site, the FBI Has Your Inbox. WIRED. (January 2014). http://www.wired.com/2014/01/tormail/

[25] B. Ramsdell and S. Turner. 2010. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification.* RFC 5751. RFC Editor. 1–45 pages. http://www.rfc-editor.org/rfc/rfc5751.txt

[26] rtyley. 2018. Spongycastle by rtyley. Github.io. (2018). https://rtyley.github.io/spongycastle/

[27] Scott Ruoti, Jeff Andersen, Scott Heidbrink, Mark O'Neill, Elham Vaziripour, Justin Wu, Daniel Zappala, and Kent Seamons. 2016. "We're on the Same Page": A Usability Study of Secure Email Using Pairs of Novice Users. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 4298–4308. https://doi.org/10.1145/2858036.2858400

[28] Scott Ruoti, Jeff Andersen, Travis Hendershot, Daniel Zappala, and Kent Seamons. 2016. Private Webmail 2.0: Simple and Easy-to-Use Secure Email. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 461–472. https://doi.org/10.1145/2984511.2984580

[29] Scott Ruoti, Jeff Andersen, Daniel Zappala, and Kent E. Seamons. 2015. Why Johnny Still, Still Can't Encrypt: Evaluating the Usability of a Modern PGP Client. *CoRR* abs/1510.08555 (2015). arXiv:1510.08555 http://arxiv.org/abs/1510.08555

[30] Scott Ruoti, Nathan Kim, Ben Burgon, Timothy Van Der Horst, and Kent Seamons. 2013. Confused Johnny: when automatic encryption leads to confusion and mistakes. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*. ACM, 5.

[31] Autocrypt Team. 2018. Autocrypt 1.0.1 documentation. https://autocrypt.org/. (2018).

[32] The Washington Times 2008. Hacker wanted to 'derail' Palin. The Washington Times. (September 19 2008). http://www.washingtontimes.com/news/2008/sep/19/hacker-wanted-to-derail-palin/

[33] Tutanota. 2018. Secure email: Tutanota makes encrypted emails easy. (2018). https://tutanota.com/

[34] Zack Whittaker. 2015. Servers of email host used in US school bomb threats seized by German police. (December 2015). http://www.zdnet.com/article/email-providers-servers-seized-by-german-police-admin-forced-to-turn-over-encryption-keys/

[35] Alma Whitten and J Doug Tygar. 1999. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0.. In *Usenix Security*, Vol. 1999.

[36] WikiLeaks. 2018. WikiLeaks — Sony Archives. WikiLeaks. (2018). https://wikileaks.org/sony/emails/

[37] Wikipedia. 2018. Gmail. Wikipedia. (2018). https://en.wikipedia.org/wiki/Gmail#Storage

[38] Robert Windrem. 2016. Payback? Russia Gets Hacked, Revealing Putin Aide's Secrets. NBC News. (27 October 2016). http://www.nbcnews.com/storyline/ukraine-crisis/payback-russia-gets-hacked-revealing-putin-aide-s-secrets-n673956