Legacy Encryption Downgrade Attacks against LibrePGP and CMS

Falko Strenzke and Johannes Roth

MTG AG, Darmstadt, Germany

Abstract. [1] This work describes vulnerabilities in the specification of the AEAD packets as introduced in the novel LibrePGP specification that is implemented by the widely used GnuPG application and the AES-based AEAD schemes as well as the Key Wrap Algorithm specified in the Cryptographic Message Syntax (CMS). These new attacks exploit the possibility to downgrade AEAD or AES Key Wrap ciphertexts to valid legacy CFB- or CBC-encrypted related ciphertexts and require that the attacker learns the content of the legacy decryption result. This can happen either due to the human recipient returning the decryption output, which has entirely pseudo-random appearance, to the attacker or due to a programmatic decryption oracle in the receiving system. The attacks effect the decryption of low-entropy plaintext blocks in AEAD ciphertexts and, in the case of LibrePGP, also the manipulation of existing AEAD ciphertexts. For AES Key Wrap in CMS, full key decryption is possible. Some of the attacks require multiple successful oracle queries. The attacks thus demonstrate that CCA2 security is not achieved by the LibrePGP and CMS AEAD or Key Wrap encryption in the presence of a legacy cipher mode decryption oracle. The proper countermeasure to thwart the attacks is a key derivation that ensures the use of unrelated block cipher keys for the different encryption modes.

Keywords: AEAD · downgrade · CMS · LibrePGP · decryption-oracle

1 Introduction

This work describes downgrade attacks that violate the confidentiality and authenticity properties of authenticated encryption, technically referred to as AEAD for "authenticated encryption with additional data", as specified in the LibrePGP protocol and in the long-standing CMS protocol [Hou07]. LibrePGP is a recent fork of the OpenPGP standard in a personal IETF draft [KT23] that introduces AEAD ciphers with the *OCB Encrypted Data packet*, referred to as *OCB Packet* throughout this work. GnuPG[2] implements the new packet starting from GnuPG version 2.3 [gnu] and also the RNP OpenPGP library[3] supports it. CMS is a widely used standard for cryptographic messages protected by encryption and signatures based on public key algorithms and using X.509 certificates. It supports AES-CCM and AES-GCM as AEAD modes [Hou07]. One of the prominent usages of CMS is in the S/MIME protocol for email encryption and signature [STR19]. AES Key Wrap with Padding [Hou09b] is used in CMS for the protection of Symmetric Key Package Content Types [Tur11]. The latter is used for instance by the Dynamic

E-mail: falko.strenzke@mtg.de (Falko Strenzke), johannes.roth@mtg.de (Johannes Roth)

[1] This work was carried out in the scope of Project 480 – "Integration von Post-Quanten Kryptografie in den E-Mail-Client Thunderbird" commissioned by the German Federal Office For Information Security (BSI)

[2] https://www.gnupg.org/

[3] https://www.rnpgp.org/software/rnp/

Symmetric Key Provision Protocol (DSKPP) [DPMN10]. In the following, we use the term *modern cipher mode* to denote AEAD as well as Key Wrap schemes.

The lack of integrity protection of the legacy encryption modes CFB, present in OpenPGP, and CBC, present in CMS, and the resulting security problems are long known. Straightforward decryption oracle attacks against these protocols have been published [KS00, JKS02, MRLG15a] already 20 years ago. Such oracle attacks exploit the fact that by performing certain transformations on an original legacy-mode ciphertext C sent by Alice to Bob and that decrypts to the plaintext P, the attacker can craft a new ciphertext C' that decrypts to different plaintext P', but where P can be reconstructed from P'. The attack is feasible since when Bob receives and decrypts C' the resulting plaintext P' will look garbled and may well lead to the assumption that something went wrong during the decryption. With or without further social engineering by the attacker, Bob may send P' back to the attacker in order to allow for an analysis of the cause of the decryption error. More sophisticated approaches to trick a user into acting as a decryption oracle are described in the more recent work [MBP+19].

Another possibility for the attacker to learn parts of the plaintext is the existence of format oracles [MRLG15b] that may be present in an application that performs automated processing of messages and generation of answers. Such oracles can be manifest in the form of an error oracle, i.e., the application returns different error messages based on the content of the plaintext, or be exploitable via timing side-channels. In a recent work [IPK+23], a format oracle in iOS Mail was successfully used for the decryption of S/MIME emails. Moreover, in the case of CBC, the possibility of a generic padding oracle is given [Vau02].

Another important previous work devising attacks that exploit the malleability of the ciphertext in OpenPGP is the rather recent Efail paper [PDM+18]. In this work, the malleability of the SED packets is exploited to inject so-called *exfiltration gadgets* into the plaintext, which effectively allow for the decryption of the plaintext or parts of it.

LibrePGP's OCB Encrypted Data packet which generally realizes AEAD encryption in this protocol, and the AES-based AEAD modes specified for CMS are in principal suited to thwart such oracle attacks that exploit the malleability of the encryption mode, as the AEAD modes protect the integrity of the ciphertext with a cryptographic checksum referred to as the *authentication tag* or simply *tag*. However, in this work we show that this goal is not met by by these two protocols when the recipient of the AEAD encrypted message also supports the decryption under the respective legacy encryption mode. In the same manner, in CMS, AES Key Wrap, which protects the integrity of the transmitted keys with a 64-bit checksum, the so-called AIV, can be attacked. The fundamental reason behind these vulnerabilities is that the decryption operation of the legacy mode enables an oracle for the block cipher operation, namely block encryption in the case of CFB and block decryption in the case of CBC, which can then be used to carry out the secret block cipher operations in order to craft or decrypt valid modern-cipher-mode ciphertexts without knowledge of the symmetric key.

As a highly relevant previous work, on which our attacks extend, in 2013, Jager, Paterson, and Somorovsky described a cross-mode attack against XML encryption [JPS13]. Specifically, their attack exploits the fact that low-entropy plaintext blocks in CCM-encrypted XML-messages can be efficiently decrypted by exploiting the legacy XML CBC decryption by using the CBC decryption result to emulate the block decryption of the underlying cipher. For a successful attack, a means is required to learn the contents of the decrypted CBC plaintext. This can be achieved either by a human user acting as a decryption oracle or another type of oracle as described above. Since this type of attack makes use of an oracle that provides the inverse direction of the cipher operation compared to the direction that is needed to decrypt a ciphertext, we refer to this type of attack as an *inverse decryption oracle attack*. The same work also points out the principle approach of the vulnerability of AES Key Wrap to downgrade attacks.

In this work, we show two principally different types of attacks that can be carried out against LibrePGP and CMS AEAD ciphertexts. The first one is only applicable to LibrePGP OCB Packets and allows the manipulation of an existing AEAD ciphertext by replacing or removing entire OCB chunks of the plaintext, a chunk being a single OCB ciphertext including the authentication tag of which a LibrePGP OCB Packet may contain multiple. This attack, like the following, is based on querying a legacy-cipher-mode decryption oracle and learning the decryption result.

The second basic type of oracle attack makes use of the legacy decryption oracle as an inverse decryption oracle first devised in [JPS13] and is applicable to AEAD in both LibrePGP and CMS. In this case, the attacker cannot straightforwardly decrypt the AEAD encrypted message, but has to be able to come up with a set of guesses for individual blocks of the plaintext and can then use the oracle to find out which of the guesses is correct. More formally speaking, the attacker may be able to decrypt low-entropy blocks at known positions within the message. With such an attack, an attacker could for instance recover a sufficiently short secret PIN code in message that is otherwise known to him.

Since applications which use LibrePGP or CMS for automated message processing and response generation are not widespread and because the aim of this work is to show up the principal protocol weaknesses, in the development of our attacks we assume fully-plaintext-revealing decryption oracles. As we explain in Sec. 2.8, we do not expect that email applications are generally vulnerable to our attacks in a straightforward manner. The reason is that an email application that is vulnerable to the AEAD downgrade attacks against LibrePGP and S/MIME (which is based on CMS), would also be vulnerable to much more reliable and efficient attacks based on the manipulation of the legacy-cipher-mode-encrypted emails, which are still in much wider use than the AEAD-encrypted ones. Accordingly, such an application must be assumed to be vulnerable against some variant of the generally more effective attacks presented in [MBP+19, PDM+18].

Note that the packets formally labelled "OCB Packet" in LibrePGP allow for both the encryption using the modes OCB and EAX. For the sake of completeness, we point out that EAX mode specified in LibrePGP is also vulnerable to attacks similar to those as we describe for OCB. However, the vulnerabilities of EAX mode are not covered by this work since the EAX mode is already deprecated in LibrePGP and the OCB packets are not yet in widespread use. Thus the analysis of this encryption mode is less relevant. Accordingly, in this work we only provide a detailed analysis of the security of the OCB mode in LibrePGP. We only want to point out that the attacks on EAX can be straightforwardly constructed by the same principle approach as that devised against OCB and allow for full decryption of plaintexts, since EAX decryption makes use only of the block cipher encryption, for which the attacker can directly exploit a CFB decryption oracle.

Besides the recently established LibrePGP protocol, there is also the so-called crypto-refresh [WHWY] as the official successor of the current OpenPGP standard, RFC 4880 [CDF+07]. The crypto-refresh has taken all hurdles in the IETF standardization process and is in the last stage of becoming an official RFC. The AEAD packets introduced by the crypto-refresh differ from the LibrePGP OCB Packet in that their processing involves a key derivation: the distributed symmetric key (typically stemming from a public-key encryption or key exchange) is not directly used for the encryption of the AEAD-encrypted data, but used as the input to a key derivation function the output of which is used as the AEAD key. Since the key derivation is also dependent on the content-encryption algorithm, key separation between the different encryption modes is achieved, which makes the crypto-refresh's AEAD packets immune to downgrade attacks.

Regarding the use of AES Key Wrap in CMS, we show straightforward full plaintext recovery attacks under the assumption of a CBC decryption oracle, which in a concrete vulnerable system may for instance be given by a padding oracle.

The buildup of the paper is as follows. In Sec. 2 and 3 we introduce the new attacks against LibrePGP and CMS, respectively. Sec. 4 reports on the responsible disclosure process and Sec. 5 discusses the appropriate countermeasures to thwart the attacks. Finally, Sec. 6 gives the conclusion.

2 Attacks against LibrePGP OCB packets

In the following subsections, we first introduce some preliminaries in Sec. 2.1. Following, in Sec. 2.2 we explain how OpenPGP legacy-mode-decryption can be leveraged to an ECB encryption oracle. Sec. 2.3, 2.4, 2.5, and 2.6 describe the novel attacks. Our proof-of-concept implementation of one of these attacks is described in Sec. 2.7. Finally, Sec. 2.8 explores the potential of real world applications based on LibrePGP for being vulnerable to our attacks.

2.1 Preliminaries: OpenPGP and LibrePGP message encryption

2.1.1 OpenPGP and LibrePGP hybrid encryption

Public-key encrypted messages in OpenPGP function according to the well-known asymmetric/symmetric hybrid encryption approach: The encrypted message begins with one or more public key encrypted session key (PKESK) packets, one for each recipient. When decrypting the respective PKESK packet with their own private key, the recipient receives the session key. Then follows the data that is symmetrically encrypted under the session key. The data to be encrypted always has to be enveloped in a valid packet, e.g., a *Literal Data* (LIT) Packet. In LibrePGP, the following types of symmetrically encrypted data packets exist:

- *Symmetrically Encrypted Data (SED) Packet*, defined in OpenPGP, i.e. RFC 4880 [CDF+07]. This packet is encrypted using a slightly modified variant of CFB encryption without any integrity protection.
- *Symmetrically Encrypted Integrity Protected Data (SEIPD) Packet*, defined in OpenPGP, i.e. RFC 4880 [CDF+07]. This packet type is used in conjunction with a Modification Detection Code (MDC) Packet: The data to be CFB-encrypted is formed by the sequence of the message, enveloped for instance in a LIT packet, and an MDC packet containing the SHA-1 hash of the LIT packet including its packet header. SEIPD packets also use CFB encryption. The verification of the SHA-1 hash of the plaintext data contained in the MDC packet provides an ad-hoc mechanism for the protection of the ciphertext's integrity.
- *OCB Packet*, defined in LibrePGP [KT23]. OCB packets encrypt data using one of the two AEAD modes OCB or EAX, where the latter is marked as deprecated.

2.1.2 Shortcomings of the SEIPD Packet

SEIPD packets specified in OpenPGP have two shortcomings: first of all, they can be downgraded to SED packets and then the malleability of SED packets can be used to modify the message [Per02, Mag15]. See also [PDM+18] for comprehensive overview of further aspects to the downgrade attacks and their exploitation in a decryption oracle attack. In this work, we show that in principle, in a considerably more complex attack, though, LibrePGP OCB packets can also be downgraded to SED packets.

A second shortcoming of SEIPD packets is that they do not achieve CCA2 security, i.e. are vulnerable to adaptively chosen ciphertext attacks, even when not considering downgrade attacks to SED. This type of weakness is for instance outlined in [Wag, Cry]. We demonstrate this weakness in App. A.

Table 1: Notation used throughout this work. Numerous items have been taken without or with minor adaptions from [KR14].

$\|k\|$	the bit length of the value k
b	The letter b denotes the block width of the underlying block cipher in bits.
$[0]^x$	The bit string formed by x zero bits.
X[i]	The i-th bit of the string S (indices begin at 0, so if X is 011, then X[0] == 0, X[1] == 1, X[2] == 1).
$X[a:b]$	The substring of the bit string X ranging from the bit positions a to b inclusive.
str2num(S)	The big-endian conversion of a bit string S into an integer (e.g., str2num(1110) == 14).
num2str(i, n)	The big-endian conversion of an integer i into a bit string of length n (e.g., num2str$(14, 4) = 1110$ and num2str$(1, 2) == 01$).
ocbDouble(S)	If $S[1] == 0$, then ocbDouble$(S) = (S[2:128] \,\|\, 0)$; otherwise, ocbDouble$(S) = (S[2:128] \,\|\, 0) \oplus ([0]^{120} \,\|\, 10000111)$.
$E_k(X)$	AES block encryption of the block X under the key k. In some instances the indexing with k is omitted, as in this work we are not concerned with any variation in the employed block cipher key.

2.1.3 The LibrePGP OCB Packet with OCB mode encryption

The OCB encryption algorithm [KR14] takes as input the key, a nonce, the plaintext, and additional data. The latter is authenticated but does not become part of the resulting ciphertext. It outputs a ciphertext at the end of which is the authentication tag.

An OCB Packet may consist of more than one OCB chunk. Each chunk is a complete OCB ciphertext ending with the corresponding authentication tag. At the very end of the OCB Packet, an empty OCB chunk is encrypted to produce the final authentication tag. The additional data input to the OCB encryption algorithm for each chunk includes the index of the chunk within the packet. The detailed buildup of the additional data used in each chunk is given in App. B.

The nonce used for the encryption of each chunk in an OCB Packet is $V \oplus I$, where $V \in \{0,1\}^{120}$ is the initialization vector supplied on the protocol level and I is the chunk index encoded as a big-endian value of the same width as V. The chunk index is counted starting from zero.

2.1.4 SED Packet encryption in OpenPGP

The encryption of SED packets makes use of the CFB mode. In this mode, encryption of a block is given as $C_i = E_k(C_{i-1}) \oplus P_i$ and the decryption as $P_i = E_k(C_{i-1}) \oplus C_i$ with $i \in \{1, \ldots, n\}$ and where $E_k()$ is the encryption of a block under the key k, and $C_0 = \text{IV}$ is the initialization vector.

The encryption of SED packets in OpenPGP does not use CFB straightforwardly, but is realized by a two-step CFB encryption which we describe in the following. Here, and throughout this work, all encryption operations are performed using the session key k resulting from the decryption of the corresponding PKESK packet without explicitly noting the use of k. Table 1 specifies various notations used in the algorithms.

- First, CFB-encrypt a bit string $Y \in \{0,1\}^{144}$ where $Y[96:127] = Y[128:143]$, i.e., the last two octets are a copy of the preceding two octets, the remaining octets chosen at random, and an IV of all zero bits.
- Let the result of this first encryption step be H.
- Set IV $\leftarrow H[16:143]$
- CFB-encrypt the payload data using the IV computed in the previous step and append the encryption result to H to form the complete OpenPGP-CFB ciphertext.

The corresponding decryption algorithm is given in Alg. 1. The condition "have quick-check" and its relevance for attacks on actual OpenPGP implementations will be discussed further down.

Algorithm 1 OpenPGP's SED two-step CFB decryption. The two arguments to SED-$\text{DEC}_K()$ are the first-step and second-step ciphertexts, respectively.

Algorithm SED-$\text{DEC}_K(H \,\|\, B_1 \,\|\, \ldots \,\|\, B_m)$ with $H \in \{0,1\}$ and $B_i \in \{0,1\}^{128}$
    $Y \leftarrow \text{CFB-DECRYPT}_K([0]^{128}, H)$ // $Y \in \{0,1\}^{128+16}$
    if have quick-check AND $Y[96:127] \neq Y[128:143]$ **then**
        Abort with error
    end if
    IV $\leftarrow H[16:143]$
    return $\text{CFB-DECRYPT}_K(\text{IV}, B_1 \,\|\, \ldots \,\|\, B_m)$
end Algorithm

Algorithm 2 ECB mode encryption realized through a SED CFB decryption-oracle for a sequence of n block cipher blocks B_i in one call. b is the block size of a cipher block in bits. For a visual depiction of the transformation from the SED decryption result to the ECB decryption result refer to Fig. 1.

Algorithm ECB-ENC-$\text{OC}_K(B_1 \,\|\, \ldots \,\|\, B_n)$
    $B_{n+1} = [0]^b$
    $P = P_1 \,\|\, \ldots \,\|\, P_{n+1} = \text{SED-DEC}_K([0]^{144} \,\|\, B_1 \,\|\, B_2 \,\|\, \ldots \,\|\, B_{n+1})$
    for $i = 1$ to n **do**
        $Q_i = P_{i+1} \oplus B_{i+1}$
    end for
    return $\{Q_i | i = 1, \ldots n\}$
end Algorithm

2.2 OpenPGP SED Packet CFB decryption as an ECB encryption oracle

In this section we show how the OpenPGP SED Packet CFB decryption can be used as an ECB encryption oracle by straightforward transformations on the returned plaintext and address the applicability of the decryption oracle to currently existing LibrePGP implementations.

If an attacker has access to the algorithm SED-DEC_K given in Alg. 1 as a decryption oracle, an ECB encryption oracle can be built on top of this as specified in Alg. 2. See Fig. 1 for an intuitive depiction of how the ECB encryption oracle is built from the CFB decryption oracle.

Table 2: Overview of OpenPGP implementations regarding aspects that influence their susceptibility to SED decryption oracle attacks.

Implementation	**Supports SED decryption**	**Enforces quick-check**
GnuPG 2.4	limited in default configuration (see text)	no
RNP 1.17.0	yes	yes

Note that if the condition "have quick-check" in Alg. 1 amounts to "true", this greatly increases the number of queries necessary to retrieve a decryption result from the CFB-decryption oracle. However, the implementation of the "quick-check" poses a vulnerability in itself [MZ06]. Accordingly, the LibrePGP specification warns in its "Security Considerations" section, to use the quick-check, if at all, then with care. While GnuPG itself does not implement the quick-check, RNP does so, as we inferred from the source code. Table 2 gives an overview of the SED decryption support of both implementations.[4]

When decrypting a SED Packet with GnuPG, the following message is output

```
gpg: WARNING: message was not integrity protected
gpg: decryption forced to fail!
```

and the file is still decrypted if the output is written to stdout, i.e., if no output file was specified on the command line. However in this case, as long as the option `ignore-mdc-error` is not set in the GnuPG configuration file, the application exits with a non-zero exit code. If an output file was specified, the file is deleted again afterwards. Also the corresponding function of the GPGME library returns an error code for SED decryption [Koc24].

2.3 A high-level view of the attacks on LibrePGP OCB Packet encryption

In the following subsections we provide the attacks against the LibrePGP OCB Packet encryption based on the presence of an OpenPGP SED decryption oracle. Specifically, we describe the approach to the initial oracle query (Sec. 2.4) that is needed for both of the subsequently described attacks, an attack that allows for the manipulation of an existing OCB Packet (Sec. 2.5), and an attack that enables the decryption of low entropy plaintext blocks (Sec. 2.6).

The underlying idea of the attacks is to perform the algorithm needed to achieve the desired computation with the help of the SED decryption oracle. Within this algorithm, whenever the block cipher encryption operation is needed, the respective CFB decryption oracle implied by the SED decryption oracle is queried to retrieve the respective operation result, making use of the fact that an SED decryption oracle can be trivially transformed to an ECB encryption oracle (see Sec. 2.2). As an optimization to reduce the number of oracle queries that are required to conduct a specific attack, the attacker provides as many blocks to the oracle in a single query as possible according to the parallelisability of the block cipher operations in the attack algorithm.

However, due to the protocol specification of OpenPGP requiring a valid packet to be found inside the decrypted data, only a subset of the initial oracle queries will return an actual decryption result. As detailed in Sec. 2.4, this leads to a high number of initial

[4] According to the Sequoia interoperability test suite, the latest RNP version 1.17.0 supports SED decryption. See https://tests.sequoia-pgp.org/#SED_encrypted_message

oracle queries, until a ciphertext is found that returns a decryption result in the first place. It should be noted that nevertheless, there is also the possibility that the attacks can be conducted with a single initial query perceived by the victim that reveals the plaintext: that is the case when there are sufficiently many recipients of the original message, in which case each recipient receives the message encrypted with the same session key. Accordingly, in this case, the attacker can distribute the set of initial queries over the whole set of simultaneous recipients.

2.4 The initial SED-Oracle query

When the attacker performs the downgrade attack from an OCB-encrypted AEAD packet to a legacy CFB-encrypted SED Packet and lets the victim decrypt the message, the CFB-decryption will result in a plaintext consisting of what appears as random bytes. However, the OpenPGP packet semantics prescribe that a decrypted plaintext must start with a valid packet from a certain set of suitable packet types. According to the LibrePGP specification, the plaintext must contain either an Encrypted Message, a Signed Message, a Compressed Message, or a Literal Message. However, we can restrict the possibility of the appearance of a valid packet at the start of the pseudo random plaintext bytes to Literal Message which is realized by a LIT packet. The reason is that all other packets allowed by the standard need further successful processing which is highly improbable to achieve by chance: Processing random data as a compressed data packet will suffer from high chance of an error during decompression, and even in case of successful decompression require the by-chance appearance of another valid packet in the decompressed data. Furthermore, processing random data as encrypted or signed packets entails the attempted decryption or signature verification which certainly fails with overwhelming probability.

Thus, in order for the SED-oracle to output a decryption result, the ciphertexts that are input to the SED-oracle in the course of the attack must, in the general case, decrypt to data that starts with a LIT packet. Accordingly, by crafting ciphertexts for carrying out the attack, meaningful and interpretable data packets within the decrypted plaintext can appear entirely only by chance. Due to the nature of CFB-mode underlying the SED-oracle, that attacker has the possibility to randomize the leading part of the SED ciphertext to randomize the beginning of the decrypted plaintext during a sequence of oracle queries. Once he has found a ciphertext for which a plaintext is returned, he can reuse the leading part of that ciphertext that is responsible for the correct header of the LIT packet in further queries and only vary the part of the ciphertext which is responsible for the body of the decrypted LIT packet. This varied part is where he places the blocks he wishes to ECB-encrypt.

Fig. 1 gives an overview of the relations between the OpenPGP SED-ciphertext, the resulting plaintext, and the semantics of the LIT packet which has to be found in it. In the top left corner we see the first-step ciphertext of the SED two-step CFB decryption, the tail of which is used as the IV of the subsequent CFB decryption. After it, in the top row follows the sequence of the random blocks, all of which are chosen independently by the attacker during his attempts to achieve a first successful query. After the last random block follow the "oracle blocks", i.e., those ciphertext blocks, that the attacker wishes to have CFB-decrypted, i.e., block-encrypted. In the initial queries the oracle blocks form a sequence of a repeated pattern of pairwise equal blocks, e.g., we have oracle-block$_1$ = oracle-block$_2$, oracle-block$_3$ = oracle-block$_4$, etc. Note that according to the specification of the CFB mode, the final oracle block is not block-decrypted and thus cannot be used to retrieve an encrypted block.

In the next row we find the operations $E()$ for block encryption under the respective session key and the bit-wise XOR operations on the blocks. Beneath that row we find

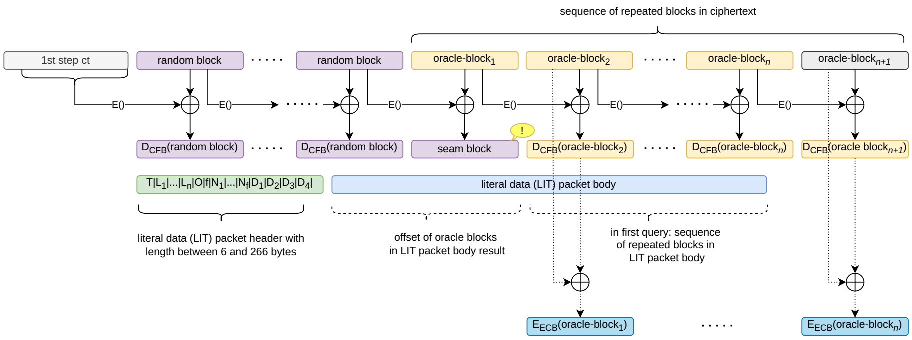

Figure 1: Overview of the structure of a LIT packet and the resulting offset of the returned oracle plaintext within the ciphertext. For the meaning of the symbols in the green block representing the LIT packet header see Table 3. The exclamation mark in the speech bubble at the seam block indicates the possible case that with the corresponding natural probability a number of the final bytes of the seam block are equal to the final bytes of the first D_{CFB}(oracle block) and thus the beginning of the block repetition pattern is recognized falsely too early.

the decryption result that starts with the decryption results of the leading random blocks which is entirely pseudorandom. The decryption of the first oracle block yields the seam block, which bears no useful information for the attacker. Then follow the blocks that are the result of the CFB-decryption of the oracle blocks.

In the next row we find the indication of the semantic interpretation of the decryption result. The green data block represents the header of the LIT packet. Table 3 shows its buildup with some rough estimates of the success probabilities to meet the criteria necessary for successful parsing of the LIT packet.

Since the filename field, the value of which is not part of the decryption result returned by GnuPG, has a variable length ranging from 0 to 255 bytes, a useful strategy for the attacker is to use a number of random blocks that ensures that they cover the maximal possible size of the header, which is 266 bytes. This ensures that the first oracle block appears in the packet body and thus the attacker receives it, since the packet body is what is returned by the decryption oracle. In this case he can determine the offset of the oracle blocks within the packet body: After the transformation to obtain the ECB decryption result, he simply looks for the beginning of the repeated pattern of pairwise identical blocks in the sequence of ECB decrypted blocks. The start of this pattern marks the position of $E_{\mathrm{ECB}}(\text{oracle-block}_1)$.

There is, however, the caveat that he cannot determine the start of the pattern with certainty in all cases. This is because with probability $1/256$, the last octet of the seam block will be identical to the last octet of each of the blocks of the first pair. This lets him determine the position of $E_{\mathrm{ECB}}(\text{oracle-block}_1)$ one octet too early (or more octets too early if there are further accidental such matches). Accordingly, a straightforward implementation of the attack fails with probability $1/256$.

2.5 Insertion of new OCB chunks

In this section we develop attacks that allow the insertion of new chunks with attacker chosen plaintext into an existing OCB Packet based on the exploitation of an SED decryption oracle. The attacker can replace chunks in an existing ciphertext or append new chunks at the end of the plaintext.

According to Alg. 4, Step 26, the authentication tag of each chunk is given as $T = E_k(s_{\tilde{n}} \oplus F_{\tilde{n}} \oplus L_{\$}) \oplus \mathrm{HASH}(K, A)$, where the variable names correspond to those used in Alg. 3 and 4.

Table 3: Buildup of an OpenPGP literal data (LIT) packet and estimated or presumed probabilities for the respective field to take on a valid value based on an entirely randomized plaintext. The symbols given in the first column are used in Fig. 1.

Symbol	**Field name**	**Size in bytes**	**Condition for validity**	**Estimated probability for validity**
T	packet tag for LIT	1	$T =$ 0xcb (new format) OR $T \in$ four valid old format tag octets	5/256
$L = (L_1, \ldots, L_n)$	body length	$\in \{0, 1, \ldots, 5\}$		unknown
O	format	1	$O =$ 0x62 (binary) OR $O =$ 0x74 (text) OR $O =$ 0x75 (UTF-8) OR $O =$ 0x6d (MIME)	1/64
f	filename-len	1	none	1
$N = (N_1, \ldots, N_f)$	filename	$\in \{0, 1, \ldots, 255\}$	valid UTF-8 [a]	unknown
$D = (D_1, \ldots, D_4)$	date	4	presumably none	1
	body	as specified by L	valid UTF-8 in case of format octet 0x74 and 0x75	unknown

[a] It is unknown to us whether a valid UTF-8 encoding is enforced during decryption in GnuPG or RNP.

Algorithm 3 Procedure for computing the value of HASH for an OCB GnuPG AEAD chunk.

Algorithm OCB-HASH(key $k \in \{0,1\}^{|\mathrm{K}|}$, additional data $A \in \{0,1\}^*$)
    $L_* = E_k([0]^{128})$
    $L_\$ = \mathrm{ocbDouble}(L_*)$
    $L_0 = \mathrm{ocbDouble}(L_\$)$
    $L_i = \mathrm{ocbDouble}(L_{i-1})$ for any integer $i > 0$
    $m = \lfloor |A|/128 \rfloor$
    parse A as $A_1 \,\|\, A_2 \,\|\, \ldots \,\|\, A_m \,\|\, A_*$ where $|A_i| = 128$ for each $1 \leqslant i \leqslant m$ and $0 \leqslant |A_*| < 128$
    $F_0 = [0]^{128}$ // Offset
    for i ← 1 to m **do**
        $F_i = F_{i-1} \oplus L_{\mathrm{ntz}(i)}$
    end for
    if $|A_*| > 0$ **then**
        $n \leftarrow m + 1$
        $F_n = F_m \oplus L_*$
        $A_n = (A_* \,\|\, 1 \,\|\, [0]^{127-|A_*|})$
    else
        $n \leftarrow m$
    end if
    $S_0 = [0]^{128}$ // Sum
    for i ← 1 to n **do**
        $S_i = S_{i-1} \oplus E_k(A_i \oplus F_i)$
    end for
    return $S = S_n$
end Algorithm

In order to achieve the computation of the tag T of an OCB chunk, the attacker runs Alg. 3. He first needs to query the CFB-oracle to receive the value L_*. Based on the knowledge of L_* he can compute $L_\$$ and L_i for $i \geqslant 0$.

We find that $S_m = \bigoplus_{i=0,\ldots m} E_k(A_i \oplus F_i)$. Note that the result of the OCB-HASH algorithm $S = S_m$ if $|A| = i \times 128$ for an integer i. Otherwise, the final non-full block receives a padding but this does not introduce any fundamental changes so for the sake of simplicity we ignore this case in our description of the attack that follows.

Algorithm 4 OCB encryption algorithm with the parameters key, nonce, additional data, and plaintext in that order. It returns the ciphertext C with $|C| = |P| + \text{taglen}$

1: **Algorithm** OCB-ENCRYPT($k \in \{0,1\}^{\text{keylen}}, N \in \{0,1\}^{120}, A \in \{0,1\}^*, P \in \{0,1\}^*$)
2: compute values L_*, $L_\$$, and L_i for $0 \leqslant i$ according to Step 2 in Alg. 3 et seq.
3: $\tilde{m} = \lfloor |P|/128 \rfloor$
4: parse P as $P_1 \,\|\, P_2 \,\|\, \ldots \,\|\, P_{\tilde{m}} \,\|\, P_*$ where $|P_i| = 128$ for each $1 \leqslant i \leqslant \tilde{m}$ and $0 \leqslant |P_*| < 128$
5: $\mathcal{N} = \text{num2str}(\text{taglen} \mod 128, 7) \,\|\, [0]^{120-|N|} || 1 || N$
6: $q = \text{str2num}(\mathcal{N}[123 : 128])$ // "bottom"
7: $f = E_k(\mathcal{N}[1 : 122] \,\|\, [0]^6)$ // "Ktop"
8: $l = f || (f[1 : 64] \oplus f[9 : 72])$ // "Stretch"
9: $G_0 = l[1 + q : 128 + q]$ // "Offset"
10: $s_0 = [0]^{128}$ // "Checksum"
11: **for** $1 \leqslant i \leqslant \tilde{m}$ **do**
12: $G_i = G_{i-1} \oplus L_{\text{ntz}(i)}$
13: $C_i = G_i \oplus E_k(P_i \oplus G_i)$
14: $s_i = s_{i-1} \oplus P_i$
15: **end for**
16: **if** $|P_*| > 0$ **then**
17: $\tilde{n} \leftarrow \tilde{m} + 1$
18: $G_{\tilde{n}} = G_{\tilde{m}} \oplus L_*$
19: $u = E_k(G_{\tilde{n}})$ // "Pad"
20: $C_{\tilde{n}} = P_* \oplus u[1 : |P_*|]$
21: $P_{\tilde{n}} = P_* \,\|\, 1 \,\|\, [0]^{127-|P_*|}$
22: $s_{\tilde{n}} = s_{\tilde{m}} \oplus P_{\tilde{n}}$
23: **else**
24: $\tilde{n} \leftarrow \tilde{m}$
25: **end if**
26: $T = E_k(s_{\tilde{n}} \oplus G_{\tilde{n}} \oplus L_\$) \oplus \text{HASH}(K, A)$
27: return $C = C_1 \,\|\, C_2 \,\|\, ... \,\|\, C_{\tilde{n}} \,\|\, T[1 : \text{taglen}]$
28: **end Algorithm**

For the creation of an entirely new AEAD-encrypted chunk for the plaintext P the attacker has to take the following steps.

1. The attacker chooses a plaintext P and determines the OCB nonce N and the additional data A according to the intended index of the chunk in the OCB Packet.

2. Compute the values in Alg. 4 from Steps 5 and 6.

3. Create the query ciphertext $R_1 = \underbrace{[0]^{128}}_{\substack{\text{plaintext} \\ \text{block for } L_*}} \,\|\, \underbrace{\mathcal{N}[1 : 122] \,\|\, [0]^6}_{\text{plaintext block for } f}$.

4. Retrieve the ECB-encryption of R_1 from the oracle and parse it into two blocks as ECB-encrypt$(R_1) = L_* \parallel f$ (Alg. 4, Step 7).

5. Compute the value of l from f (Alg. 4, Step 8).

6. Compute the required values of L^* and $\{L_i\}$ according to Alg. 3, Steps 2 and following.

7. Compute from the $\{L_i\}$ all required values of $\{F_i\}$ according to Alg. 3, Steps 8, 10, and 14.

8. Compute from l, L^*, and $\{L_i\}$ all required values of $\{G_i\}$ according to Alg. 4, Steps 9, 12, and 18 .

9. Compute the value $s_{\tilde{n}} = \bigoplus_{i=1,\dots,\tilde{n}} P_i$ according to Alg. 4, Steps 14 and 22.

10. Create the oracle ciphertext
$$R_2 = \underbrace{P_1 \oplus G_1 \parallel P_2 \oplus G_2 \parallel \dots \parallel P_{\tilde{m}} \oplus G_{\tilde{m}}}_{\text{regular ciphertext encryption}} \parallel \underbrace{G_{\tilde{n}}}_{\text{for } P_*} \parallel \underbrace{s_{\tilde{n}} \oplus F_{\tilde{n}} \oplus L_\$}_{\text{for tag computation}} \parallel \underbrace{A_1 \oplus F_1 \parallel \dots}_{\text{for HASH}(K,A)}$$
and use it to query the ECB encryption oracle a second time.

11. He thus receives from the ECB encryption oracle all values necessary to compute all the ciphertext blocks $C_1, \dots, C_{\tilde{n}}$ using Alg. 4, Steps 13 and 20.

12. Note: the values needed for the computation of HASH(K, A) according to Alg. 3 have already been retrieved through the second oracle query above (namely $E_k(A_i \oplus F_i)$) and the first query (namely $E_k([0]^{128})$).

13. He computes the tag $T = \underbrace{E_k(s_{\tilde{n}} \oplus F_{\tilde{n}} \oplus L_\$)}_{\text{from query } R_2} \oplus \underbrace{\text{HASH}(K, A)}_{\text{using query } R_2}$.

2.6 Decryption of low entropy blocks

If the plaintext of a LibrePGP OCB Packet contains low entropy blocks, i.e. blocks for the contents of which the attacker can create a reasonably short list of possible values for each block, he can find the correct guess out of this list by the following attack. For the sake of simplicity, we give a description of the attack for only attacking a single block within the ciphertext, excluding the final potential non-full block.

- He has a set of h guesses $\mathcal{U} = \{U_i \mid 1 \leqslant i \leqslant h \text{ and } U_i \in \{0,1\}^{128}\}$ for the plaintext block P_t at block position t for a ciphertext C.

- He conducts the first query in the same way as in the procedure described in Sec. 2.5 and thus can compute the values $\{G_i\}$ (see Step 8 in that section).

- He then computes the set of oracle blocks $\{Q_i = U_i \oplus G_t | U_i \in \mathcal{U}\}$.

- He creates a second query ciphertext $r_2 = Q_1 \parallel Q_2 \parallel \dots \parallel Q_h$ and feeds it to the ECB encryption oracle.

- The ECB-encryption result returned by the oracle is parsed into blocks as $D_1 \parallel D_2 \parallel \dots \parallel D_h$

- He computes $\{X_i = G_t \oplus D_i | 1 \leqslant i \leqslant h\}$

- If there is one $X_j = C_t$, then the attacker knows that $U_j = P_t$.

The attack functions since the blocks $X_i = G_t \oplus E_k(U_i + G_t)$ are the expected ciphertext blocks for the plaintext guesses in $\mathcal{U}$ at the plaintext position t.

We want to point out that in the special case where the attacker has partial control over the plaintext of the attacked message, such an attack can be extended to a full plaintext recovery by successively modifying the block-offset into the unknown plaintext and thus applying a divide-and-conquer style attack to recover one unknown plaintext byte at a time. This type of attack is referred to as *blockwise chosen-boundary attack* by the original authors [DR]. In [JPS13] it is applied to the inverse decryption oracle attacks against XML encryption. This potential extension of the attack also applies to the equivalent attacks against CMS presented in Sec. 3.3.

2.7 Practical attacks on GnuPG

We implemented and successfully tested an attack which replaces an OCB chunk with plaintext containing only whitespaces of the same size as the original chunk within an existing OCB-encrypted LibrePGP OCB Packet using AES as the cipher by performing the procedure described in Sec. 2.5. The OCB Packet to be modified was created with GnuPG 2.4.3 with the command

```
gpg --output <out-file> -a --recipient <recipient> --force-aead --aead-algo=ocb -z0 --chunk-size
      =6 --encrypt <plaintext-file>
```

Compression was disabled with the option `-z0` since otherwise the changes in the middle of the plaintext lead to a decompression error with high probability. The chunk size was set to 64 bytes in order make the attack feasible for a small ciphertext. The execution of the attack according to Sec. 2.5 involves the initial query step, in which the random leading part of the OpenPGP CFB ciphertext, having a length of 256 bytes, is varied until a successful answer is returned. The leading random pattern is followed by 100 oracle blocks. When running the initial query step against GnuPG 2.2.27 as the decryption oracle for the SED ciphertext, it takes between a few ten to a few hundred queries until a decryption result is returned by GnuPG. This is a considerably higher success rate than what should be expected according to the estimation in Tab. 3. However, we did not investigate the reason for this discrepancy. When a decryption result is returned, our attack application identifies the start of the oracle block pattern by searching for the repeated block pattern and optionally verifies with the help of the correct session key provided as a command line option, that the oracle blocks were correctly ECB-encrypted. We point out that for reasons of simplicity, our implementation of the attack deviates from the optimized procedure described in Sec. 2.5 in that it only uses a fixed oracle block in the initial oracle query and thus needs one more oracle query than the described optimized procedure.

For the oracle queries following the initial one, the leading part of the ciphertext determined in the first query is reused and thus the algorithmic oracle calls only afford a single query to the SED decryption. The modified AEAD ciphertext is then successfully decrypted using GnuPG 2.4.3.

2.8 Vulnerable applications using GnuPG

As stated in Section 2.1.4, in the default configuration, i.e. without setting the configuration option `ignore-mdc-error`, the GnuPG command line application outputs the plaintext on the stdout and exits with a non-zero exit code when decrypting an SED Packet. An application using GnuPG that rejects the output in case of a non-zero exit code is thus not vulnerable to the attacks. Note that this still does not mean that sending LibrePGP packets with GnuPG is safe, since the vulnerability is a principal problem of the LibrePGP protocol and its manifestation depends on the implementation choices of the recipient's client.

Regarding the question under which circumstances real world application using LibrePGP OCB packets could be vulnerable against our attacks, we suggest two main scenarios. The first one is given where a human user reveals the full plaintext, due to it appearing as “garbled”, to the attacker. The large number of initial queries, which is in the domain of multiple tens or hundreds could potentially be distributed over a set of users. In this case, the victim, against whom the full attack is conducted, would only observe two oracle queries (in the case of the attack from Sec. 2.5) which they have to answer with the resulting “garbled” decryption result.

However, note that conducting this attack against LibrePGP-based email encryption is not realistic, since an email client that is vulnerable against our attacks would also expose the vulnerability of SEIPD packets being downgradeable to SED Packets, which was exploited in the Efail attack [PDM+18]. This vulnerability can be exploited much more straightforwardly, i.e., with only a single modified ciphertext. Accordingly, for any application that also supports OpenPGP SEIPD packets, which clearly is the case for current email clients, the vulnerability of SEIPD packets would be the much greater problem.

The second conceivable scenario for a realistic decryption oracle would be the presence of format oracles [MRLG15a]. A format oracle is given when an error message due to an invalid message format reveals information about the plaintext. Such an oracle might possibly also be exploitable via a timing side channel, which might allow to recover detailed information about the type of error and the position in the plaintext. On the one hand, a setting where format oracles reveal the values of individual octets in the decrypted plaintext during automated processing would lead to an even larger number of queries. But such a setting would also make the execution of a large number of queries generally more feasible than in the case of required user interaction. However, we did not analyze this possibility further due to the absence of widely used applications that use OpenPGP or LibrePGP encryption in a protocol that makes oracle attacks feasible. Accordingly, this attack vector remains a hypothetical one.

3 Attacks on AES-CCM, AES-GCM, and AES Key Wrap in CMS

In this section we describe our novel attack against AES-CCM and AES-GCM in CMS. After giving some preliminaries in Sec. 3.1 and 3.2, we introduce our new attack in Sec. 3.3. In Sec. 3.4 we then discuss briefly the applicability of the attacks to S/MIME-encrypted emails and show up the possibility of CBC-downgrade attacks against AES Key Wrap in CMS in Sec. 3.5.

3.1 Hybrid encryption in CMS

Like OpenPGP, CMS employs a hybrid encryption model that realizes the content-encryption with two message parts. First, a symmetric content-encryption key (CEK) is delivered by means of a *RecipientInfo* structure. The CEK is used to encrypt the content that follows as the second message part. This makes it possible to create an encrypted CMS message using the same CEK to multiple recipients, by supplying one RecipientInfo for each of them. CMS defines a number of RecipientInfo types [Hou09a] that transfer the CEK using public-key and secret-key methods.

3.2 AEAD modes in CMS

RFC 5084 [Hou07] defines AES-CCM [Mor04] and AES-GCM [Mor07] as AEAD modes for CMS. Both of these AEAD modes employ the CTR mode for the data encryption with a publicly known nonce used as the basis for the derivation of the counter block, the encryption of which yields the key stream S:

$$S = (S_0, S_1, \ldots S_n) = (\text{AES-E}_k(\text{CTR}_0), \text{AES-E}_k(\text{CTR}_1), \ldots, \text{AES-E}_k(\text{CTR}_n)),$$

where CTR_i is the counter block derived from the block index i and the AEAD nonce value. The details of the derivation of the counter blocks CTR_i for AES-CCM and AES-GCM in CMS are irrelevant to our attacks and thus are omitted here. Given the key stream S, the counter mode encryption is performed as $C = (C_0, C_1, \ldots, C_n)$ with $C_i = P_i \oplus S_i$ and P_i being the i-th plaintext block. A non-full final block P_n is handled by truncating S_n and C_n to the same length as P_n.

3.3 A cross-mode attack against CMS AEAD modes

In the following, we describe an attack in which the attacker generates a CBC ciphertext that, when decrypted by a CBC decryption oracle, allows the decryption of a low entropy block in an AES-CCM or AES-GCM ciphertext. It is in principle analogous to the previously published attack against XML encryption [JPS13], but improves upon it regarding the number of necessary oracle queries: while the previous work only decrypts a single block guess in one oracle query, in our attack the decryption of all block guesses are performed simultaneously in one oracle query. This improvement is, however, only effective for a decryption oracle which returns the complete plaintext to the attacker. When a padding oracle or format oracle is exploited, the parallelisation of block decryption queries will naturally not work, since such an oracle typically only allows at most the determination of a single byte per oracle query.

The attack is executed as follows:

- Generate the set of h guesses $\{U_i\}$ with $i \in \{1, \ldots, h\}$ for the target plaintext block P_t at position t in the original CMS AEAD ciphertext. The corresponding ciphertext block in the target CTR ciphertext is labelled C_t.
- Compute the corresponding set of guessed key stream blocks $\{Q_i = U_i \oplus C_t | i = 1, \ldots, h\}$.
- Create the CBC ciphertext as the oracle input:
  - Choose CBC-IV as Q_0 arbitrarily.
  - Form the CBC ciphertext as the sequence of the guess blocks $(Q_i | i = 1, \ldots h)$.
- Then the CBC decryption oracle will compute the sequence of plaintext blocks: $(P_i = \text{AES-D}_k(Q_i) \oplus Q_{i-1} | i = 1, \ldots h)$.
- The attacker receives the plaintext (P_i) and computes the block sequence $(X_i = P_i \oplus Q_{i-1} | i = 1, \ldots h)$. This sequence represents the block-wise decryption of (Q_i).
- Let H_t be the counter block at position t in the AEAD ciphertext, i.e., for the correct guess of the target key stream block Q_v at index $v \in \{1, \ldots, h\}$ we have $Q_v = \text{AES-E}_k(H_t)$.
- Note that for the correct guess we have $X_v = \text{AES-D}_k(Q_v) \oplus Q_{v-1} \oplus Q_{v-1} = \text{AES-D}_k(Q_v) = H_t$.
- Thus, if the attacker finds a block $X_v = H_t$ in $(X_i | i = 1, \ldots h)$ then
  - the guess $Q_v = U_v \oplus C_t$ for the key stream block is correct
  - and thus the corresponding guess U_v for the plaintext block P_t is correct.

3.4 Applicability to S/MIME-encrypted emails

S/MIME [STR19] builds upon CMS, making it directly vulnerable to attacks on CMS. In the case of email, in principle, a decryption oracle can easily be realized by a human user viewing the "garbled" mail in his mail user agent (MUA) and replying with the quoted original message to the attacker, as described in Sec. 1. However, as already pointed out in that section and also in Sec. 2.8, the ability to display such a "garbled" message implies other more severe vulnerabilities regarding the manipulation of legacy-mode-encrypted messages. Accordingly, we found that for instance the widely used MUA Thunderbird refuses to display CBC-encrypted emails where a single byte was manipulated (leading to a full random binary plaintext block according to the properties of CBC).

However, as the work by Ising et al. [IPK+23] shows, there might exist more sophisticated attack vectors exploiting automated and observable message processing by MUAs.

3.5 Downgrade Attacks against AES Key Wrap in CMS

As pointed out already by Jager et al. [JPS13], the availability of a CBC decryption oracle allows to decrypt keys encrypted with AES Key Wrap according to RFC 3394 [SH02a]. This is due to the fact that the Key Wrap decryption algorithm employs the block cipher decryption operation under the respective key encryption key (KEK) as the only secret operation. Accordingly, an attack can be mounted by running the Key Wrap and querying the CBC decryption oracle analogously to the attack given in Sec. 3.3 whenever the block cipher block decryption is required.

As an example for a potentially vulnerable setup, RFC 6160 specifies the cryptographic protection of a Symmetric Key Package [Tur11] in CMS. It prescribes that AES Key Wrap with Padding [Hou09b] be used as the content-encryption algorithm. Accordingly, attacks downgrading AES Key Wrap to CBC encryption are possible. The attacker simply starts the execution of the "Extended Key Unwrapping Process" defined in [Hou09b], and whenever he needs the block decryption operation, he queries the CFB decryption oracle by submitting the same CMS message, exchanging only the encrypted content to regular CBC encrypted content and placing the block to be decrypted as the ciphertext. Figure 2 depicts the attack on the basis of the CMS data structures. Note that CMS generally specifies the typical redundant padding scheme to be applied when the content-encryption algorithm operates on the granularity of blocks, where each unused byte in the final block has the value of the number of unused bytes [Hou09a, Sec. 6.3]. Thus a generic padding oracle is indeed conceivable in CMS implementations.

Such an attack is always possible when an AES Key Wrap algorithm is used as the content-encryption algorithm and a block decryption oracle for the same content-encryption key is available, here given through a CBC decryption oracle. The vulnerability is thus manifest in CMS independently of the specific application context of the Symmetric Key Package content. It also equally applies to the original AES Key Wrap without padding [SH02b, Mor12].

4 Responsible disclosure

Prior to the publication of our results, we informed both the GnuPG and the RNP project as the only implementations of LibrePGP OCB Packets known to us. The maintainer of the GnuPG project declared that GnuPG was not vulnerable to the described attacks without setting the `ignore-mdc-error` configuration property to allow for SED decryption. However, as described in Sec. 2.2, due to the specific way GnuPG behaves without this configuration property being set when decrypting SED packets, our attacks may still be seen as applicable to the default configuration of GnuPG (given the non-zero exit code from

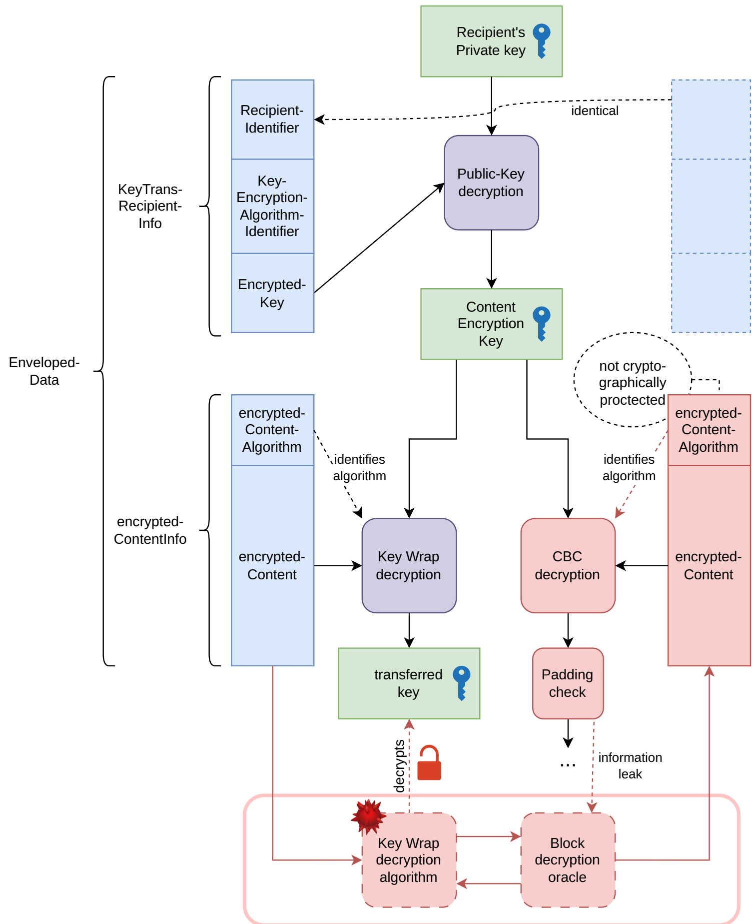

Figure 2: Depiction of the attack against CMS EnvelopedData for the protection of Symmetric Key Package Content Types [Tur11]. In the upper half, the asymmetric decryption of the RecipientInfo structure is shown, which is the same for the decryption of the regular ciphertext as well as during the attack. In the lower half, on the left hand side the processing of the original symmetric ciphertext is shown, where the symmetric decryption is performed using the AES Key Wrap algorithm. At the bottom the attack algorithm is depicted. It is given by simply executing the AES Key Wrap decryption algorithm by using the CBC decryption oracle whenever the AES block decryption operation under the content-encryption key (CEK) must be performed. On the right hand side, the processing of the corresponding attacker-modified symmetric ciphertext is shown. Since the encryptedContentAlgorithm field, which signifies the symmetric decryption algorithm to use, is not cryptographically protected, the attacker can cause the query ciphertext to be decrypted under the same CEK as for the original ciphertext. The encrypted content is CBC-decrypted, and the attacker may learn the decryption result from a CBC padding oracle.

GnuPG does not prevent the decrypted content from being displayed). Moreover, LibrePGP is a general standard and other implementations may behave differently regarding whether and how SED decryption is supported.

We disclosed the vulnerability of the AES-based AEAD schemes in CMS on the LAMPS mailing list[5] as it might have affected the ongoing standardization of the KEMRecipientInfo. This working group inside the IETF is responsible for amending the CMS standards framework. As a result, a new draft to address the vulnerability was adopted by the LAMPS working group, as further explained in Sec. 5.1.

The vulnerability of AES Key Wrap in CMS was not separately disclosed by us. We did not see any necessity for this, since, as explained in Sec. 5, the countermeasure proposed for the AES-based AEAD schemes equally prevents the attack on AES Key Wrap, at least the straightforward one described in Sec. 3.5.

5 Countermeasures

We discuss appropriate countermeasures to thwart the presented attacks against AEAD in LibrePGP and CMS.

5.1 Employment of a key derivation

The appropriate countermeasure to thwart AEAD downgrade attacks is to apply a key derivation to the session key to ensure that each symmetric encryption mode uses a different content-encryption key. The underlying paradigm is known as *key separation*. Such a measure has been realized in the crypto-refresh, which is in the process of being published as an RFC as the official successor of the currently effective OpenPGP standard [CDF+07]. One important feature of the key derivation function is that it may not be built from the block cipher encryption under the input key, as in that case the key derivation itself might be subject to downgrade attacks.

According to the feedback that we received from the GnuPG maintainer, a corresponding countermeasure in the LibrePGP specification is not foreseen.

For CMS, as the consequence of the revelation of our attack, a draft[6] was created and adopted by the LAMPS working group. This draft specifies an optional key derivation using HKDF [KE] to be applied to the key encrypted in the RecipientInfo in order to derive the content-encryption key. The content-encryption algorithm ID enters the key derivation as the additional *info* input parameter to the HKDF and thus makes the derived key dependent on the content-encryption algorithm.

This proposed countermeasure in CMS effectively prevents both the attacks against the AES-based AEAD schemes as well as against AES Key Wrap as a content-encryption algorithm. However, besides functioning as a content-encryption algorithm, there are other potential uses of AES Key Wrap in CMS which are not affected by this countermeasure, such as in the KEKRecipientInfo. In the KEKRecipientInfo, a previously exchanged symmetric key encryption key (KEK) is identified as to be used for the decryption of the content-encryption key. If a receiving implementation accepted AES-CBC as a key-encryption algorithm specified in the KEKRecipientInfo, then it would still be vulnerable. A cursory test of the OpenSSL command line CMS tool showed that it does not perform decryption based on a KEKRecipientInfo that specifies AES-CBC as the content decryption algorithm.[7] Furthermore, there is the remote possibility that the KEK used in the KEKRecipientInfo

[5] https://mailarchive.ietf.org/arch/msg/spasm/TTtMQlcpGRq_bThfJl-HnqqGLGI/

[6] https://datatracker.ietf.org/doc/draft-ietf-lamps-cms-cek-hkdf-sha256

[7] In the master branch of https://github.com/openssl/openssl/blob/master/crypto/cms/cms_env.c, viewed on 2024-05-07, the routine responsible for the decryption of the KEK in a KEKRecipientInfo, `cms_RecipientInfo_kekri_decrypt`, always assumes the usage of a Key Wrap variant as the key-encryption algorithm.

can be used for content decryption in a different context. This could be the case in a custom key management mechanism, as can be implemented in CMS via the generic OtherRecipientInfo structure. Then again, it is conceivable that in such a custom setup the previously exchanged KEK can be used for content decryption and thus a CBC decryption oracle could reveal the content-encryption key. Accordingly, though we could not identify a straightforward attack route against such uses of AES Key Wrap, insecure uses of it that are vulnerable to CBC downgrade attacks cannot be entirely precluded in the context of CMS.

Besides providing key separation between the different encryption modes, it is advisable to also ensure key separation between different ciphers. Otherwise, should it be the case that one cipher supported by the protocol can be used to leak the key through the decrypted plaintext, then a cross-cipher decryption oracle attack might be used to reveal the key of a strong cipher. For the proposed solution in the LAMPS draft, key separation between ciphers is fulfilled, since the algorithm identifiers in CMS are specific for the encryption mode as well as for the cipher [Hou07].

5.2 Disabling SED decryption in LibrePGP

A straightforward implementation countermeasure effective for the current state of LibrePGP is to not support the deprecated SED decryption. However, this countermeasure has the drawback that it can only be enforced by the receiving client and for the sender of an OCB Packet it would remain unknown if the receiving client is vulnerable or not. Furthermore, it would make it impossible for users to decrypt stored legacy data that was still encrypted with SED. The current LibrePGP specification only discourages but does not forbid the support of SED decryption and mandates precaution measures for it: "This [i.e., SED] packet is obsolete. An implementation MUST NOT create this packet. An implementation MAY process such a packet but it MUST return a clear diagnostic that a non-integrity protected packet has been processed. The implementation SHOULD also return an error in this case and stop processing."[8] This prescription is of course ambiguous, since "MAY process" and "SHOULD [...] stop processing" are obviously conflicting choices. Accordingly, it may be said that GnuPG realizes this countermeasure, but nevertheless the attack is still possible when a calling application or a human user of GnuPG's command line interface accepts the decryption result despite the non-zero exit code of GnuPG, as explained in Sec. 2.2.

The analogous countermeasure for CMS is completely infeasible, since contrary to the SED encryption in OpenPGP, CBC-encryption is still widely used in CMS for instance in S/MIME-encrypted emails.

6 Conclusion

In this work we introduce novel attacks exploiting the same principal design error in LibrePGP and CMS AEAD encryption, as well as certain uses of the AES Key Wrap in CMS, namely the absence of an algorithm-dependent key derivation for the content-encryption key in order to ensure key separation between the different encryption modes. The attacks against CMS are analogous to those previously published for XML encryption [JPS13]. While the attacks are shown to be realistic under the assumption of a fully-plaintext-revealing legacy-cipher-mode decryption oracle, we cannot point out any vulnerable real world application. On the contrary, we come to the conclusion that the presumably most widely deployed application layer protocol for these two cryptographic protocols, namely email encryption, will be affected by our attacks to a lesser degree than by previous

[8] https://www.ietf.org/archive/id/draft-koch-librepgp-00.html#name-symmetrically-encrypted-dat

attacks against legacy-mode-encrypted emails like for instance Efail [PDM+18]. However, experience has shown that the reliance on the unexploitability of principal cryptographic vulnerabilities due to the specifics of typical protocol and application implementations is often unfounded. This is apparent for instance regarding the recurrence of padding oracle attacks against CBC [Bod14, AFP13] and PKCS#1 v1.5 encryption [Ble98, BSY] in TLS. Works like [IPK+23] show that through unexpected features of applications, observable decryption oracles may exist even where the nature of the protocol does not suggest their presence. Consequently, we hold that our results for LibrePGP and CMS should be taken seriously and appropriate countermeasures be integrated into these protocols.

References

[AFP13] Nadhem J. Al Fardan and Kenneth G. Paterson. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, 2013. doi:10.1109/SP.2013.42.

[Ble98] Daniel Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1. In *CRYPTO*, pages 1–12. Springer-Verlag, 1998.

[Bod14] Bodo Möller (Google Security Team). This POODLE bites: exploiting the SSL 3.0 fallback, 2014. https://googleonlinesecurity.blogspot.co.uk/2014/10/this-poodle-bites-exploiting-ssl-30.html.

[BSY] Hanno Böck, Juraj Somorovsky, and Craig Young. Return Of Bleichenbacher's Oracle Threat (ROBOT). SEC'18: Proceedings of the 27th USENIX Conference on Security Symposium, August 2018, Pages 817–832, https://eprint.iacr.org/2017/1189.pdf.

[CDF+07] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. RFC 4880 – OpenPGP Message Format, November 2007. https://datatracker.ietf.org/doc/html/rfc4880.

[Cry] Cryptography StackExchange Post. Why is plain-hash-then-encrypt not a secure MAC? https://crypto.stackexchange.com/questions/16428/why-is-plain-hash-then-encrypt-not-a-secure-mac/16431#16431.

[DPMN10] A. Doherty, M. Pei, S. Machani, and M. Nystrom. RFC 6063 — Dynamic Symmetric Key Provisioning Protocol (DSKPP), December 2010. https://datatracker.ietf.org/doc/html/rfc6063.

[DR] T. Duong and J. Rizzo. Here come the ⊕ Ninjas. Unpublished manuscript, 2011, https://nerdoholic.org/uploads/dergln/beast_part2/ssl_jun21.pdf.

[gnu] gnupg.org. gnupg/NEWS. https://dev.gnupg.org/source/gnupg/browse/master/NEWS.

[Hou07] R. Housley. RFC 5084 – Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS), November 2007. https://datatracker.ietf.org/doc/html/rfc5084.

[Hou09a] R. Housley. RFC 5652 – Cryptographic Message Syntax (CMS), 2009. https://tools.ietf.org/html/rfc5652.

[Hou09b] Housley, R. and Dworkin, M. RFC 5649 — Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm, August 2009. https://datatracker.ietf.org/doc/html/rfc5649.

[IPK+23] Fabian Ising, Damian Poddebniak, Tobias Kappert, Christoph Saatjohann, and Sebastian Schinzel. Content-Type: multipart/oracle - tapping into format oracles in email End-to-End encryption. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4175–4192, Anaheim, CA, August 2023. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/ising.

[JKS02] Kahil Jallad, Jonathan Katz, and Bruce Schneier. Implementation of Chosen-Ciphertext Attacks against PGP and GnuPG. In Agnes Hui Chan and Virgil Gligor, editors, *Information Security*, pages 90–101, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[JPS13] Tibor Jager, Kenneth G. Paterson, and Juraj Somorovsky. One bad apple: Backwards compatibility attacks on state-of-the-art cryptography. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*, 2013. https://www.ndss-symposium.org/ndss2013/ndss-2013-programme/one-bad-apple-backwards-compatibility-attacks-state-art-cryptography/.

[KE] H. Krawczyk and P. Eronen. RFC 5869 — HMAC-based Extract-and-Expand Key Derivation Function (HKDF).

[Koc24] Werner Koch. Private communication, 2024.

[KR14] T. Krovetz and P. Rogaway. RFC 7253 – The OCB Authenticated-Encryption Algorithm, 2014. https://datatracker.ietf.org/doc/html/rfc7253.

[KS00] Jonathan Katz and Bruce Schneier. A Chosen Ciphertext Attack Against Several E-Mail Encryption Protocols. In *9th USENIX Security Symposium (USENIX Security 00)*, Denver, CO, August 2000. USENIX Association. URL: https://www.usenix.org/conference/9th-usenix-security-symposium/chosen-ciphertext-attack-against-several-e-mail-encryption.

[KT23] W. Koch and R. H. Tse. LibrePGP Message Format, November 2023. https://www.ietf.org/archive/id/draft-koch-librepgp-00.html#section-5.14.

[Mag15] J. Magazinius. Openpgp seip downgrade attack, October 2015. http://www.metzdowd.com/pipermail/cryptography/2015-October/026685.html.

[MBP+19] Jens Müller, Marcus Brinkmann, Damian Poddebniak, Sebastian Schinzel, and Jörg Schwenk. Re: What's Up Johnny? – Covert Content Attacks on Email End-to-End Encryption, 2019. arXiv:1904.07550.

[Mor04] Morris Dworkin. NIST Special Publication 800-38C – Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality , 2004.

[Mor07] Morris Dworkin. NIST Special Publication 800-38D – Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC , 2007.

[Mor12] Morris Dworkin. NIST Special Publication 800-38F – Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping, December 2012. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38F.pdf.

[MRLG15a] Florian Maury, Jean-René Reinhard, Olivier Levillain, and Henri Gilbert. Format Oracles on OpenPGP. In Kaisa Nyberg, editor, *Topics in Cryptology — CT-RSA 2015*, pages 220–236, Cham, 2015. Springer International Publishing. https://www.ssi.gouv.fr/uploads/2015/05/format-Oracles-on-OpenPGP.pdf.

[MRLG15b] Florian Maury, Jean-René Reinhard, Olivier Levillain, and Henri Gilbert. Format Oracles on OpenPGP. In Kaisa Nyberg, editor, *Topics in Cryptology — CT-RSA 2015*, pages 220–236, Cham, 2015. Springer International Publishing.

[MZ06] Serge Mister and Robert Zuccherato. An Attack on CFB Mode Encryption as Used by OpenPGP. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography*, pages 82–94, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[PDM+18] Damian Poddebniak, Christian Dresen, Jens Müller, Fabian Ising, Sebastian Schinzel, Simon Friedberger, Juraj Somorovsky, and Jörg Schwenk. Efail: Breaking s/mime and openpgp email encryption using exfiltration channels. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, page 549–566, USA, 2018. USENIX Association.

[Per02] T. Perrin. Openpgp security analysis, September 2002. https://www.ietf.org/mail-archive/web/openpgp/current/msg02909.html.

[SH02a] J. Schaad and R. Housley. Advanced Encryption Standard (AES) Key Wrap Algorithm, September 2002. https://datatracker.ietf.org/doc/html/rfc3394.

[SH02b] J. Schaad and R. Housley. RFC 3394 — Advanced Encryption Standard (AES) Key Wrap Algorithm, September 2002. https://datatracker.ietf.org/doc/html/rfc3394.

[STR19] J. Schaad, S. Turner, and B. Ramsdell. RFC 8551 – Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification , 2019. https://tools.ietf.org/html/rfc8551.

[Tur11] S. Turner. RFC 6160 — Algorithms for Cryptographic Message Syntax (CMS) Protection of Symmetric Key Package Content Types, April 2011. https://datatracker.ietf.org/doc/html/rfc6160.

[Vau02] S. Vaudenay. Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS. In *Advances in Cryptology – EUROCRYPT 2002*, pages 543–545. Springer-Verlag, 2002.

[Wag] David Wagner. Email Subject: Re: BIG question about using and storing IV's. http://www.cs.berkeley.edu/~daw/my-posts/mdc-broken.

[WHWY] Paul Wouters, Daniel Huigens, Justus Winter, and Niibe Yutaka. OpenPGP. https://datatracker.ietf.org/doc/draft-ietf-openpgp-crypto-refresh/13/.

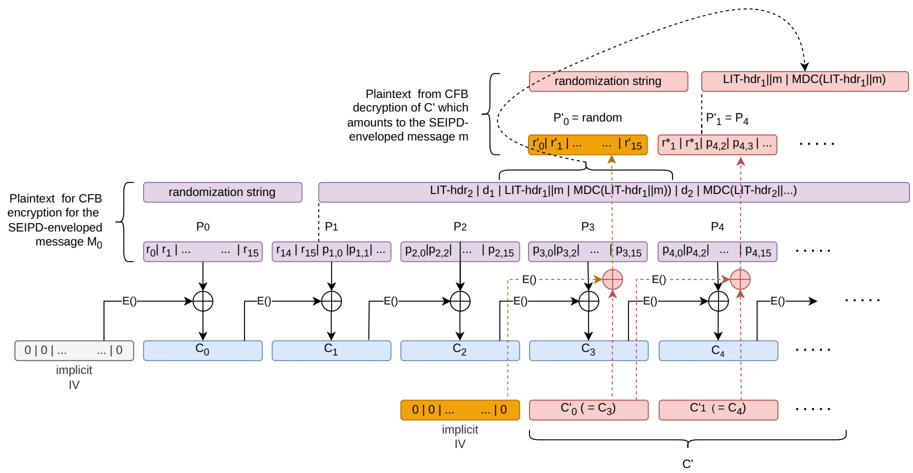

Figure 3: Depiction of an adaptively chosen ciphertext attack against SEIPD encryption assuming a block size of 16 bytes. Here, $E()$ indicates block encryption under the given block cipher key and $\oplus$ denotes XOR. The violet plaintext consisting of the bytes $p_{1,0}, p_{1,1}, \ldots$ is the message M_0 prepared by the attacker enveloped in the outer LIT packet with the header LIT-hdr$_2$. Above the blocks $P_0, P_1, \ldots$, the semantic interpretation of the decrypted plaintext is depicted in the same color. The light blue boxes show the resulting ciphertext C. The red and orange boxes represent the manipulated ciphertext C' and the plaintext resulting from its decryption. Here the red boxes represent repositioned ciphertext and plaintext blocks and the orange ones entirely different blocks. The red and orange arrows and XOR operators indicate the data flows and operations during decryption of C'.

A Insecurity of OpenPGP SEIPD packets under adaptively chosen ciphertext attacks

In the CCA2 game, the adversary has to submit two challenge plaintexts M_0 and M_1 of the which the challenger encrypts a random one. The adversary receives the challenge ciphertext C, which is thus the encryption of either M_0 or M_1 and he wins if, after further decryption queries for arbitrary ciphertexts excluding the challenge ciphertext, he can determine which of the two messages was encrypted to yield C. In order to break the MDC scheme he submits two challenge plaintexts where M_0 is built as $d_1 \,||\,$ LIT-hdr$_1 \,||\,$ MDC(LIT-hdr$_1 \,||\, m) \,||\, d_2$, i.e., which contains a sequence of arbitrary data d_1, then a LIT packet enveloping the message m ("LIT-hdr" denoting the header of the LIT packet) and the MDC packet associated with the preceding LIT packet, and then arbitrary data d_2. Before the CFB-encryption, the message M_0 will be enveloped as LIT-hdr$_2 \,||\, d_1 \,||\,$ LIT-hdr$_1 \,||\, m \,||\,$ MDC(LIT-hdr$_1 \,||\, m) \,||\, d_2 \,||\,$ MDC(LIT-hdr$_2 \,||\, \ldots$). M_1 is just a random plaintext of the same length. After receiving the challenge ciphertext C, which is either the encryption of M_0 or M_1, the adversary can determine which of the two messages was encrypted by stripping-off from C the packet header of the outer LIT and d_1 at the beginning as well as d_2 at the end and feeding this modified ciphertext C' to the decryption oracle. C' will only be a valid ciphertext if C was the encryption of M_0, in which case it will decrypt to the plaintext message m. He thus wins the CCA2 game.

To apply this attack to OpenPGP SEIPD packets, M_0 needs to be chosen with proper alignment of the inner LIT packet. Fig. 3 depicts the required alignments.

Table 4: Buildup of the additional data for the GnuPG AEAD chunks. The individual fields' encodings are omitted here since they are irrelevant to the described attack.

field	**size in bytes**	**in chunks ...**
Packet Tag in new format encoding	1	all
version number	1	all
cipher algorithm	1	all
encryption mode	1	all
chunk size	1	all
big-endian chunk index	8	all
msg. total bytes length	8	only in final

B Buildup of the additional data in a LibrePGP OCB Packet OCB chunk

The buildup of the additional data of each chunk in a LibrePGP OCB Packet is shown in Table 4.