Why Joanie Can Encrypt: Easy Email Encryption with Easy Key Management

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

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

Jason Nieh
Columbia University
New York, NY
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 moving a single private key between devices, we implement per-device key pairs: only public keys need be synchronized via a simple verification step. Compromising an email account or server only provides access to encrypted emails. We implemented this scheme on several platforms, showing it works with PGP and S/MIME, is compatible with widely used mail clients and email services including Gmail, 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

ACM Reference Format:
John S. Koh, Steven M. Bellovin, and Jason Nieh. 2019. Why Joanie Can Encrypt: Easy Email Encryption with Easy Key Management. In *Fourteenth EuroSys Conference 2019 (EuroSys '19), March 25–28, 2019, Dresden, Germany*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3302424.3303980

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 [23], the 2016 email hack of one of Vladimir Putin's top aides [44], the email leaks of former Vice President candidate Sarah Palin and CIA Director John Brennan [12, 39], and other similar cases [20]. 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 [30, 41], or compromise them, such as in the Sony Pictures email leaks [43].

The common thread is that a compromise exposes the *entire history* of affected users' emails after a single breach. With the explosive growth in cloud storage, it is easy to keep gigabytes of old emails at no cost. Gmail's massive storage capacity—up to 15 GB for free, or 30 TB for paid options [16]—opens 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 secure email models are effective against attackers but rarely used. Examples include Pretty Good Privacy (PGP) [3], and Secure/Multipurpose Internet Mail Extensions (S/MIME). Both are too complicated for most users because all email correspondents must comprehend public key cryptography. The current paradigm places too much of a burden on senders who must correctly encrypt emails and manage keys [36, 42]. The result is even technical users rarely encrypt their email. End-to-end encryption for email seeks absolute security at the expense of usability, creating a chasm between absolute security via encrypting all emails via PGP or S/MIME, and protecting no emails at all.

We introduce an approach to encrypted email that addresses the gaping void between unusable but absolute security, and usable but no security. We change the problem from sending encrypted emails to *storing* them since it is a user's *history* of emails that is most tantalizing to attackers. Our goal is to mitigate the attacks often publicized in the news where email account credentials or servers are compromised. The attackers have access to emails stored on servers but not 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 the affected emails would be 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 safeguards any emails received prior to a compromise. Furthermore, such a defense

must be usable for non-technical users, and compatible with 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. The onus is no longer on the sender to figure out how to use PGP or S/MIME. Instead, email clients automatically encrypt received email without user intervention. E3 protects all emails received prior to any email account or server compromise for the emails' lifetime, with threat models similar to those of 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 if 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 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.

E3 is easy to implement and use. We have implemented it for multiple environments, including retrofitting existing Android mail clients with E3 for use with mobile devices, implementing an extension for the Google Chrome web browser to use E3 as a Gmail web client, and implementing a daemon-like Python client that allows users to use existing unmodified mail clients. We tested that the Android and Python prototypes work with popular email services, including Gmail, Yahoo! Mail, and AOL Mail. We also quantified the performance of E3 on Android. Our measurements 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 [14]; 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, a user installs an E3 client that is either a separate app or a full mail client. The latter may support E3 natively or via an add-on. E3's setup is similar to a normal mail client which asks for the user's email service and its credentials. If we assume a user uses only one device to access email, then once that device's E3 mail client is setup, the client will begin encrypting all email on receipt. The user then continues using whatever email client he wants exactly as before, including sending and receiving email, except that the E3 client transparently encrypts emails on receipt. Mail clients that support encrypted emails identify them with visual indicators to avoid being *too* transparent [36] as it should be obvious whether an email was encrypted or not.

Modern email users often use multiple devices to access email, and E3 is specifically designed for and encourages users to configure multiple devices with E3. To do so, users participate in a simple, brief, and platform-independent two-way verification process for each new device as summarized in Figure 1. Suppose that a user's initial E3 client is on his smartphone, and now he wants to configure E3 on his laptop

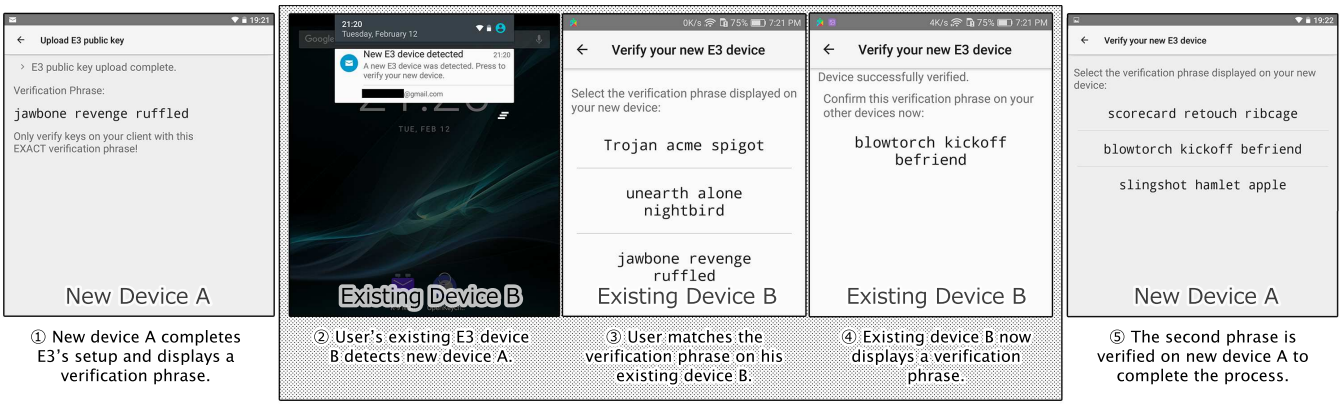

Figure 1. E3's two-way verification process.

computer. The user wants the smartphone and laptop clients to trust each other with email access, so they participate in a two-way verification. The user performs the E3 client setup process on the laptop, except it will show the user a verification phrase at the end. The smartphone client detects the laptop's client and prompts the user with a choice of several phrases on the smartphone and asks the user to select the one that was displayed on the laptop. After selecting the correct one, the user repeats to process with the laptop and smartphone swapped; the smartphone displays a verification phrase which the user must select correctly on the laptop. This completes the two-way verification, and the smartphone and laptop now trust updates from one another. If the user wishes to add a third device, say a tablet, he performs the two-way verification process with any of the previously configured devices. If he verifies the tablet on his smartphone, then his laptop can transitively trust the tablet via the phone.

If the user does not select the correct verification phrase within a time limit, the verification process is canceled and the user will need to restart the E3 verification process on the new device. When the user succeeds in verifying a new device, the user is informed that it will take some time for any previously encrypted emails to become readable on the new device. The reason is because these emails need to be re-encrypted so that the new device can read them.

Importantly, users rarely set up new devices or mail clients, so re-encrypting emails is an uncommon cost. Adding a new device generally happens in the following situations: (1) a user replaces an existing device, or (2) a user obtains an entirely new device. If a replacement, then in many cases the old device's data is cloned to the new device so that neither verification nor re-encryption is necessary. Case (2) is an uncommon occurrence, but a new device means the user will need to verify it and re-encrypt emails; however, any future replacements will fall under case (1).

When a new device is added, the clients on all previously added devices display a notice to the user that his emails are being re-encrypted. The user has the option to cancel this process and return the emails to their original state. Upon cancellation, the 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. A user removes a device by deleting it by name from any device configured with E3. The remaining clients then re-encrypt all email to exclude the deleted device.

A user may occasionally no longer be able to use a device, because it has been damaged or is no longer operational. If the user has multiple E3 clients as would commonly be the case for users that have multiple devices to access email, he can still access his E3 encrypted email on his remaining device(s). If the user only had one device with E3, a backup of the old device's data can be simply cloned to a replacement device to regain access to email on the replacement device. With mobile devices which are more easily damaged, backups are increasingly common. If it is desired to support users who use only one E3 mail client device that is never backed up, E3 can provide the user with a recovery password which the user must save by printing it out or recording it somewhere safe. Users use the recovery password on a new E3 client to access to their emails. No recovery password is needed for users who use multiple devices unless they fear they may lose all of them simultaneously.

While most mail clients support encrypted email, one exception is web browser clients such as the Gmail website and other webmail services. Browser add-ons for encrypted mail exist (and we have also written one), and it is reasonable to expect native browser support if the demand is great enough. For now, web browser extensions can integrate with webmail services to decrypt E3-encrypted email.

E3 assumes that email should only be accessible from trusted devices. Given the ubiquity of mobile devices and that most users use them for accessing email [37], this assumption is quite reasonable for modern users. E3 is not compatible with using untrusted computers such as those at an Internet cafe, nor should it be if users care about their email privacy given that such computers may be compromised. Attempts to use such untrusted computers to read email will not work; they will only provide access to encrypted emails.

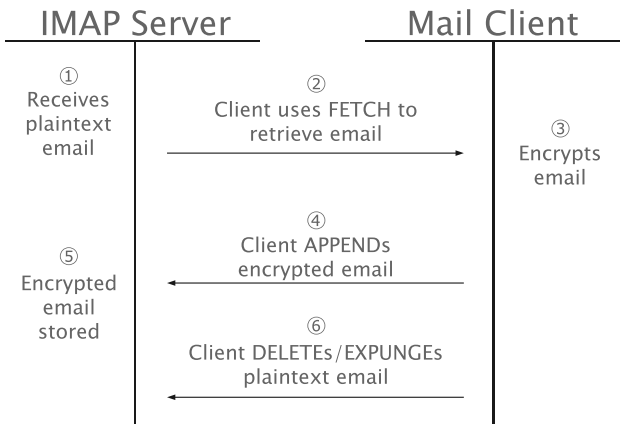

Figure 2. Communications between an E3 mail client and an IMAP server to encrypt email.

4 Architecture

The E3 architecture consists of two main components, an encrypt on receipt mechanism and a per-device key (PDK) architecture. For simplicity, we first describe the encrypt on receipt mechanism using one E3-enabled device, then describe how multiple devices are supported using the per-device key (PDK) architecture.

Figure 2 presents a high-level view of E3's encrypt on receipt mechanism. An 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) [8]. S/MIME implementations rely on X.509 certificates and the S/MIME standard as documented in RFC 5280 [7] and RFC 5751 [31]. PGP implementations follow the OpenPGP standard in RFC 4880 [3].

4.1 Keypairs without 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 other people. They are self-generated and self-signed, and require no PKI for the user to understand. Previous work [42] 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.

4.2 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 2 shows how these four IMAP commands encrypt email on receipt with existing IMAP servers. E3 is 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 works on any IMAP message. It does not matter what mailbox or folder the message is in. The same process is even applied to a user's copies of his sent emails which are appended to the IMAP server (these appear as "Sent" emails to users). All these IMAP commands execute in the background, decoupling them from the critical path of reading email.

E3 requires multiple round-trip times (RTTs) with the server because IMAP does not support message replacement. Optimizations may be possible in the future. The proposed REPLACE command [2] substitutes for the APPEND, DELETE, and EXPUNGE commands. This RFC extension is not yet supported. The REPLACE command would eliminate the multiple RTTs associated with DELETE and EXPUNGE thereby significantly improving performance when replacing many small emails. This is because RTTs have a constant cost that dominates the brief time it takes to encrypt and replace small emails. In contrast, the RTTs account for a small percentage of the total time for processing large emails and are unnoticeable.

E3 uses approaches similar to existing IMAP clients in dealing with race conditions since multiple clients may try to encrypt the same message which could result in duplicated encrypted emails. Currently, the blessed way of achieving pseudo-atomicity when modifying IMAP messages is to use the IMAP CONDSTORE extension [26]. 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 E3 client which wishes to encrypt a message adds a flag (either `\Flagged` or `\E3Encrypting` depending on custom flag support) to a message using the UNCHANGEDSINCE modifier with the IMAP STORE command so that it will only succeed if the message has been unchanged; this also updates the mod-sequence value of the message, so any other clients who try to issue the same command will fail since the message was already modified.

The flag and mod-sequence value act like a lock, thereby alerting other clients that this message is being encrypted. Then, the client with the lock can issue IMAP commands

without racing others. One issue is the client may crash before it completes its work and leave a dangling lock. A basic solution is to use a heuristic based on a message's received timestamp. A client periodically scans the mailbox for messages with the `\E3Encrypting` flag, and based on the timestamp heuristic, determines if too much time has passed since each message was received. For example, if a message is unencrypted for three hours since it was received but has the `\E3Encrypting` flag, the client may obtain the lock on the message and encrypt it.

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 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.

4.3 Ciphertext Format

E3 uses the widely supported OpenPGP message or S/MIME Enveloped-Data formats depending on client preference. While E3 can be implemented as a full standalone mail client, it can also be implemented as a program that provides just the encrypt on receipt mechanism. Users can then use existing unmodified mail clients that support S/MIME, including Apple Mail, Mozilla Thunderbird, and Microsoft Outlook, to access E3 mail in S/MIME format, assuming the E3 private key is available on the device to both the encryption program and the existing mail client. The same holds for PGP.

These formats only encrypt the body text, so 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 supports re-encrypting existing encrypted email to a newer crypto standard.

E3's encryption does not interfere with spam filters. Spam filters often exist either on servers or clients. When they are on the server, such as with Gmail, the mail service filters 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.

4.4 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 locally in the clear, many mail clients that support encryption 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 is to store ciphertexts locally, but this provides no real benefits since the key is also stored locally, and would also interfere with local searching.

A limitation of encrypted email schemes is that unmodified email servers cannot search the body content of encrypted emails. (Headers, including the `Subject:` and other metadata fields, are searchable.) If the server can be modified, SSARES [1] is a scheme for searchable encrypted email without access to private keys, making it compatible with E3's threat model. 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 often 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 or requested, which would provide the exact same search capability with or without E3.

4.5 Key Management, Migration, and Recovery

E3 eliminates manual public key exchanges. This simplifies the key management by removing half of it. What remains is the problem of private keys when using multiple devices. 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 call this the per-device key (PDK) scheme as depicted in Figure 3. With PDK, a user does not need to move any private keys among his devices. Instead, each of his clients automatically makes available its *public key* to his other devices. The result is that any E3 client can then encrypt the user's emails using the public keys from all of his devices. Consequently,

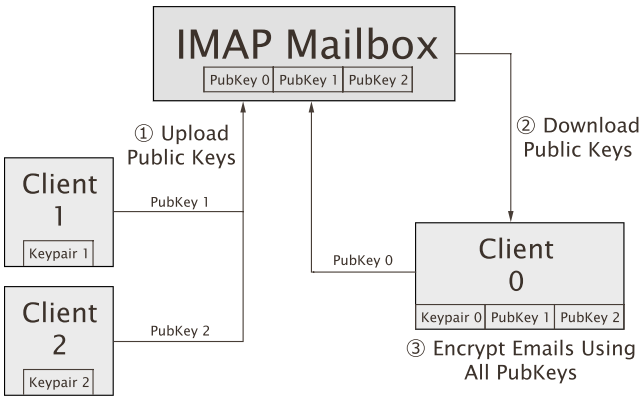

Figure 3. The per-device key (PDK) architecture.

any of a user's multiple E3 clients can encrypt emails while making them readable on any other client. The principle is similar to when a traditional PGP or S/MIME user encrypts an email to multiple people. The email is not encrypted multiple times for each public key, but is encrypted only once using a symmetric key which in turn is encrypted to each public key. E3 takes this paradigm and applies it in a new way by encrypting emails on receipt using every verified public key belonging to the user. When a new key is added, clients re-encrypt already-encrypted emails to the new keys.

PDK's primary features are as follows:

1. Private keys never move or leave a device.
2. A private key is "revoked" by re-encrypting emails to all public keys except for the revoked one.
3. Private key recovery, normally mitigated with private key backups, is reduced to device data backup which is easier. As long as one device is available, emails can be decrypted without a recovery process.
4. Public keys are automatically distributed using the email account.
5. Keys are self-generated and self-signed so users can freely add new devices.
6. Private keys use local secure storage when available without relying on a user password so there is no password to target in phishing attacks.

E3 clients upload their public keys to the mailbox as ordinary emails with the keys as attachments. Other E3 clients detect these key emails and store the public keys locally. Table 1 shows custom MIME headers used in E3 key emails to support PDK. Invalid or missing headers (when they are required) cause a key email to be rejected immediately. The concert of these headers is used to support key verification.

Public keys in the user's mailbox cannot be blindly trusted. Clients must securely confirm whether a new public key really belongs to the user, and ideally, the method to do so should be compatible with any kind of device whether a desktop or mobile one. The following solution satisfies these requirements. A given client periodically scans for

Header	Description
X-E3-NAME	A custom name for this E3 public key.
X-E3-VERIFICATION	The verification phrase as a space-separated string.
X-E3-TIMESTAMP	The signed timestamp of when this key was uploaded.
X-E3-DIGEST	The digest (fingerprint) of this E3 key.
X-E3-RESPONSE	The digest of the key that this key is in response to.
X-E3-KEYS	The public keys known to the uploader.
X-E3-DELETE	The public key deleted from the uploader.
X-E3-SIGNATURE	Signature of all fields using the uploader's private key.

Table 1. Custom headers in uploaded E3 key emails.

new keys, and as a first heuristic, ensures that the sender (i.e., the "From:" field) of any detected key email 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 spoof this field or gain access to the email account and upload a malicious key with the correct sender address. We therefore augment this check by requiring temporal proximity and a two-way verification step. 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 will expire.

Temporal proximity relies on verified and signed timestamps. A newly configured client uploads its public key along with a signed timestamp obtained from services such as Roughtime [15], and existing clients verify if the timestamp is within the allowed time window and trustworthy. For example, a 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, clients 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.

E3 also uses a two-way verification process with a verification phrase that is easy for humans to recognize and match. When a new client uploads its key, it adds a randomly generated verification phrase to the key email which is prominently displayed. The user then needs to confirm this verification phrase on one of his existing E3 clients. Once he completes the verification on any existing client, it will display a second verification phrase. The user then needs to confirm this second phrase on his new client to complete the two-way verification.

The catch is that when the user confirms a verification phrase, it must be selected from among two randomly generated incorrect phrases. The user must select the correct verification phrase in order to verify the key. This multiple choice confirmation reduces the chances of a user accidentally accepting a key that isn't his. The words in the phrases are selected from a curated pool such as the PGP Word List [21].

As shown in [11], 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 use a recognizable but randomly selected or generated image. Further research is needed to better understand what kinds of strings or images real users can correctly recall and verify while making minimal errors.

To concretely visualize how adding and deleting E3 clients works, we will describe the one device, two device, three device, and n device cases for PDK. To represent uploaded key emails, we use the notation $KeyEmail_d(Key_d, \{h\})$ where d is the device which uploaded the key email, Key_d is the public key of the uploader, and $\{h\}$ can be any of the values shown in Table 1 with X-E3- removed for spacing reasons; we also elide the required VERIFICATION and TIMESTAMP headers but note that they are necessary in each $KeyEmail$ we describe. To denote what public keys a given device d knows about, we use $d[Key_{d0}, Key_{d1}, ..., Key_{dn}]$.

One Device. Since there are no devices to synchronize keys with, a user simply sets up an E3 client on his single device and begins encrypting emails on receipt.

Two Devices. Let us consider two devices, A and B, where A is a device with E3 already configured on it, and a user wants to add B to his E3 ecosystem. Thus, the initial state of knowledge is $A[Key_A]$ and $B[Key_B]$. The user sets up an E3 client on B and it uploads $KeyEmail_B(Key_B, \{\})$, then shows the user a verification phrase. Now device A detects $KeyEmail_B$ and requests the user to verify it with the phrase shown on device B. If the user succeeds, device A now knows device B's public key, but since $KeyEmail_B$ did not contain X-E3-RESPONSE, device A knows it needs to upload its own set of keys so that device B can learn about existing public keys. Device A therefore uploads

$$KeyEmail_A(Key_A, \{\text{RESPONSE}[Key_B], \text{KEYS}[Key_A, Key_B], \text{SIGNATURE}_A\}).$$

A then displays its own verification phrase to the user, which he must verify on device B after it detects $KeyEmail_A$. If this second verification succeeds, now both devices A and B know about their public keys and can trust future updates from each other. The final state is $A[Key_A, Key_B]$ and $B[Key_B, Key_A]$.

Three Devices. The same process for two devices holds for adding a new third device C because A and B trust each other, so if C is verified and added to A, A will upload

$$KeyEmail_A(Key_A, \{\text{RESPONSE}[Key_C], \text{KEYS}[Key_A, Key_B, Key_C], \text{SIGNATURE}_A\})$$

which B trusts because of the signature, so B can automatically add Key_C to itself. Then once the user does the response verification of A on C, C will trust A as well and can add Key_B. So the final state is $A[Key_A, Key_B, Key_C]$, $B[Key_B, Key_A, Key_C]$ and $C[Key_C, Key_A, Key_B]$.

N Devices. Now consider a user who has built up his E3 ecosystem over time and has $N-1$ devices already synchronized with each other, and now he wishes to add device N. The user completes the two-way verification process with device N and any device K in $0, ..., N-1$. Then K automatically distributes N's public key to every other device by leveraging transitive trust because the other devices already trust K. Since K is manually verified on device N by the user, N can trust the keys that K provides.

Clients must re-encrypt all emails for new public keys, but a user may wish to undo adding a new device. If the user stops and reverses the re-encryption process, the client reprocesses the emails it re-encrypted, and re-encrypts them again to the original set of keys. However, the now-defunct key must be revoked first.

The general case of revocation is done via an advertised deletion. When a user revokes a client, he deletes the key by name from any client's list of keys. The client which performs the deletion announces this by uploading a signed key email with the X-E3-DELETE header so that other clients can also exclude the revoked one.

PDK achieves a streamlined key verification process where, for every newly added key, the user only ensures that the verification phrase matches the one he recognizes two times. This is in contrast to key verification for end-to-end encrypted email which often relies on confusing public key fingerprint matching, QR code scanning which is unavailable without a camera, and understanding of PKI. Although E3 keys can be verified with these techniques, the higher guarantee (and difficulty) they provide is unnecessary given the unique environment in which E3 operates. Another issue with end-to-end encrypted 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. As a side note, advanced users may prefer fingerprint matching or QR code scanning. These are only available as an advanced option that is not enabled by default.

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. As discussed in Section 3, for these users, a backup of the old device's data can be simply cloned to a replacement device to regain access to email on the replacement device. For users that only one E3 mail client device that is never backed up, PDK key recovery uses the traditional method of encrypting the user's private key with a password, presented as a "recovery key" to users, 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) [18].

PDK is also compatible with re-encrypting emails to future-proof them against changes in crypto standards. Algorithms age, so ciphers and key sizes that are secure today may not be in the future. PDK supports this use case since a user can generate and add new keys while deleting old keys at will.

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.

4.6 E3 Configurations

While we have assumed that all E3 clients encrypt on receipt and perform PDK, it is possible to also configure some E3 clients to only perform PDK when using multiple devices. At least one E3 client needs to encrypt on receipt to protect a user's email. Clients which only perform PDK configure a user's device to decrypt emails and do not encrypt emails. An example is a one-time use app or add-on which configures a user's existing, unmodified mail client with an E3 private key. These clients *only* perform the key management functionality described in Section 4.5 and none of the encryption, and are a strict subset of E3 clients which do encrypt.

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 encrypted email, E3 protects sent and received mail assuming all correspondents use E3. Senders can encrypt their sent email copies as stored on their IMAP server. Unlike end-to-end encryption, which *requires* 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 encrypted 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 encrypted 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 emails that arrive via unprotected SMTP connections. That said, attackers tapping such backbone links is out of scope for E3 and in general is difficult for any party but an intelligence agency.

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 assuming all email correspondents use it. Nevertheless, 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 is a minuscule percentage of total emails. New emails are important, but it is clear that encrypting the majority of emails is better than none.

Email account compromise happens in many ways but it is 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 even though it does not password-protect private keys since the device is assumed to be secure (it is better to rely on OS level protections such as seen in Apple Mail and Autocrypt [38], and also there is now no password for an attacker to phish). The critical aspect is that E3 makes informed decisions about private key storage and management based on the user's platform and device, so users are never requested to manage their private keys in contrast to PGP and S/MIME which require a user to actively manage and move around a private key. Thus, (non-technical) users have no knowledge of where the E3 private key is stored. This latter intrinsic property of E3 also raises the bar for an attacker to trick a user into providing his private key since the user does not know where it is. Attackers would therefore need to provide detailed instructions unique to platform and device for users to find the private key.

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. In general, most solutions assume that a user has a single keypair which is either copied to all his devices, or carried on his person such as on a security token or USB device. We have designed PDK as a departure from these approaches. It provides a secure and usable scheme that leverages users' tendency to access email on multiple devices, and also 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 spoof the sender address or have access to the email account allowing him to craft legitimate emails with the correct sender. 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 needs to also guess the correct verification phrase. An attacker with access to the mailbox could wait for the user to upload a new key, duplicate the key email but attach his malicious key instead, and delete the real key email. This would allow the attacker to construct a key email with the correct verification phrase, and this may go unnoticed by the user and his other E3 clients. However, this attack requires immediate temporal proximity, i.e., as soon as the user uploads a new key, and moreover, the client that performed the key upload can detect this attack even if other clients cannot. To do this, the uploading client polls the server to see if the key email it uploaded was deleted or moved, or if another key email with the same phrase was uploaded. The client can distinguish the real email from a fake one in any case simply by referencing the real key email's IMAP UID which is generated by the IMAP server, not the client. As soon as the client detects an issue, it warns the user that an attack may be occurring.

Another possible attack is to try to exploit E3's automatic public key distribution approach by either trying to propagate a malicious key to valid clients, or trying to delete valid clients. To propagate a malicious key addition or deletion, an adversary could upload a fake key email for either case. A fake key addition email would not be verified by a user, and thus the attack would fail. A fake key deletion email would not be accepted by any valid clients because the signature (X-E3-SIGNATURE) would be incorrect.

An adversary may 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, clients rate limit requests to add new keys and show a warning to the user. As a final measure, clients also immediately discard 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 servers and devices that are malicious from the beginning as out of scope. E3 cannot protect against an IMAP server that is run by a dishonest service provider. This then begs the question of whether popular email services can be trusted. As a case study, Google's retention policy [17] states that when a user requests a deletion, Google immediately begins deleting that data from all its systems, but it may take some time for the data to be completely removed from every internal Google server. At the least, the data is no longer accessible from user-facing interfaces such as Gmail thus preventing any external adversaries from gaining access to deleted emails. Google clearly states that it does delete data completely, so if it were to do otherwise, it would be subject to US law [5, 6] which prohibits "deceptive practices" by any entity engaging in commerce. Similar laws apply in other regions as well.

E3 also does not protect against compromise of the user's devices or mail clients, but neither does end-to-end encrypted email. Similarly, if a user's device is stolen, E3 cannot 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

To demonstrate that E3 is easy to implement, we built four different E3 prototypes for various platforms: a K-9 S/MIME client, a K-9 PGP client, a Python encryption client, and a Google Chrome extension.

We implemented E3 in K-9 Mail, a popular open-source Android mail client, using S/MIME. K-9 Mail's developers by design include no crypto libraries and offload crypto to separate crypto provider applications. However, K-9 has no S/MIME support since no such provider for S/MIME currently exists. Our K-9 S/MIME implementation therefore includes the Spongy Castle [32] crypto library and performs

all key generation and management on its own. K-9 S/MIME represents a worst case scenario where nearly email crypto functionality is implemented from scratch. Excluding third party libraries such as Spongy Castle, which adds 8.6K lines of code (LOC), our K-9 S/MIME implementation only added roughly 2.5K LOC. The entire K-9 codebase is around 210K LOC, excluding XML code which adds another 200K LOC, thus suggesting that E3 comparatively represents a modest amount of complexity. We also implemented a more optimized E3 K-9 S/MIME version by replacing Spongy Castle with precompiled OpenSSL libraries to leverage hardware encryption support on Android devices. While there was no change to the E3 code needed, the OpenSSL libraries are substantially larger than Spongy Castle, roughly 390K LOC. Although Android includes its own OpenSSL as a system library, the version included is heavily modified and strips many features including the S/MIME functions required for our implementation.

We also implemented E3 in K-9 Mail using PGP by relying on the OpenKeychain Android app, which is both a keychain and crypto provider. K-9 offloads all PGP and key operations to OpenKeychain which exposes an external cross-application API, so it was not necessary to add a crypto library to K-9. We modified K-9 Mail and OpenKeychain to support E3. We added an API call to OpenKeychain (OpenPGP-API) for storing E3 keys, and changes to make OpenKeychain 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. Our E3 K-9 PGP client had nicer UI features compared to the K-9 S/MIME client, adding 3.3K LOC, with much of the additions being UI boilerplate code. Our changes to OpenKeychain were about 250 LOC, while the entire OpenKeychain codebase is 590K LOC, excluding 124K LOC of XML. Without the need for additional crypto libraries, the total amount of additional code to support E3 was only 3.6K LOC out of the over a million LOC required for K-9 and OpenKeychain.

We implemented a Python E3 daemon for Windows, Linux, and macOS that generates an E3 keypair and encrypts on receipt, but does not currently automatically add the private key for use with existing mail clients; users must manually perform this step, so the daemon is currently intended for use by more technical users. The implementation is only 1K LOC. We sketch out what automatically adding the key to mail clients 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 [9]. On Android, the KeyChain API [10] stores 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 implemented a Google Chrome extension to interface with the Gmail website and support reading E3 encrypted emails and key management. This extension was a proof of concept to show that reading E3 email on web mail clients is possible and practical, but does not perform encrypt on receipt. It is about 750 LOC plus 7.5K LOC for external Javascript libraries for crypto and other important functionality. The extension requests access to the user's Gmail API to process raw emails instead of scraping Gmail's DOM. When a user loads an encrypted email in Gmail, the extension checks if it can be decrypted, fetches the email, decrypts it, and injects its contents into the page. The extension uses the Gmail API to also perform the necessary key management functionality for E3. However, Google Chrome by design provides no secure storage whether for extension data or browser cookie data. It instead relies on its own and OS security features to protect sensitive data. Thus, we store the E3 keypair in Chrome's local storage.

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

Table 2. Tested servers and their compatibility with E3.

7 Experimental Results

We verify that E3 works with existing IMAP services, measure its performance overhead, and evaluate its usability with real users. We used K-9 S/MIME for performance testing, and K-9 PGP for usability testing.

7.1 Compatibility and Interoperability

To verify that E3 is compatible with existing IMAP and S/MIME systems, we tested our prototypes on several of the most popular commercial and open-source email servers. Table 2 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

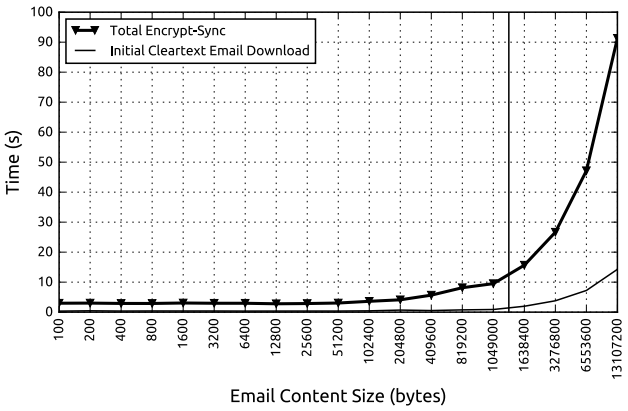

Figure 4. Time spent for the one-time "encrypt/synchronize" compared to the cleartext download. Points right of the line are emails with a JPG.

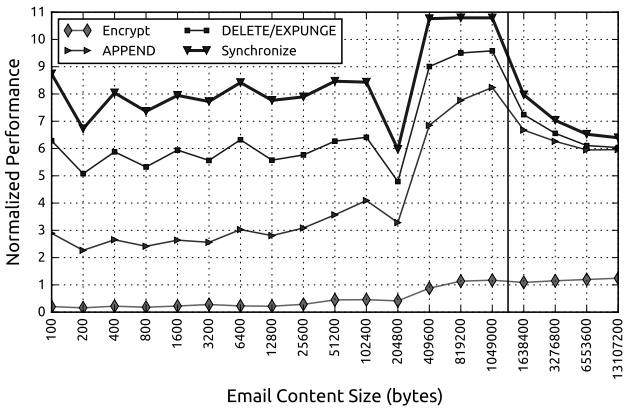

Figure 5. Normalized time spent for the one-time "encrypt/synchronize" relative to the cleartext download (not pictured). Points right of the line are JPG emails.

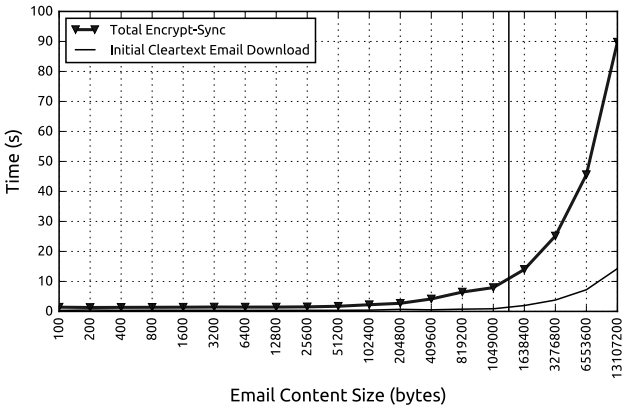

Figure 6. Expected time for the one-time "encrypt/synchronize" with REPLACE compared to the cleartext download. Points right of the line are JPG emails.

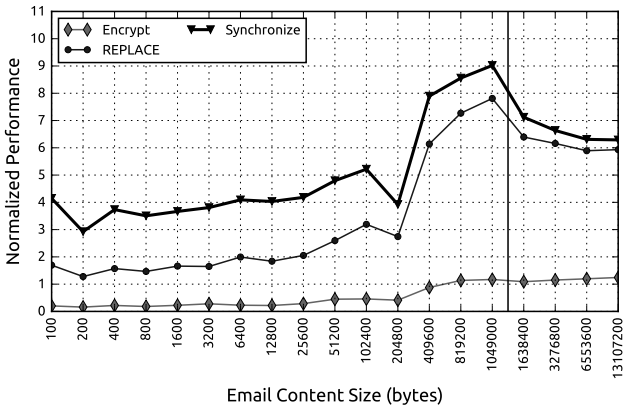

Figure 7. Normalized expected time for "encrypt/synchronize" with REPLACE relative to the cleartext download (not pictured). Points right of the line are JPG emails.

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 E3 K-9 S/MIME 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 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 4 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 4 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 4 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 5 shows the same encrypt/sync measurements as Figure 4, 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.

IMAP currently does not support replacing a message in a single operation. The proposed IMAP REPLACE extension [2] would eliminate the DELETE/EXPUNGE, so REPLACE's overhead will resemble APPEND alone. We approximate this by taking Figure 4 and removing DELETE/EXPUNGE. This leaves Encrypt and APPEND as visible in Figure 6. Normalized performance can be seen in Figure 7. Like Figure 5, Figure 7 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. The main difference with E3 versus a regular mail client involves the initial setup of E3 before a user can start sending and receiving emails. We therefore focus on the usability of the mail client setup.

We administered an IRB-approved[1] user study with nine participants who used and compared our E3 K-9 PGP client versus an unmodified K-9 client with and without PGP. Participants used three devices we provided to them in each session: a Nexus 7 Android tablet, a Huawei Honor 5X, and a Samsung Galaxy S7. Each user was also supplied with an empty Gmail account. All participants had some experience with mobile device mail clients. They consisted of six non-technical users aged 31 to 60, and two technical users aged 21 to 30. The non-technical users all worked in blue-collar occupations or were self-employed. The technical users worked or had worked in technology, and one was a Ph.D student who specializes in computer security and mobile computing. Both had never used PGP but were familiar with its design.

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 minute session comprised three 20 minute sessions, each devoted to using vanilla K-9, E3 K-9 PGP, or K-9 with PGP. During each 20 minute session, we instructed the user to configure the selected mail client with a Gmail account then send and receive emails to obtain tax forms from three separate people. More specifically, the user first set up the respective email client with an empty Gmail account on one of the three mobile devices, requested a tax form from the first person, and verified the response was encrypted (for E3 and PGP) by checking for the visual encryption flag indicator on the K-9 client. Upon successfully completing the first email exchange, users then configured a second device with the same Gmail account, which essentially tested E3 and PGP's key management. E3

[1]The Institutional Review Board (IRB) is the United States' approach to an ethics committee that oversees human subjects testing.

Soln.	Count	Mean	Std. Dev	Min.	Q1	Median	Q3	Max
K-9	8	81.25	14.88	62.50	70.00	76.25	96.25	100.00
E3	8	74.38	18.84	47.50	60.00	76.25	90.63	97.50
PGP	8	41.25	12.03	27.50	27.50	45.00	50.00	57.50

Table 3. System Usability Scale summarized scores.

required completing the two-way verification to distribute E3's public keys, and PGP required transferring their single private key. If successful, users then requested the tax form from the second person. This was then repeated for the third device and person.

We provided users with a visual setup guide for both E3 and PGP, and they could ask the study coordinator for help with specific errors (for example, if they unknowingly made a typo or didn't know how to go to the home screen), but we provided no in-depth help. Our reasoning was to strike a balance between providing consistent help to all users for both solutions while also preventing cases where users would get stuck on a simple mistake unrelated to the study goals. 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) [27], an industry-standard questionnaire also used in many similar studies [33–36], for the system they had just used. At the end of the study, participants completed 14 additional survey questions specific to our research, and a final free-form question requesting any comments. To ensure that participants actually understood each email solution they used, the study coordinator explained each system prior to completing the 14 additional survey questions.

The summarized SUS scores are presented in Table 3. A higher score means better usability. The results for K-9 and E3 were quite close while K-9 PGP received remarkably low ratings. This suggests that users felt that E3 was almost as easy to use as K-9, while PGP was significantly worse. All users except the one technical user who specializes in computer security and mobile computing failed to complete K-9 PGP's tasks in the time limit even with copious help. The pain point in PGP where users struggled was the private key management when they had to transfer their PGP keypair to their other devices. On the other hand, all users succeeded in every instructed step for E3 K-9 PGP in 10 to 15 minutes. The average completion time for K-9 was 8 minutes.

We also asked users to compare the email solutions which we summarize in Table 4. Responses are on a scale of 1 (Strongly Disagree) to 5 (Strongly Agree). Subjects in general agreed that E3 was easier to use than PGP, but E3 still introduced noticeable extra setup time compared to K-9. For many of these questions, users choose a score of 3 despite giving

#	Question (1 = Strongly Disagree, 5 = Strongly Agree)	Mean	Std.	Min.	Med.	Max
31	I found it easy to use K-9.	4.50	0.55	4	4.5	5
32	I found it easy to use K-9 with PGP.	2.17	0.75	1	2	3
33	I found it easy to use K-9 with E3.	3.83	0.98	3	3.5	5
34	I could see myself using K-9 on a regular basis.	4.00	0.89	3	4	5
35	I could see myself using E3 on a regular basis.	3.83	0.75	3	4	5
36	I could see myself using PGP on a regular basis.	2.00	0.89	1	2	3
37	I thought E3 takes too long to set up each time on a new device.	2.00	0.89	1	2	3
38	I thought PGP takes too long to set up each time on a new device.	3.67	1.21	2	3.5	5
39	I thought that E3 was easier to use than PGP.	4.17	0.98	3	4.5	5
40	I thought that using the QR code scanner was harder than verifying a three word phrase.	3.50	1.22	2	4	5
41	I thought that transferring my key in PGP was harder than verifying a three word phrase in E3	4.17	0.98	3	4.5	5
42	The extra security with E3 is worth the extra steps compared to regular email.	4.17	0.98	3	4.5	5
43	The extra security with PGP is worth the extra steps compared to regular email.	2.50	0.84	1	3	3
44	The extra security with PGP is worth the extra steps compared to E3.	2.00	0.63	1	2	3

Table 4. Summarized scores for added survey questions. (Questions are abbreviated for spacing reasons.)

similar usability scores for K-9 and E3 as seen in Table 3. This suggests that another factor unrelated to usability influenced their responses to our custom questions. The most likely culprit is that most of the non-technical users did not consider themselves important enough to use encryption. They thus tended to be indifferent and responded with the middle-ground score of 3 for any question concerning the encrypted email solutions.

Free responses included saying "PGP sucks" and "I had no idea what I was doing with [PGP]." Several subjects commented on how much easier E3's two-way verification was compared to PGP's key exchange and private key import/export. Most users felt unimportant enough to use encryption, but could still see the value in having an easier to use email encryption solution for people who do handle sensitive data.

It is important to note that our user study places a large emphasis on configuring email clients, which is a relatively infrequent occurrence. Furthermore, it is not uncommon for non-technical users to ask others, technical support in the context of an enterprise organization or customer support when purchasing a device, for assistance in setting up a device. The fact that the main usability difference between E3 and vanilla email is in the client configuration and that there is no difference for sending and receiving email suggests that E3 usability is likely to be even better in practice.

We draw these conclusions: (1) E3 is easy to use even for non-technical users. (2) E3 is much more usable and intuitive than PGP. (3) 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.

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 [42]. They found that correctly sending encrypted email in an end-to-end encrypted email setting is outstandingly difficult.

Many works following "Johnny" have tried to tackle the problem of end-to-end encrypted email by attempting to make the process easier or more transparent. STREAM [13] uses SMTP/POP proxies which opportunistically encrypt email by finding keys or generating them on the fly, but key management is problematic. Verifying keys involves out-of-band commnuincation such as phone calls, and delivering keys requires users to know how to extract keys from email, install them, and use them. STEED [22] extends the IMAP standard to support transparent end-to-end encryption, but requires modified clients and servers, and does not address key management at all. Pwm [36] and Pwm 2.0 [34] attempt to make end-to-end encryption transparent by integrating with popular mail providers and relying on a third-party identity-based encryption (IBE) server to manage keys, but users authenticate to the IBE server via their email account which does not protect against compromised accounts. Confidante [24] leverages users' ubiquitous use of social media accounts to ease public key management and verification via a third-party service. It however relies on users being able to correctly identify social media accounts.

Various commercial services that provide end-to-end encryption try to address the key management directly by taking a walled garden approach. Lavabit [25], Posteo [28], and Tutanota [40] create closed platforms where the service handles all key management on its servers, but users are restricted to encrypting messages only to other users of the same platform or to redirecting recipients to the service's website to gain access to an encrypted file. Services such as Lavabit maintain master keys which could decrypt all emails, making them vulnerable to compromises and subpoenas.

Autocrypt [38] is a decentralized and incrementally deployable system for distributing public keys to support end-to-end encryption by making public key management more usable. Only clients need to be modified to support Autocrypt. Autocrypt includes the sender's public key in an email and an indicator whether the sender prefers encryption. The receiver replies in the same manner, including his public key in the email and an indicator whether encryption is preferred. Autocrypt thereafter will send encrypted email between the two parties if any of three criteria are satisfied: the sender requests encryption, the received email was encrypted, or all parties explicitly prefer encryption. Autocrypt does not encrypt all of a user's email, for example an email from someone who does not prefer encryption. Given that Autocrypt use remains limited, it may not protect a substantial portion of a user's emails if a compromise occurs. This is in contrast to E3, which will protect all of user's email before a compromise occurs. E3 could be used to complement Autocrypt, most obviously by encrypting plaintext emails with non-Autocrypt correspondents. Another critical difference is that Autocrypt eases PGP public key distribution but does not address private key management and has no solution for making it easy to read encrypted email on multiple devices.

E3's encrypt on receipt approach has been proposed using other mechanisms. Most examples modify one's Mail Transfer Agent or Mail Delivery Agent to encrypt emails before delivering them to the client [4, 19], but this is too complicated for non-technical users. Posteo [29] provides support for encrypting emails on receipt, but their approach is server-side and only works on their servers. Unlike E3, none of these approaches work with existing unmodified IMAP servers and clients, and none of them address the issue of client-side key management.

9 Conclusions

Easy Email Encryption (E3) introduces new client-side encrypt-on-receipt and per-device keys (PDK) mechanisms compatible with the existing IMAP standard and servers. E3 email clients automatically encrypt received email without user intervention, making it easy for users to protect the confidentiality of all emails received prior to any email account or server compromise. E3 uses keys that are self-generated and self-signed, and PDK makes it easy to use them to access encrypted email across multiple devices. Users no longer need to understand or rely on public key infrastructure, coordinate with recipients, or figure out how to use PGP or S/MIME. We show that E3 is easy to implement on a variety of platforms including Android, Windows, Linux, and even Google Chrome, and show that it works with popular IMAP-based email services including Gmail, Yahoo! Mail, AOL, and Yandex Mail. Our user study results show that real users, even non-technical ones, consider E3 easy to use even when compared to using regular unecrypted email clients and vastly easier to use over the state of the art for PGP. Our measurements using E3 with Gmail services show that performance overheads are modest and acceptable in practice.

Almost exactly 20 years ago, Johnny was unable to encrypt. In the current modern era, the explosive growth of ubiquitous and always-on, always-connected mobile devices has provided the necessary foundation for putting a new and usable spin on the idea of receiver-controlled encryption. Johnny could not encrypt in his time, but Joanie in the modern age certainly can.

10 Acknowledgments

Matt Blaze and Nadia Heninger contributed some early ideas that led to E3. This work was supported in part by NSF grant CNS-1717801.

References

[1] Adam J. Aviv, Michael E. Locasto, Shaya Potter, and Angelos D. Keromytis. 2007. SSARES: Secure Searchable Automated Remote Email Storage. In *23rd Annual Computer Security Applications Conference, 2007 (ACSAC '07)*. IEEE, ACSA, Miami Beach, Florida, USA, 129–139.

[2] Stuart Brandt. 2019. *IMAP REPLACE Extension.* RFC 7162. IETF. 11 pages. https://www.rfc-editor.org/rfc/rfc8508.txt

[3] Jon Callas, Lutz Donnerhacke, Hal Finney, David Shaw, and Rodney Thayer. 2007. *OpenPGP Message Format.* RFC 4880. IETF. 89 pages. http://www.rfc-editor.org/rfc/rfc4880.txt

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

[5] US Federal Trade Commission. 1914. 15 USC CHAPTER 2, SUBCHAPTER I: FEDERAL TRADE COMMISSION. http://uscode.house.gov/view.xhtml?req=granuleid%3AUSC-prelim-title15-chapter2-subchapter1&edition=prelim. (Sept. 1914).

[6] US Federal Trade Commission. 2016. Federal Trade Commission Act - Section 5: Unfair or Deceptive Acts or Practices. https://www.federalreserve.gov/boarddocs/supmanual/cch/ftca.pdf. (Dec. 2016).

[7] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and William Polk. 2008. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* RFC 5280. IETF. 151 pages. http://www.rfc-editor.org/rfc/rfc5280.txt

[8] Mark R. Crispin. 2003. *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1.* RFC 3501. IETF. 108 pages. https://www.rfc-editor.org/rfc/rfc3501.txt

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

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

[11] 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, Miami, Florida, USA, 417–428. https://doi.org/10.1145/2500423.2500428

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

[13] 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, Boston, MA, USA, 1–4. http://dl.acm.org/citation.cfm?id=1123196.1123245

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

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

[16] Google. 2019. Google One - More storage and extra benefits from Google. https://one.google.com/about. (Feb. 2019).

[17] Google. 2019. How Google retains data we collect – Privacy & Terms – Google. https://policies.google.com/technologies/retention?hl=en. (2019).

[18] Dale Gustafson, Mike Just, and Magnus Nystrom. 2004. *Securely Available Credentials (SACRED)—Credential Server Framework.* RFC 3760. IETF. 22 pages. http://www.rfc-editor.org/rfc/rfc3760.txt

[19] Björn Jacke. 2008. How to automatically PGP/MIME encrypt incoming mail via procmail. https://www.j3e.de/pgp-mime-encrypt-in-procmail.html. (Oct. 2008).

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

[21] Patrick Juola and Philip Zimmermann. 1996. Whole-Word Phonetic Distances and the PGPfone Alphabet. In *Proceeding of 4th International Conference on Spoken Language Processing (ICSLP '96)*, Vol. 1. IEEE, Philadelphia, PA, USA, 98–101. https://doi.org/10.1109/ICSLP.1996.607046

[22] Werner Koch and Marcus Brinkmann. 2011. *STEED–Usable End-to-End Encryption.* White Paper. https://g10code.com/docs/steed-usable-e2ee.pdf

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

[24] Ada Lerner, Eric Zeng, and Franziska Roesner. 2017. Confidante: Usable Encrypted Email: A Case Study with Lawyers and Journalists. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, Paris, France, 385–400. https://doi.org/10.1109/EuroSP.2017.41

[25] Lavabit LLC. 2004. Lavabit. https://lavabit.com/. (2004).

[26] Alexey Melnikov and Dave Cridland. 2014. *IMAP Extensions: Quick Flag Changes Resynchronization (CONDSTORE) and Quick Mailbox Resynchronization (QRESYNC).* RFC 7162. IETF. 52 pages. https://www.rfc-editor.org/rfc/rfc7162.txt

[27] 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

[28] Posteo. 2009. Email green, secure, simple and ad-free - posteo.de - Features. (2009). https://posteo.de/en/site/features#featuresprivacy

[29] Posteo. 2015. Help - How do I activate inbound encryption with my public PGP key? - posteo.de. (2015). https://posteo.de/en/help/how-do-i-activate-inbound-encryption-with-my-public-pgp-key

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

[31] Blake Ramsdell and Sean Turner. 2010. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification.* RFC 5751. IETF. 45 pages. http://www.rfc-editor.org/rfc/rfc5751.txt

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

[33] 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, San Jose, California, USA, 4298–4308. https://doi.org/10.1145/2858036.2858400

[34] 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, Tokyo, Japan, 461–472. https://doi.org/10.1145/2984511.2984580

[35] 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 (Oct. 2015), 5. arXiv:1510.08555 http://arxiv.org/abs/1510.08555

[36] 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 9th Symposium on Usable Privacy and Security (SOUPS '13)*. ACM, ACM, Newcastle, United Kingdom, 19.

[37] Bettina Specht. 2018. Email Client Market Share Trends for the First Half of 2018 – Litmus Software, Inc. https://litmus.com/blog/email-client-market-share-trends-first-half-of-2018. (13 July 2018).

[38] Autocrypt Team. 2016. Autocrypt 1.0.1 documentation. https://autocrypt.org/. (2016).

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

[40] Tutanota. 2011. Secure email: Tutanota makes encrypted emails easy. (2011). https://tutanota.com/

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

[42] Alma Whitten and J. D. Tygar. 1999. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0.. In *Proceedings of the 8th USENIX Security Symposium*. USENIX Association, Washington, D.C., USA, 169–184.

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

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