"Johnny, you are fired!" – Spoofing OpenPGP and S/MIME Signatures in Emails

Jens Müller[1], Marcus Brinkmann[1], Damian Poddebniak[2], Hanno Böck, Sebastian Schinzel[2], Juraj Somorovsky[1], and Jörg Schwenk[1]

[1]*Ruhr University Bochum*
[2]*Münster University of Applied Sciences*

Abstract

OpenPGP and S/MIME are the two major standards to encrypt and digitally sign emails. Digital signatures are supposed to guarantee authenticity and integrity of messages. In this work we show practical forgery attacks against various implementations of OpenPGP and S/MIME email signature verification in five attack classes: (1) We analyze edge cases in S/MIME's container format. (2) We exploit in-band signaling in the GnuPG API, the most widely used OpenPGP implementation. (3) We apply MIME wrapping attacks that abuse the email clients' handling of partially signed messages. (4) We analyze weaknesses in the binding of signed messages to the sender identity. (5) We systematically test email clients for UI redressing attacks.

Our attacks allow the spoofing of digital signatures for arbitrary messages in 14 out of 20 tested OpenPGP-capable email clients and 15 out of 22 email clients supporting S/MIME signatures. While the attacks do not target the underlying cryptographic primitives of digital signatures, they raise concerns about the actual security of OpenPGP and S/MIME email applications. Finally, we propose mitigation strategies to counter these attacks.

1 Introduction

Email is still the most important communication medium on the Internet and predates the World Wide Web by roughly one decade. At that time, message authenticity was not a major concern, so the early SMTP and email [1, 2] standards did not address confidentiality or the authenticity of messages.

Email Authenticity As the ARPANET evolved into the Internet, email usage changed. Email is now used for sensitive communication in business environments, by the military, politicians and journalists. While technologies such as SPF, DKIM, and DMARC can be used to authenticate the domain of the sending SMTP server, these are merely helpful in mitigating spam and phishing attacks [3]. These technologoies, however, do not extend to authenticating the sending *person*, which is necessary to provide message authenticity.

Two competing email security standards, OpenPGP [4] and S/MIME [5], offer end-to-end authenticity of messages by digital signatures and are supported by many email clients since the late 1990s. Digital signatures provide assurance that a message was written by a specific person (i.e., authentication and non-repudiation) and that it was not changed since then (i.e., integrity of messages). Adoption is still low[1] due to severe usability issues, but both technologies have a large footprint either in the industry (S/MIME) or with high-risk roles such as journalists, lawyers, and freedom activists (OpenPGP). One example: Debian (a volunteer group with over 1000 members) relies on the authenticity of signed emails for voting on project leaders and proposals. We thus ask: *Is it possible to spoof a signed email such that it is indistinguishable from a valid one even by an attentive user?*

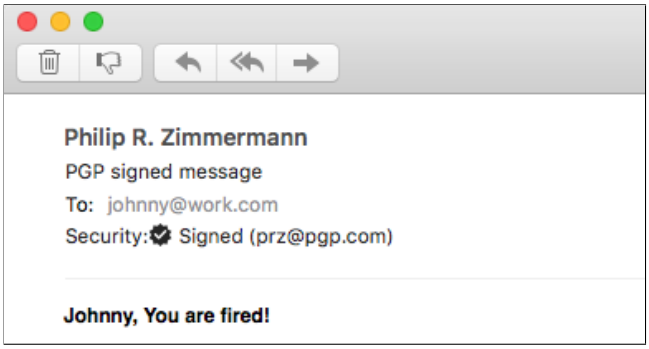


Figure 1: Screenshot of a spoofed PGP signature in Apple Mail, based on wrapping a signed email published by Phil Zimmermann.

Spoofing Valid Signatures Signature verification in the context of email is complex. For example, emails can be signed by an entity other than the sender or signed emails may be forwarded (resulting in partly signed messages).

[1]We examined OpenPGP keyservers and measured over 20k key uploads per month consistently over the last 4 years. We also found that Thunderbird reports 150k daily users (on work days) for the PGP-plugin Enigmail.

Also, signatures are an optional feature of email and therefore generally not enforced. Most importantly, the result of the verification must be presented to the user such that there is no room for misinterpretation. Any failure to do so may lead to a signature spoofing attack as shown in Figure 1. This paper describes the results of our analysis of the most widely used email clients supporting PGP or S/MIME signatures.

Attack Classes Our attacks do not break the cryptography in digital signatures, but rather exploit weaknesses in the way PGP and S/MIME signatures are verified by email clients, and how the verification outcome is presented to the user.

We define the following five attack classes:

1. **CMS attacks.** Cryptographic Message Syntax (CMS) is a versatile standard for signed and encrypted messages within the X.509 public-key infrastructure. We found flaws in the handling of emails with contradicting or unusual data structures (such as multiple signers) and in the presentation of issues in the X.509 trust chain.
2. **GPG API attacks.** GnuPG is the most widely used OpenPGP implementation, but it only offers a very restricted command line interface for validating signatures. This interface is vulnerable to injection attacks.
3. **MIME attacks.** The body of an email is conceptually a MIME *tree*, but typically the tree has only one leaf which is signed. We construct non-standard MIME trees that trick clients into showing an unsigned text while verifying an unrelated signature in another part.
4. **ID attacks.** The goal of this attack class is to display a valid signature from the identity (ID) of a trusted communication partner located in the mail header, although the crafted email is actually signed by the attacker.
5. **UI attacks.** Email clients indicate a valid signature by showing some security indicators in the user interface (UI), for example, a letter with a seal. However, several clients allow the mimicking of important UI elements by using HTML, CSS, and other embedded content.

Contributions We make the following contributions:

- We present the results of our structured analysis of OpenPGP and S/MIME signature verification in 25 widely used email clients.
- We present three new attack classes: attacks based on CMS evaluation, on the syntax of the GnuPG machine interface, and on wrapping signed content within invisible subtrees of the MIME tree.
- We adapt two attack classes of web security to the email context, namely UI redressing and ID spoofing.
- We show that our attacks bypass signature validation in about 70% of the tested email clients, including Outlook, Thunderbird, Apple Mail, and iOS Mail.

Coordinated Disclosure We reported all our attacks and additional findings to the affected vendors and gave advice on appropriate countermeasures.

2 Background

2.1 End-to-End Email Authenticity

The digital signature parts of the OpenPGP and S/MIME standards provide end-to-end authenticity for email messages. First and foremost, both technologies are configured on the endpoints and technically-versed users can therefore choose to use them independently of the email server configuration. In both standards, the keys are bound to users and thus authenticate users independently of the transport.

OpenPGP Email Signing Phil Zimmerman invented PGP (Pretty Good Privacy) in 1991 and due to its popularity, PGP was standardized as *OpenPGP* by the IETF [6]. The most popular implementation is GnuPG.[2] There are two common ways to use OpenPGP in emails. With *Inline PGP*, the email body directly contains the OpenPGP data. The MIME multipart standard is not used and the MIME type is `text/plain`. With *PGP/MIME*, the email has a `multipart/signed` MIME structure, where the signed message is the first part and the detached signature is the second part. Some email clients support PGP natively, but most (in particular Thunderbird, Apple Mail, and Outlook) need a plugin, which provides an intermediate layer between the mail client and a PGP implementation like GnuPG.

S/MIME Email Signing In 1999, the IETF published S/MIME (Secure/Multipurpose Internet Mail Extension) version 3 as an extension to the MIME standard with certificate-based cryptography [5]. S/MIME is the result of a long history of secure email protocols and can be seen as the first Internet standards-based framework to digitally sign, authenticate, or encrypt emails. S/MIME uses the *Cryptographic Message Syntax (CMS)* as its underlying container format. The signatures themselves are always CMS encoded, but the signed message can either be included in the CMS (*opaque* signature) or be transmitted as the first part of a `multipart/signed` message (*detached* signature).

2.2 Trust and Validity

Verifying the cryptographic integrity of a signature is often not sufficient. In addition to this verification, the public key

[2] W. Koch, *GNU Privacy Guard*, `https://gnupg.org/`.

that generated the signature must be connected to some actual person or entity, such as an email address, by a certificate. S/MIME certificates are issued by certificate authorities which are trusted by the email clients. It is easy for users to order S/MIME certificates and sign messages which are accepted by all clients. PGP as a product of the cypherpunk movement distrusts central authorities, so user IDs in PGP are only self-signed by default. This does not provide any protection against spoofing and puts responsibility for trust management into the hands of users and applications. In fact, no version of the OpenPGP standard defines a trust model for user ID binding signatures. Historically, users of PGP were encouraged to participate in the Web of Trust, a decentralized network of peers signing each other's user IDs, paired with a scoring system to establish *trust paths* between two peers that want to communicate. Signatures and keys are exchanged through a network of public keyservers. This approach has been found difficult to use, privacy invasive, and hard to scale, so some email clients implement their own trust model (e.g., OpenKeychain, R2Mail2, and Horde/IMP).

3 Attacker Model and Methodology

In our scenario we assume two trustworthy communication partners, Alice and Bob, who have securely exchanged their public PGP keys or S/MIME certificates. The goal of our attacker Eve is to create and send an email with arbitrary content to Bob whose email client falsely indicates that the email has been digitally signed by Alice.

Attacker Model We assume that Eve is able to create and send arbitrary emails to Bob. The email's sender is spoofed to Alice's address, for example, by spoofing the `FROM` header, a known impersonation technique which should be prevented by digital signatures. This is our default attacker model with the weakest prerequisites. It is sufficient for the UI attack class, and some CMS and GPG API attacks.

For the MIME attack class and some CMS attacks, we also assume that the attacker has a single valid S/MIME or OpenPGP signature from Alice which may have been obtained from previous email correspondence, public mailing lists, signed software packages, signed GitHub commits, or other sources. This is a weak requirement as well, because digital signatures are usually not kept secret. In fact, in many cases digital signatures are used explicitly to give public proof that some content was created by the signer.

For the ID attack class and one attack in the GPG API class, we assume that Bob trusts Eve's signatures. For S/MIME this condition always holds because Eve can easily obtain a valid certificate from a trusted CA for her own email address. For OpenPGP, Bob must import Eve's public key and mark it as valid. This is a stronger condition, but holds if Eve is a legitimate communication partner of Bob.

An overview of the attack classes and attacker models is given in Table 1. Each attack class is described in section 4 and the subscript identifies the specific attack, i.e., M_1 identifies attack 1 in the MIME attack class.

	Attacker Model		
Attack class	**Mail only**	**Need signature**	**Key trusted**
CMS (4.1)	C_3, C_4	C_1, C_2	–
GPG (4.2)	G_1	–	G_2
MIME (4.3)	–	M_1, M_2, M_3, M_4	–
ID (4.4)	–	–	I_1, I_2, I_3
UI (4.5)	U_1	–	–

Table 1: Attacker's capabilities for all test cases in each attack class.

Our attacker model does not include any form of social engineering. The user opens and reads received emails as always, so awareness training does not help to mitigate the attacks.

Methodology We define that the *authenticity of a signed email is broken in the context of an email client UA* if the presentation of a crafted email in *UA* is indistinguishable from the presentation of a "valid" signed email (either as *perfect* or *partial* forgery). Furthermore, we document cases where we could forge *some, but not all* GUI elements required for indistinguishability (i.e., a *weak* forgery).

- **Perfect forgery** (●) If a presentation is identical at any number of user interactions, regardless of any additional actions the user takes within the application, we call the forgery "perfect" (e.g., Figure 1).
- **Partial forgery** (◐) If a presentation is only identical at the first user interaction (i.e., when an email is opened and the standard GUI features are visible), we call the forgery "partial" (e.g., Figure 14).
- **Weak forgery** (○) If a presentation contains contradicting GUI elements at the first user interaction, with some *but not all* elements indicating a valid signature, we call this forgery "weak" (e.g., Figure 15).

We suspect that partial forgeries already go unnoticed by unwitting users, so we classify perfect and partial forgeries as successful attacks. Weak forgeries show signs of spoofing at the first glance. As part of our evaluation, we provide screenshots of interesting cases to illustrate the differences.

Selection of the Clients We evaluate our attacks against 25 widely-used email clients given in Table 2 and Table 3. Of these, 20 support PGP and 22 are capable of S/MIME signature verification. They were selected from a comprehensive list of over 50 email clients assembled from public software directories for all major platforms (i.e., Windows, Linux, macOS, Android, iOS, and web). Email clients were

excluded when they did not support PGP or S/MIME signatures, were not updated for several years, or 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, where required.

4 Attacks

4.1 CMS Attack Class

The Cryptographic Message Syntax (CMS, the container format used by S/MIME) is a versatile standard for signed and encrypted emails. It not only supports a broad range of use cases (e.g., multiple signers), but also copes with legacy problems like lack of software support and misbehaving gateways. This made the standard more complex; several values in a CMS object are optional, or may contain zero or more values.[3] Furthermore, two different signature formats are defined. This makes it difficult for developers to test all possible combinations (either plausible or implausible).

Opaque and Detached Signatures The CMS and S/MIME standards define two forms of signed messages: *opaque* and *detached* signatures [7] (also called *embedded* and *external* signatures). The signature is always a CMS object, but the corresponding message can either be embedded into this object or transmitted by other means.

When signing in opaque mode, the to-be-signed content (i.e., "the message text") is embedded into the binary CMS signature object via a so called `eContent` (or "embedded content") field (see Figure 2a).

In detached signatures, the `eContent` must be absent in order to signal that the content will be provided by other means. This is what the multipart/signed structure does; the email is split into two MIME parts, the first one is the content and the second one is the CMS signature without the `eContent` field (see Figure 2b).

eContent Confusion (C_1) A confusing situation arises when the `eContent` field is present *even though* the multipart/signed mechanism is used. In this case, the client can choose which of the two contents (i.e., either the opaque or detached contents) to verify *and* which of the two contents to display. Clearly, it is a security issue when the verified content is not equal to the content which is displayed.

The "eContent Confusion" allows perfect forgeries of arbitrary signed emails for a person from which we already have a signed email. Because opaque signatures can be transformed into detached signatures and vice versa, any signed email will work for the attack.

[3] Here *optional* means potentially absent, which is different from present but empty.

```
From: Alice
To: Bob
Subject: Opaque signature
Content-Type: application/pkcs7-mime; smime-type=signed-data
Content-Transfer-Encoding: base64

<base64-encoded CMS object with eContent>
```

(a) A message with an opaque signature. The message is embedded in the CMS object and is not directly readable by a human.

```
From: Alice
To: Bob
Subject: Detached signature
Content-Type: multipart/signed; protocol="application/pkcs7-
     signature"; boundary="XXX"

--XXX
Content-Type: text/plain

Hello, World! This text is signed.
--XXX
Content-Type: application/pkcs7-signature;
Content-Transfer-Encoding: base64

<base64-encoded CMS object without eContent>
--XXX--
```

(b) A message with a detached signature. The message is in the first MIME part and directly readable by a human. Legacy software tended to damage line endings or encoding in such emails, which broke the signature verification.

Figure 2: Opaque and detached signatures as used in S/MIME.

Attack Refinement. Although the `eContent` is not shown in the email client, it is easily revealed under forensic analysis that an old email was reused for this attack. Interestingly, the attack can be refined to remove the original content entirely without affecting the outcome of the signature verification.

Similarly to opaque and detached signatures, where an absent `eContent` signals that the content is provided somewhere else, CMS supports so called "signed attributes", whose absence or presence signals what was signed. If a `signedAttrs` field is present, the signature covers the exact byte sequence of the `signedAttrs` field and not the content per se. If naively implemented, this would of course leave the content unauthenticated. Therefore, the `signedAttrs` field *must* contain a hash of the content [7]. If the `signedAttrs` field is absent, the signature covers the `eContent` directly.

This indirect signing allows the replacement of the original content with the exact byte sequence of the `signedAttrs` field without affecting the signature verification outcome. An email modified in this way will appear "empty" or contain seemingly "garbage" (because the `signedAttrs` is interpreted as ASCII). In either case, this can be used to hide where the old signature originated from. We consider this a noteworthy curiosity.

Multiple Signers (C_2) S/MIME and CMS allow multiple signers in parallel to sign the same content [7]. Obviously, the outcome of the verification may differ for each signer

and the user interface should make that clear. However, it is reasonable to show a simplified version. We consider it a forgery if an *invalid* signature is marked as "valid" due to the presence of an unrelated valid signature.

No Signers (C_3) A CMS signature object may contain zero or more signers. Although RFC 5652 gives limited advice regarding zero signers, it does not state explicitly what to do with "signed messages" without a signer.

Trust Issues (C_4) In contrast to OpenPGP, S/MIME is built upon trust hierarchies. Certificates are authentic as long as they can be traced back to a valid root certificate. In practice, this means that most S/MIME certificates (in the Internet PKI) are indirectly trusted. However, clients must check the validity of the certificate chain. We consider it a forgery if a client accepts invalid certificates, such as self-signed certificates or otherwise untrusted or non-conforming certificates.

4.2 GPG API Attack Class

GnuPG, a stand-alone OpenPGP implementation, provides a stateful text-based command-line interface with approximately 380 options and commands. The complexity of the API and the need for consumers to parse GnuPG's string messages provides a rich attack surface.

GnuPG Status Lines GnuPG provides a machine-readable interface by emitting *status lines* in a simple text-based format (see Figure 3), via the `--status-fd` option. Each status line starts with `[GNUPG:]` and one of approximately 100 possible keywords (such as `GOODSIG`), followed by additional text specific to the keyword.

Although some documentation exists, it does not cover all possible sequences of status lines and their significance for any given operation. In fact, due to streaming processing, the complexity of the API reflects the overall complexity of the OpenPGP message format. In particular, we note the following risk factors:

- The API contains text under the attacker's control (e.g., the `<user-id>` in `GOODSIG`), which must be escaped to prevent injection attacks.
- The number and order of status lines depend on the OpenPGP packet sequence, which is under the attacker's control. Applications must handle all combinations correctly that are allowed by GnuPG.
- The API is stateful, i.e., the semantics of a status line can depend on its position in the sequence. For example, the validity indicated by a `TRUST_*` line applies only to the signature started by the previous `NEWSIG`.

```
$ gpg --status-fd 2 --verify
[GNUPG:] NEWSIG
[GNUPG:] GOODSIG 88B08D5A57B62140 <alice@good.com>
[GNUPG:] VALIDSIG 3CB0E84416AD52F7E186541888B08D5A57B62140
     2018-07-05 1530779689 0 4 0 1 8 00 3
     CB0E84416AD52F7E186541888B08D5A57B62140
[GNUPG:] TRUST_FULLY 0 classic
```

(a) Example output for a single trusted signature (excerpt).

```
NEWSIG [<signers-user-id>]
GOODSIG <key-id> <user-id>
BADSIG <key-id> <user-id>
VALIDSIG <fingerprint> <date> <create-timestamp> <expire-
     timestamp> <version> <reserved> <public-key-algorithm> <
     hash-algorithm> <signature-class> [<primary-key-
     fingerprint>]
TRUST_NEVER <error-token>
TRUST_FULLY [0 [<validation-model>]]
PLAINTEXT <format> <timestamp> <filename>
PLAINTEXT_LENGTH <length>
```

(b) Important status lines for signature verification from GnuPG.

Figure 3: Status lines output by GnuPG as a side-effect of streaming message processing.

- The use of the API requires a good understanding of OpenPGP and trust models as implemented in GnuPG. The `GOODSIG`, `VALIDSIG` and `TRUST_*` lines have very specific technical meaning that is not always apparent from the inconsistent terminology in the interface.
- By default, GnuPG runs in the context of the user's home directory, using their configuration files and keyrings, which can influence the output of GnuPG within, and outside of, the status line API.

We focus our work on injection attacks and applications parsing the interface. First, we review the source code of GnuPG to identify places where an attacker could inject untrusted data at trusted positions in the API. Then we review all open source mail clients to identify exploitable mistakes in the API parser.

In-band Injection Attacks (G_1) There are various places in the GnuPG status line API that contain untrusted data under the attacker's control. For example, the `<user-id>` in a `GOODSIG` status line is an arbitrary string from the public key that can be crafted by an attacker. A naive idea is to append a newline character followed by a spoofed status line into the user ID of a public key. Normally, GnuPG protects against this naive attack by properly escaping special characters.

In addition to the status line API, we also review the logging messages for injection attacks. This is due to a common pattern, where applications using GnuPG conflate the status API and the logging messages by specifying the same data channel `stdout` for both (using the command line option `--status-fd 2`). Best practice requires separate channels to be used, but technical limitations can make this difficult for some plugins and cross-platform applications.

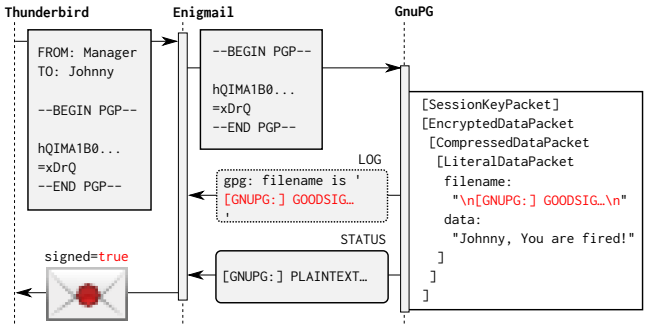


Figure 4: In-band injection of GnuPG status lines via log messages.

If an injection attack is successful, it can be very powerful, as the attacker can spoof the status lines entirely and provide arbitrary data to the application. Such spoofed status lines can include forged indications of a successful signature validation for arbitrary public keys. A valid PGP message containing the injection payload is showed in Figure 4.

Attacks on Email Clients Using GPG (G_2) The GnuPG interface provides only limited functionality; for example, it is not possible to validate a signature against a single public key in the keyring, but only against *all* public keys contained therein. Since this is insufficient for validating the sender of an email, GnuPG returns a string containing the user ID and the result of the validation. By manipulating this string, mail clients can be tricked into giving false validation results.

Applications using the GnuPG status line API have to parse the status lines and direct a state machine, keeping track of verification and decryption results under a wide range of different inputs, user configurations and GnuPG versions. Thus, application developers often use a common design pattern to deal with the complexity, such as: Iterating over all status line messages, parsing the details of those status lines that have information relevant to the task the application is interested in, and ignoring all other unknown or unsupported messages. This can lead to a number of serious vulnerabilities. For example, if an application is unprepared to handle multiple signatures, it might not reset the signature verification state at the `NEWSIG` status line, conflating the verification result of multiple signatures into a single result state. This might allow an attacker to add more signatures to an existing message to influence the result. Another example is the use of regular expressions that are not properly anchored to the status line API data format, thereby allowing an attacker to inject data that, although it is properly escaped by GnuPG, is then misinterpreted by the application.

4.3 MIME Attack Class

In this section we discuss attacks on how email clients handle partially signed messages in the case that the signed part is wrapped within the MIME tree of a multipart message. For this class of attacks, the attacker is already in possession of at least one message and a corresponding valid signature from the entity to be impersonated. The obtained message can be in *Inline PGP*, *PGP/MIME*, or *S/MIME* as all formats can be embedded as sub-parts within a multipart message.

Prepending Attacker's Text (M_1) Email clients may display a valid signature verification status even if only a single MIME part is correctly signed. In such a scenario of partially signed emails, the attacker can obfuscate the existence of the correctly signed original message within a multipart email. For example, this can be achieved by prepending the attacker's message to the originally signed part, separated by a lot of newlines, resulting in a weak forgery.

Hiding Signed Part with HTML (M_2) Another option is to completely hide the original part with HTML and/or CSS, resulting in a perfect forgery. There are several ways to do this. One way occurs if the email client renders the output of multiple MIME-parts within a single HTML document presented to the user, then the signed part can simply be commented out, for example, using HTML comments. Furthermore, it can be embedded in (and therefore hidden within) HTML tags, or wrapped into CSS properties like `display:none`. An example for such a MIME-wrapping attack based on a hidden signed part is shown in Figure 5.

```
From: manager@work.com
To: johnny@work.com
Subject: Signed part hidden with CSS/HTML
Content-Type: multipart/mixed; boundary="XXX"

--XXX
Content-Type: text/html

Johnny, you are fired!
<div style="display:none"><plaintext>

--XXX
---BEGIN PGP SIGNED MESSAGE---
Hash:  SHA512

Congratulations, you have been promoted!
---BEGIN PGP SIGNATURE---
iQE/BAEBAgApBQJbW1tqIhxCcnVjZSBXYXluZSA8YnJ1Y2V3YXluZTQ1...
---END PGP SIGNATURE---

--XXX--
```

Figure 5: Signature spoofing attack based on MIME wrapping. The signed part is hidden by HTML/CSS, while the message "Johnny, You are fired!" is displayed by the email client.

Hiding Signed Part in Related Content (M_3) Even if there is a strict isolation between multiple MIME parts, it can be broken using `cid:` references (see RFC 2392). This can be achieved by constructing a `multipart/related` message consisting of two parts. The first MIME part contains

```
From: Philip R. Zimmermann <prz@pgp.com>
To: johnny@work.com
Subject: PGP signed message
Content-Type: multipart/related; boundary="XXX"

--XXX
Content-Type: text/html

<b>Johnny, You are fired!</b>
<img src="cid:signed-part">

--XXX
Content-ID: signed-part

---BEGIN PGP SIGNED MESSAGE---
A note to PGP users: ...
---BEGIN PGP SIGNATURE---
iQA/AwUBOpDtWmPLaR3669X8EQLvOgCgs6zaYetj4JwkCiDSzQJZ1ugM...
---END PGP SIGNATURE---

--XXX--
```

Figure 6: Multipart email with a `cid:` reference to the signed part.

an attacker-controlled text and a `cid:` reference to the original signed part, which is placed into the second MIME part. An example email to demonstrate such an attack is given in Figure 6. It contains an HTML message part and a signed text which was written and published by Phil Zimmermann back in 2001.[4] The `cid:` reference enforces the signed (but invisible) part to be parsed by the mail client, which indicates a valid signature for the shown message (from the first part). This allows us to impersonate Phil Zimmermann[5] for arbitrary messages. A corresponding screenshot of Apple Mail (GPG Suite) is given in Figure 1 on the first page.

Hiding Signed Part in an Attachment (M_4) Even without using HTML, the originally signed part can be hidden by defining it as an attachment. This can be done by placing it into a sub-part with the additional header line shown below:

```
Content-Disposition: attachment; filename=signature.asc
```

4.4 ID Attack Class

In this section we discuss attacks on how email clients match a signed message to a sender's identity. These attacks are less powerful than those previously discussed, because indistinguishability is rarely given at all levels of user interaction, i.e., many clients allow the user to check the signature details, which may reveal signs of manipulation.

Not Checking If Sender = Signer (I_1) When dealing with digital signatures, the question *Signed by whom?* is important. If Bob's email client simply displayed "valid signature" for any PGP or S/MIME signed message, Eve could sign her message and send it to Bob with Alice set as the sender. This is due to a lack of binding between the user ID from the signature and the address given in the `FROM` header.

[4] P. Zimmermann, A (Inline PGP signed) note to PGP users, `https://philzimmermann.com/text/PRZ_leaves_NAI.txt`

[5] PGP key ID `17AFBAAF21064E513F037E6E63CB691DFAEBD5FC`

```
From: Alice <eve@evil.com>
```

(a) Display name and email address in `FROM` header.

```
From: alice@good.com <eve@evil.com>
```

(b) Display name set to email address `FROM` header.

```
From: alice@good.com
From: eve@evil.com
```

(c) Multiple `FROM` header.

```
Sender: alice@good.com
From: eve@evil.com
```

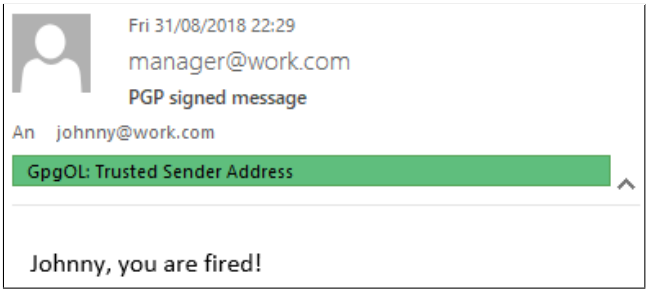
(d) `SENDER` and `FROM` headers.

Figure 7: Examples how to fool signature verification logic if the PGP user ID or the Internet mail address in the signer's S/MIME certificate is compared to sender address in the email header.

Display Name Shown as Signer (I_2) There are two options to handle this problem. First, a mail client can explicitly display the signer's identity somewhere in the UI and let the user compare it to the sender address. Second, the email client can check whether the signer's identity (i.e., email address) equals the sender's address, and show a warning if this is not the case. The second option gives a lot of room for attacks; RFC 2632 for S/MIME signed messages states that "receiving agents *must* check that the address in the From or Sender header of a mail message matches an Internet mail address in the signer's certificate". However, in practice email clients can only check if the `FROM` header *contains* the signer's email address because RFC 5322 allows additional display names to be associated with a mailbox (e.g., `Name <foo@bar>`). If an email client only shows the display name to the user, Eve can simply set *Alice* as display name for her own sender address (see Figure 7a). Also, in such a scenario the display name itself could be set to an email address such as *alice@good.com* that is presented to the user, see Figure 7b.

An example screenshot for Outlook, which required additional effort, is given in Figure 8a. The source code of this mail can be found in Figure 8b. While Outlook always shows the full sender address (*eve@evil.com*), it can simply be "pushed out of the display" by appending a lot of whitespaces to the display name (*manager@work.com*).

From/Sender Header Confusion (I_3) Another problem is how email clients deal with multiple `FROM` fields in the mail header—especially if PGP or S/MIME support is not implemented directly by the email client, but offered through a



(a) Screenshot of a spoofed PGP signed message in Outlook which was actually signed by the attacker (*eve@evil.com*).

```
From: manager@work.com [whitespaces] x <eve@evil.com>
To: johnny@work.com
Reply-to: manager@work.com
Subject: Signed by whom?
Content-Type: multipart/signed; boundary="XXX";
  protocol="application/pgp-signature"

--XXX

Johnny, you are fired!

--XXX
Content-Type: application/pgp-signature

[valid signature by eve@evil.com]
--XXX--
```

(b) Proof-of-concept email source code to forge PGP signatures.

Figure 8: PGP signature spoofing attack against Outlook/GpgOL, based on the RFC 5322 display name shown as the signer's identity.

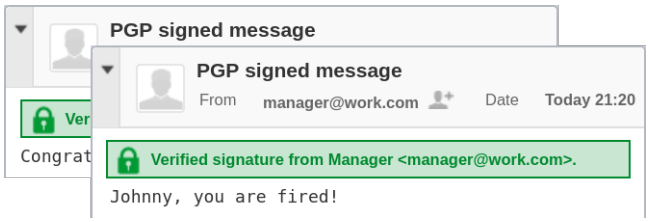
third-party plugin—as there may be different implementations on how to obtain the sender address of an email. For example, the email client could display the email address given in the first `FROM` header while the plugin would perform its checks against any occurrence of this header field (see Figure 7c). Also, the plugin could respect the `SENDER` header in its checks which "specifies the mailbox of the agent responsible for the actual transmission of the message" [8] while the mail client would display the email address taken from the `FROM` email header (see Figure 7d). Note that if Eve sets an additional `Reply-to: alice@good.com` header, which instructs the email client to reply to Alice, such attacks go unnoticed by Bob when replying to the email. They can, however, be detected in most clients by reviewing the signature details and spotting Eve as the real signer.

4.5 UI Attack Class (U_1)

In this section, we discuss UI redressing attacks that exploit the presentation of signature verification results to the user. The attacks are successful if the spoofed message is indistinguishable from a message with a real valid signature.

This attack class exploits that various email clients display the status of the signature within the email content itself. This part of the UI is under the control of the attacker. With HTML, CSS, or inline images it is easy to reproduce security-critical UI elements displaying a "valid signature".

Figure 9a shows a signed and a spoofed email in Roundcube. The spoofed email is based on displaying a valid signature indicator with HTML and CSS (an example of UI redressing). The HTML code is provided in Figure 9b.



(a) Two emails with visually indistinguishable signature indicators. One is PGP-signed, the other is a specially crafted HTML email.

```
From: manager@work.com
To: johnny@work.com
Subject: UI redressing
Content-Type: text/html

<div class="message-part">
<div id="enigma-message1" class="enigma-notice1" style="margin-
     left: -0.25em; margin-bottom: 5px; padding: 6px 12px 6px
     30px; font-weight: bold; background: url(enigma_icons.png
     ) 3px -171px no-repeat #c9e6d3; border: 1px solid #008a2e
     ; color: #008a2e">
      Verified signature from Manager <manager@work.com>.
</div><div class="pre">Johnny, You are fired!</div></div>
```

(b) Forging a PGP signature in Roundcube with UI redressing.

Figure 9: Security indicators in Roundcube. The indicator is in the attacker-controlled HTML area, which allows trivial spoofing.

5 Evaluation

Of the tested 20 clients with PGP support, 15 use *GnuPG* to verify signatures and call it either directly, or through some kind of plugin such as *Enigmail*[6] or *GPG Suite*,[7] or by using the *GPGME*[8] wrapper library. The remaining clients use *OpenPGP.js*,[9] *OpenKeychain*,[10] or a proprietary solution. Of the tested 22 clients with S/MIME support, only five require third party plugins. The results of signature spoofing attacks tested on the various email clients are shown in Table 2 for OpenPGP and in Table 3 for S/MIME.

The results of our evaluation show a poor performance of the overall PGP and S/MIME ecosystems when it comes to trustworthiness of digital signatures; for ten OpenPGP capable clients and seven clients supporting S/MIME we could spoof visually indistinguishable signatures on all UI levels (resulting in perfect forgeries). On four additional OpenPGP

[6] P. Brunschwig, *Enigmail*, `https://enigmail.net/`
[7] L. Pitschl *GPG Suite*, `https://gpgtools.org/`
[8] W. Koch, *GPGME*, `https://github.com/gpg/gpgme`
[9] ProtonMail, *OpenPGP.js* `https://openpgpjs.org/`
[10] Cotech, *OpenKeychain*, `https://openkeychain.org/`

capable clients and eight clients supporting S/MIME, we could spoof visually indistinguishable signatures on the first UI level (resulting in partial forgeries). While none of the attacks directly target the underlying cryptographic primitives, the high success rate raises concerns about the practical security of email applications. We discuss the results for each class of attack in this section. We also published proof-of-concept attack emails and screenshots of partial and weak forgeries in a public repository.[11]

5.1 CMS Attack Class

eContent Confusion (C_1) We found that Thunderbird, Postbox, MailMate, and iOS Mail are vulnerable to eContent confusion. Given a valid S/MIME signature for a specific user, this allows a perfect forgery for any message of this user. Note that opaque signed emails can be transformed into detached signed emails and vice versa. Thus, having *any* signed email from a target is enough to forge an arbitrary number of new messages. Mozilla assigned CVE-2018-18509 to this issue.

Multiple Signers (C_2) Evolution coerces multiple signers into one "valid signature" UI element (see Figure 10). However, the UI reveals the erroneous signature upon further inspection of the UI. By definition, this is a partial forgery.

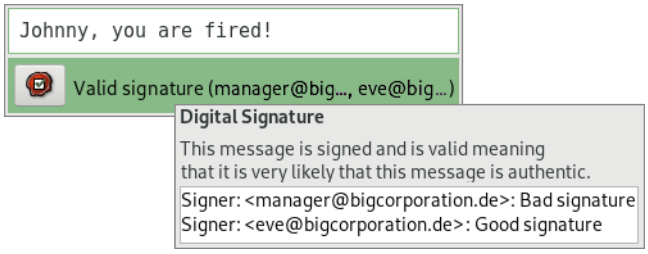


Figure 10: This email has two signers. The first signer did not sign this email. Evolution coerced both signers into one security indicator showing "valid signature". (Minor details were removed from the screenshot to make it smaller.)

No Signers (C_3) In the case of no signers, three email clients, namely Outlook, Mutt, and MailDroid, show some UI elements suggesting a valid signature and some UI elements doing the opposite. We consider this a weak forgery, because there is no clear indication that signature validation has succeeded/failed. In Outlook an otherwise very prominent red error bar is not shown and a seal indicates a valid signature (although it should show a warning sign), see Figure 11. Interestingly, clicking through the UI in Outlook may *strengthen* the illusion of a validly signed message because of the wording in the subdialogs.

[11] `https://github.com/RUB-NDS/Johnny-You-Are-Fired`

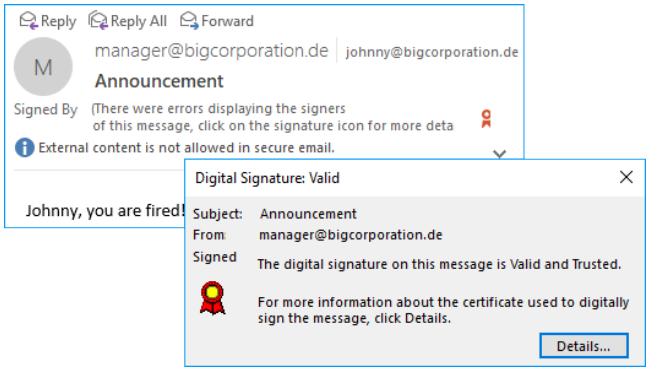


Figure 11: "Signed email" in Outlook with no signers at all.

Trust Issues (C_4) Although none of the clients accepted "extended certificates", i.e., certificates re-signed by a leaf certificate with no `CA=true` flag, we observed that Trojitá and Claws display conflicting UI elements on untrusted certificates (i.e., "success: bad signature"). Nine and MailDroid do not display information about the origin of a certificate on the first level of the UI. This means that, although there are security indicators suggesting a signed email, the origin may be completely untrustworthy, resulting in partial forgery.

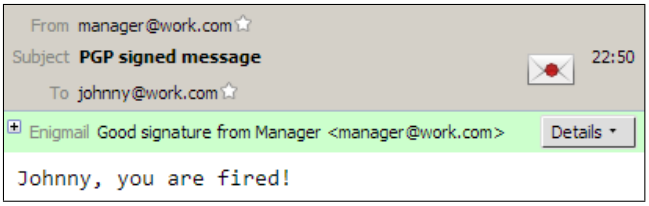
5.2 GPG API Attack Class

We found an injection attack for the "embedded filename" log message in GnuPG, as well as several issues in Enigmail's status line parser and state machine.

Status Line Injection through Embedded Filename (G1). OpenPGP Literal Data Packets, which contain the actual plaintext of a message, contain some metadata, in particular an *embedded filename* that is usually set by the sender to the original filename of a signed or encrypted file. We found that GnuPG's logging message for the embedded filename does not escape newline and other special characters. If an application combines the logging messages with the status line API in a single data channel, an attacker can use the embedded filename to inject arbitrary status lines (up to 255 bytes, which can be sufficient to spoof the `GOODSIG`, `VALIDSIG` and `TRUST_FULLY` lines for a single signature). Figure 12b shows the injected text highlighted. The successful attack is shown in Figure 12a.

Using this attack, we were able to spoof arbitrary signature verification results in Enigmail, GPG Suite, and Mailpile.[12] The attack only assumes our weakest attacker model, as all relevant status lines can be injected into the embedded filename. In fact, the message does not even need to be signed at all; the attack, however, has the additional requirement

[12] Manual signature verification using GnuPG on the command line is also affected; the embedded filename can contain arbitrary terminal escape sequences, allowing the attacker to overwrite any part of the terminal.



(a) Screenshot of a spoofed PGP signature in Thunderbird.

```
$ gpg --status-fd=2 --verbose message.gpg
gpg: original file name=''
[GNUPG:] GOODSIG 88B08D5A57B62140 Manager <manager@work.com>
[GNUPG:] VALIDSIG 3CB0E84416AD52F7E186541888B08D5A57B62140
      2018-07-05 1530779689 0 4 0 1 8 00
      3CB0E84416AD52F7E186541888B08D5A57B62140
[GNUPG:] TRUST_FULLY 0 classic
gpg:  ''
[GNUPG:] PLAINTEXT 62 1528297411 '%0A[GNUPG:]%20GOODSIG[...]
[GNUPG:] PLAINTEXT_LENGTH 56
```

(b) Example output for GnuPG status line injection (excerpt).

Figure 12: Status line injection attack on GnuPG.

that the user has enabled the `verbose` configuration option. This option is not enabled by default, but it is often used by experts and part of several recommended GnuPG settings.

The vulnerability is present in all versions of GnuPG until 2.2.8. Our finding is documented as CVE-2018-12020. Due to the severity of the attack, we also reviewed non-email applications for similar vulnerabilities, see subsection 7.5.

State Confusion and Regular Expressions in Enigmail (G2). We found two flaws in the way Enigmail handles status messages for multiple signatures:

- If the status of the *last* signature is `GOODSIG`, `EXPKEYSIG`, or `REVKEYSIG`, Enigmail will overwrite the signature details (e.g., fingerprint, creation time, algorithms) with those from the *first* `VALIDSIG`, confusing the metadata of two signatures. The attacker can change the state of a signature to good, expired, or revoked by adding a corresponding second signature.
- If *any* of the signatures is `TRUST_FULLY` or `TRUST_ULTIMATE`, and the last signature is good, expired, or revoked, then Enigmail will display the information from the first `VALIDSIG` as trusted.

We also found regular expressions that were not anchored to the beginning of status lines. This allows for injection of fake `VALIDSIG` and other status lines in malicious user ID strings. Combining this with the above, this allows an attacker to control the signature details completely. The attack involves two signatures on a single message, which Enigmail combines in a complex way to a single signature that is shown to the user. We assume that the attacker is trusted, and further assume that the attacker can poison the user's keyring with arbitrary untrusted keys. This is not a strong assumption because in PGP the local keyring is just an insecure cache of public keys. For example, the attacker might rely on automatic key retrieval from public keyservers or include the key as extra payload when sending her regular public key to Bob.

The *first signature* in the attack is by a key with a malicious user ID, crafted by the attacker, that injects a spoofed `VALIDSIG` status line. Because Enigmail processes this line twice using different parsers, some information needs to be duplicated to make the attack work. The *second signature* is a regular valid signature over the same message by the trusted attacker's key. This signature sets the global flag in Enigmail indicating that a signature is made by a trusted key, but otherwise it is completely ignored. Figure 13 shows which data in the attack is used by Enigmail. The fingerprint will be used to resolve further details such as the user ID of the (spoofed) signing key. The vulnerability is present in all versions of Enigmail until 2.0.7, affecting Thunderbird and Postbox. Our finding is documented as CVE-2018-12019.

```
[GNUPG:] NEWSIG
[GNUPG:] GOODSIG 8DA07D5E58B3A622 x 1527763815 x x x 1 10 x
      4F9F89F5505AC1D1A260631CDB1187B9DD5F693B VALIDSIG x x 0 4
      F9F89F5505AC1D1A260631CDB1187B9DD5F693B
[GNUPG:] TRUST_UNDEFINED
[GNUPG:] NEWSIG
[GNUPG:] GOODSIG 88B08D5A57B62140 <eve@evil.com>
[GNUPG:] TRUST_FULLY
```

Figure 13: The status line API as seen by Enigmail (abbreviated).

5.3 MIME Attack Class

On five PGP email clients, including popular products such as Thunderbird and Apple Mail, we could completely hide the original signed part (resulting in perfect forgeries) using multiple techniques, such as wrapping it into attacker-controlled HTML/CSS (M_1), referencing it as an "image" (M_2), or hiding it as an "attachment" (M_3). On another three clients we could only obfuscate the original signed message part (resulting in weak forgeries) by appending it and using a multitude of newlines to cover its existence (M_4). Our findings are documented as CVE-2017-17848, CVE-2018-15586, CVE-2018-15587, and CVE-2018-15588.

Our evaluation shows that S/MIME is less vulnerable to signature wrapping attacks by design; all but one tested email clients only show a valid signature verification if the whole email including all sub-parts is correctly signed (i.e., `Content-Type: multipart/signed` is the root element in the MIME tree). Unfortunately, it seems hard to get rid of partially signed emails or mark them as suspicious, as can be done for partially encrypted messages – a countermeasure Enigmail applied to the EFAIL attacks [9] – because in the PGP world, partially signed is quite common (e.g., mailing lists, forwards, etc.). There seems to be a different philosophy in the S/MIME world; for example, S/MIME forwarding

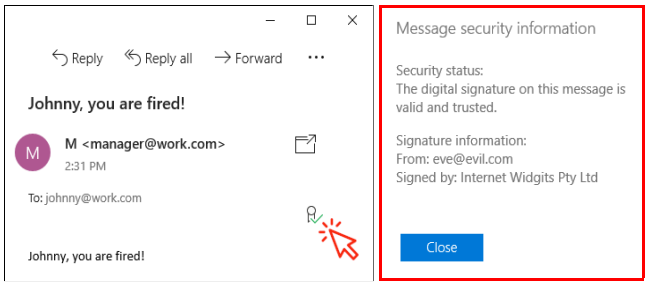


Figure 14: Partial forgery in the Windows 10 mail app. The email is actually signed by Eve which is only visible in the signature details.

intentionally breaks the signature because the forwarding entity should re-sign the message.

5.4 ID Class

Eleven PGP email clients and twelve S/MIME clients explicitly show the signer's identity (i.e., PGP user ID or Internet mail address in the signer's S/MIME certificate) when receiving a signed email. This can be considered safe because a user gets direct feedback regarding the *signed by whom?* question. The other clients do not show the signer's identity on the first level of the UI. Of those, two PGP email and three S/MIME mail clients, such as the Windows 10 mail app (see Figure 14), do not perform any correlation between the sender address and the signer's identity at all. They only show "signed" with no further information, making them easy targets for ID attacks (resulting in partial forgeries). The other seven PGP email clients and eight S/MIME clients compare the sender's address to the email address found in the public key or certificate matching the signature. This process is error-prone. For four PGP email clients, including GpgOL for Outlook, and eight S/MIME email clients, the correlation could be fully bypassed using the various techniques described in subsection 4.4. If Bob does not manually view the signature details, there is no indicator that the email was signed by Eve instead of Alice (resulting in partial forgery). For two of these clients (GpgOL for Outlook and Airmail) no signature details were available, resulting in perfect forgery.

5.5 UI Attack Class

Five tested PGP email clients and four S/MIME clients display the status of signatures within the email body, which is a UI component controlled by the attacker. This allowed us to create fake text or graphics implying a valid signature using HTML, CSS and inline images visually indistinguishable from a real signed message. Only further investigation of the email, such as viewing signature details, could reveal the attack (resulting in partial forgery). Three of these clients do not not even have an option for further signature details, resulting in perfect forgery. Spoofing signatures was especially easy for clients where the PGP or S/MIME plugin simply injects HTML code into the email body. We could just copy the original HTML snippet of a valid signature and re-send it within the body of our spoofed email.

Another seven PGP clients and nine S/MIME clients show the results of signature verification in, or very close to, the email body and could be attacked with limitations (causing weak forgeries). Some of these clients have additional indicators in other parts of the UI pointing out that the email is actually signed, but those indicators are missing in the case of spoofed signatures based on UI redressing (see Figure 15). Furthermore, in some of these clients the spoofed signature was not 100% visually indistinguishable.

6 Countermeasures

Similarly to other RFC documents, S/MIME [10] and OpenPGP [6] contain a section on *security considerations*. While these sections discuss cryptographic best practices (e.g., key sizes and cryptographic algorithms), they do not discuss how to design secure validation routines and interfaces. In this section, we discuss several possible countermeasures against the presented attacks in order to give guidance for implementing secure email clients.

6.1 CMS Attack Class

eContent Confusion (C_1) Both S/MIME signing variants are commonly used[13] and standard compliant clients are expected to support them. Thus, special care must be taken to only display the content which was subject to verification.

The relevant standards, RFC 5652 (CMS) and RFC 5751 (S/MIME), do not give any advice on how to handle the case were both variants are present. We recommend to display neither of them and show an error instead. In fact, Claws reports a "conflicting use".

[13] Outlook 2016 sends opaque signed messages by default and Thunderbird sends detached messages by default.

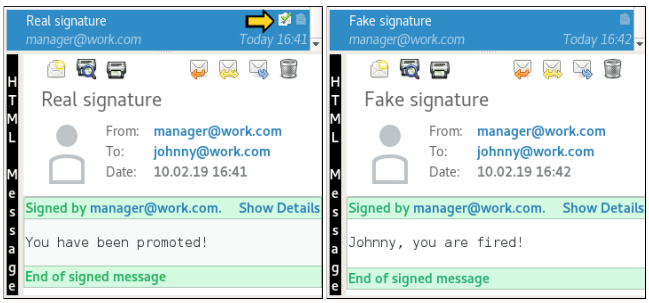


Figure 15: "Weak" forgery in KMail. The check mark UI indicatior in the upper right cannot be spoofed using simple UI redressing.

OS	Client	Plugin	GPG	MIME	ID	UI	Weaknesses
Windows	Thunderbird (52.5.2)	Enigmail (1.9.8)	●	●	–	○	G_1, G_2, M_2, M_3, U_1
	Outlook (16.0.4266)	GpgOL (2.0.1)	–	–	●	○	I_2, U_1
	The Bat! (8.2.0)	GnuPG (2.1.18)	–	○	○	–	M_1, I_1
	eM Client (7.1.31849)	native	–	–	–	◐	U_1
	Postbox (5.0.20)	Enigmail 1.2.3	●	–	–	–	G_1, G_2
Linux	KMail (5.2.3)	GPGME (1.2.0)	–	–	–	○	U_1
	Evolution (3.22.6)	GnuPG (2.1.18)	–	●	–	○	M_4, U_1
	Trojitá (0.7-278)	GPGME (1.2.0)	–	–	◐	●	I_2, I_3, U_1
	Claws (3.14.1)	GPG plugin (3.14.1)	–	○	–	–	M_1
	Mutt (1.7.2)	GPGME (1.2.0)	–	–	–	○	U_1
macOS	Apple Mail (11.2)	GPG Suite (2018.1)	●	●	–	○	G_1, M_1, M_2, M_3, U_1
	MailMate (1.10)	GPG Suite (2018.1)	–	●	○	●	M_1, M_2, M_3, I_2, U_1
	Airmail (3.5.3)	GPG-PGP (1.0-4)	–	●	●	–	M_3, I_2
Android	K-9 Mail (5.403)	OpenKeychain (5.2)	–	–	◐	–	I_2
	R2Mail2 (2.30)	native	–	–	◐	○	I_1, U_1
	MailDroid (4.81)	Flipdog (1.07)	–	○	–	◐	M_1, U_1
Web	Roundcube (1.3.4)	Enigma (git:48417c5)	–	–	–	●	U_1
	Horde/IMP (7.7.9)	GnuPG (2.1.18)	–	–	–	–	–
	Mailpile (1.0.0rc2)	GnuPG (2.1.18)	●	–	◐	–	G_1, I_1
	Mailfence (2.6.007)	OpenPGP.js (2.5.3)	–	–	–	–	–

● Indistinguishable signature on all UI levels (perfect forgery) ○ Signature can be spoofed with limitations (weak forgery)
◐ Indistinguishable signature on first UI level (partial forgery) – No vulnerabilities found

Table 2: Out of 20 tested email clients 14 were vulnerable to our OpenPGP signature spoofing attacks (perfect or partial forgery).

Disallow Multiple Signers (C_2) It is difficult to give good advice on the presentation of multiple signers, as different clients may implement different UI concepts. Furthermore, introducing UI elements might worsen the usability or introduce security problems on its own (e.g., UI redressing).

However, a simple solution is to not implement multiple signer support at all. Many up-to-date clients do not support multiple signers, e.g., Thunderbird and Outlook 2016. Additionally, we know of no client which is able to *produce* a message with multiple signers. Thus, it seems reasonable to us to not support this feature.

Error Out If No Signers (C_3) Messages with no signer should be treated as errorneous or as not signed. In either case there should be no UI element indicating a signed message. We recommend to not show the message and show an error instead. This is due to the possible application of signature stripping attacks as demonstrated by [11] and [9].

Redesign Trust Management and Workflow (C_4) Clients must validate the complete certification path and fail the signature verification on trust issues. Furthermore, certificates should be checked automatically. Clients must not accept self-signed certificates. If needed, a separate trust chain should be configured on the device or in the application.

6.2 GPG API Attack Class

GnuPG developers can improve the documentation and the API, but they have to consider backwards compatibility and extensibility. GnuPG must track attacker controlled data and always escape newlines and other special characters in all outputs. GnuPG should also validate the structure of OpenPGP messages and provide clear guidelines on how to achieve common tasks such as certificate pinning.

Frontend developers can harden the invocation of the backend (e.g. by using dedicated channels for log and status lines or adding `--no-verbose` to disable logging), their status line parsers (e.g., by anchoring all regular expressions), and the state machine aggregating the results (e.g., by keeping track of multiple signatures, as indicated by `NEWSIG`). However, applications that are too strict risk incompatibilities with future backend upgrades or unconventional user configurations.

The OpenPGP standard should be updated to provide a strict grammar for valid message composition, as the present flexibility (such as arbitrary nesting of encrypted, signed, and compressed messages) is unjustified in practice and puts the burden on implementations to define reasonable limits. Specifically, the OpenPGP standard should only allow one optional encryption layer, one optional compression layer, and one possibly signed literal data packet. More complex message composition (e.g., sign+encrypt+sign to allow for

OS	Client	Plugin	CMS	MIME	ID	UI	Weaknesses
Windows	Thunderbird (52.5.2)	native	●	–	○	–	C_1, I_3
	Outlook (16.0.4266)	native	○	–	–	○	C_3, U_1
	Win. 10 Mail (17.8730.21865)	native	–	–	◐	○	I_1, U_1
	Win. Live Mail (16.4.3528)	native	–	–	◐	○	I_2, U_1
	The Bat! (8.2.0)	native	–	–	◐	–	I_1
	eM Client (7.1.31849)	native	–	–	–	◐	U_1
	Postbox (5.0.20)	native	●	–	◐	–	C_1, I_3
Linux	KMail (5.2.3)	native	–	–	–	○	U_1
	Evolution (3.22.6)	native	◐	–	–	○	C_2, U_1
	Trojitá (0.7-278)	native	○	–	◐	●	C_4, I_2, I_3, U_1
	Claws (3.14.1)	GnuPG (gpgsm) (2.1.18)	○	–	–	–	C_4
	Mutt (1.7.2)	native	○	–	–	○	C_3, U_1
macOS	Apple Mail (11.2)	native	–	–	–	○	U_1
	MailMate (1.10)	native	●	●	○	○	$C_1, M_1, M_2, M_3, I_2, U_1$
	Airmail (3.5.3)	S/MIME (1.0-10)	–	–	●	–	I_2
Android	R2Mail2 (2.30)	native	–	–	◐	○	I_2, I_3, U_1
	MailDroid (4.81)	FlipDog (1.07)	◐	–	–	◐	C_3, C_4, U_1
	Nine (4.1.3a)	native	◐	–	◐	○	C_4, I_1, U_1
iOS	Mail App (12.01)	native	●	–	–	–	C_1
Web	Roundcube (1.3.4)	rc_smime (git:f294cde)	–	–	–	●	U_1
	Horde/IMP (6.2.21)	native	–	–	–	–	–
	Exchange/OWA (15.1.1034.32)	Control (4.0500.15.1)	–	–	–	○	U_1

●	Indistinguishable signature on all UI levels (perfect forgery)	○	Signature can be spoofed with limitations (weak forgery)
◐	Indistinguishable signature on first UI level (partial forgery)	–	No vulnerabilities found

Table 3: Out of 22 tested email clients 15 were vulnerable to our S/MIME signature spoofing attacks (perfect or partial forgery). Some clients with weak forgery, conflicting UI elements or unusual workflows are documented in more detail in the appendix.

early spam mitigation) may be desirable for certain applications in the future. These should then be covered in future standard revisions to ensure that they can be supported without introducing new risk factors. Until then, they can be supported either by nesting several distinct messages explicitly or as non-standard extensions that are disabled by default in compliant implementations.

6.3 MIME Attack Class

There are two approaches to counter signature spoofing attacks on partially signed messages which are hidden in the MIME tree. Either email clients should show exactly which part of the message was signed, for example, by using a green frame. However, note that this is hard to implement in a secure way because all edge cases and potential bypasses, such as internal `cid:` references, need to be considered, and it must be made sure that the frame cannot be drawn by the attacker using HTML/CSS. Or email clients should be conservative and only show a message as correctly signed if the whole message (i.e., the MIME root) was correctly signed. While this will break digital signatures in some mailing lists and in forwarded emails, such an *all-or-nothing* approach can be considered as more secure and is preferred by the authors of this paper. Furthermore, if the signature contains a timestamp, it should be shown to the user to draw suspicion on re-used and wrapped old signatures. For incoming new messages the timestamp could be compared to the current system time and an alert could be given if the signature contains a timestamp older than a certain threshold. This can also protect from attacks such as non-realtime surreptitious forwarding.

6.4 ID Attack Class

Approaches to encrypted and digitally signed email headers like Memory Hole [12] (a non-standard OpenPGP extension) or RFC 7508 (Securing Header Fields with S/MIME) aim to guarantee a cryptographic binding between the signature and the sender address. However, few email clients support these standards. Furthermore, clients supporting Memory Hole (Thunderbird/Enigmail, R2Mail2) still accept signed emails without header protection for backwards compatibility.

Even worse, Enigmail did not guarantee consistency between unprotected email headers and headers protected by Memory Hole in our tests. Clients remain vulnerable to ID

attacks unless further mitigations are applied. Hence, it can be considered a good practice to explicitly show the signer user IDs when displaying a PGP signed message. A comparison to the `FROM` or `SENDER` header fields may not be sufficient because—as our evaluation shows—that approach is error prone and hard to implement in a secure way.

6.5 UI Attack Class

The results of signature verification should not be shown in attacker-controlled parts of the UI, such as the message content itself, which may contain arbitrary graphics. In the context of webmail and native email clients using HTML5 for their UI, such as Electron[14] or XUL,[15] it must be ensured that there is a strict isolation (e.g., separate DOM) between the message content and the rest of the UI. Otherwise, it may be possible to influence the appearance of the UI, for example by injecting CSS properties into the message content. Following the example of browsers with regard to HTTPS, the trend is to avoid positive UI indicators and only show indicators if something is wrong. These systems aim to be secure by default. However, this is obviously infeasible for email signatures as long as most emails are unsigned.

7 Additional Findings

7.1 Crashes

We discovered multiple crashes during testing. For example, we found a nullpointer dereference in Mozilla's NSS library, which is also used in Evolution and Postbox. Although the security impact is rather low, sending a specifically crafted email does lead to a permanent denial of service, as email clients will cache this email and crash upon startup. We experienced a similar issue with iOS Mail, but did not evaluate the origin of the crash. Additionally, we observed crashes in MailMate, R2Mail2, Maildroid (Exception), Roundcube, and Windows 10 Mail.

7.2 Airmail Accepts Invalid PGP Signatures

We found that the Airmail GPG-PGP plugin does not properly validate OpenPGP signatures, accepting even invalid ones, irregardless of their status. This makes signature spoofing attacks trivial.

Also, Airmail does not correctly verify the validity of the signing key even for good signatures, allowing impersonation attacks by injecting public keys into the user's keyring with the email address that should be spoofed.

The vulnerability is present in all versions of Airmail GPG-PGP until "1.0 (9)". Our finding is documented as CVE-2019-8338.

[14] GitHub Inc., *Electron*, `https://electronjs.org/`

[15] Mozilla Foundation, *XUL*, `https://developer.mozilla.org/XUL`

7.3 OpenPGP Message Composition Attacks

OpenPGP messages and keys are sequences of (possibly nested) packets (see Fig. 16). This structure is under control of the attacker, so it must be validated by the implementation. According to the OpenPGP standard, any arbitrary nesting of encryption, compression, and signatures is allowed without restrictions. This flexibility is unjustified, and seems to be an oversight in the specification, as only a small number of combinations are actually meaningful in practice. It also opens PGP up to vulnerabilities such as decompression attacks [13]. In practice, implementations must enforce additional limits; for example, GnuPG allows up to 32 levels of nesting.

```
[SessionKeyPacket]
[EncryptedDataPacket
 [CompressedDataPacket
  [OnePassSignaturePacket]
  [LiteralDataPacket]
  [SignaturePacket]
 ]
]
```

(a) Example structure of an OpenPGP message that is signed with one signing key, compressed, and encrypted to one recipient key.

```
message :- encrypted | signed | compressed | literal.
encrypted :- SessionKeyPacket*, EncryptedDataPacket(message).
signed :- OnePassSignaturePacket, message, SignaturePacket.
compressed :- CompressedDataPacket(message).
literal :- LiteralDataPacket.
```

(b) Grammar for OpenPGP message composition from RFC 4880 (simplified excerpt). This grammar does not include rules for compatibility with older PGP versions.

Figure 16: Valid OpenPGP message and its grammar specification.

Status lines are emitted by GnuPG as packets are processed recursively. However, the status lines do not represent packet boundaries nor the depth of nesting. As a consequence, the status interface is a flat projection of the nested structure, and some information is lost in the process.

Message Composition Attacks GnuPG outputs status lines as a side-effect of recursive processing of packets in OpenPGP messages. This has led to signature spoofing attacks in the past, where an attacker can prepend or append additional unsigned plaintext to a message [14]. We verified that current versions of GnuPG handle this correctly, and could not find any similar issues for signature verification.

Encryption Spoofing Some attacks to spoof signature verification can also be used to spoof decryption results, causing the email client to indicate an encrypted message where in fact the plaintext was transmitted in the clear. Although by itself this is not a security violation, it is concerning and might be a precursor or building stone for other attacks.

Besides the obvious adaptation of our UI redressing and status line injection attacks, we found a flaw in the message composition verification of GnuPG. Since 2006 [14], GnuPG only allows at most one plaintext (i.e., one Literal Data Packet) in a message. However, GnuPG does not verify that the plaintext of an encrypted (non-conforming) message is actually contained within the encrypted part of the message. By replacing the plaintext in an Encrypted Data Packet with a dummy packet ignored by GnuPG (the OpenPGP standard makes a provision for private/experimental packet types), and prepending or appending the (unencrypted) Literal Data Packet, we can cause GnuPG to output the same status lines as for a properly encrypted message, excluding the order. The following output shows the result for a properly encrypted message (differences in red and bold):

```
[GNUPG:] BEGIN_DECRYPTION
[GNUPG:] PLAINTEXT 62 0
[GNUPG:] DECRYPTION_OKAY
[GNUPG:] END_DECRYPTION
```

The next output shows the result for an empty Encrypted Data Packet, followed by a Literal Data Packet in the clear:

```
[GNUPG:] BEGIN_DECRYPTION
[GNUPG:] DECRYPTION_OKAY
[GNUPG:] END_DECRYPTION
[GNUPG:] PLAINTEXT 62 0
```

Both messages are displayed identically (resulting in a perfect forgery) in Thunderbird, Evolution, Mutt, and Outlook, revealing the flexibility of the PGP message format, GnuPG's parser, and the GnuPG status line parsers in email client applications.

7.4 Short Key PGP IDs

Short key IDs of 32 bit (the least significant 4 bytes of the fingerprint) were used in the PGP user interface, on business cards, by key servers, and other PGP-related utilities in the past until pre-image collisions were demonstrated to be efficient in practice [15]. Unfortunately, the Horde/IMP email client still uses short key IDs internally to identify public keys and automatically downloads them from key servers to cache them in internal data structures. Our attempts to exploit these collisions for ID attacks were inconsistent due to caching effects, which is why we did not include these attacks in the evaluation. Horde/IMP should mitigate these attacks by using full-length fingerprints to identify PGP keys.

7.5 GPG API Attacks Beyond Email

Based on source code searches on GitHub[16] and Debian,[17] we looked for software applications or libraries other than email clients which might be susceptible to API signature spoofing attacks. Candidates were programs that invoke GnuPG with `--status-fd 2`, thereby conflating the logging messages with the status line API, and programs that do not correctly anchor regular expressions for status lines (involving `[GNUPG:]`). We identified three broad classes of programs using the GnuPG status line API: (1) Wrapper libraries that provide an abstraction layer to GnuPG, usually for particular programming languages. (2) Applications using certificate pinning to verify the integrity of files when stored under external control (cloud storage), including version control systems like Git. (3) Package managers that use GnuPG for integrity protection of software packages.

We found vulnerable software in all three categories, but due to the large number of libraries and applications using GnuPG, we could not yet perform an extensive review. Also, we found that the available code search engines are not a good match for the task of identifying applications calling an external application through a shell interface. We therefore fear that there may still be a significant number of vulnerable applications using GnuPG out there.

Python-GnuPG Python-gnupg[18] is a library interface to GnuPG for the Python language. It uses the status line interface and conflates it with the logging messages, making 717 applications using it[19] potentially susceptible to the embedded filename injection attack described above, depending on how and in which context they use the Python library.

Bitcoin Source Code Repository Integrity The Bitcoin project uses the Git version control system, which supports signatures on individual software patches to verify the integrity of the whole repository as it changes over time. A shell script using GnuPG to verify the integrity of all commits is included in the distribution.[20] This script uses the status line API and does not anchor the regular expressions, making it susceptible to the malicious user ID injection attack described above. An attacker who can inject arbitrary keys into the keyring of the user can simply re-sign modified source code commits and thus bypass the verification script.

The Bitcoin source code is frequently used as the basis for other crypto-currencies,[21] and thus this error may propagate to many other similar projects such as Litecoin.

Signature Bypass in Simple Password Store Pass[22], a popular password manager for UNIX, uses the GnuPG status line API to encrypt password files and digitally sign configuration files and extension plugins. It does not anchor the regular expressions, making it susceptible to the malicious

[16] *GitHub Code Search*, `https://github.com/search`

[17] *Debian Code Search*, `https://codesearch.debian.net/`

[18] V. Sajip, *python-gnupg* – A Python wrapper for GnuPG, `https://pythonhosted.org/python-gnupg/`

[19] According to *libraries.io*, https://libraries.io/pypi/python-gnupg/usage

[20] *Bitcoin Source Code*, `https://github.com/bitcoin/bitcoin/blob/master/contrib/verify-commits/gpg.sh`

[21] According to *GitHub* the Bitcoin code has 21379 forks as of Nov. 2018

[22] J. A. Donenfeld, *pass*, `https://www.passwordstore.org/`

user ID injection attack described above. If `pass` is used to synchronize passwords over untrusted cloud storage, an attacker with control over that storage can add their public key to the configuration, which causes `pass` to transparently re-encrypt the passwords to the attackers key (in addition to the user's key) over time. Also, if extension plugins are enabled, the attacker can get remote code execution. This finding is documented as CVE-2018-12356.

Yarn Package Manager Yarn is a package manager by Facebook for the JavaScript runtime Node.js. The Yarn installer script[23] primarily relies on TLS to secure the integrity of the installation package itself. However, it also attempts to use GnuPG signature verification, presumably to secure the integrity of the installer from the build server to the download server, which can be different from the server that hosts the installer script (e.g., for nightly builds).

Unfortunately, Yarn fails to do any form of certificate pinning or trust management, and will accept any valid signature by any key in the local keyring, even by untrusted keys. If the attacker can inject a public key into the user's keyring, and perform a MiTM attack against one of Yarn's download servers, the attacker can replace the installation package with one containing a backdoor, thereby gaining remote code execution on the user's machine. This finding is documented as CVE-2018-12556.

7.6 Unsuccessful Cryptograpic Attacks

We analyzed 19 of 20 OpenPGP email clients from Table 2 (all but Airmail, which we could not test, see subsection 7.2) and all 22 email clients supporting S/MIME signatures from Table 3 if they are vulnerable to well-known attacks on the PKCS#1v1.5 signature scheme for RSA with exponent $e = 3$. Specifically, we checked for mistakes in the handling of the padding [16] and ASN.1 structures [17]. All tested clients resisted our attempts.

8 Related Work

The OpenPGP standard [6] only defines how to sign the email body. Critical headers such as `SUBJECT` and `FROM` are not signed if no further extensions, such as Memory Hole [12], Secure Header Fields [18] or XML-based techniques as described in [19], are applied. Levi et al. [20] developed a GUI for users to better understand the limitations of S/MIME digital signatures for emails, i.e., showing which parts of the email are actually signed by whom. Usability papers such as [21] discuss the difficulties that inexperienced users have to manually verify the validity of a signature and understand the different PGP trust levels.

[23] *Yarn installer script*, `https://yarnpkg.com/install.sh`

It is well known that messages signed by a certain entity can be reused in another context. For example, a malicious former employer in possession of a signed "I quit my job" message by Alice can simply re-send this message to Alice's new employer. Such attacks have been referred to as "surreptitious forwarding" in 2001 by Davis [22] who also showed how to strip signatures in various encryption schemes. Gillmor [23] touches upon the problems of partially-signed Inline PGP messages as well as non-signed attachments and the UI challenge such constructs present for MUAs. Furthermore, he demonstrates a message tampering attack through header substitution: by changing the encoding, a signed message with the content *€13/week* can be presented as *£13/week*, while the signature remains valid.

Recently, Poddebniak et al. [9] described two attacks to directly exfiltrate the plaintext of OpenPGP encrypted messages. One works by wrapping the ciphertexts into attacker-controlled MIME parts. This is related to our MIME wrapping attacks on signatures, as both techniques exploit missing isolation between multiple MIME parts. However, we present attacks which do not require mail clients to put trusted and untrusted input into the same DOM, using advanced approaches such as `cid:` URI scheme references.

GnuPG had signature spoofing bugs in the past. In 2006, it was shown that arbitrary unsigned data can be injected into signed messages [14]. Until 2007, GnuPG returned a valid signature status for a message with arbitrary text prepended or appended to an Inline PGP signature, allowing an attacker to forge the contents of a message without detection [24].

In 2014, Klafter et al. [15] showed practical collisions in 32-bit PGP key IDs, named the "Evil 32" attack, which can be dangerous in case a client requests a PGP key from a keyserver using the 32-bit ID. Collisions with 64-bit IDs are also feasible but require more computing power, therefore only full 160-bit fingerprints should be used.

"Mailsploit" [25] enables attackers to send spoofed emails, even if additional protection mechanism like DKIM are applied, by abusing techniques to encode non-ASCII chars inside email headers as described in RFC 1342. In 2017, Ribeiro [26] demonstrated that CSS rules included in HTML emails can be loaded from a remote source, leading to changes in the appearance of—potentially signed—emails after delivery.

Our partial (◐) and weak (○) forgery attacks can potentially be detected by carefully inspecting the GUI or manually clicking to receive more signature details. A user study analyzing user behavior would be necessary to reveal the real impact of such inconsistencies. Lausch et al. [27] reviewed existing cryptographic indicators used in email clients and performed a usability study. With their 164 "privacy-aware" participants they were able to select the most important indicators. They argue that further research with participants with general skills should be performed in the future work. It would be intersting to extend this user study with specific

corner cases from our paper. Similar studies were performed to analyze the usage of TLS indicators in web browsers. Schechter et al. performed laboratory experiments where they analyzed user behavior in respect to different browser warnings [28]. For example, they found out that all 63 tested users used their banking credentials on banking websites delivered over pure HTTP. Sunshine et al. [29] performed a study on SSL warning effectiveness. Their conclusion is that blocking unsafe connections and minimizing TLS warnings would improve the warning effectiveness and user security. Felt et al. studied TLS indicators in different browsers, including Chrome, Firefox, Edge, and Safari [30]. They performed a user study with 1329 participants to select the most appropriate indicators reporting positive as well as negative TLS status. The selected indicators have then been adopted by Google Chrome. Other researchers concentrated on special cases like the evaluation of Extended Validation certificates [31] or TLS indicators in mobile browsers [32].

9 Future Work

User Study On most clients the evaluation results were obvious. In the case of perfect or partial forgery, little to no discussion is needed because it should be clear that a user can not distinguish between a valid and a spoofed signature. However, some clients displayed conflicting information, for example, an erroneous seal *and* a success dialog. Other clients expect a more elaborate workflow from the user, such as clicking through the UI.

We currently classify a weak forgery as "not vulnerable" because the user has at least a chance to detect it and we did not measure if users actually detected it. A user study could clarify whether our weak forgery findings (e.g., conflicting security indicators) would convince email users, and answer the following research questions: Do users pay attention to email security indicators in general? Do users examine digital signature details (in particular for partial forgeries)? How do users react once they detect broken signatures?

We believe that such a study would help to understand the current email security ecosystem and our paper lays the foundations for such a study.

S/MIME Signatures in AS2 *Applicability Statement 2 (AS2)* as described in RFC 4130 is an industry standard for secure Electronic Data Interchange (EDI) over the Internet. It relies on HTTP(S) for data transport and S/MIME to guarantee the authenticity and integrity of exchanged messages. As critical sectors such as the energy industry heavily rely on AS2 for their business processes, it would be interesting to evaluate if our attacks (e.g., in the CMS class) can be applied to AS2. Unfortunately, we did not have the opportunity to test commercial AS2 gateways yet.

Email Security Fuzzing As described in subsection 7.1, our tests with different attack vectors led to unintentional crashes in several email applications. More rigorous testing and systematic fuzzing with MIME, S/MIME and PGP structures could uncover further vulnerabilities.

10 Conclusion

We demonstrated practical email signature spoofing attacks against many OpenPGP and S/MIME capable email clients and libraries. Our results show that email signature checking and correctly communicating the result to the user is surprisingly hard and currently most clients do not withstand a rigorous security analysis. While none of the attacks directly target the OpenPGP or S/MIME standards, or the underlying cryptographic primitives, they raise concerns about the practical security of email applications and show that when dealing with applied cryptography, even if the cryptosystem itself is considered secure, the overall system needs to be looked at and carefully analyzed for vulnerabilities. Implementing countermeasures against these vulnerabilities is very challenging and, thus, we recommend that OpenPGP, MIME, and S/MIME offer more concrete implementation advices and security best practices for developing secure applications in the future.

Acknowledgements

The authors would like to thank Kai Michaelis and Benny Kjær Nielsen for insightful discussions about GnuPG and its secure integration into the email ecosystem, and our anonymous reviewers for many insightful comments.

Juraj Somorovsky was supported through the Horizon 2020 program under project number 700542 (FutureTrust). Jens Müller was supported by the research training group 'Human Centered System Security' sponsored by the state of North-Rhine Westfalia.

References

[1] J. Postel, "Simple Mail Transfer Protocol," August 1982. RFC0821.

[2] D. Crocker, "Standard for the format of ARPA internet text messages," August 1982. RFC0822.

[3] H. Hu and G. Wang, "End-to-end measurements of email spoofing attacks," in *27th USENIX Security Symposium (USENIX Security 18)*, (Baltimore, MD), pp. 1095–1112, USENIX Association, 2018.

[4] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer, "OpenPGP message format," November 1998. RFC2440.

[5] B. Ramsdell, "S/MIME version 3 message specification," June 1999. RFC2633.

[6] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, "OpenPGP message format," November 2007. RFC4880.

[7] R. Housley, "Cryptographic Message Syntax (CMS)," September 2009. RFC5652.

[8] P. Resnick, "Internet message format," October 2008. RFC5322.

[9] 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)*, (Baltimore, MD), pp. 549–566, USENIX Association, 2018.

[10] B. Ramsdell and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) version 3.2 message specification," January 2010. RFC5751.

[11] F. Strenzke, "Improved message takeover attacks against S/MIME," Feb. 2016. `https://cryptosource.de/posts/smime_mta_improved_en.html`.

[12] D. K. Gillmor, "Memory Hole spec and documentation." `https://github.com/autocrypt/memoryhole`, 2014.

[13] "CVE-2013-4402." Available from MITRE, 2013.

[14] "CVE-2006-0049." Available from MITRE, 2006.

[15] R. Klafter and E. Swanson, "Evil 32: Check your GPG fingerprints." `https://evil32.com/`, 2014.

[16] D. Bleichenbacher, "Forging some RSA signatures with pencil and paper." Presentation in the rump Session CRYPTO 2006, Aug. 2006.

[17] A. Furtak, Y. Bulygin, O. Bazhaniuk, J. Loucaides, A. Matrosov, and M. Gorobets, "BERserk: New RSA signature forgery attack." Presentation at Ekoparty 10, 2014.

[18] B. Ramsdell, "S/MIME version 3 certificate handling," June 1999. RFC2632.

[19] L. Liao, *Secure Email Communication with XML-based Technologies*. Europ. Univ.-Verlag, 2009.

[20] A. Levi and C. B. Güder, "Understanding the limitations of S/MIME digital signatures for e-mails: A GUI based approach," *computers & security*, vol. 28, no. 3-4, pp. 105–120, 2009.

[21] A. Whitten and J. D. Tygar, "Why Johnny can't encrypt: A usability evaluation of PGP 5.0," in *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, SSYM'99, (Berkeley, CA, USA), pp. 14–14, USENIX Association, 1999.

[22] D. Davis, "Defective sign & encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML," in *Proceedings of the General Track: 2001 USENIX Annual Technical Conference*, (Berkeley, CA, USA), pp. 65–78, USENIX Association, 2001.

[23] D. K. Gillmor, "Inline PGP signatures considered harmful." `https://dkg.fifthhorseman.net/notes/inline-pgp-harmful/`, 2014.

[24] "CVE-2007-1263." Available from MITRE, 2007.

[25] S. Haddouche, "Mailsploit." `https://mailsploit.com/`, 2017.

[26] F. Ribeiro, "The ROPEMAKER email exploit," 2017.

[27] J. Lausch, O. Wiese, and V. Roth, "What is a secure email?," in *European Workshop on Usable Security (EuroUSEC)*, 2017.

[28] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer, "The emperor's new security indicators," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, pp. 51–65, May 2007.

[29] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor, "Crying wolf: An empirical study of SSL warning effectiveness," in *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM'09, (Berkeley, CA, USA), pp. 399–416, USENIX Association, 2009.

[30] A. P. Felt, R. W. Reeder, A. Ainslie, H. Harris, M. Walker, C. Thompson, M. E. Acer, E. Morant, and S. Consolvo, "Rethinking connection security indicators," in *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, (Denver, CO), pp. 1–14, USENIX Association, 2016.

[31] R. Biddle, P. C. van Oorschot, A. S. Patrick, J. Sobey, and T. Whalen, "Browser interfaces and extended validation SSL certificates: An empirical study," in *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, (New York, NY, USA), pp. 19–30, ACM, 2009.

[32] C. Amrutkar, P. Traynor, and P. C. van Oorschot, "Measuring SSL indicators on mobile browsers: Extended life, or end of the road?," in *Information Security* (D. Gollmann and F. C. Freiling, eds.), (Berlin, Heidelberg), pp. 86–103, Springer Berlin Heidelberg, 2012.