Comparison of RFC 9580 and LibrePGP

Version 1.3
July 2, 2025

Johannes Roth and Falko Strenzke
MTG AG
{johannes.roth|falko.strenzke}@mtg.de

Table of changes

Version	Date	Authors	Changes
v1.0	2025-02-05	Johannes Roth, Falko Strenzke	Created intial version
v1.1	2025-03-24	Johannes Roth, Falko Strenzke	Added Executive Summary, extended Section 3.2, worked in comments from BSI received on 2025-03-24
v1.2	2025-06-04	Johannes Roth, Falko Strenzke	Added description of countermeasure for v5/v3 signature reinterpretation attack. Removed one section from Chapter 5. Updated statement in Section 3.4.2.1 about OpenPGP PQC encryption key version restrictions. Corrected Section 4.3.2 about nonce reuse probabilities. Updates to UTF-8-reencoding and unprotected LIT fields.
1.3	2025-07-02	Johannes Roth, Falko Strenzke	Editorial and formatting changes.

Executive Summary

This document compares OpenPGP as per RFC 9580 with LibrePGP version 03, the most recent version at the time of writing. The differences of both specifications to their common ancestor, RFC 4880, with the inclusion of the additional specifications given in RFC 6637 and RFC 5581, are also documented. We outline differences in data formats, cryptographic operations, and the current state of Post-Quantum-Cryptography algorithms in LibrePGP and OpenPGP, as well as give a brief overview of the history of those documents.

The document starts with an introductory part followed by a technical comparison of the specifications, including data formats and the interplay of different packet versions. For this, all three specifications have been analyzed in-depth. The overview of the notable differences can be found in tabular format in Section 3.5. We further provide a unified and unambiguous representation for the different transferable public key formats (also called certificates).

The second major part of the document deals with cryptographic mechanisms and examines the security of individual components and the vulnerability to certain attacks for OpenPGP and LibrePGP.

The most notable differences that we found between LibrePGP and OpenPGP, some of which have relevance to the protocol security, are the following:

- The current draft for the introduction of PQC in the OpenPGP working group (WG) and the LibrePGP specification exhibit numerous technical differences regarding the realization of PQC. Besides each being based on different packet versions, they differ in the set of defined algorithms: While LibrePGP currently only defines PQC encryption with ML-KEM in composite combination with ECDH, the WG draft also introduces ML-DSA and SLH-DSA as signature algorithms. They furthermore differ in the manner in which parametrization occurs: while RFC 9580 fixes algorithms and security parameters for each code point, LibrePGP allows generic parametrization under a single algorithm code point. The PQC integration is addressed in Section 3.4.2.

- LibrePGP does not employ proper key separation for the newly defined AEAD schemes. This constitutes a vulnerability on the protocol level that leads to a principle vulnerability to downgrade attacks if for a cipher with AEAD mode, also the legacy mode SED-mode is available for that cipher. While concrete real-world attacks based on this principal vulnerability are not known and do not seem a realistic threat, this cannot be ruled out completely. Formally, LibrePGP AEAD-encrypted messages do not achieve IND-CCA2 security. See Section 4.3.1.

- In Section 4.4 we describe that for LibrePGP existential forgeries can be crafted when reinterpreting v5 signatures as v3 document signatures. A straightforward and backwards-compatible countermeasure that fully mitigates the vulnerability is suggested.

- In contrast to the transparent IETF processes that the OpenPGP specifications were developed in, the LibrePGP specification is developed in an intransparent process. While contributions from the public are possible and corrections are welcome, it cannot be assumed that the specification will be subjected to the same scrutiny as for standardization as an RFC since there is no apparent quality assurance process that is comparable to that which an IETF RFC undergoes before publication. This subject is covered in Section 3.2.

Contents

List of Figures 6

1 Introduction 7

2 Basics of the OpenPGP Protocol 9
- 2.1 OpenPGP Certificates . . . 9
- 2.2 OpenPGP Messages . . . 12
  - 2.2.1 OpenPGP and LibrePGP Hybrid Encryption . . . 12
  - 2.2.2 Shortcomings of the v1 SEIPD Packet . . . 12

3 Overview of Differences 15
- 3.1 Existing Analysis . . . 15
- 3.2 Notes on the Maintenance of the Specifications Subject to Comparison . . . 15
- 3.3 Structure of Certificates . . . 16
  - 3.3.1 Notation for Certificate Structure . . . 16
  - 3.3.2 v4 Certificates . . . 17
  - 3.3.3 v6 and v5 Certificates . . . 17
- 3.4 Overview of Features of v4, v5, and v6 Keys and Planned Integration of PQC . . . 19
  - 3.4.1 Allowed Packet Pairings in RFC 9580 and LibrePGP . . . 19
  - 3.4.2 Planned Integration of PQC in OpenPGP and LibrePGP . . . 19
    - 3.4.2.1 PQC in OpenPGP . . . 19
    - 3.4.2.2 PQC in LibrePGP . . . 22
- 3.5 Tabular Overview of the Differences between RFC 4880, RFC 9580, and LibrePGP . . . 22

4 Differences of RFC 9580 and LibrePGP in Cryptographic Mechanisms 30
- 4.1 Signature Salt . . . 30
  - 4.1.1 Assessment . . . 30
- 4.2 Signing Literal Data Packets . . . 30
  - 4.2.1 Assessment . . . 30
- 4.3 AEAD Packets . . . 31
  - 4.3.1 Legacy Cipher Downgrade Attacks . . . 31
  - 4.3.2 Nonce-Reuse Bounds under Session Key Reuse . . . 31
    - 4.3.2.1 Nonce-Reuse in RFC 9580 . . . 32
    - 4.3.2.2 Nonce-Reuse in LibrePGP . . . 32
- 4.4 Possible Reinterpretation of v5 Signatures as v3 Signatures . . . 34
- 4.5 Analysis of Conceivable Signature Ambiguities in v4 and v6 Signatures . . . 36
  - 4.5.1 Analysis for v6 Signatures . . . 36
  - 4.5.2 Analysis for v4 Signatures . . . 39
  - 4.5.3 Analysis for LibrePGP's v5 Signatures . . . 39
- 4.6 UTF-8 Recoding in RFC 9580 . . . 39

5 Analogous Security Features of RFC 9580 and LibrePGP **41**
5.1 Key Overwrite Attacks 41

6 Bibliography **42**

A Figures to the Impossible v5 Document-Key-Signature Aliasing **43**

List of Figures

1.1 Overview of the history of OpenPGP standards. 8

2.1 Example of a v6 certificate structure 11
2.2 OpenPGP sample messages 13

3.1 Differences in the v4 certificate format specification of the different standards. 18
3.2 Differences between the v6 and v5 certificate format specification. 18
3.3 Overview of the signature variants for OpenPGP 20
3.4 Overview of the signature variants for LibrePGP 20
3.5 Overview of the encryption variants for OpenPGP with PQC 21
3.6 Overview of the encryption variants for LibrePGP with PQC 21

4.1 OpenPGP AEAD nonce derivation. 32
4.2 LibrePGP AEAD nonce derivation. 33
4.3 Hypothetical v6 document signature and signature over a key aliasing attack 37
4.4 Hypothetical v6 document signature and signature over a key reverse aliasing attack . . 38

A.1 Hypothetical v5 document signature and signature over a key aliasing attack 44
A.2 Hypothetical v5 document signature and signature over a key reverse aliasing attack . . 45

Chapter 1

Introduction

This document gives an overview of the difference between three OpenPGP-related data formats:

- RFC 4880 [RFC4880], the OpenPGP standard that was the current standard from 2007 until 2024,
- RFC 9580 [RFC9580], the updated standard that was being worked on in the recent years,
- LibrePGP [LPGP01], a recent fork of the OpenPGP standard.

Figure 1.1 gives an overview of the evolution of the OpenPGP and related specifications. RFC 2440, 4880, and 9580 are the successive official OpenPGP standards. LibrePGP is a fork of the standard that was created from a working draft to update RFC 4880 that was called "4880bis".

It shall be noted that LibrePGP version 03, that is subject to the analysis in this document, can at any time be updated to a newer LibrePGP version. The statements made in this document are therefore potentially limited with respect to the time frame of their applicability to the most recent LibrePGP specification.

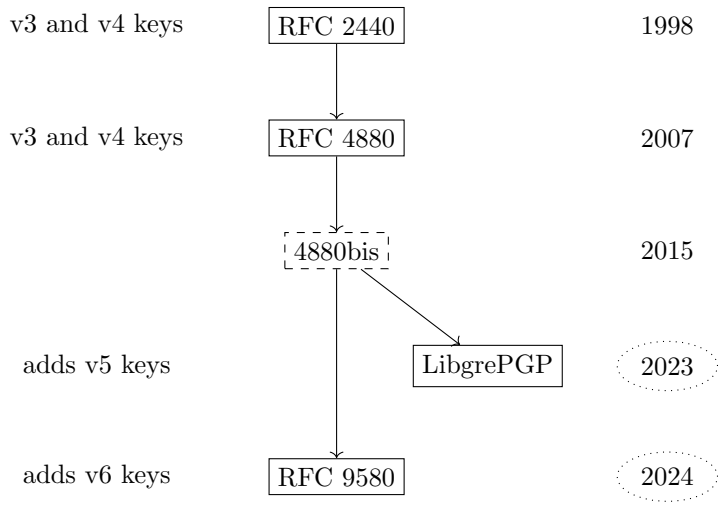

Figure 1.1: Overview of the history of OpenPGP standards. To the left of the graph the key format versions associated with each specification are indicated. Other changes and package versions are omitted for simplicity and clarity of the overview.

Chapter 2

Basics of the OpenPGP Protocol

2.1 OpenPGP Certificates

OpenPGP Certificates (also referred to as "Transferable Public Keys") are data structures that contain OpenPGP public keys and associated information. Their purpose is the distribution and storage of a user's own keys and those of parties they want to communicate with. A certificate contains a primary key as well as any number of keys that are subordinate to the primary key, referred to as "subkeys". The primary key, typically in conjunction with an associated User ID, is the key that other users place trust in. A certificate in general contains various signatures made by its own keys that ensure the authenticity of information contained in it. Signatures are for instance used to authenticate the subkeys contained in the certificate by the primary key.

Figure 2.1 shows an example structure of a v6 certificate with the most commonly used packets. Note that Revocation Signatures are only present when the key the refer to has actually been revoked. Please also note that a typical v6 certificate contains further subpackets, which have been omitted here for the sake of simplicity and clarity. A User Attribute Packet contains Attribute Subpackets. One such subpacket that is specified in all three standards compared in this work is the Image Attribute Subpacket. The term "v6 certificate" refers to a certificate with a v6 primary key, the v6 key format being the new key format introduced in the current OpenPGP specification RFC 9580. The certificate is built up by a sequence of OpenPGP packets. The root node "v6 certificate" in the figure is only for the purpose of illustration and does not represent any actual surrounding structure in the data format. The first packet is always the primary key. This key has to have signing capabilities, as it is used to create various signatures in the certificate. It may additionally have the capability for public key encryption, as in the case of RSA.

In order to indicate that a primary key and thus also the complete corresponding certificate is revoked, a revocation signature is included as the next packet. In a v6 certificate, the revocation signature is created by the primary key itself. Revocation signatures may be pre-generated and escrowed to parties which are then able to revoke the respective primary key in the future. Revocation signatures can also be applied to individual subkeys and to certification signatures by placing them directly after the respective subkey or certification signature packet. In case of a revoked subkey in a v6 certificate, the revocation signature had to be made by the primary key. In RFC 4880 and in LibrePGP, besides revocation signatures issued by the primary key itself, also designated revocation keys can be part of the certificate. This key type has been deprecated in v6.

The next packet in a v6 certificate – after the primary key and the potentially existing revocation signature – must be a Direct Key Signature. The purpose of this signature is to bind various attributes to the primary key in its signature subpackets. Then follows the User ID packet. This packet is optional for a v6 certificate. In the case of a certificate used for securing e-mail communication, it contains the key owner's e-mail address.

The subsequent Positive Certification Signature is a created by the primary key and authenticates the User ID packet and optionally stores further information relating to the User ID in the signature's subpackets. An OpenPGP certificate can also contain third party certification signatures. These are signatures by other users confirming the key ownership with respect to the User ID. Based on such third party signatures, other users who trust the signers can also place trust in the key ownership claimed in the User ID packet.

Next follows a subkey packet for a signature key. If a user wants to use the primary key also for the purpose of signing messages, then the inclusion of a signature subkey packet is not necessary. After the potential revocation signature packet that would revoke the subkey, a Subkey Binding Signature follows, which authenticates the subkey by the primary key. A valid Subkey Binding Signature is required for a subkey to be usable for encryption or verification. A valid Subkey Binding Signature of signature-capable key must always contain an Embedded Signature Packet Signature Subpacket in its unhashed subpackets which contains a Primary Key Binding Signature. Each signature packet can contain subpackets in its unhashed subpackets area or in its hashed subpackets area. Only the latter is included in the signed data when computing the signature packet's signature value. A Primary Key Binding Signature authenticates the primary key by the signature subkey. This prevents an attack where a user integrates a foreign signature subkey into his certificate and can thus claim to have made a signature that in fact was made by another user.

Next in the figure follows an encryption subkey, which is again followed by a Subkey Binding Signature (after a possible Revocation Signature).

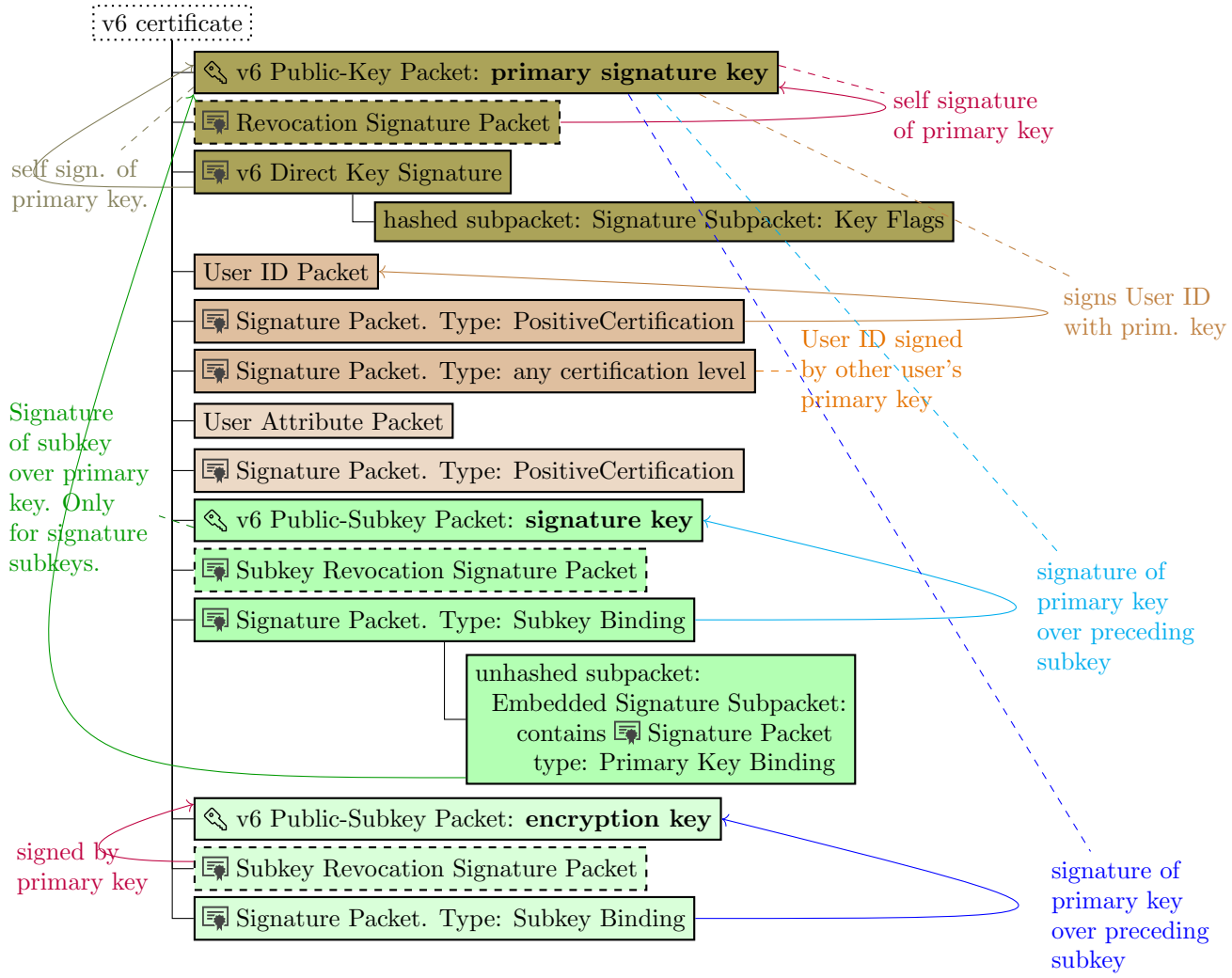

Figure 2.1: Example of a v6 certificate structure.

2.2 OpenPGP Messages

In this section we describe the most important features of OpenPGP messages. The upper part of Figure 2.2 shows two variants of OpenPGP signed messages. In either case multiple signatures can be applied to the message. An example of an OpenPGP encrypted message is is shown in the lower part of the Figure. It shows how the encryption of a message happens based on a plaintext given by the second variant of a signed message.

When encryption and signing are combined in one message, the message is first signed and then the message and signature(s) are encrypted, as it is stated in Section 2.1 of the current standard [RFC9580] and is the common implementation practice. However, it should be noted that Section 10.3 of that same document, which specifies the message grammar formally, allows also encrypt-then-sign. Thus it is possible that certain implementations will also process encrypted-then-signed messages.

The following subsections explain the technical aspects of message encryption in OpenPGP further.

2.2.1 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 OpenPGP, the following types of symmetrically encrypted data packets exist:

- *Symmetrically Encrypted Data (SED) Packet*, is the legacy symmetric encryption mechanism defined in OpenPGP since RFC 2440. This packet is encrypted using a slightly modified variant of CFB encryption without any integrity protection.
- *Symmetrically Encrypted Integrity Protected Data (SEIPD) Packet*. Two versions of this packet type exist:
  - v1 SEIPD is defined in OpenPGP since RFC 4880. 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.
  - v2 SEIPD is defined in RFC 9580. It uses AEAD encryption where the encryption key is derived from the session key using HKDF. The key derivation is done in a manner that ensures key separation between the AEAD algorithms and the legacy SED and v1 SEIPD encryption modes.
- *OCB Packet*, defined in LibrePGP. OCB packets encrypt data using one of the two AEAD modes OCB or EAX, where the latter is marked as deprecated.

2.2.2 Shortcomings of the v1 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 [Efail] for a comprehensive overview of further aspects to the downgrade attacks

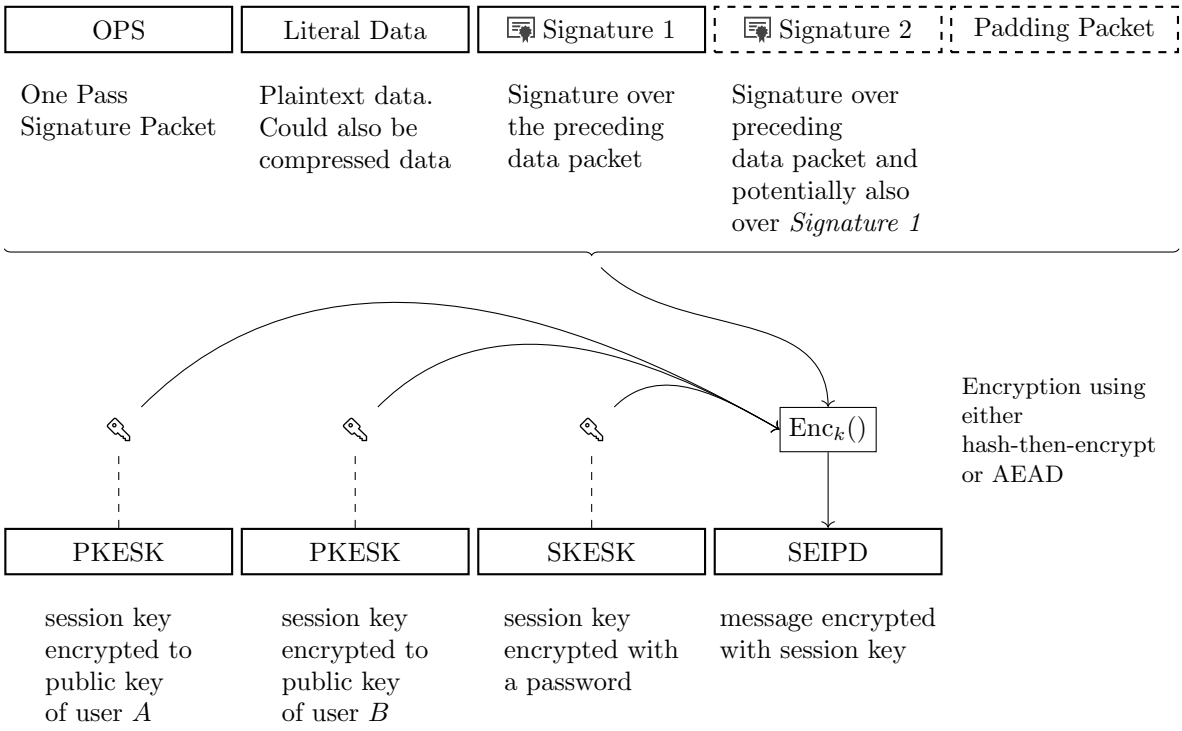

Figure 2.2: Typical examples of Open PGP messages. The signed message starts with a One Pass Signature Packet which specifies the hash algorithm to use for the signature digest computation. Multiple Signature Packets are supported where the appended signatures might be over the same message or the message and the preceding signatures. In messages compliant with RFC 9580, Padding Packets can appear anywhere in a packet sequence. Their purpose is to achieve hiding the length of messages when encryption is applied. Below the signed messages a sample OpenPGP encrypted message is shown, the plaintext of which is the second variant of the signed message. Any combination of PKESK (Public Key Encrypted Session Key) or SKESK (Secret Key Encrypted Session Key) Packets, but at least one, precedes an SEIPD (Symmetrically Encrypted and Integrity Protected Data) Packet.

and their exploitation in a decryption oracle attack. In [SR24], it is shown that in principle, in a considerably more complex attack, though, LibrePGP OCB packets can also be downgraded to SED packets. This fact is listed in the analysis section and specifically handled in Section 4.3.1.

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, CryStackExch]. The weakness is concretely demonstrated in the Appendix of [SR24].

Chapter 3

Overview of Differences

3.1 Existing Analysis

The following existing analysis of the differences between RFC 9580 and LibrePGP was used in the compilation of the differences. Daniel Huigens compiled a list of the changes introduced RFC 9580 and LibrePGP in relation to RFC 4880.[1]. Justus Winter summarized the differences between LibrePGP and RFC 9580 with a focus on security on the openpgp-email list[2]. An e-mail on gnupg-devel addresses some differences regarding symmetrically encrypted packets.[3]

3.2 Notes on the Maintenance of the Specifications Subject to Comparison

The three standards that are subject to comparison are RFC 4880, which was the effective OpenPGP standard until the release of RFC 9580 as the result of the working draft referred to as the "crypto-refresh". Both of these specifications are the result of the IETF process for the creation and release of an RFC. On the other hand, the LibrePGP standard is only technically managed as an IETF draft. Neither is it adopted by any IETF working group, nor is any intention of the authors apparent to reach an adoption. It is also unlikely that this could be achieved, as the OpenPGP working group has already released RFC 9580 as the official successor to RFC 4880 and thus the adoption of a standard version that itself, too, is a successor of that previous RFC, while ignoring the recently published RFC 9580, is not realistic.

In the following, we provide an estimation of the level of quality assurance that is applied to the LibrePGP draft versions as well as a judgement of the quality of the recently published LibrePGP draft versions. When in June 2024 inconsistencies regarding the description of the v4 certificate structure and v6 signatures in the LibrePGP-01 standard were pointed on the LibrePGP-discuss mailing list, as the reaction corresponding corrections were applied to the draft in the GitHub repository and later published in the 02 version.[4] Furthermore, in version 02, previously existing statements about the usage of v6 signatures were removed. The changelog notes "Drop the remains of support for V6 signatures." Version 03 of the LibrePGP draft introduces further breaking changes, i.e., changes that are not backwards-compatible, namely the removal of the Attested Certifications subpacket and the use of the extended v4 fingerprint in a revocation certificate. This can be readily verified in the diff-view

[1] https://mailarchive.ietf.org/arch/msg/openpgp/aqBy97lj2P4DVxTdsOeKZDVdmms/

[2] https://lists.hostpoint.ch/hyperkitty/list/openpgp-email@enigmail.net/thread/4APRJGPZGYFIH2MWRMYSYR7G2MWDZPEA/

[3] https://lists.gnupg.org/pipermail/gnupg-devel/2023-February/035276.html

[4] See https://lists.gnupg.org/pipermail/librepgp-discuss/2024/000077.html and the further messages in the thread.

that the IETF tooling generates.[5]. In the case of the Attested Certifications subpacket, the change log of version 03 presents the removed subpacket retroactively as "experimental" – its experimental nature was not indicated in version 02. These examples show how previous updates to the LibrePGP specification introduce breaking changes, which obviously can happen at any point in time through the publication of a new version. This makes it unpredictable to implementers which version is the currently effective specification and when exactly a new version takes effect.

Based on these observations and the general information gained from observing the LibrePGP mailing list[6], which is the medium where the discussion of the specification is supposed to take place[7], we come to the conclusion that the specification is maintained by and receives contributions from only a small number of contributors. The decision and process for the publication of a new draft version is completely at the discretion of the maintainer of the specification. A body comparable to that of an IETF WG, that is involved in this process and takes on tasks of reviewing and approving a new official version, does apparently not exist.

The quality of the standard, with respect to version-internal and cross-version consistency and transparency of the release system is not comparable with that which can be expected from an IETF RFC. The latter typically receives input from numerous contributors and has to pass the so-called working group last call, a distinct WG-internal review phase before it is declared as final by the WG. After that, it further goes through Area Director (AD) and IESG review phases before it is published.

3.3 Structure of Certificates

The following subsections give a comparison of the specifications for a v4 certificate in all three standards and a comparison of the newly defined v6 and v5 certificate structures. To facilitate the comprehension, we first extend the notation that the standards use for a more concise representation.

3.3.1 Notation for Certificate Structure

In accordance with the notation in the standards, the brackets notation `[ X ]` is used to denote that `X` is an optional element and an ellipsis denotes repetition of the previous element or group.

In extension to the notation in the standards, we furthermore introduce three new elements:

- the parenthesis notation `( X Y Z )` to group the enclosed elements so that for instance a subsequent ellipses will refer to the whole group of elements;
- and the curly brace notation `{ X Y Z }` denotes that the enclosed top-level groups represented by `X`, `Y`, and `Z` can appear in any order;
- and the vertical bar is used to denote alternatives, i.e., `X | Y` results in either `X` or `Y`. In combination with an ellipsis, in each repetition any of the alternatives can be choosen.

The introduction of these additional notations is necessary in order to provide a more refined formalized specification of the certificate structures. The formalized specifications given in the three standards are all not exact. To understand the exact requirements for certificates in all three standards, the text in Section "Transferable Public Keys"[8] in each standard must be carefully studied.

[5] https://author-tools.ietf.org/iddiff?url1=draft-koch-librepgp-02&url2=draft-koch-librepgp-03&difftype=--html

[6] https://lists.gnupg.org/pipermail/librepgp-discuss/

[7] https://lists.gnupg.org/pipermail/librepgp-discuss/2023/000000.html

[8] The section number of "Transferable Public Keys" in RFC 4880 and in LibrePGP is 11.1, in RFC 9580 it is 10.1.

3.3.2 v4 Certificates

Figure 3.1 shows the formalized specification of the v4 certificate structure as given in RFC 4880, RFC 9580, and LibrePGP.

The fact that the RFC 4880 and LibrePGP specifications refer to the revocation signatures for subkeys as "Binding-Signature-Revocation" seems to be simply an error, as the texts in those standards clearly say that the revocation signature is computed over the key itself. This is thus only verbally corrected in the specification of RFC 9580. The same is true for the element labelled as "Primary-Key-Binding-Signature", which has to interpreted as a "Subkey-Binding-Signature".

An important difference between RFC 9580 and LibrePGP is that in the LibrePGP standard, v5 subkeys are allowed as part of certificates with a v4 primary key.

Furthermore, both RFC 9580 and LibrePGP introduce breaking changes to the v4 certificate specification in relation to RFC 4880. This pertains to the presence of the User ID packet as well as the required number of signatures after each User ID or User Attribute packet, as can bee seen in Figure 3.1. RFC 9580 introduces a breaking change by making the User ID optional. LibrePGP introduces a breaking change in requiring also for v4 certificates at least one signature after each User ID packet[9].

The signatures following a User ID or User Attribute Packet are certification signatures which authenticate the User ID. They may be generated by either a certificate's own certification-capable key (typically the primary key) or by a certification-capable key from another certificate.

3.3.3 v6 and v5 Certificates

Figure 3.2 shows a comparison of the v6 certificate specification in RFC 9580 and the v5 certificate specification in LibrePGP.

The most notable changes introduced in RFC 9580 are making the User ID optional and making a Direct Key Signature on the primary key mandatory.

Furthermore, the RFC 9580 makes the User ID Packet optional for v4 certificates.[10] Formally, this is a wire-format breaking change, since v4 certificates created without a User ID Packet may not be accepted by implementations conforming to RFC 4880.

[9] This can be seen in the Transferable Public Key structure in https://www.ietf.org/archive/id/draft-koch-librepgp-01.html#name-transferable-public-keys

[10] LibrePGP-01 contained the same statement, but this was changed to a mandatory User ID Packet in LibrePGP-02.

```
Primary-Key
 [Revocation-Self-Signature]
 [Direct-Key-Signature ...]
 {
  (User-ID [Signature ...])
  [
    (User-ID | User-Attribute)
    [Signature ...]
  ] ...
 }
 [[Subkey [Binding-Signature-Revocation]
  Primary-Key-Binding-Signature] ...]
```

v4 certificate in RFC 4880.

```
Primary Key
 [Revocation-Signature]
 [Direct-Key-Signature ...]
 [
  (User-ID | User-Attribute)
  [Signature ...]
 ] ...
 [Subkey [Subkey-Revocation-Signature ...]
  Subkey-Binding-Signature ...] ...
```

v4 certificate in RFC 9580.
v6 subkeys are *not* allowed in v4 certificates.

```
Primary-Key
 [Revocation-Self-Signature]
 [Direct-Key-Signature...]
 {
  (User-ID Signature [Signature ...])
  [
    (User-ID | User-Attribute)
    Signature [Signature ...]
  ] ...
 }
 [[Subkey [Binding-Signature-Revocation]
  Primary-Key-Binding-Signature] ...]
```

v4 certificate in LibrePGP-02.
v5 subkeys are allowed in a v4 certificate.

Figure 3.1: Differences in the v4 certificate format specification of the different standards.


```
Primary Key
   [Revocation Signature ...]
    Direct Key Signature ...
   [(User-ID | User-Attribute)
           [Certification Revocation Signature ...]
           [Certification Signature ...]] ...
   [Subkey [Subkey Revocation Signature ...]
           Subkey Binding Signature ...] ...
   [Padding]
```

v6 certificate in RFC 9580
Only v6 subkeys are allowed in v6 certificates

```
Primary-Key
[Revocation-Self-Signature]
[Direct-Key-Signature...]
{
 (User-ID Signature [Signature ...])
 [
   (User-ID | User-Attribute)
   Signature [Signature ...]
 ] ...
}
[[Subkey [Binding-Signature-Revocation]
 Primary-Key-Binding-Signature] ...]
```

v5 certificate in LibrePGP-02
Note that the structure is the same as for v4 in LibrePGP

Figure 3.2: Differences between the v6 and v5 certificate format specification. The grouping of the elements with the parenthesis notation introduced by us is based on the textual clarifications in the LibrePGP specification (namely LibrePGP-02 draft, Section 11.1 https://www.ietf.org/archive/id/draft-koch-librepgp-02.html#name-transferable-public-keys).

3.4 Overview of Features of v4, v5, and v6 Keys and Planned Integration of PQC

This section gives an overview of the possible pairings of key, signature and encryption-related packet versions in RFC 9580 and LibrePGP. Furthermore, the current state of these two specifications with respect to the integration of PQC is explained.

Please note that for simplicity the figures in this section use "RFC4880" to mean "the old formats", e.g., v4 key and signature packets and v3 PKESK packets, however, there are minor changes in the successive specifications, as described in 3.3.3.

3.4.1 Allowed Packet Pairings in RFC 9580 and LibrePGP

Figure 3.3 shows an overview of the signing keys and signature formats allowed by RFC 9580. There is a one-to-one correspondence between v4 keys and v4 signatures on the one hand as well as v6 keys and v6 signatures on the other hand. Figure 3.4 shows an overview of the signing keys and signature formats allowed by LibrePGP.

Figure 3.5 shows the relations among v4 and v6 signature keys on the one hand and v3 and v6 PKESK ("Public-Key Encrypted Session Key") Packets as well as v1 and v2 SEIPD ("Symmetrically Encrypted and Integrity Protected Data") Packets on the other hand. LibrePGP does not introduce a new PKESK or SEIPD packet version. Instead, as shown in Figure 3.6, it introduces the OCB Packet as a new packet type for content encryption that can be used instead of the v1 SEIPD packets.

Neither RFC 9580 nor LibrePGP allows the creation of v3 signatures. Both specifications allow the verification of v3 signatures, however.

3.4.2 Planned Integration of PQC in OpenPGP and LibrePGP

Both, OpenPGP and LibrePGP are making efforts to specify the integration of PQC. In the following, an overview is given, first for OpenPGP and then LibrePGP.

3.4.2.1 PQC in OpenPGP

For OpenPGP two drafts exist that add composite encryption algorithms, as well as composite and standalone signature algorithms.

- signature algorithms:
  - draft-ietf-openpgp-pqc: SLH-DSA as standalone and composite combinations of ML-DSA with Ed25519 and Ed448,
  - draft-ehlen-openpgp-nist-bp-comp: composite combinations of ML-DSA mit ECDSA using NIST- and Brainpool-curves,
- encryption algorithms:
  - draft-ietf-openpgp-pqc: composite combinations of ML-KEM with X25519 and X448,
  - draft-ehlen-openpgp-nist-bp-comp: composite combinations of ML-KEM with ECDH using NIST- and Brainpool-curves.

For all composite combinations the newly introduced Algorithm IDs each represent one concrete combination of a PQC algorithm with an ECC algorithm with fixed security parameters resp. EC Domain Parameters. Furthermore, the current draft-ietf-openpgp-pqc[11] restricts PQC signatures to v6 keys and thus also to v6 signatures. It allows ML-KEM-768+X25519 for v4 keys but restricts ML-KEM-1024+X448 to v6 keys.

[11] https://www.ietf.org/archive/id/draft-ietf-openpgp-pqc-09.html

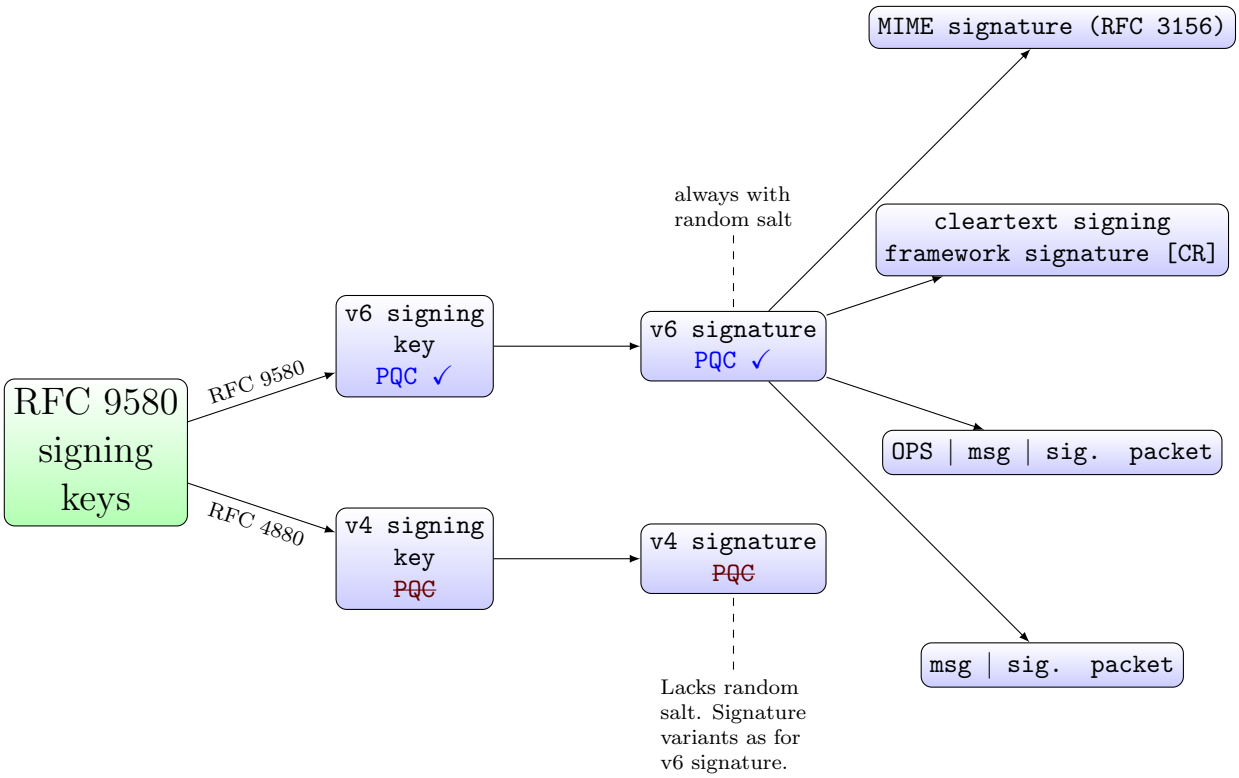

Figure 3.3: Overview of the signature variants for OpenPGP. v6 signatures may only be issued by v6 keys and vice versa for v4 keys. The four signature variants shown for v6 signatures are the same for v4 signatures even though they are omitted there for the purpose of a better overview.

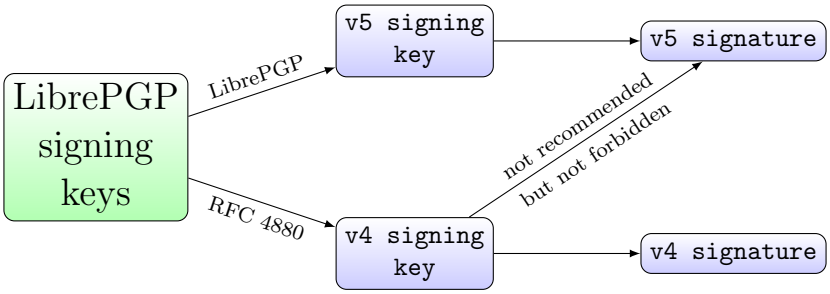

Figure 3.4: Overview of the signature variants for LibrePGP. v5 keys may only issue v5 signatures. For v4 keys it is recommended to issue only v4 signatures, but it is not forbidden to issue a v5 signature. The signature variants shown on the right side of Figure 3.3 also apply here. PQC signatures have not been specified for LibrePGP as of now.

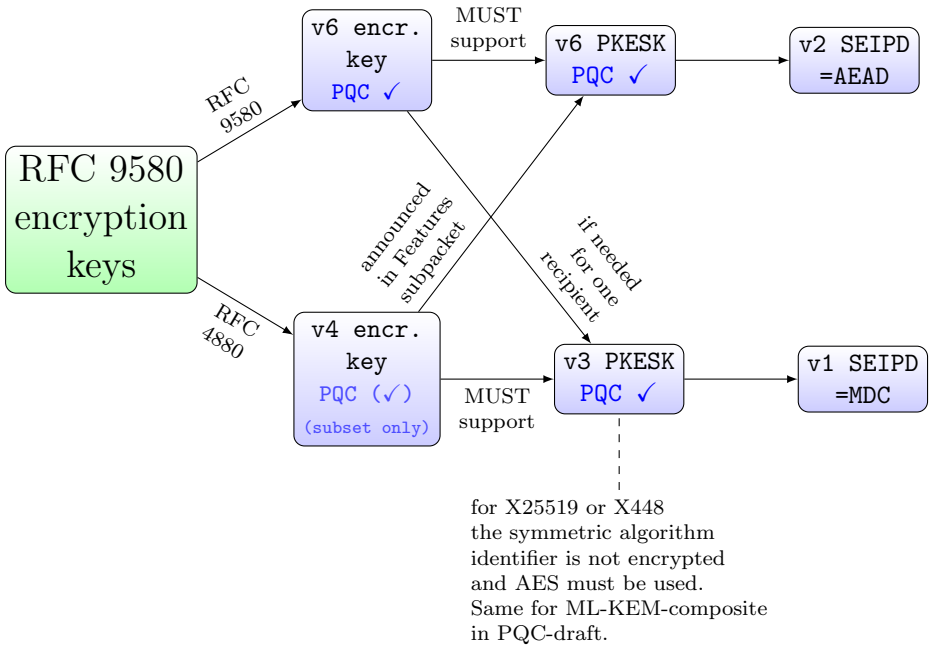

Figure 3.5: Overview of the encryption variants for OpenPGP with PQC.

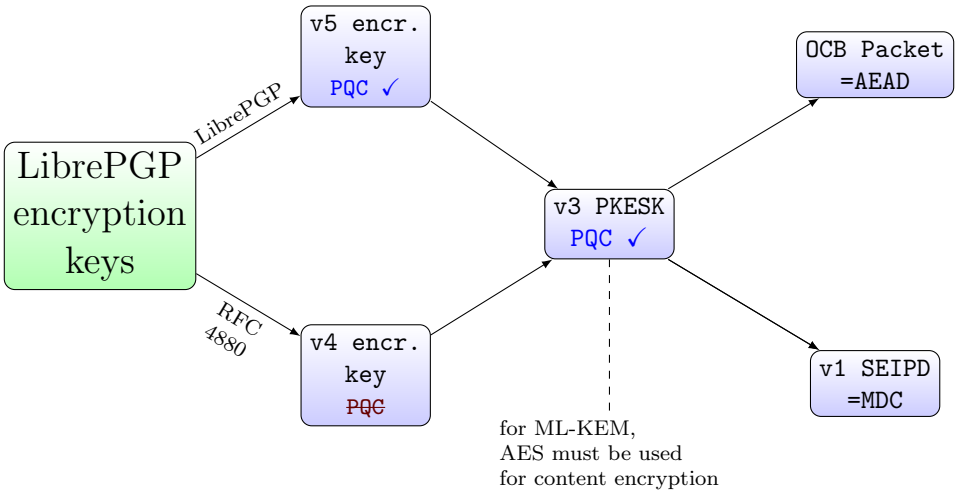

Figure 3.6: Overview of the encryption variants for LibrePGP with PQC.

3.4.2.2 PQC in LibrePGP

LibrePGP introduces "ML-KEM" as a new algorithm. Despite the name, this is also a combination of ML-KEM and ECDH. In contrast to the proposals in the OpenPGP drafts for PQC, it allocates only a single Algorithm ID and at the same time allows for flexible combinations of ML-KEM with ECDH-KEMs through parameterization.

While the LibrePGP standard does not explicitly preclude the use of ML-KEM in v4 keys, this restriction is understood by us to be implicitly given since the `fixed-info` field in the KEM-Combiner requires a 32-byte fingerprint which is only fulfilled for a v5 key.[12]

LibrePGP does not yet introduce PQC signature algorithms. Yet, at the point it should introduce them for v5 signatures, one important difference to OpenPGP is that v5 signatures do not hash a random salt before the message as v6 signatures do (see Section 4.1). This is a measure of v6 signatures that ensures the independence of the signature algorithm's security from the collision-resistance of the hash function used for the pre-hashing of the message. For SLH-DSA the addition of the salt can replace the use of the optional non-deterministic message randomizer. Since OpenPGP follows the hash-and-sign paradigm, message hashes are signed and not messages themselves. Accordingly, without the random salt, SLH-DSA would loose its independence from the collision resistance of the hash-function.

3.5 Tabular Overview of the Differences between RFC 4880, RFC 9580, and LibrePGP

Legend:

The feature or aspect in question is not specified for this standard.
A feature is allowed only passively, i.e., the implementation may accept the respective data structures, but not produce them.
A change in the specification formally causes wire-format incompatibility to implementations conforming to an older standard. This can for instance be given when a feature is completely deprecated both for generation and consumption.
No change compared to RFC 4880.
The features is newly introduced by the standard.

If multiple classifications should apply to a single tabular cell, then the one coming first in the list given above has precedence over the others.

Note that when writing RFC4880 in this comparison, we mean RFC4880 with the additions of RFC6637 and RFC5581, i.e., the additions of ECC and Camellia to OpenPGP.

Table 3.1: Overview of the differences in data structure encoding

Aspect	RFC 4880	RFC 9580	LibrePGP
Packet header formats[13]	"If interoperability is not an issue, the new packet format is RECOMMENDED."	SHOULD not use old packet format to generate new data.[14]	Equal to RFC 4880.

Continued on next page

[12] see https://mailarchive.ietf.org/arch/msg/openpgp/79WLHz5OgeAblsv-0v4tbpXyBw0/

[13] Section 4.2 in all three standards.

[14] This differing statement about the usage of the old packet format does not seem to formally introduce a difference to RFC 4880.

Table 3.1 – Differences in data structure encoding – cont'd from prev. page

Packet and/or version	**RFC 4880**	**RFC 9580**	**LibrePGP**
Appending optional checksum in ASCII armored encoding	Optional.	Allowed but discouraged.	Equal to RFC 4880.
time fields	Unix timestamps. Note: time fields overflow in the year 2106		
Literal Data Packet Header	defines special file name "_CONSOLE" as indicating a sensitive message	deprecates the special handling of the file name "_CONSOLE"	Equal to RFC 4880

Table 3.2: Overview of the differences in cryptographic packet types and versions

Aspect	**RFC 4880**	**RFC 9580**	**LibrePGP**
v2 public key packet	MUST NOT generate a v2 key, but MAY accept / continue to use it.		
v3 public key packet	MUST NOT generate a v3 key, but MAY accept / continue to use it.		
v4 public key packet	Specified, allowed for use.		
v5 public key packet	Not specified.		Introduces v5 key packets.
v6 public key packet	Not specified.	Introduces v6 key packets.	Not specified.
Secret Key Packets (IDs 5, 7) – Encryption of secret key material[15]	SED-style or v1-SEIPD-style encryption ⚠ **vulnerable to key-overwrite attacks**. See Section 5.1.	Introduces AEAD-encryption	Introduces OCB-encryption
		AEAD with inclusion of public key in additional data to prevent attacks based on public-key-tampering.	
v3 signature packet	Allows generation of v3 signatures.	Disallows generation of v3 signatures, they still "MAY" be accepted.	
v4 signature packet	Recommends generation of v4.[16]	Generation of v4 signatures allowed by v4 keys, but not by higher key versions.[17]	
v5 signature packet	Not specified.		Introduces v5 signature packets.

Continued on next page

[15] Secret Key Packet encryption is treated in Section 5.5.3 in all three standards.

[16] https://datatracker.ietf.org/doc/html/rfc4880#section-5.2

[17] RFC 9580, 10.3.2.2. Packet Versions in Signatures. https://www.ietf.org/archive/id/draft-koch-librepgp-01.html#section-5.2

Table 3.2 – Differences in cryptographic packet types and versions – cont'd from prev. page

Packet and/or version	RFC 4880	RFC 9580	LibrePGP
v6 signature packet	Not specified.	Introduces v6 signature packets.	Mentions v6 signature packets[18] as an option, but does not fully specify them. Partly, the specification is contradicting the specification of RFC 9580 regarding v6 signature packets.[19]
v3 PKESK	Specified, allowed for use.		
v6 PKESK	Not specified.	Introduces this packet version. Requires v2 SEIPD.	Not specified.
v4 SKESK	Specified, allowed for use.		
v5 SKESK	Not specified.	Not specified.	Introduces this packet version. The session key is OCB-encrypted.
v6 SKESK	Not specified.	Introduces this packet version. Requires v2 SEIPD. The session key is AEAD-encrypted.	Not specified.
SED packet	Specified, allowed for use. ⚠ **No integrity protection.**	MUST NOT be generated and processed only with precaution measures.	
v1 SEIPD packet	Specified, allowed for use. Ad-hoc integrity protection through hash-then-encrypt without key separation. ⚠ **SED-downgrade possible.**		
v2 SEIPD packet	Not specified.	Introduces this packet version. Realizes AEAD encryption with key separation.	Not specified.
OCB Encrypted Data Packet (Tag 20)	Not specified.		Introduces this packet. Realizes AEAD encryption without key separation. ⚠ **SED-downgrade possible, see Section 4.3.1 .**

Table 3.3: Overview of the differences in packets and subpackets

Packet	RFC 4880	RFC 9580	LibrePGP
Signature Type: Attested Key Signature.			Defines this new subpacket.

Continued on next page

[18] Any mentioning of v6 signatures will be removed in LibrePGP-02

[19] For instance in https://www.ietf.org/archive/id/draft-koch-librepgp-01.html#section-5.2.4, v6 signatures are addressed but the hashing of the signature salt is omitted.

Table 3.3 – Differences in packets and subpackets – cont'd from prev. page

Aspect	RFC 4880	RFC 9580	LibrePGP
Signature Subpacket: Preferred Encryption Modes	Not specified.		Introduces and at the same time deprecates this subpacket
Signature Subpacket: Revocation Key	Specified, allowed for use.	"Applications MUST NOT generate such a subpacket."	Specified, allowed for use.
Signature Subpacket: 30 – Features	Available features: v1 SEIPD support.	Features added: v2 SEIPD support.	Features added: OCB Encrypted Data Packet, and Version 5 Public-Key Packet support.
Signature Subpacket: Notation Data	Specified, allowed for use.		Additionally defines a registry for some predefined notation types.
Signature Subpacket: Key Block	Not specified.		Defines this new subpacket.
Signature Subpacket: Attested Certifications	Not specified.		Defines this new subpacket.
Signature Subpacket: Key Flags	Defines this subpacket.		Adds two new key usages for time stamping and "restricted encryption key".
Padding Packet	Not specified.	Introduces this new packet which provides a mechanism of obscuring the length of encrypted data.	Not specified.

Table 3.4: Overview of the differences in cryptographic algorithms

Aspect	RFC 4880	RFC 9580	LibrePGP
IDEA	Allows this cipher.	Forbids the generation of ciphertexts.	Allows this cipher.
TripleDES	Allows this cipher.	Forbids the generation of ciphertexts.	Allows this cipher.
CAST5	Allows this cipher.	Forbids the generation of ciphertexts.	Allows this cipher.
Camellia	Added in RFC 5581.	Incorporates RFC 5581.	Incorporates RFC 5581
DSA	Allows this algorithm.	Forbids all cryptographic operations for this algorithm.	Allows this algorithm.
DSA key sizes	q MUST NOT be below 160 bits. Public-key size SHOULD NOT be below 1024 bits.	N/A	Equal to RFC 4880.
ElGamal	Allows this algorithm with a minimal key size of 1024 bits.	Forbids key generation and encryption.	Equal to to RFC 4880.

Continued on next page

Table 3.4 – Differences in cryptographic algorithms – cont'd from prev. page

Aspect	RFC 4880	RFC 9580	LibrePGP
RSA	Allows this algorithm. ⚠ **RSA encryption in OpenPGP implies the use of PKCS#1 v1.5 encoding.**	Keys SHOULD not be generated.	Allows this algorithm.
RSA key sizes	SHOULD NOT be below 1024 bits.	MUST NOT be below 2048 bits, SHOULD NOT be below 3072 bits.	Equal to RFC 4880.
Generic ECDH	Reserves Code Points, defined in RFC 6637 with NIST curves	Adds and deprecates X25519 from rfc4880bis which MAY be used with v4. Incorporates RFC 6637. Adds brainpool curves.	Incorporates RFC 6637. Adds Curve25519, X448, and brainpool curves.
Generic ECDSA	Reserves Code Points, defined in RFC 6637 with NIST curves	Incorporates RFC 6637. Adds brainpool curves.	Incorporates RFC 6637. Adds brainpool curves.
Generic EdDSA	Not specified.	Adds and deprecates Ed25519 from rfc4880bis. MAY be used with v4.	Adds Ed25519 and Ed448
Ed25519	Not specified.	New code point ("MUST") for this curve.	Specified as generic EdDSA algorithm.
X25519	Not specified.	New code point ("MUST") for this curve.	Specified as generic ECDH algorithm.
Ed448	Not specified.	New code point ("SHOULD") for this curve.	Specified as generic EdDSA algorithm.
X448	Not specified.	New code point ("SHOULD") for this curve.	Specified as generic ECDH algorithm.
AEAD: OCB mode	N/A	Introduces OCB mode (MUST).	
AEAD: EAX mode	N/A	Introduces EAX mode. (MAY)	Introduces EAX mode, but also deprecates it.
AEAD: GCM mode	N/A	Introduces GCM mode (MAY).	Not supported.

Table 3.5: Overview of the differences in the certificate structure

Aspect	RFC 4880	RFC 9580	LibrePGP
v4 key: User ID Packet	Presence is mandatory.	Presence is optional. This is a wire-format breaking change.	Presence is mandatory.
v4 key: number of certification signatures on User ID packet	Zero or greater.	Zero or greater.	One or greater.

Continued on next page

Table 3.5 – Differences in the certificate structure – cont'd from prev. page

Aspect	RFC 4880	RFC 9580	LibrePGP
v5 / v6 key: User ID Packet	N/A	Presence is optional for a v6 key.	At least one is required for a v5 key.[20]
v5 / v6 key: Certification Signature on User ID or User Attribute packets	N/A	Presence is optional for a v6 key.	At least one is required for a v5 key.
v5 / v6 key: Subkey-Binding-Signature	N/A	Presence is mandatory for a v6 key.	Presence is mandatory for a v5 key.
v4 key: Direct Key Signature 0x1F	Presence is optional for a v4 key.		
v5 / v6 key: Direct Key Signature 0x1F on primary key	N/A	Is mandatory for a v6 key.	Is optional for a v5 key.[21]

Table 3.6: Overview of the differences in cryptographic mechanisms

Aspect	RFC 4880	RFC 9580	LibrePGP
PKESK	v3 PKESK	v6 PKESK	v3 PKESK
Identification of recipient	8 byte Key ID	20 byte (v4) or 32 byte (v6) key fingerprint	Equal to RFC 4880.
Session-Key-Reuse	Not specified.	Introduces a feature for session key reuse by including a salt value in the PKESK Packet that ensures fresh content-encryption keys. Enables replies to messages without knowing the other party's encryption key.	Not specified.
AEAD	N/A	v2 SEIPD Packet	OCB Packet
session key for AEAD algorithms		key derivation binds the message key to the symmetric encryption algorithm	⚠ **lack of key derivation leads to security vulnerability in LibrePGP, refer to Sec. 4.3.1**
form of additional data of each AEAD chunk		does not include chunk index (as it is already used for the nonce-derivation)	includes the chunk index (although it is already used to derive the nonce)

Continued on next page

[20] https://www.ietf.org/archive/id/draft-koch-librepgp-01.html#section-11.1
[21] Not mentioned in Section 11. of LibrePGP-01.

Table 3.6 – differences in cryptographic mechanisms – cont'd from previous page

Aspect	RFC 4880	RFC 9580	LibrePGP
Signature	v4 Signature	v6 Signature	v5 Signature
Signature meta data – Sec. 4.2	⚠ **formatting octet, file name and date field in header of a Literal Data Packet are not protected by the signature**	Equal to v4 signatures. ⚠ **Meta data unprotected by signature, see column for RFC 4880. Warns about the lack of integrity protection**[22].	Introduces hashing of Literal Data Packet meta data in message digest .
Signatures Trailer Aliasing – Sec. 4.5.3	No vulnerability.	No vulnerability.	⚠ **v5-to-v3-cross-version-aliasing in signature trailer leads to existential forgery vulnerability in LibrePGP**
Randomization of signatures through a salt		v6 signatures include a random salt that is fed into the message digest before any other data, thus randomizing any signature.	v5 signatures do not use salting.
Signature hash computation – recoding applied to message before hash computation	converting line endings to <CR><LF>	Additionally introduces recoding of the signed data for "document" type signature into UTF-8.[23] Text data in Literal Data Packets contents must be encoded in UTF-8.[24] [25]. Section 4.6 addresses this change in some more detail.	Equal to RFC 4880.
X25519, X448 Derivation of KEK	N/A	New X25519/X448 algor. HKDF(ephemeral-public-key, recipient-public-key, shared-point-coordinate)	Generic ECDH construction Hash(shared-point-coordinate, id-of-key-wrap-algorithm-for-session-key, curve-OID, public-key-alg-ID, KEK-alg-ID-for-AES-Key-Wrap, recipient-fingerprint)[26]

[22] https://www.rfc-editor.org/rfc/rfc9580.html#section-5.9-4. See Section 4.6 for a discussion why the lack of integrity protection for the LIT header fields in v6 signatures is still considered as a potential security problem.

[23] https://www.rfc-editor.org/rfc/rfc9580.html#section-5.2.4-3

[24] https://www.rfc-editor.org/rfc/rfc9580.html#section-5.9-3.4.1

[25] Related discussion in a GitLab issue in the RFC 9580 Repository: https://gitlab.com/openpgp-wg/rfc4880bis/-/merge_requests/111

[26] This specification of the inputs to the hash function is only giving the cryptographically relevant inputs. For a complete definition of the input to the hash function see Section 13.5. in [LPGP01].

Table 3.7: Overview of differences in string-to-key (S2K) algorithms

Aspect	**RFC 4880**	**RFC 9580**	**LibrePGP**
Simple S2K	Specified, allowed for use.	Generation forbidden ("MUST NOT")[27]	Specified, allowed for use.
Salted S2K	Specified, allowed for use.	Generation discouraged ("SHOULD NOT") except when string "is [of] high entropy".	Specified, allowed for use.
Iterated and Salted S2K	Specified, allowed for use.		
Argon2	Not specified.	Introduces this algorithm.	Not specified.

[27] RFC 9580 warns that the Simple S2K algorithm may lead to session key and IV reusage for password reusage. While identical passwords may indeed lead to session key reusage, the IV reusage is irrelevant. This is due to the fact that the randomization of the CFB encryption in OpenPGP SED and v1 SEIPD packets does not depend on the IV – which is always set to a zero-block – but is instead based on a randomized block at the start of the plaintext. This mechanism ensures a randomized encryption even under session key reuse.

Chapter 4

Differences of RFC 9580 and LibrePGP in Cryptographic Mechanisms

In this chapter we give a detailed analysis of specific features of RFC 9580 and LibrePGP.

4.1 Signature Salt

While RFC 9580 introduces a random salt that is prepended to any v6 signature, LibrePGP's v5 signatures do not introduce any comparable features.

4.1.1 Assessment

The introduction of a signature salt ensures collision-resistance of the signature algorithm even if the collision-resistance of the hash function is weakened. Accordingly, v6 signatures rely on weaker security assumptions than v5 signatures.

4.2 Signing Literal Data Packets

Literal Data Packets are the standard packages for encapsulating data that is subject to signing and encryption. A key difference in RFC 9580 and LibrePGP is that RFC 9580 inherits from the v4 signatures the feature that the Literal Data Packet's header field are not signed [1]. LibrePGP's v5 packets on the other hand introduce hashing of the Literal Data Packet's header fields (content type, file name, and date) at least in the case of signatures of the type of document signatures.[2]. LibrePGP also introduces a signature subpacket for v4 signatures that includes the Literal Data Packet's header fields in the signature.[3]

4.2.1 Assessment

The lack of integrity protection of signed-only emails in v6 signatures may lead to security implications if applications falsely rely on the integrity of the file name or date header fields.

[1] https://www.rfc-editor.org/rfc/rfc9580.html#section-5.9-4
[2] https://www.ietf.org/archive/id/draft-koch-librepgp-01.html#section-5.2.4-10
[3] https://www.ietf.org/archive/id/draft-koch-librepgp-01.html#section-5.2.3.33

4.3 AEAD Packets

4.3.1 Legacy Cipher Downgrade Attacks

The lack of a key derivation in LibrePGP's AEAD packets is analysed in [SR24]. In this work it is shown that under the assumption of fully-revealing CFB decryption oracles, LibrePGP's AEAD packets can be decrypted via such a CFB decryption oracle given through the decryption of SED packets.

4.3.2 Nonce-Reuse Bounds under Session Key Reuse

In this section we address the risk of nonce-reuse or nonce-collisions, as an alternative term, in AEAD as specified in RFC 9580 and LibrePGP, that may occur under session key reuse. Since nonce reuse in the AEAD schemes that are defined by the specifications at hand leads to security vulnerabilities, they must be avoided with sufficiently high probability. Thus the bounds on the number of AEAD encryptions we derive here correspond to the maximum number of messages that may be encrypted using the same session key while keeping the nonce-collision probability below a required upper bound.

Note that both specifications differ with respect to the standing that session key reuse has: While RFC 9580 explicitly foresees the possibility of a mechanism for session key reuse[4], LibrePGP does not define or foresee such a mechanism and instead specifies per-message unique session keys[5]. Nevertheless, the AEAD specification of LibrePGP OCB packets accounts for session key reuse by requiring random nonces. Thus an implementation might decide to use this feature also for LibrePGP under certain conditions. Accordingly, a comparison of the bounds on the number of messages that may be encrypted under the same session key before reaching a critical nonce-collision probability might turn out to be useful for implementers that have to consider session key reuse. However, due to the different standing that session key reuse has in both specifications, the result should not be seen as quantitative comparison of a security feature.

In the following two subsections, we derived upper bounds for the number of OpenPGP and LibrePGP AEAD messages under session key reuse that bound the nonce-collision probability. For our approximations we use the relation describing the well-known birthday problem

$$p_{\text{coll}} \approx \frac{n^2}{2S}$$

with p_{coll} being the probability for at least one collision, S the size of the nonce space, and n the number of AEAD encryptions. See for instance [Mat24] for a derivation of this approximation of the probability.

In the following analysis, we employ as the tolerable nonce collision probability the value $p_{\text{coll}} = 2^{-32}$, which is a typical value for this bound.[6] With $S = 2^N$, with N being the width of the nonce in bits, we arrive at the relation

$$n_* \approx 2^{(N-32)/2}, \tag{4.1}$$

where n_* is the bound for the number AEAD-encrypted messages after exceeding which $p_{\text{coll}} \geqslant 2^{-32}$.

Equation 4.1 describes the case where each nonce is fully random and messages consist only of a single AEAD chunk. In the following subsections, this equation is adapted to the specific conditions that hold for the OpenPGP and LibrePGP AEAD specifications.

[4] RFC 9580, Section 5.13.2: "The KDF mechanism provides key separation between cipher and AEAD algorithms. Furthermore, an implementation can securely reply to a message even if a recipient's certificate is unknown by reusing the Encrypted Session Key packets and replying with a different salt that yields a new, unique message key."

[5] draft-koch-librepgp-03, Section 2.1: "The sending LibrePGP generates a random number to be used as a session key for this message only. "

[6] The bound 2^{-32} for the nonce-collision probability is for instance defined in NIST's GCM specification SP 800-38d. Note that this is not necessarily a bound that guarantees secure operation for all application contexts.

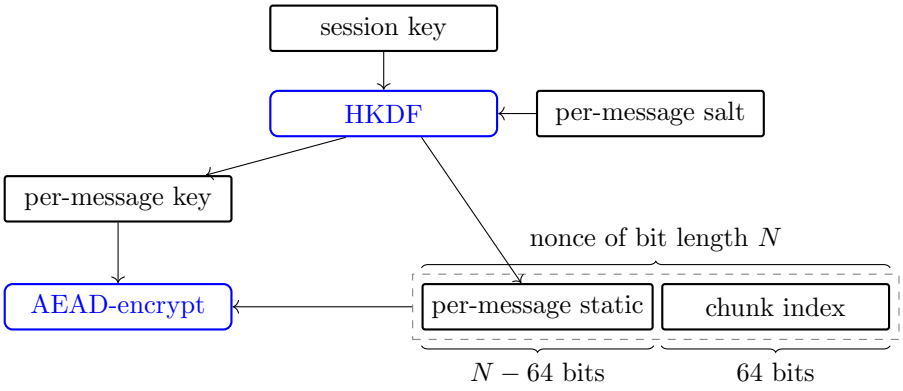

Figure 4.1: OpenPGP AEAD nonce derivation.

4.3.2.1 Nonce-Reuse in RFC 9580

In the v2 SEIPD Packets defined in RFC 9580, the nonce-reuse probability is reduced compared to that of the raw AEAD scheme due to the salt-based derivation of a new data encryption key for each message. As depicted in Figure 4.1, in the v2 SEIPD Packets the high part of the nonce is derived from the session key and the per-message salt. This part amounts to $N - 64$ bits, where N is the bit length of the nonce for the given AEAD scheme. The low 64 bits of the nonce are given by the 64-bit-encoded chunk index.

Accordingly, a collision occurs when the per-message key and the high part of the nonce collide. The probability for a such a collision is actually independent of the session key reuse due to the salt-based derivation of the per-message-key. In the following, for the sake of a unified terminology, we nevertheless refer to this case as a nonce collision, since in the case of LibrePGP, due to the absence of the key derivation that occurs in OpenPGP v2 SEIPD, a key collision is always given in the case of session key reuse.

Using Equation 4.1, we find as the bound for the number of messages encrypted with session key reuse for OpenPGP v2 SEIPD as

$$n_*^{\text{OpenPGP}} \approx 2^{(K+N-64-32)/2},$$

where K is the size of the per-message key in bits. Table 4.1 gives the concrete values for this message bound for the three AEAD schemes specified in RFC 9580 when using a 128-bit data encryption key. The result shows that a virtually unlimited number of messages can be produced without risking a nonce collision.

4.3.2.2 Nonce-Reuse in LibrePGP

For the LibrePGP OCB Packets, we determine the message bound n_*^{LibrePGP} for OCB and EAX as the only supported AEAD algorithms. For the sake of simplicity, we limit the analysis to applications that use fixed sized messages of a length of 2^M bytes, with M being a non-negative integer. The nonce handling in LibrePGP's AEAD encryption is depicted in 4.2. The main difference to AEAD in OpenPGP that affects the nonce-collision probability is that the session key is used directly for the message encryption. Accordingly, for the nonce-collision probability, only a collision of the nonce itself has to be considered.

In LibrePGP AEAD the initial nonce value is a random bit string of the full length of the nonce. The encoded chunk index is XORed to the rightmost part of the initial nonce. This means that each message occupies a randomly placed window of nonces[7] within the nonce space the size of which

[7]The XORing of the chunk index into the non-zero initial nonce causes "jumps" in the sequence of nonces for each

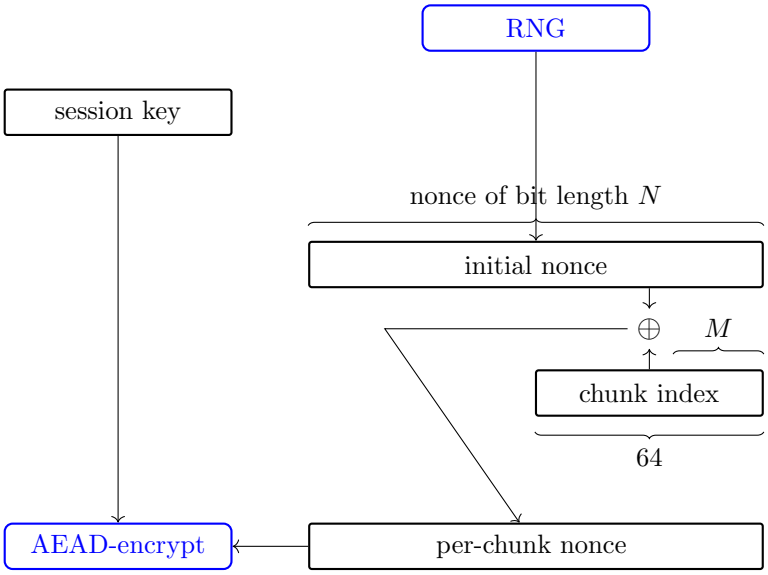

Figure 4.2: LibrePGP AEAD nonce derivation. M with $0 < M \leqslant 64$ represents the number of bits that are occupied by the maximal chunk index of the message.

Table 4.1: Bounds of the number of AEAD messages with session key reuse at which a nonce collision occurs with a probability of 2^{-32}. For the case of RFC 9580, 128-bit message encryption keys are assumed.

RFC 9580				**LibrePGP**	
Algorithm	**Nonce-length in bits (N)**	**Size of random part in bits (N-64)**	**Message bound n_*^{OpenPGP}**	**Message bound n_*^{LibrePGP}**	
				1 chunk	**2^{26} chunks**
OCB	120	56	$2^{(128+56-32)/2} = 2^{76}$	$2^{(120-32)/2} = 2^{44}$	$2^{(120-26-32)/2} = 2^{31}$
EAX	128	64	$2^{(128+64-32)/2} = 2^{80}$	$2^{(128-32)/2} = 2^{48}$	$2^{(128-26-32)/2} = 2^{35}$
GCM	96	32	$2^{(128+32-32)/2} = 2^{64}$	N/A	

window is equal to the number of chunks in the message[8]. Partial overlaps cannot occur for such messages: When the high $N - M$ bits are differ, there is no overlap, and when the high $N - M$ bits are equal, there is full overlap, i.e., each nonce of one message collides with one nonce of the other message.

In reality, partial overlaps, i.e., overlaps for only a subset of the nonces, would potentially occur for massages with differing numbers of chunks or when they are consisting of x chunks with $x \neq 2^y$ for any integer y. This would lead to a certain deviation of the message bounds from the simplified model applied here.

The formula for the message bound for messages consisting of 2^M chunks is

$$n_*^{\text{LibrePGP}} \approx 2^{(N-M-32)/2},$$

In order to enable a meaningful comparison to OpenPGP, where the message bounds are independent of the number of chunks per message, we provide the minimal and a maximal message bound, occurring for messages with a single chunk and those with the maximum number of chunks, respectively. As no actual maximal bound exists for the maximum number of chunks, we use as the basis the maximal OpenPGP packet length of 2^{32} bytes that can be specified with a single packet length field. We point out that in principle, partial body lengths allow arbitrarily long plaintexts in OpenPGP and LibrePGP. The smallest possible chunk size is 64 bytes. This leads to a presumed maximal number of chunks of $= 2^{32}/64 = 2^{26}$. The resulting message bounds are given in the two rightmost columns of Table 4.1.

As a conclusion, LibrePGP, despite its lack of a per-message-key derivation, allows for a reasonably high number of AEAD messages with session key reuse, before critical nonce-collision probabilities occur, even under the rather unrealistic aggravating assumption of a small chunk size for large messages. They are, nevertheless, considerably lower than those found for OpenPGP.

4.4 Possible Reinterpretation of v5 Signatures as v3 Signatures

In a GitLab issue[9] in the repository of RFC 9580 document, Demi Marie Obenour reports an ambiguity with respect to the encoding of the signature meta data trailer, or short *signature trailer*, for the different signature versions. This was affecting the new signature format specified in draft version of RFC 9580 at that point. The *signature trailer* is term for the meta data that is hashed in the message digest computation after the message has been hashed. The v6 signature format as specified in the final version of RFC 9580 was modified to avoid the ambiguity. As apparent from Table 4.2, v6 signatures can be reliably distinguished from v4 signatures due to the aligned version bytes in the meta data trailer. Both v4 and v6 signatures could be reinterpreted as v3 signatures if the implementation would accept the signature type 0xFF. However, this signature type is not defined and RFC 9580 explicitly forbids it.

The ambiguity is preserved, though, in LibrePGP v5 signatures. The presence of such an ambiguity results in the possibility of existential signature forgeries: Given an OpenPGP message signed with one signature version it is possible to modify the message, the signature meta data including the signature version such that the signature is still valid. The following details the vulnerability of the v5 signatures.

In the signature computation for v3 and v4 signatures, first the message body is hashed, then signature version specific trailer made up of signature meta data. v6 signatures differ from this only in the fact that prior to the message body, the signature salt is hashed.

message. Nevertheless, since here we are only considering collisions between messages made up of the same number 2^M of chunks, a contiguous nonce window is occupied by each message, since the lower M bits of the nonce take on all possible values.

[8] Here we ignore the final empty chunk for simplicity.

[9] https://gitlab.com/openpgp-wg/rfc4880bis/-/issues/130

v3:	...	...	...	...	...	**...msg**	sig-type	4-byte crea. time
v4:	...	...	...	...	**hashed-subp bd**	0x04	0xFF	4-byte *hashed length*
v6:	...	...	...	...	**hashed-subp bd**	0x06	0xFF	4-byte *hashed length*
v5:	...	0x05	0x0FF	hl1	hl2	hl3	$\underbrace{\text{hl4}}_{0 \text{ or } 1}$	last 4 bytes of *hashed length*

Table 4.2: Signature trailers that are input into the message digest for v3, v4, v5 (LibrePGP), and v6 signatures. "hashed-subp bd" stands for "hashed-subpacket body". Each column represents a single byte except where noted otherwise or where bold text is used. The four bytes in the last row labelled "hl" followed by a number are the high order bytes belonging to the same field *hashed length* as the final 4 octets in that row. The signature type 0xFF is not defined and explicitly disallowed in RFC 9580.

Since OpenPGP signatures do not feature a leading length field of the body data, the meta data trailers can only be parsed from the end of the stream. As shown by Table 4.2, LibrePGP v5 signatures can be reinterpreted as v3 signatures, if the value of *hashed length* in the v5 signature causes the byte hl4, which is the 5th lowest byte of the big endian length encoding of *hashed length*, to have a value that coincides with a valid signature type. For the meaning of the field *hashed length*, see Figure 4.3 and the accompanying text. The only possibly reachable v3 signature types are document signatures of type 0x00 ("Binary Signature") or 0x01 ("Text Signature"), since it is not possible to create a v6 signature packet with sufficiently large *hashed length* so that the byte hl4 of the v5 signature, which aliases to the signature-type byte of the v3 signature, becomes larger than 1. This limitation is due to the fact that the overall size of a signature packet cannot be more than 2^{32} bytes (4 GiB).[10]

Thus, in general, the LibrePGP v5 signatures can be modified to v3 signatures over different data. The creation time of this signature may be unrealistic in some cases, but this will not stop applications from verifying such a signature correctly. The difference in the data is that the further trailer fields of the v5 signature appear in the message body.

If arbitrarily high values of *hashed length* could be reached, v5 signatures could also be interpreted as v4 or v6 signatures. However, that would require a size of *hashed length* of more than 4 TB for v4 and more than 6 TB for v6, in order to match the 5th-to-last and 6th-to-last bytes of the corresponding signature trailers. Since there is no apparent way to create signature packets with more than 4 GB, the maximum length allowed by the 4-byte length field of their header, these ambiguities can be precluded.

The possibility of the inverse reinterpretation of v3 signatures as v5 signatures naturally principally exists, however, it would require the *signature creation time* hashed at the end of the v3 signature to coincide with the *hashed length* of the v5 signature. Furthermore, the creation of v3 signatures is disallowed by both LibrePGP[11] and RFC 9580. Their acceptance is specified as "MAY" in both standards.[12]

As a counter-measure that prevents the above described reinterpretation attack, LibrePGP could either specify v5 signatures that do not allow for this kind of signature trailer aliasing, or forbid the verification of v3 signatures with v5 keys.

[10] Actually, the *hashed length* can in principle be larger than the size of the encoded packet contents, if there is hashed data contributing to *hashed length* that is not encoded in the signature packet. Currently, the only such data is given by the Literal Data Packet meta data which is hashed in the case of v5 signatures. Its size is limited to 262 bytes. However, this contribution is insignificant for our analysis here.

[11] https://www.ietf.org/archive/id/draft-koch-librepgp-01.html#section-5.2.3-3

[12] https://www.ietf.org/archive/id/draft-koch-librepgp-01.html#name-signature-packet-tag-2

4.5 Analysis of Conceivable Signature Ambiguities in v4 and v6 Signatures

4.5.1 Analysis for v6 Signatures

In this section we analyze further potential ambiguities in the signature hash. Specifically, we evaluate the idea of the reinterpretation of the hashed data for a document and the signature over a key, such as a subkey binding signature, of the same signature version. The corresponding attempted buildup of the hashed data of a document crafted for this purpose is shown in the left column of Figure 4.3. It carries the signatures type 0x00. In order to interpret the same hashed data as a binding signature, the signature type must amount to 0x18. In order to achieve this difference in the signature type, the area in the hashed data that represents the signature packet body, must have a different location in both interpretations. This is reached by making the start of the signature packet body of the key signature the final part of the signed document, as shown in Figure 4.3. However, this approach fails due to the fact that the *hashed length* – after having been reduced modulo 2^{32} – is encoded at the end of the hashed data. *hashed length* amounts to the byte length of the hashed part of the signature packet body, as shown in Figure 4.3. The *hashed length* in both interpretations cannot be equal since in the case of the signature over the key, the signature packet body always contains the whole signature packet body for the document signature additionally to its other contents. Thus the *hashed length* for the signature over the key will always be larger. In principle, the encoded value for the signature over the key could be brought to equality with the encoded value for the document by enlarging the hashed subpackets of the key appropriately, namely such that the modular reduction leads to equality.

However, this would require subpackets of a size of more than 2^{32} bytes, which is not possible, due to the whole signature packet's size being constrained to $2^{32} - 1$ bytes by the available length encoding in the header. The available length for the encoding of signature subpackets is further reduced since the signature packet also contains the encoded fields of the signature salt and the signature value itself.

Under the assumption of the following changes to the standard it would be possible to reach equality of the two encoded values (after the modular reduction) of *hashed length* in Figure 4.3 and thus enable the attack depicted above:

- Allowing a larger signature packet size by either
  - specifying a new encoding of packet lengths of more than $2^{32} - 1$ bytes
  - or allowing partial body lengths (Sec. 4.2.1.4 in RFC 9580) for signature packets, leading to unlimited body lengths.
- The specification of packets which cause more data to be hashed than what they encode. This could be the case for instance if a subpacket for signature context data was specified, where the context is omitted from the packet encoding in order to force the verifier to reconstruct it according to the actual circumstances. If such a packet existed for signatures over a key, then it would be possible to make it part of the signature over the key in order to reach the 2^{32} byte overflow as described above.[13]

[13] A packet with this characteristics included in the hashed data of the document signature cannot be used to compensate the difference in the values of *hashed length*, as it can only increase this value equally in both cases.

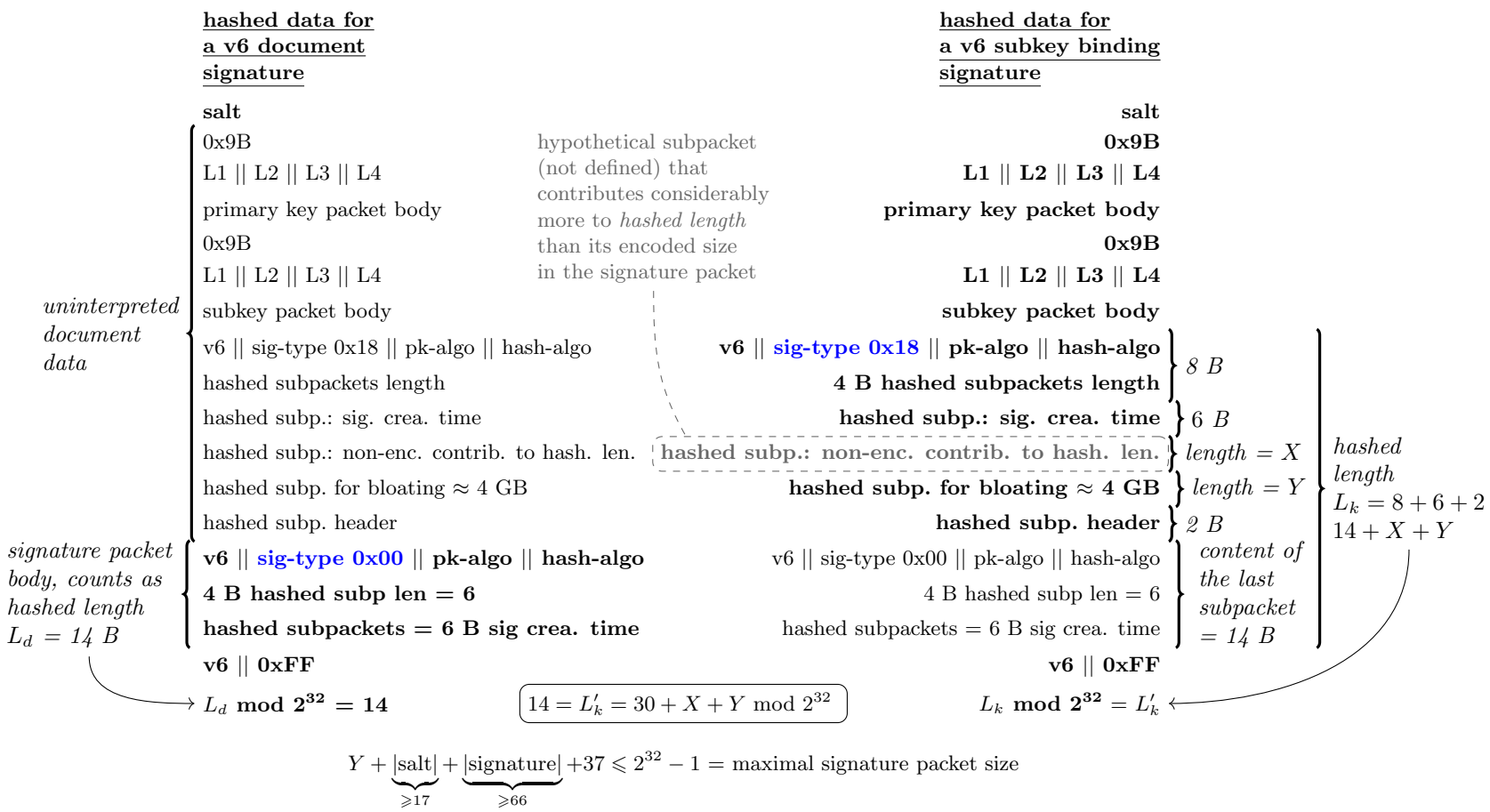

Figure 4.3: Depiction of an attempted but impossible signature forgery applied of a v6 signature over a key through a v6 document signature. Both columns show the same hashed data for two different signatures. The left column shows their interpretation as in a document signature of type 0x00, the right as in a signature over a key. The values set in bold indicate interpreted data, the normal font indicates uninterpreted data for the respective signature type. Fields surrounded by a dashed line indicate fields that are not fully encoded in the signature packet in which it is interpreted. In order to mount a successful attack, one has to craft a document in such a way that in the interpretation as a key signature, the hashed data following the document data in the document signature is so to say hidden in the contents of a subpacket in the interpretation as a signature over a key. The forgery would only be possible if equality between the values of *hashed length* for both signatures would be achieved. This would require a large enough contribution to *hashed length* of length X to cause the wrap-around at 2^{32} in the encoding of *hashed length* of the subkey binding signature, where at the same time the encoding of "hashed subpackets leading" is small enough not cross the 2^{32} byte boundary for the encoding of the whole signature packet. This is not possible with the existing signature subpackets in v6.

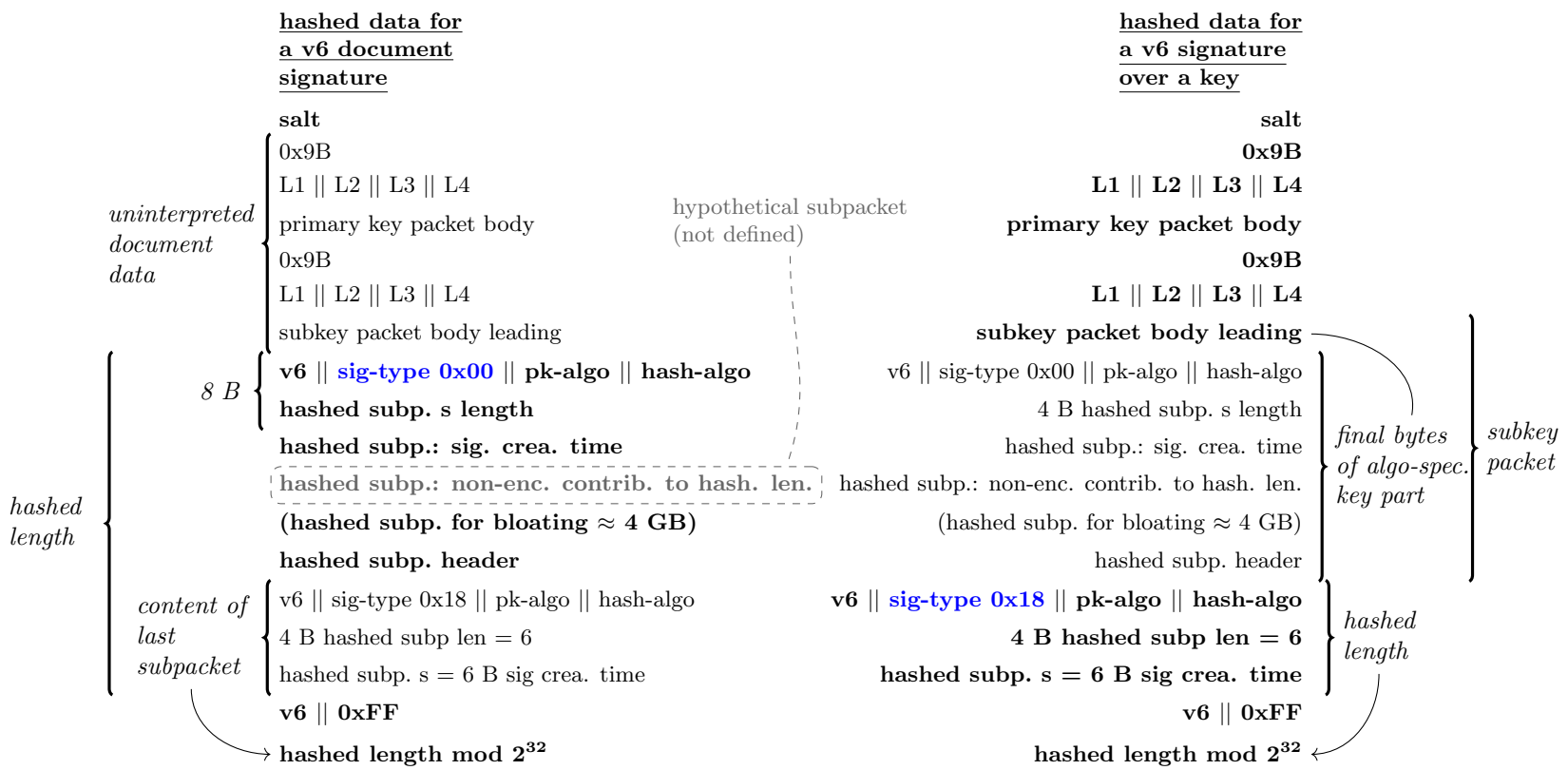

Figure 4.4: Depiction of the principle of an inverse document-to-key-type attack for v6 that turns out to be impossible. Here, the algorithm-specific public-key value is to be chosen resp. crafted such that its tail aligns with 8-byte start of the document signature's trailer. The signature creation time included in the document signature's trailer poses a special challenge as it must either be predicted by the attacker when crafting the key or be aligned to a modifiable field in the already existing key. The attack is not possible since *hashed length* on both sides cannot be brought to equality since the *hashed length* for the document signature will always be larger than that of the signature over a key. Furthermore, the wrap-around at 2^{32} bytes of the encoded value of *hashed length* cannot be reached since the signature packet's encoded size is maximally $2^{32} - 1$ bytes and in it space is reserved for the salt and the signature value. As soon as there was a mechanism in v6 signatures allowing for a significant amount of data contributing to *hashed length* that is not encoded in the signature packet, similarly to the Literal Data Packet meta data being hashed in v5 signatures, the wrap-around at 2^{32} bytes of the encoded value of *hashed length* may indeed be reached with a sufficiently high value of the remainder and the attack could succeed. However, in order to generally bloat the *hashed length* of the document signature in order for a limited non-encoded contribution to it being able to cause the wrap-around, the "hashed subpacket for bloating" indicated in the figure would have to be used, the contents of which count toward *hashed length* only for the document signature. Nevertheless, the key would have to contain this almost 4 GB long string as part of its public key, which is of course unrealistic. Accordingly, for a realistic attack, the conceived mechanism for the non-encoded contribution to *hashed length* would have to be bring a contribution in the domain of 2^{32} byte (4 GB) by itself.

4.5.2 Analysis for v4 Signatures

The case of v4 signatures differs from that of v6 signatures only marginally:

- v4 signatures do not use a salt. This has no effect on the potential attack described in Section 4.5.1, except that due to the missing salt, the encoded size of the signature packet's area that is not counted towards *hashed length* is smaller and thus the space left in the packet in order to cause the wrap-around at 2^{32} bytes is somewhat larger.
- The whole area of signature subpackets of a v4 signatures can be at most of a size of $2^{16}-1$ bytes. This greatly increases the gap for reaching the wrap-around of *hashed length* at 2^{32} compared to the case of v6 signatures.

4.5.3 Analysis for LibrePGP's v5 Signatures

LibrePGP's signature computation differs in two aspects that are relevant for the signature disambiguation in the sense of the analysis in Section 4.5.1:

- In document signatures (of types 0x00 and 0x01), the header data of the Literal Data packet, namely the format byte, the file name with a preceding length octet, and the signature date, are part of the hashed data, yet they are not encoded anywhere in the signature packet.[14] However, this does not help in the compensation of the length difference between the values of *hashed length* in both cases, as it can only lead to even larger values of *hashed length* for the document signature.
- *hashed length* is encoded in 8 bytes without a modular reduction. Accordingly, it is not possible to use the modular reduction to reach equality of the values of *hashed length* for both types of signatures.

A depiction of the attempted aliasing in the case of v5 signatures can be found in Appendix A.

Note that LibrePGP also introduces a signature subpacket for v4 signatures that causes the Literal Data Packet's header to be hashed.[15]

4.6 UTF-8 Recoding in RFC 9580

RFC 4880 specifies in Section 5.9 for the format specifier field of Literal Data Packets: "*If it is a 't' (0x74), then it contains text data, and thus may need line ends converted to local form, or other text-mode changes. The tag 'u' (0x75) means the same as 't', but also indicates that implementation believes that the Literal Data contains UTF-8 text.* "

RFC 9580 deprecates the creation of such packets using the format "t" (SHOULD NOT), and defines also that when receiving a message with this format specifier be interpreted also as indicating UTF-8 encoding of the text.

Section 5.9 in RFC 9580 is updated to:

> *Older versions of OpenPGP used t (0x74) to indicate textual data, but did not specify the character encoding. Implementations SHOULD NOT emit this value. An implementation that receives a Literal Data packet with this value in the format field SHOULD interpret the packet data as UTF-8 encoded text, unless reliable (not attacker-controlled) context indicates a specific alternate text encoding. This mode is deprecated due to its ambiguity.*

[14] https://www.ietf.org/archive/id/draft-koch-librepgp-01.html#section-5.2.4-10
[15] https://www.ietf.org/archive/id/draft-koch-librepgp-01.html#section-5.2.3.33

Furthermore, RFC 9580 specifies the recoding of the text documents before the digest computation: "*For text document signatures (type ID 0x01), the implementation MUST first canonicalize the document by converting line endings to <CR><LF> and encoding it in UTF-8 (see [RFC 3629]). The resulting UTF-8 byte stream is hashed.*"

Formally, the UTF-8 reencoding that is applied here is a mapping

$$\text{UTF-8-reencoding} : \{\text{input-encodings}\} \times \{\text{wire-messages}\} \mapsto \{\text{messages-for-verification}\}$$

In principle, both reencodings, the line-ending conversion and the UTF-8-reencoding, that happen before the signature hash computation, are potentially non-injective mappings, i.e., multiple elements from the input space may be mapped to a single element of the output space. A non-injective mapping allows the manipulation of a message without invalidating the signature and thus formally leads to a EUF-CMA[16] violation. This formal security gap is, however, mitigated if it does not lead to an ambiguity in the display of a validly signed message to the user.

For the previously existing line ending conversion, it is not apparent how this can lead to semantic differences in two messages. For the newly introduced UTF-8 reencoding, since it is not clear how a client determines the original message encoding, it is not ultimately possible to judge whether or not any such ambiguities in the message representation may occur. In case all relevant clients display the message in the same encoding as is used as the input-encoding to the UTF-8 reencoding, no security problem can arise. However, RFC 9580 does not state whether and under what circumstances the reencoding operation is safe from a security point of view or provides any security analysis thereof.

Note that further, it cannot be excluded that the formatting octet that is part of the header of LIT packets may influence the decision about whether and using which input-encoding the UTF-8-reencoding is applied. The formatting octet, however, as it is not included in the signature message digest of v4 or v6 signatures, is subject to manipulation by an attacker. Also for this aspect, RFC 9580 provides no security analysis.

[16] Existential unforgeability under chosen message attack. EUF-CMA is a standard security notion that is required from a secure signature scheme.

Chapter 5

Analogous Security Features of RFC 9580 and LibrePGP

5.1 Key Overwrite Attacks

In [BHP22], so called "key overwrite" attacks are devised that allow an attacker to recover private keys if he can

- manipulate the public key material that the victim has stored locally and that will be used in the decryption procedure along with the private key material,
- and in the case of encryption keys:
  - can trigger the decryption by the victim of ciphertexts crafted by him
  - and receives information about whether the decryption operation was successful or not.
- in the case of signature keys:
  - receive signatures generated using the manipulated public-key material.

One example of such attacks given in the paper is the manipulation of the modulus n in the RSA decryption without the use of CRT. Another example is the manipulation of the algorithm ID that is part of the public key material and thus trick the victim's implementation into using a private key of one public-key scheme for the secret operation of a different scheme.

The secret key protection methods specified in RFC 4880 do not include any integrity protection of the public key and thus implementations based on this standard are potentially vulnerable to such attacks. Both RFC 9580 and LibrePGP solve this problem by specifying an AEAD variant for the secret key encryption that also includes the public-key material in the additional data as a parameter in the AEAD encryption.

Chapter 6

Bibliography

[BHP22] L. Bruseghini, D. Huigens, and K. G. Paterson. Victory by ko: Attacking openpgp using key overwriting. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 411–423, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3548606.3559363.

[CryStackExch] 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.

[Efail] Poddebniak, D. et al. Efail: Breaking S/MIME and OpenPGP Email Encryption using Exfiltration Channels. https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-poddebniak.pdf.

[LPGP01] W. Koch and R. Tse. LibrePGP Message Format – Version 01, May 2024. https://www.ietf.org/archive/id/draft-koch-librepgp-01.html.

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

[Mat24] J. P. Mattsson. Collision-Based Attacks on Block Cipher Modes - Exploiting Collisions and Their Absence. Cryptology ePrint Archive, Paper 2024/1111, 2024. URL https://eprint.iacr.org/2024/1111.

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

[RFC4880] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, R. Thayer . RFC 4880: OpenPGP Message Format , 2007.

[RFC9580] E. Wouters, P, D. Huigens, J. Winter, and Y. Niibe. RFC 9580 – OpenPGP, July 2024. https://www.rfc-editor.org/rfc/rfc9580.html.

[SR24] F. Strenzke and J. Roth. Legacy Encryption Downgrade Attacks against LibrePGP and CMS. Cryptology ePrint Archive, Paper 2024/1110, 2024. URL https://eprint.iacr.org/2024/1110. https://eprint.iacr.org/2024/1110.

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

Appendix A

Figures to the Impossible v5 Document-Key-Signature Aliasing

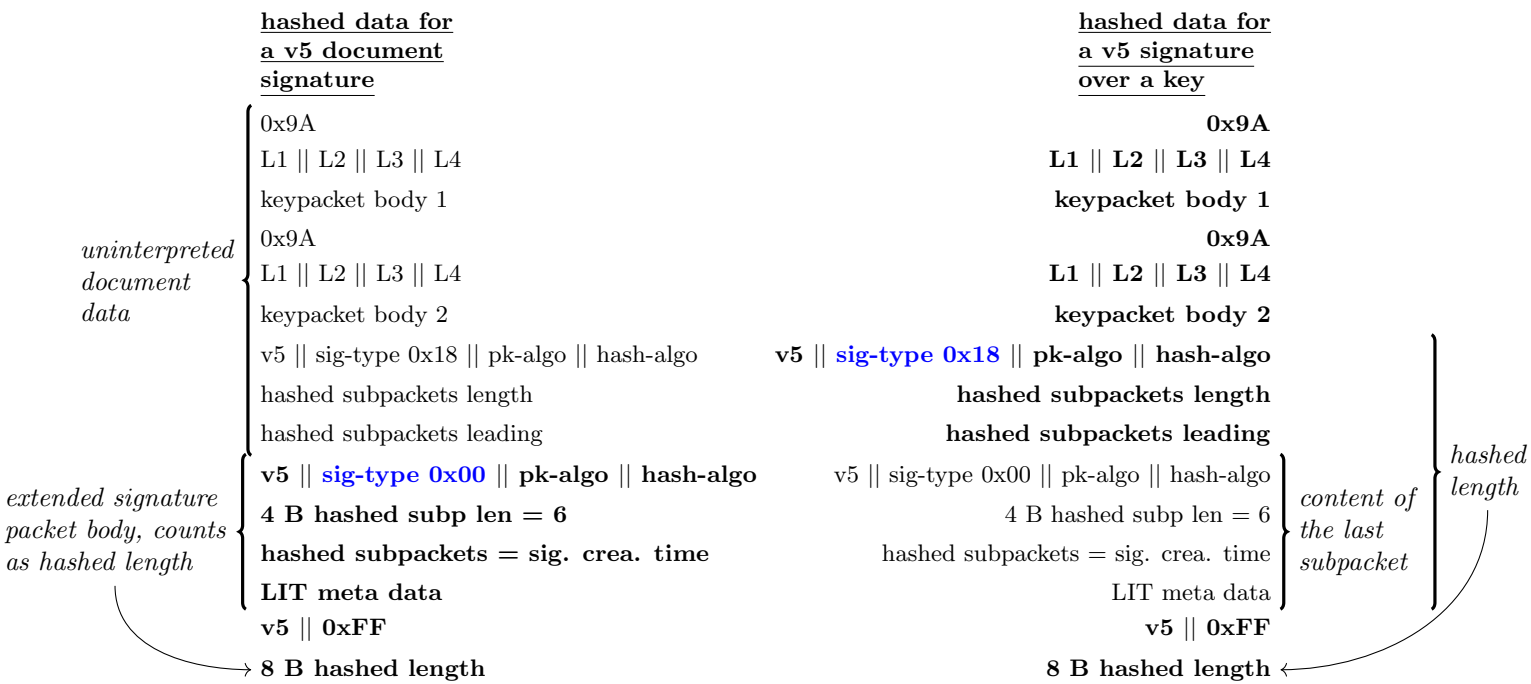

Figure A.1: Depiction of the attempted reinterpretation of a document signature as a signature over a key in the case of LibrePGP's v5 signatures. Reaching equality between the values of *hashed length* for both signature types is impossible, as increasing *hashed length* on either side cannot compensate the existing difference.

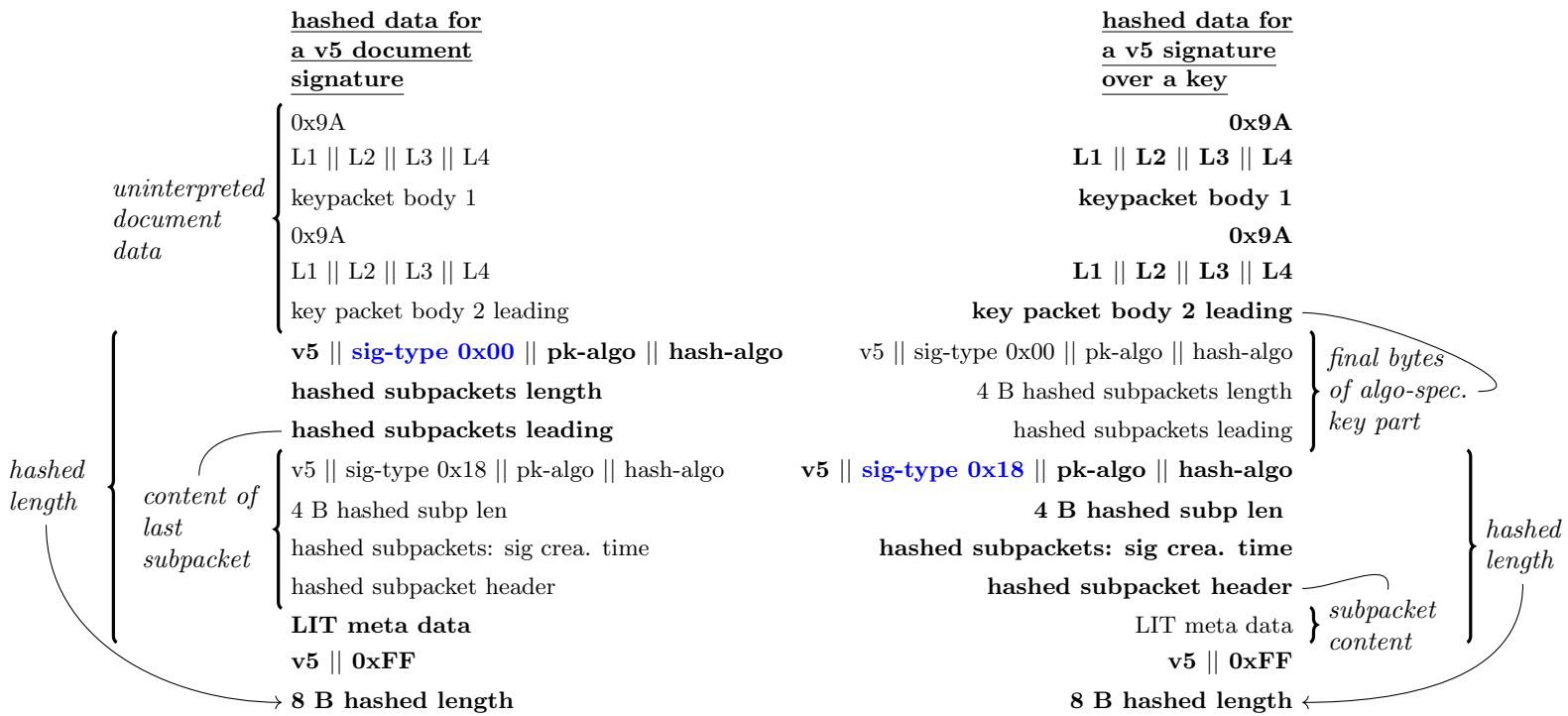

Figure A.2: Depiction of the principle of an inverse document-to-key-type attack for v5 which turns out to be impossible. The *hashed length* on both sides cannot be brought to equality since the hashed length for the document signature will always be larger than that of the signature over a key. The alternative attempt to place the trailer of the signature over a key (starting with "v5") into the "LIT meta data" is also not helpful as it only further increases the difference in *hashed length* on both sides and furthermore places further content into the algorithm-specific key part.