openpgp
Internet-Draft
Updates: 9580 (if approved)
Intended status: Standards Track
Expires: 7 May 2026

A. Gallagher, Ed.
PGPKeys.EU
D. K. Gillmor
ACLU

3 November 2025

OpenPGP Signatures and Signed Messages
draft-gallagher-openpgp-signatures-02

Abstract

This document specifies several updates and clarifications to the OpenPGP signature and message format specifications.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://andrewgdotcom.gitlab.io/openpgp-signatures. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-gallagher-openpgp-signatures/.

Discussion of this document takes place on the OpenPGP Working Group mailing list (mailto:openpgp@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/openpgp/. Subscribe at https://www.ietf.org/mailman/listinfo/openpgp/.

Source for this draft and an issue tracker can be found at https://gitlab.com/andrewgdotcom/openpgp-signatures.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 7 May 2026.

Table of Contents

1. Introduction

OpenPGP signatures have a rich vocabulary, however this is often under-specified. This document attempts to address this by:

* Expanding on specifications where [RFC9580] does not fully describe the existing or expected behaviour of deployed implementations.

* Adding clarification where deployed implementations differ in their interpretation of [RFC9580] and its predecessors.

* Deprecating unused or error-prone features.

* Constraining the formal message grammar.

This document does not specify any new wire formats.

2. Conventions and Definitions

The term "OpenPGP Certificate" is used in this document interchangeably with "OpenPGP Transferable Public Key", as defined in Section 10.1 of [RFC9580].

The term "Component key" is used in this document to mean either a primary key or subkey.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Signature Types

Several signature types are specified in incomplete, confusing or contradictory ways. We update their specifications as follows.

3.1. Certification Signature Types (0x10..0x13)

Section 5.2.1 of [RFC9580] defines four types of certification signature (0x10..0x13). All may be created by either the key owner or a third party, and may be calculated over either a User ID packet or a User Attribute packet. In addition, a Certification Revocation signature revokes signatures of all four types.

Historically, certifications were only made by third parties. First-party self-certifications only became customary later, and were made mandatory when preference subpackets were introduced.

3.1.1. Generic, Casual and Positive Certifications (0x10, 0x12, 0x13)

The semantic distinctions between the certification signature types are ill-defined. Since no definition of the phrase "some casual verification" (Section 5.2.1.6 of [RFC9580]) was ever issued, there is no consensus on the semantics of a Casual Certification or how it differs in practice from the other certification types.

The following convention has evolved over time [ASKCERTLEVEL], and is hereby specified:

* 0x10 Generic Certification SHOULD only be used for third-party certifications.

* 0x12 Casual Certification is deprecated and SHOULD NOT be created.

* 0x13 Positive Certification SHOULD only be used for self-certifications.

A receiving implementation MUST treat a third-party certification of any of the above types as equivalent to a type 0x10 signature, and a first-party certification of any of the above types as equivalent to a type 0x13 signature.

3.1.2. Persona Certifications (0x11)

0x11 Persona Certification signatures are an exceptional case, because by default many implementations do not consider them when calculating trust values. This follows from Section 5.2.1.5 of [RFC9580]:

> The issuer of this certification has not done any verification of the claim that the owner of this key is the User ID specified.

A receiving implementation therefore MUST NOT attribute any trust statement to the presence of a Persona Certification. In addition, since an unverified self-certification is both a meaningless and reckless statement ("I have not checked whether this is my own identity"), a generating implementation MUST NOT generate Persona self-certifications, and a receiving implementation MUST ignore them.

Although a Persona Certification has no intrinsic semantic value, the semantics of signatures may be altered by adding subpackets such as notations. A generating implementation MAY use a third-party Persona Certification to make a verifiable statement about a User ID (for example, by adding a notation) without making any trust statement about the relationship between the User ID and the primary key.

3.2. Primary Key Binding Signature (Type 0x19)

Section 5.2.1.9 of [RFC9580] defines the Primary Key Binding Signature as:

> This signature is a statement by a signing subkey, indicating that it is owned by the primary key.

Section 10.1.5 of [RFC9580] gives additional details:

> For subkeys that can issue signatures, the Subkey Binding signature MUST contain an Embedded Signature subpacket with a Primary Key Binding signature (Type ID 0x19) issued by the subkey on the top-level key.

The motivation for this requirement is not contained in any of the RFCs, and the terms "signing subkey" and "subkeys that can issue signatures" are imprecise. We hereby address these omissions:

> An attacker could issue a Subkey Binding signature over a public subkey that belongs to a victim, and publish it as part of the attacker's own certificate. A third party might then look up the subkey using the Issuer Key ID or Issuer Fingerprint subpacket from a signature made by the victim, and find the attacker's certificate instead. The attacker could then use this to impersonate the victim to the third party. The Primary Key Binding signature mitigates this attack, by requiring the subkey's owner to consent for it to be bound to a particular primary key.
>
> A Primary Key Binding signature is REQUIRED in any Subkey Binding signature that contains one or more Key Flags whose specification requires one. A receiving implementation MUST reject any Subkey Binding signature that contains one or more of these Key Flags and does not contain a valid Subkey Binding signature. A Primary Key Binding signature is OPTIONAL otherwise.

Initially, the only Key Flags for which a Primary Key Binding signature is REQUIRED are 0x02 (Literal Data Signature Category), 0x0008 (Timestamping Category) and ((TBC)) (Countersignature Category) (Section 11.2).

3.3. Primary Key Revocation Signature (Type 0x20)

Section 5.2.1.11 of [RFC9580] defines the Key Revocation Signature as:

> This signature is calculated directly on the key being revoked. A revoked key is not to be used. Only Revocation Signatures by the key being revoked, or by a (deprecated) Revocation Key, should be considered valid Revocation Signatures.

The name and description are potentially confusing, as it can only revoke a Primary Key and not a Subkey -- other OpenPGP artifacts that are named "Key" without a qualifier (such as the "Key Flags" and "Key Expiration Time" subpackets) apply to both Primary Keys and Subkeys.

We therefore rename the 0x20 signature type to "_Primary_ Key Revocation Signature" for clarity, and update its definition as follows:

> This signature is calculated directly on the primary key being revoked. A revoked primary key is not to be used. Only Revocation Signatures by the primary key being revoked, or by a (deprecated) Revocation Key, should be considered valid Primary Key Revocation Signatures.

3.4. Subkey Revocation Signature (Type 0x28)

Section 5.2.1.11 of [RFC9580] defines the Subkey Revocation Signature as:

> This signature is calculated directly on the primary key and the subkey being revoked. A revoked subkey is not to be used. Only Revocation Signatures by the top-level signature key that is bound to this subkey, or by a (deprecated) Revocation Key, should be considered valid Revocation Signatures.

The phrasing "top-level signature key that is bound to this subkey" is confusing. Instead, we update the definition for clarity:

> This signature is calculated directly on the primary key and the subkey being revoked. A revoked subkey is not to be used. Only Revocation Signatures by the primary key, or by a (deprecated) Revocation Key, should be considered valid Subkey Revocation Signatures.

There are several other places in [RFC9580] that use the term "top-level key" instead of "Primary Key", but this is not explicitly defined. It MUST be interpreted as a synonym for "Primary Key" in all these contexts.

3.5. Certification Revocation Signature (Type 0x30)

Section 5.2.1.13 of [RFC9580] defines the Certification Revocation Signature as:

> This signature revokes an earlier User ID certification signature (Type IDs 0x10 through 0x13) or Direct Key signature (Type ID 0x1F). It should be issued by the same key that issued the revoked signature or by a (deprecated) Revocation Key. The signature is computed over the same data as the certification that it revokes, and it should have a later creation date than that certification.

Section 5.2.4 of [RFC9580] is clear that Direct Key signatures and Certification Signatures have completely different constructions. This implies that there are two different ways to construct a Type 0x30 signature, each of which appears in a different part of an OpenPGP certificate.

The above definition dates back to [RFC2440], except for the "or Direct Key Signature" clause which was added to the first sentence in [RFC4880]. But the third sentence still defines the construction unconditionally by reference to "the certification that it revokes", even though it does not necessarily revoke a certification.

The use of a Certification Revocation Signature to revoke a Direct Key Signature is imprecise and not widely supported, and is hereby deprecated. Since Direct Key Signatures have no intrinsic semantics, the ability to revoke a Direct Key Signature is not necessary. To retract a previous statement made by a Direct Key Signature, it is sufficient to create a new Direct Key Signature with a different set of subpackets.

(See also Section 6.1)

3.5.1. Distribution of Certification Revocations

Distribution of revocations has historically been unreliable. In particular, a keystore that enforces self-sovereignty (Section 8 of [I-D.dkg-openpgp-abuse-resistant-keystore]) cannot be relied upon to distribute third-party certification revocations. In addition, it is not normally possible to distribute a self-certification revocation over a User ID without also distributing the contents of that User ID. This is a significant impediment to reliable implementation of self-sovereign User ID redaction.

One possible solution is to allow the use of Embedded Signatures in User Attributes (Section 3.2.1 of [I-D.gallagher-openpgp-user-attributes]).

3.6. Timestamp Signature (0x40)

Section 6.2.1 of [RFC1991] defined the Timestamp signature as:

> <40> - time stamping ("I saw this document")
>
> Type <40> is intended to be a signature of a signature, as a notary seal on a signed document.

The second statement implies that a v3 0x40 sig is made over a signature packet. But the first statement implies a signature over a document, just with different semantics.

By Section 5.2.1.14 of [RFC9580], this has changed to:

> 0x40: Timestamp signature. This signature is only meaningful for the timestamp contained in it.

This avoids the apparent contradiction of [RFC1991], but is less informative. And there is no explicit construction given in Section 5.2.4 of [RFC9580].

We note also that [RFC9580] introduced a Key Flag for timestamping. This indicates that timestamping documents is sufficiently different from signing them that separate keys should be used. This is consistent with the idea that "I wrote this document" and "I saw this document" are distinct statements with different consequences. This is crucial in the case of an automated timestamping service that makes no claims about the accuracy of document contents.

We define type 0x40 Timestamp signatures as follows:

> A type 0x40 Timestamp signature is made over a Literal Data Packet, and is constructed the same way as a type 0x00 Binary Document Signature. If the message is a text document, it MUST already be in Canonical Text form. By default a Timestamp signature conveys no opinion about the validity of the document; it only claims that the document existed at the timestamp of signature creation. This interpretation MAY be modified by adding notation subpackets, the meaning of which are application-dependent. It can be made over an otherwise unsigned document, or it can be one of many signatures over the same document. The Cleartext Signature Framework MUST NOT be used with Timestamp signatures.

Countersigning a Signature packet only (including blind countersigning) is done using the type 0x50 Third-Party Confirmation signature.

3.7. Third-Party Confirmation Signature (0x50)

Section 5.2.1.15 of [RFC9580] defines a Third-Party Confirmation signature as:

> This signature is a signature over some other OpenPGP Signature packet(s). It is analogous to a notary seal on the signed data. A Third-Party Confirmation signature SHOULD include a Signature Target subpacket that identifies the confirmed signature.

A concrete construction is provided, but the placement and semantics are still not well-defined. We clarify these as follows:

> By default, a Third-Party Confirmation signature makes no claim about the validity of the other signature, just its existence, and makes no claim whatsoever about the subject of that signature. This interpretation MAY be modified by adding notation subpackets, the meaning of which are application-dependent. It MAY be included in an Embedded Signature packet in the unhashed area of the signature it notarizes. Otherwise, it SHOULD be distributed as a detached signature. A Signature Target subpacket SHOULD NOT be included.

3.7.1. Terminology Subtleties

Implementers should note that "Third-Party Confirmation" signatures (type 0x50) are distinct from "third-party Certification" signatures (types 0x10..0x13 when issued by a primary key other than the one signed over), and beware that older RFCs do not always use sufficiently precise terminology to distinguish them.

3.7.2. Deprecation of the Signature Target Subpacket

The Signature Target subpacket (Section 5.2.3.33 of [RFC9580]) fulfils the following roles:

* In a Timestamp or Third-Party Confirmation signature, it identifies the signature that is being countersigned

* In a Revocation signature, it identifies the signature being revoked

The Signature Target subpacket is vaguely defined however:

> (1 octet public key algorithm, 1 octet hash algorithm, N octets hash)
>
> This subpacket identifies a specific target signature to which a signature refers. For Revocation Signatures, this subpacket provides explicit designation of which signature is being revoked. For a Third-Party Confirmation or Timestamp signature, this designates what signature is signed. All arguments are an identifier of that target signature.

> The N octets of hash data MUST be the size of the signature's hash. For example, a target signature with a SHA-1 hash MUST have 20 octets of hash data.

It is unclear whether the hash field refers to the digest that the target signature is made over, or a digest over the resulting signature packet. The definition of a Third-Party Confirmation signature in Section 5.2.1 of [RFC4880] gives us a hint however:

> A third-party signature SHOULD include Signature Target subpacket(s) to give easy identification. Note that we really do mean SHOULD. There are plausible uses for this (such as a blind party that only sees the signature, not the key or source document) that cannot include a target subpacket.

(Beware that "third-party signature" in the above should be read as "Third-Party Confirmation signature"; see Section 3.7.1.)

The only way that a blind party would be unable to generate a Signature Target subpacket is if the hash is the digest that the original signature was made over. But if so it is not a unique identifier of a signature packet, since multiple distinct signatures can be made over the exact same material, including subpackets. In particular, if there was no Issuer Key ID or Issuer Fingerprint subpacket in the target signature's hashed area, a Signature Target subpacket could not distinguish between the original signature or an otherwise valid one issued by a completely different signing key.

The Signature Target subpacket is therefore not functional when used in a Third-Party Confirmation signature. A more reliable mechanism for identifying the target of a Third-Party Confirmation signature is to include it an Embedded Signature subpacket, directly in the unhashed area of the signature being countersigned.

The other specified use for the Signature Target subpacket is in a revocation signature. Certification Revocations are customarily understood to mean "I retract all my previous statements that this key is related to this user" (Section 6.2.1 of [RFC1991]), so a Certification Revocation is not specific to any particular Certification signature. No other signature types can be revoked - primary key and subkey revocation signatures revoke the key, not the previous binding signature(s) (Section 8), and so are not specific to any particular binding signatures either.

The Signature Target subpacket is therefore not functional when used in a revocation signature. An alternative mechanism is to allow the inclusion of Intended Recipient subpackets in signatures over key material (Section 3.2.1 of [I-D.gallagher-openpgp-user-attributes]).

Timestamp signatures as specified in Section 3.6 do not require a Signature Target subpacket, since the signed message grammar identifies the material being signed over.

We therefore deprecate the Signature Target subpacket in all contexts.

3.7.3. Use of Third-Party Confirmation Signatures by Applications

We may wish to allow the application layer to make validity claims using countersignatures. For example, a key server may wish to record that it has verified a User ID by automated means. The key server may not wish to make a Certification signature, to prevent the cumulation of many such automated signatures (Section 9.5). For the same reason, it may not wish to embed its countersignature in an unhashed area of a signature packet in the certificate.

It could make a Third-Party Confirmation signature over the most recent self-certification, and distribute it as a detached signature, perhaps in a certificate bundle (Section 8.1 of [I-D.gallagher-openpgp-hkp]).

((TBC))

4. Message Grammar

The accepted convention is that a prefixed Signature packet signs over the next literal packet only, skipping any intervening signatures - however this is not explicitly specified in [RFC9580]. Historically, PGP 2.X treated a prefixed Signature packet as applying to the entire following sequence of packets, but this usage is deprecated [FINNEY1998]. See Section 4.3 for an alternative construction.

In addition, One-Pass Signature (OPS) nesting semantics are complex, and under-specified [SCHAUB2022]. Section 5.4 of [RFC9580] defines the nesting octet as:

> A 1-octet number holding a flag showing whether the signature is nested. A zero value indicates that the next packet is another One-Pass Signature packet that describes another signature to be applied to the same message data.

The terminology is imprecise, and non-zero "nesting" flags are completely unspecified. One self-consistent interpretation is as follows:

* A zero nesting octet means that the following OPS and its counterpart signature are not signed over by the current OPS.
  - This process is recursive if multiple sequential OPS packets have a nesting octet of zero.
* To add multiple OPS signatures over the same message data, all OPS constructions except the innermost one have the nesting octet zeroed.
  - It is not clear what happens if the innermost nesting octet is zero but no OPS packet follows.

The above implies that an OPS with a nonzero nesting octet signs over all packets between the OPS packet and its matching signature packet, including any further signatures, however it is not clear whether any current implementation supports this.

This is further expanded in [OPENPGPDEVBOOK].

This still leaves us with an overly complex grammar that resists rigorous formalization. We attempt to improve the formalism below.

4.1. OPS Message Constraints

We constrain OPS structures to a subset of previously-allowed configurations:

* A prefixed Signature packet signs over the next literal or compressed packet, ignoring any intervening signature or OPS packets.
* Prefixed signatures and OPS signatures MUST NOT both be used in the same message.
* When generating an OPS packet that is not followed by another OPS packet, the nesting octet SHOULD be set to 1.
  - Otherwise, the nesting octet SHOULD be set to 0.
* When consuming an OPS packet, the nesting octet MUST be ignored.

This effectively deprecates the nesting octet, while maintaining backwards compatibility with legacy code.

4.2. Subject Normalization

The _subject_ of an OpenPGP signature refers to the packet(s) that are signed over. The _type-specific data_ of an OpenPGP signature refers to the section of the data stream that is passed to the signature's digest function after the optional salt and before the trailer. The type-specific data differs from the subject in that it has been normalized, the details of which are dependent on the signature type.

The subject of a signature in the Literal Data category (Section 6) is the Literal Data packet that immediately follows one or more prefixed signatures, or is enclosed by one or more OPS constructions. If no Literal Data packet is present, the signature is malformed.

The following normalization steps are applied to the subject of the signature to produce the type-specific data:

* The framing of the Literal Data packet is discarded, and any partial-length packets are concatenated.

* If the Signature Type is 0x01, the Literal Data packet body is converted to Canonical Text, by converting line endings to CRLF and removing any trailing whitespace (Section 4.2.1).

A One-Pass Signature over a Literal Data packet, a prefixed Signature over the same packet, and a detached signature over a file containing the body of the same packet are all calculated the same way. This means that they can be losslessly transformed into each other with the exception of the Literal Data metadata fields, which an application MAY assume contain their recommended default values as per Section 5.9 of [RFC9580].

A signature of Type 0x01 MUST NOT be made over arbitrary binary data, only over UTF-8 text.

4.2.1. Line Ending Normalization

When normalizing line endings, only bare linefeeds (an LF control character that is not preceded by a CR) are normalized to CRLF. In particular, bare carriage returns MUST NOT be converted to CRLF.

4.3. Nested Signatures

To sign over an entire signed message together with its signatures, the wire format of the inner message SHOULD first be encapsulated in a Literal Data packet. A Canonical Text signature MUST NOT be made over such a nested message, and the Cleartext Signature Framework MUST NOT be used.

Beware that the outer signature will thus be sensitive to the inner message's packet framing, i.e. the otherwise inconsequential choice of packet header format and partial body lengths. If the inner message is parsed and re-serialized unmodified, but using a different framing, the outer signature will no longer validate.

4.4. Formal Grammar

The message grammar in Section 10.3 of [RFC9580] is therefore updated to:

* Literal Message:
  Literal Data Packet.

* Encrypted Session Key:
  Public Key Encrypted Session Key Packet | Symmetric Key Encrypted Session Key Packet.

* Encrypted Data:
  Symmetrically Encrypted Data Packet | Symmetrically Encrypted and Integrity Protected Data Packet.

* Encrypted Message:
  Encrypted Data | Encrypted Session Key, Encrypted Message.

* Prefixed Signed Message: Signature Packet, Prefixed Signed Message | Literal Message.

* Multiply One-Pass Signed Message:
  One-Pass Signature Packet (with nesting octet 0), One-Pass Signed Message, Corresponding Signature Packet.

* Singly One-Pass Signed Message:
  One-Pass Signature Packet (with nesting octet 1), Literal Message, Corresponding Signature Packet.

* One-Pass Signed Message:
  Multiply One-Pass Signed Message | Singly One-Pass Signed Message.

* Signed Message:
  Prefixed Signed Message | One-Pass Signed Message.

* Optionally Signed Message:
  Signed Message | Literal Message.

* Compressed Message:
  Compressed Data Packet.

* Unencrypted Message:
  Compressed Message | Optionally Signed Message.

* Optionally Padded Unencrypted Message:
  Unencrypted Message | Unencrypted Message, Padding Packet.

* OpenPGP Message:
  Encrypted Message | Unencrypted Message.

In addition to these rules, a Marker packet (Section 5.8 of [RFC9580]) can appear anywhere in the sequence.

4.5. Unwrapping Encrypted and Compressed Messages

[RFC9580] permits an encrypted message to contain another encrypted message, and a compressed message to contain another compressed message, possibly recursively. Such messages require potentially unbounded resources for negligible added utility, and therefore MUST NOT be created.

In addition, encrypt-then-sign messages are not idiomatic OpenPGP, and MUST NOT be generated.

The guidance in Section 10.3.1 of [RFC9580] is therefore updated to:

* Decrypting a version 2 Symmetrically Encrypted and Integrity Protected Data packet MUST yield a valid Optionally Padded Decrypted Message.

* Decrypting a version 1 Symmetrically Encrypted and Integrity Protected Data packet or -- for historic data -- a Symmetrically Encrypted Data packet MUST yield a valid Decrypted Message.

* Decompressing a Compressed Data packet MUST yield a valid Optionally Signed Message.

4.6. Marker Packet

Section 5.8 of [RFC9580] defines the Marker Packet as follows:

> The body of the Marker packet consists of:
>
> - The three octets 0x50, 0x47, 0x50 (which spell "PGP" in UTF-8).
>
> Such a packet MUST be ignored when received.

We update this to include:

* If a receiving implementation encounters a Marker Packet with any other contents, the entire packet sequence SHOULD be rejected.

* A Marker Packet MAY be added by an application to notify non-OpenPGP software that a data stream contains OpenPGP data. If so, the Marker Packet SHOULD be the first packet in the sequence, and SHOULD NOT use a Legacy header, so that it can be easily detected.

5. Signature Packets

Receiving implementations currently have insufficient guidance for when to reject non-idiomatic signature packets.

5.1. Recursive Embedding Inside Signature Subpackets

Section 5.2.3 of [RFC9580] specifies two subpackets which could recursively include a signature inside a signature:

* Embedded Signature (type 32): contains a signature packet

* Key Block (type 38, experimental): contains an entire certificate, which may itself include signature packets

In order to prevent excessive recursion via nested signature subpackets:

* Signatures contained within Embedded Signature subpackets MUST NOT contain any Embedded Signature subpackets:

  - An Embedded Signature subpacket MUST contain a signature of an Embeddable signature type.

  - An Embeddable signature MUST NOT contain Embedded Signature subpackets.

- Initially, only the Primary Key Binding and Third-Party Confirmation signature types are specified as Embeddable.

* A Key Block subpacket MUST only be used inside a signature type in the Literal Data Signature Category.

A receiving implementation MUST invalidate any signature that does not conform to the above guidance.

5.2. Subpackets with Conflicting Information

Section 5.2.3.9 of [RFC9580] gives a receiving implementation significant leeway in interpreting conflicting combinations of subpackets:

> It is certainly possible for a signature to contain conflicting information in subpackets. For example, a signature may contain multiple copies of a preference or multiple expiration times. In most cases, an implementation SHOULD use the last subpacket in the hashed section of the signature, but it MAY use any conflict resolution scheme that makes more sense.

We hereby tighten this guidance:

> A signature MUST NOT contain more than one subpacket of any given type in its hashed subpackets area, unless otherwise specified. A receiving implementation MUST invalidate a signature that contains in its hashed area more than one subpacket of any type for which this is not explicitly permitted.

Multiple copies of the following subpacket types are already explicitly permitted:

* Issuer Key ID Section 5.2.3.9 of [RFC9580]

* Notation Data Section 5.2.3.24 of [RFC9580]

* Intended Recipient Fingerprint Section 5.2.3.36 of [RFC9580]

In addition, the following interpretations are natural extensions of specified behaviour, and hereby permitted:

5.2.1. Multiple Revocation Key Subpackets

If multiple Revocation Key subpackets are present, any of the listed keys MAY generate key revocation signatures.

5.2.2. Multiple Preferred Keyserver Subpackets

If multiple Preferred Keyserver subpackets are present, updates MAY be obtained from any of the listed keyservers.

5.2.3. Multiple Embedded Signature Subpackets

If multiple Embedded Signature subpackets are present, a receiving implementation SHOULD attempt to process them all in turn.

5.2.4. Multiple Key Block Subpackets

If multiple Key Block subpackets are present, a receiving implementation SHOULD attempt to process them all in turn.

5.2.5. Multiple Issuer Fingerprint Subpackets

Multiple Issuer Fingerprint subpackets are permitted, with the same interpretation as multiple Issuer Key ID subpackets.

Note however that Section 5.2.3.35 of [RFC9580] states of the Issuer Fingerprint subpacket:

> If the version of the issuing key is 4 and an Issuer Key ID subpacket (Section 5.2.3.12) is also included in the signature, the Key ID of the Issuer Key ID subpacket MUST match the low 64 bits of the fingerprint.

Generalizing to multiple subpackets, we replace this with:

> If both Issuer Key ID and Issuer Fingerprint subpackets are included in a signature then each Issuer Key ID subpacket MUST match the low 64 bits of only one v4 Issuer Fingerprint subpacket, and all v4 Issuer Fingerprint subpackets MUST have a corresponding Key ID subpacket.

6. Signature Categories

Signature Type code points are spaced out into identifiable ranges of types with similar semantics. These also mostly correspond to the various Key Flags. These ranges and their mapping to the Key Flags are not specified in [RFC9580].

We define Signature Categories to cover each range of type values:

* Literal Data Signature Category (0x00..0x07)
  - 0x00 Signature over a Binary document

    - 0x01 Signature over a Canonical Text document
    - 0x02 Standalone signature (null document)
  * Unassigned (0x08..0x0F)
  * Certification Category (0x10..0x17)
    - 0x10 Generic certification
    - 0x11 Persona certification
    - 0x12 Casual certification
    - 0x13 Positive certification
    - (0x16 Approved certifications)
  * Key Binding Category (0x18..0x1F)
    - 0x18 Subkey bind
    - 0x19 Primary key bind
    - 0x1F Direct key (self bind)
  * Primary Key Revocation Category (0x20..0x27)
    - 0x20 Primary Key revocation
  * Subkey Revocation Category (0x28..0x2F)
    - 0x28 Subkey revocation
  * Certification Revocation Category (0x30..0x37)
    - 0x30 Certification revocation
  * Unassigned (0x38..0x3F)
  * Timestamping Category (0x40..0x47)
    - 0x40 Timestamp
  * Unassigned (0x48..0x4F)
  * Countersignature Category (0x50..0x57)

    - 0x50 Third-Party confirmation

* Unassigned (0x58..0x5F)

* Private and Experimental Range (0x60..0x6F)

* Unassigned (0x70..0xFE)

* RESERVED (0xFF)

We have defined a Private and Experimental signature type range. This is 0x60..0x6F (96..111) for consistency with the existing private and experimental range in other registries. It does not form a Category and does not have a corresponding Key Flag.

Self-certifications over v4 Primary User IDs are used to convey the same information as Key Binding signatures. Therefore, unless specifically stated otherwise, any stipulations that apply to Key Binding signatures also apply to self-certifications over v4 Primary User IDs.

6.1. Key Flags

A Key Flags subpacket SHOULD be included in a Direct Key or Subkey Binding signature (or for v4 keys, a self-certification over the primary User ID). It applies only to a single key material packet; for a Direct Key signature (or primary User ID self-cert) it applies to the primary key only, and for a Subkey Binding signature, it applies only to that subkey.

Previously, it was also specified for use in third-party Certification Signatures. This is not widely supported and is hereby deprecated.

The following Key Flags permit the creation of signatures in one or more Signature Categories:

* 0x01.. Third-party signatures in the Certification and Certification Revocation Categories

* 0x02.. Literal Data Signature Category

* 0x0008.. Timestamping Category

* ((TBC)) Primary Key Revocation Category

* ((TBC)) Countersignature Category

The following exceptional usages are always permitted regardless of Key Flags:

* Primary keys are always permitted to make self-signatures in the Certification, Key Binding, Certification Revocation, Primary Key Revocation and Subkey Revocation Categories.

* Subkeys with signing-capable algorithms are always permitted to make Primary Key Binding signatures.

* Any key is permitted to make a signature in the Private and Experimental range.

Otherwise:

* A signature made by a key that does not have the corresponding Key Flag MUST be considered invalid.

* A key with no Key Flags subpacket MUST NOT create signatures.

Section 5.2.1.10 of [RFC9580] also explicitly allows keys with the 0x01 Key Flag to create third-party 0x1F Direct Key Signatures. These are used for trust delegation in [SQ-WOT].

6.2. Authentication Signatures

OpenPGP defines no authentication signature types, but does have an authentication Key Flag. Traditionally, authentication is performed by converting the key material into that of another protocol (usually OpenSSH) and performing authentication in that protocol.

Beware that cross-protocol usage can be exploited to evade the domain separation protections of Key Flags. For example, there is no distinction between document signing, certification and authentication usage in OpenSSH, and once converted an OpenPGP authentication key may be used as a OpenSSH CA or to sign git commits.

((TODO: Guidance for the use of authentication keys should be provided.))

7. Signature Subpacket Categories

Signature subpacket types may also be categorized, depending on where they are used:

7.1. General subpackets.

These may be attached to any signature type, and define properties of the signature itself. Some of these subpackets are self-verifying (SV), i.e. they contain hints to locate the issuing key that can be confirmed after the fact. It MAY be reasonable to place self-verifying general subpackets in the unhashed area. All other general subpackets MUST be placed in the hashed area.

Subpacket types: Signature Creation Time, Signature Expiration Time, Issuer Key ID (SV), Notation Data, Signer's User ID, Issuer Fingerprint (SV).

(Notation subpackets are categorized here as general subpackets, however the notations within them may have arbitrary semantics at the application layer)

7.2. Context subpackets.

These have semantics that are meaningful only when used in signatures of a particular type or category:

7.2.1. Direct subpackets.

These are normally only meaningful in a direct self-sig (or for v4 keys, a self-cert over the primary User ID) and define usage preferences for the certificate as a whole. They MAY be used in self-certs over other User IDs, in which case they define usage preferences for just that User ID (but this is not always meaningful or universally supported). They SHOULD NOT be used elsewhere. They MUST be placed in the hashed area.

A Direct subpacket MUST be ignored if it is in a self-cert made over a User ID by a v6 or later primary key.

Subpacket types: Preferred Symmetric Ciphers, Revocation Key (deprecated), Preferred Hash Algorithms, Preferred Compression Algorithms, Key Server Preferences, Preferred Key Server, Features, (Preferred AEAD Algorithms), Preferred AEAD Ciphersuites, Replacement Key.

The Replacement Key subpacket MAY also be used as a key revocation subpacket.

7.2.2. Revocation subpackets.

These are only meaningful in signatures of the Key Revocation, Subkey Revocation or Certificate Revocation categories. They SHOULD NOT be used elsewhere. They MUST be placed in the hashed area.

Subpacket types: Reason for Revocation, Replacement Key (Primary Key Revocations only).

7.2.3. Key Binding subpackets.

These are only meaningful in a signature of the Key Binding category (or for v4 keys, a self-cert over the primary User ID) and define properties of that particular component key. They SHOULD NOT be used elsewhere. They MUST be placed in the hashed area.

A Key Binding subpacket MUST be ignored if it is in a self-cert over a User ID that is not currently the primary User ID, or in a self-cert made over a User ID by a v6 or later primary key.

Subpacket types: Key Expiration Time, Key Flags.

7.2.4. First-party Certification subpackets.

These are only meaningful in a self-certification over a User ID, and define properties of that User ID. They SHOULD NOT be used elsewhere. They MUST be placed in the hashed area.

Subpacket types: Primary User ID

7.2.5. Third-party Certification subpackets.

These are only meaningful in third-party certification signatures and define properties of the Web of Trust. They SHOULD NOT be used elsewhere. They MUST be placed in the hashed area.

Subpacket types: Exportable Certification, Trust Signature, Regular Expression, Revocable, Policy URI, (Trust Alias).

7.2.6. Literal Data subpackets.

These are only meaningful in signatures of the Literal Data category, and define properties of the document or message. They SHOULD NOT be used elsewhere. Some of these subpackets are self-verifying (SV) and MAY be placed in the unhashed area. All other Literal Data subpackets MUST be placed in the hashed area. (Beware that the usefulness of all of these subpackets has been questioned)

Subpacket types: Intended Recipient Fingerprint, (Key Block (SV)), (Literal Data Metadata).

7.2.7. Attribute Value subpackets.

These are only meaningful in signature types whose specification explicitly requires them. They SHOULD NOT be used elsewhere. It MAY be reasonable to place Embedded Signature subpackets in the unhashed area. All other Attribute Value subpackets MUST be placed in the hashed area. They have no intrinsic semantics; all semantics are defined by the enclosing signature.

Subpacket types: Signature Target, Embedded Signature, (Delegated Revoker), (Approved Certifications).

7.3. Subpackets summary

Type	Name	Category	Critical	Unhashed	Context	Notes
0	Reserved	-				never used
1	Reserved	-				never used
2	Signature Creation Time	General	SHOULD			MUST always be present in hashed area
3	Signature Expiration Time	General	SHOULD			
4	Exportable Certification	Third-Party	MUST IFF false			boolean, default true
5	Trust Signature	Third-Party				
6	Regular Expression	Third-Party	SHOULD			
7	Revocable	Third-				boolean, default false (Se

```
ction 8.11)   |
   |     |(deprecated)   |Party     |        |        |       |
                 |
   +-----+---------------+----------+--------+--------+-------+-------------------------
--------------+
   |8    |Reserved       |-         |        |        |       |never used
                 |
   +-----+---------------+----------+--------+--------+-------+-------------------------
--------------+
   |9    |Key Expiration |Key       |SHOULD  |        |       |
                 |
   |     |Time           |Binding   |        |        |       |
                 |
   +-----+---------------+----------+--------+--------+-------+-------------------------
--------------+
   |10   |Placeholder for|-         |        |        |       |PGP.com proprietary featur
e                |
```

	backwards compatibility					
11	Preferred Symmetric Ciphers	Direct				
12	Revocation Key (deprecated)	Direct				
13-15	Reserved	-				never used
16	Issuer Key ID	General		MAY		issuer fingerprint is preferred
17-19	Reserved	-				never used
20	Notation Data	General				notations may be further classified
21	Preferred Hash Algorithms	Direct				
22	Preferred Compression Algorithms	Direct				
23	Key Server Preferences	Direct				
24	Preferred Key Server	Direct				

25	Primary User ID	First Party				boolean, default false
26	Policy URI	Third-Party				(effectively a human-readable notation)
27	Key Flags	Key Binding	SHOULD			
28	Signer’s User ID	General				
29	Reason for Revocation	Revocation				free text field is effectively a human-readable notation
30	Features	Direct				

31	Signature Target (deprecated)	Attr Value		0x50 3-p conf		Section 3.7.2
32	Embedded Signature	Attr Value		MAY	0x18 sbind	
33	Issuer Fingerprint	General		MAY		
34	Reserved	-				
35	Intended Recipient Fingerprint	Literal Data	SHOULD			
36	(Delegated Revoker)	Attr Value	MUST		TBD	[I-D.dkg-openpgp-revocation]
37	Reserved (Approved Certifications)	Attr Value			0x16 1pa3pc	[I-D.dkg-openpgp-1pa3pc]
38	Reserved (Key Block)	Literal Data		MAY		
39	Preferred AEAD Ciphersuites	Direct				
40	(Literal Data Metadata)	Literal Data				[I-D.gallagher-openpgp-literal-metadata]

41	(Trust Alias)	Third-Party				
TBD	Replacement Key	Direct, Revocation	SHOULD NOT			[I-D.ietf-openpgp-replacementkey]

Table 1: OpenPGP Signature Subpacket Types

Three subpacket types are Boolean, with different default values for when they are absent (two true, one false). It is RECOMMENDED that these subpackets not be used to convey their default values, only the non-default value. The default value SHOULD instead be conveyed by the absence of the subpacket.

Unless otherwise indicated, subpackets SHOULD NOT be marked critical. In particular, a critical subpacket that invalidates a self-signature will leave the previous self-signature (or no self-signature!) as the most recent valid self-signature from the PoV of some receiving implementations. A generating implementation MUST be sure that all receiving implementations will behave as intended if a signature containing a critical subpacket is invalidated. Otherwise, with the possible exception of Literal Data signatures, it is NOT RECOMMENDED to set the critical bit.

It is RECOMMENDED that a signature's creator places all subpackets in the hashed area, even self-verifying subpackets for which this is not strictly necessary. The unhashed area MAY be used for informational subpackets attached by third parties (which can be safely stripped).

7.4. Guidance for management of the Signature Subpacket Registry

* Future boolean subpackets SHOULD NOT contain an explicit value; a value of TRUE SHOULD be indicated by the presence of the subpacket, and FALSE otherwise.

* Specification of new subpackets SHOULD address classification, criticality and self-verification as outlined above.

* Subpackets SHOULD be implemented in the private/experimental area first, then reassigned to a permanent code point.

7.5. Unhashed Subpacket Deduplication

Unhashed subpacket areas are malleable and so may have subpackets added or removed in transit, either innocently or maliciously. A receiving implementation SHOULD clean the unhashed area of subpackets that are not meaningful or trustworthy outside the hashed area. If two signature packets are bitwise identical apart from differences in their unhashed subpacket areas, an implementation MAY merge them into a single signature. If two unhashed subpackets in the merged signature are bitwise identical, they MUST be deduplicated. Otherwise, the unhashed subpacket area of the merged signature SHOULD contain the useful subpackets from both original signatures, even if this means multiple subpackets of the same type.

8. Revoking Signatures and Keys

((TODO: merge parts of this section into Section 3))

There are three kinds of signatures that perform revocation: Key Revocation (0x20), Subkey Revocation (0x28), and Certification Revocation (0x30).

* A Key Revocation Signature (0x20) directly invalidates a Primary Key packet, and thereby (indirectly) revokes a full OpenPGP certificate (a.k.a. "Transferable Public Key").

* A Subkey Revocation Signature (0x28) directly revokes a Subkey packet, without affecting other key material attached to the same Primary Key.

* A Certification Revocation Signature (0x30) revokes:

  - all previous Certification Signatures (0x10..0x13) over the same primary key and User ID or User Attribute, or

  - all previous Direct Key Signatures (0x1f) over the same primary key, that were made by the same key that made the revocation.

All revocation types are permanent and cannot be un-revoked. A key may be temporarily invalidated by specifying a Key expiry date on a new Direct Key, Subkey Binding, or (for v4 keys) Primary User ID self-certification. A User ID or User Attribute may be temporarily invalidated by specifying a Signature expiry date on a self-certification. This expiry date can then be overridden on a later signature of the same type.

8.1. Revoking a Primary Key

A Key Revocation signature invalidates the Primary Key packet that it is made over. By implication, this revokes the entire certificate (Transferable Public Key) anchored by the Primary Key.

8.1.1. Key Retirement

A key owner may wish to retire a key, for example if it is using an older algorithm or it is no longer required. This can be achieved by making a Key Revocation Signature with a soft revocation reason (see Section 8.8) and publishing it directly.

8.1.2. Key Compromise

If a key owner loses control of their private key material, for example if their storage device is stolen or their computer is infected with malware, they will normally wish to invalidate their key. This can be achieved by making a Key Revocation Signature with a Reason for Revocation of "Key Compromise" (0x02) and publishing it directly.

If the key owner has also lost access to their private key material, for example if all copies of it were stolen, they cannot generate a new revocation and must follow the "Loss of Access" procedure in the next section.

8.1.3. Loss of Access

If a key owner loses access to their private key material, for example if they forget an encryption passphrase or a storage medium is destroyed, they will generally wish to invalidate their key. This can be achieved by publishing an escrowed Key Revocation Signature (see Section 8.10).

If the key owner does not have such a revocation stored safely, there is nothing further that they can do cryptographically. In such circumstances, they will need to inform their correspondents by other means. See Section 8.3 below for possible alternative methods in a controlled environment.

8.2. Revoking a Subkey

A Subkey packet may be revoked because its private key material has been compromised. It is possible for a Subkey to be compromised without the Primary Key being affected, for example if the private Subkey and Primary Key material are stored on separate devices. In such a case, it is not necessary for a Subkey Revocation Signature to be generated ahead of time and escrowed, since the Primary Key is still usable and can generate a revocation as required.

FIXME: are other kinds of subkey revocation meaningful, see draft-revocations #1 ABG

8.3. Revoking a Certification

User ID and User Attribute self-certifications and Direct Key self-signatures can be explicitly expired or replaced by the keyholder by issuing a superseding signature, so the only reason for a certification revocation is for third-party certifications.

Section 6.2.1 of [RFC1991] provided guidance for the interpretation of Certification Revocations as follows:

> <30> - public key packet and user ID packet, revocation ("I retract all my previous statements that this key is related to this user") (*)

While this language was not carried over into later specifications of the OpenPGP standard, neither was it explicitly contradicted.

When Alice revokes her third-party certification over Bob’s Primary Key and User ID, she is saying one of the following:

* Key is Compromised (0x02): "I believe that this key has been compromised"

* User ID No Longer Valid (0x32): "I no longer believe that this primary key should be associated with this identity"

Hard third-party certification revocations are useful in an environment where Alice is treated as an authority (say as a member of a corporate IT department) but does not have control over Bob’s key material or access to an escrowed revocation of Bob’s key.

FIXME: we need to specify what a receiving application should do when seeing an 0x02 certification revocation made by a trusted authority; ABG

Alice’s Certification Revocation signature packet could get attached to Bob’s certificate by several methods:

* By submitting it to a keystore that performs an unauthenticated merge; this is however vulnerable to abuse

* By submitting it to a keystore whose administrators can override Bob’s published certificate, for example a corporate directory

* By attaching it to her own key as an Embedded Signature subpacket, as specified in Section 3.2.1.1 of [I-D.gallagher-openpgp-user-attributes]

Alice could issue a superseding certification of her own over Bob’s User ID or User Attribute instead of using a soft revocation type, however she may wish to be explicit about the finality of her decision.

FIXME: Given an initial certification at time T, if the superseding certification has a timestamp of T+1, then will a verifier that cares about the certification still accept signatures from the key based on the User ID that were made exactly at time T? Alternately, if the superseding certification has a timestamp of time T exactly, will verifiers prefer its expiration date or the initial certification’s expiration date (or lack thereof)?

8.4. Challenges with OpenPGP Revocation

This section describes a number of outstanding challenges with implementing OpenPGP revocation in the state of the field before this document. Some of these problems are fixed by this document.

8.4.1. Obtaining Revocation Information

How does the user know that they have the correct revocation status? Where do they look for revocations from? With what frequency?

When the keyholder changes to a new certificate, how do they distribute revocations over older certificates?

8.4.1.1. Revocation Stripping

Given the chance to tamper with an OpenPGP certificate, the simplest thing that an adversary can do is to strip signature packets. Stripping a revocation signature packet is trivial, and the resulting certificate looks valid.

An OpenPGP implementation needs a reliable channel to fetch revocation signatures from, and a reliable and well-indexed storage mechanism to retain them safely to avoid using revoked certificates.

8.4.2. Revocations Using Weak Cryptography

What if we find a Key Revocation signature made using SHA1 or MD5?

Should we consider the indicated key revoked?

8.4.3. Revoking Primary Key Binding Signatures

Primary keys sign Subkey Binding Signatures. Signing-capable subkeys sign Primary Key Binding Signatures.

A Primary Key Binding Signature is only valid in an Embedded Signature subpacket of another signature. If the enclosing signature is revoked, the embedded signature is no longer meaningful. It is therefore not necessary to define a method to explicitly revoke a Primary Key Binding Signature.

8.4.4. Implications for Revoked Key Material

If a primary key is revoked with Reason for Revocation 2 (key has been compromised), then an implementation MAY infer that any other certificate containing the same key material has also been compromised. Note that testing key material for equality is nontrivial due to flexibility in representation, and is therefore outside the scope of this document.

If a primary key is revoked for any reason other than key compromise, an implementation MUST NOT infer anything about any other certificate containing the same key material.

If a subkey is revoked for any reason, an implementation MUST NOT infer anything about any other certificate containing the same key material. This is because a key owner can create a valid subkey revocation signature over a subkey containing arbitrary key material:

* embedded Primary Key Binding Signatures are not required in Subkey Revocation Signatures

* an earlier valid Subkey Binding Signature is not required to validate a later Subkey Revocation Signature

Encryption subkeys cannot create embedded Primary Key Binding Signatures, but a malicious subkey binding over an arbitrary encryption subkey has no security implications, since the only person adversely affected would be the attacker themselves, whose correspondents would encrypt to the wrong key. By contrast, if a malicious revocation over such a subkey was interpreted as a valid revocation over the original key material, the key's actual owner might no longer be able to receive encrypted messages at all.

Therefore, the meaning of a Subkey Revocation Signature MUST be limited to the context of the primary key that made the revocation signature.

8.4.5. No Revocation Expiration

Key material can be marked with an expiration date (e.g. in a self-signature). Signatures themselves can also be marked with an expiration date.

While Revocation Signatures are signatures, the act of revocation is permanent, so expiration is not applicable to revocations.

An implementation generating a Revocation Signature MUST NOT include an Signature Expiration Time subpacket or a Key Expiration Time subpacket in either the hashed subpackets area or the unhashed subpackets area of the signature packet. An implementation encountering a Revocation Signature packet that contains either expiration subpacket MUST ignore the subpacket.

8.4.6. Reasons for Revocation Mismatch

How should an implementation interpret a Key Revocation signature or Subkey Revocation signature with Reason for Revocation subpacket with ID 32 ("User ID information is no longer valid")?

How should an implementation interpret a Certification Revocation with a Reason for Revocation with, say, ID 1 ("Key is superseded")?

Do we just say these Revocation signatures are invalid? Do we ignore the Reasons for Revocation subpacket?

8.5. Revocation Signature Subpacket limitations

When generating a revocation signature, an implementation:

* SHOULD include a Signature Creation Time subpacket
* SHOULD NOT set the critical bit for any subpacket
* MUST NOT set the critical bit for any subpacket other than Signature Creation Time
* MUST NOT place Signature Creation Time or Reason for Revocation packets in the unhashed area

When consuming a revocation signature, an implementation:

* MUST ignore the critical bit for every subpacket
* MUST ignore any Signature Creation Time or Reason for Revocation subpacket in the unhashed area

An implementation MUST support the Signature Creation Time subpacket. If a revocation signature does not contain a valid Signature Creation Time subpacket, a receiving implementation MAY treat it as if it was created in the infinite past.

8.6. What About Revocations From the Future?

If a Revocation signature appears to have been made in the future, its interpretation will depend on whether it is hard or soft:

* If a hard revocation is from the future, then its creation date is irrelevant, since hard revocations are retrospective. Hard revocations MUST be treated as if their creation date was in the infinite past, regardless of the value of the creation date subpacket.

* If a soft revocation is from the future, then the revocation SHOULD NOT take effect until that date.

8.7. Dealing With Revoked Certificates

Implementations MUST NOT encrypt to a revoked certificate. Implementations MUST NOT accept a signature made by a revoked certificate as valid unless the revocation is "soft" (see Section 8.8) and the timestamp of the signature predates the timestamp of the revocation. Implementations MUST NOT use secret key material corresponding to a revoked certificate for signing, unless the secret key material also corresponds to a non-revoked certificate.

Implementations MAY use the secret key material corresponding to a revoked certificate.

8.8. Hard vs. Soft Revocations

Reasons for Revocation subpacket allows different values.

Some of them suggest that a verifier can still accept signatures from before the timestamp of the Revocation. These are "soft" revocations.

All the rest require that a verifier MUST treat the certificate as "hard" revoked, meaning that even signatures that have creation timestamps before the creation timestamp of the revocation signature should themselves be rejected.

8.8.1. When is Soft Revocation Useful?

Expiration makes just as much sense as a soft revocation in many circumstances, and is typically better supported. Soft revocation can however be useful if the signer wishes to explicitly indicate that their decision is final. Since revocations are permanent, a correspondent who sees a soft revocation does not need to poll for further updates to see whether an expiration date has been extended. The only reason to poll for an update to a soft-revoked key would be to check whether the soft revocation had been upgraded to a hard revocation.

Since a public encryption subkey is not useful to third parties for historical purposes, only for creating new encrypted data, there is no practical distinction between soft and hard revocation reasons, and all soft encryption subkey revocations SHOULD be treated as hard revocations with reason "none" (0x00). Note however that for some public key algorithms (such as ECDH) the owner may need to keep the public subkey in order to decrypt historical data, if the secret key material only exists on an OpenPGP card.

8.9. Revocation Certificates

A revocation certificate indicates that a given primary key is revoked.

This can take two common forms. Each form is a sequence of OpenPGP packets:

* A standalone Key Revocation signature packet by key X over X (this form is valid only for primary keys earlier than version 6)

* Primary Key X + Key Revocation signature by X over X

Additionally, there is a deprecated form:

* Primary Key X + Direct Key Signature with Revocation Key subpacket pointing to Y + Key Revocation signature by Y over X (this form is valid only for primary keys earlier than version 6)

8.9.1. Handling a Revocation Certificate

When an implementation observes any of the above forms of revocation certificate for a certificate with primary key X, it should record it and indicate that X has been revoked and is no longer to be used, along with all of its User IDs and Subkeys.

8.9.2. Publishing a Revocation Certificate

FIXME: talk about interactions with HKP, VKS, WKD, OPENPGPKEY (DANE), or other key discovery methods?

8.10. Escrowed Revocation Certificate

An escrowed revocation certificate is just a valid revocation certificate that is not published. The parties who can retrieve or reassemble the escrowed revocation certificate can publish it to inform the rest of the world that the certificate has been revoked. It is described in Section 13.9 of [RFC9580].

Since the reason for publishing an escrowed revocation cannot be known in advance, escrowed revocations SHOULD NOT include a Reason for Revocation subpacket. If such a subpacket is included, it SHOULD explicitly state a reason of "none" (0x00).

Since the reason for publishing an escrowed revocation cannot be known in advance, escrowed revocations SHOULD NOT include a Reason for Revocation subpacket. If such a subpacket is included, it SHOULD explicitly state a reason of "none" (0x00).

In what circumstances does escrowed revocation work? When is it inappropriate?

8.10.1. Escrowed Hard Revocation Workflow

An escrowed hard revocation certificate covers the use case where where the keyholder has lost control of the secret key material, and someone besides the keyholder may have gotten access to the secret key material.

At key creation time, keyholder creates a hard revocation certificate. Optionally, they encrypt it to a set of trusted participants. The keyholder stores the revocation certificate somewhere they or one of the trusted participants will be able to access it.

If the keyholder sends it to any trusted participant immediately, that participant can trigger a revocation any time they like. In this case, the keyholder and the trusted participants should clarify between themselves what an appropriate signal should be for when the trusted participant should act

If physical access is retained by the keyholder, then the keyholder has to be capable of consenting for the revocation to be published.

8.10.2. Escrowed Soft Revocation Workflow

Do regular updates of the escrowed revocation (e.g. after each signing). Store them somewhere safe?

8.10.3. K-of-N Escrowed Revocation

FIXME: how to split an escrowed revocation certificate among N parties so that any K of them can reconstruct it. ((ABG: I think this is out of scope))

8.11. Deprecation of the "Revocable" Signature Subpacket

The "Revocable" subpacket is not commonly supported, and when used as described is effectively non-functional. It is hereby deprecated.

8.11.1. Non-functionality of the "Revocable" Signature Subpacket

Section 5.2.3.20 of [RFC9580] states:

> Signatures that are not revocable have any later revocation signatures ignored. They represent a commitment by the signer that he cannot revoke his signature for the life of his key. If this packet is not present, the signature is revocable.

But this is not an effective constraint on the key owner’s future behaviour:

* Since there is no such thing as a document revocation signature, this is only applicable to self-signatures and third party certifications.

* If a key is compromised, then the timestamp on any revocation can be trivially backdated. So this must only apply to revocations by valid or soft-revoked keys.

* A soft revocation of a self-signature or third-party certification is functionally equivalent to a later signature with an expiry date, which is not covered by the "Revocable" semantics.

* A hard revocation has the same semantics regardless of its creation date. In particular, an escrowed revocation signature (such as the revocation signatures commonly made at key generation time) will have a creation time significantly in the past compared to when it is typically published. A "non-revocable" certification created after the escrowed revocation sig cannot prevent the escrowed revocation taking effect.

Therefore any "non-revocable" signature can still be effectively "revoked" by one of the following unremarkable events:

* by a later signature with an explicit expiry date, which has the same practical effect as a soft revocation,

* or by an escrowed hard revocation, which has the same practical effect as a later hard revocation.

The "revocable" subpacket is therefore non-functional.

9. Time Evolution of Signatures

Validation of a Signature packet is performed in several stages:

1. Formal Validation (the signature packet is well-formed and parseable)

2. Cryptographic Validation (the signature data was calculated correctly)

3. Structural Validation (the signature packet is placed in the correct context)

4. Temporal Validation (the signature has not expired or been revoked)

5. Issuer Validation (the signature was made by a valid key)

Included in the Issuer Validation stage is validation (including Temporal Validation) of the binding signatures in the issuer’s certificate. If the Web of Trust is in use, this process is potentially recursive.

9.1. Conflicting Requirements in Current Specifications

Section 5.2.3.10 of [RFC9580] states:

> An implementation that encounters multiple self-signatures on the same object MUST select the most recent valid self-signature and ignore all other self-signatures.

But Section 5.2.3.31 of [RFC9580] states:

> If a key has been revoked because of a compromise, all signatures created by that key are suspect.

These requirements are in explicit conflict and must be resolved by further specification.

In addition, the use of the unqualified term "valid" is ambiguous. If read inclusively to mean that expired or revoked signatures are not "valid" for the purposes of this statement, it results in complex key validity calculations with questionable added utility and obscure failure modes.

We therefore update the first statement above to read:

> An implementation that encounters multiple self-signatures on the same object MUST select the most recent _cryptographically valid_ self-signature and ignore all other self-signatures, _unless there is a revocation signature over the same object_.

9.2. Key and Certification Validity Periods

Key Expiration Time subpackets are a rich source of footguns:

1. They specify an offset rather than an timestamp, but are not usable without first converting to a timestamp.

2. The offset is calculated relative to the creation timestamp of a different packet (the component key packet).

3. Some implementations interpret them as being inheritable in their raw form, so that the same offset value gets applied to different creation timestamps.

4. It is unclear how to interpret Key Expiration Time subpackets in a v4 self-signature over a non-Primary User ID.

Further, their semantics overlaps that of Signature Expiration Time:

1. If the binding signature over a key expires, but the key does not, the key is nevertheless unusable due to lack of signatures.

2. If a key expires, but the signature over it does not, the signature is unusable.

This means there are effectively two expiration dates on a Key Binding signature, the key expiration and the signature expiration, but without distinct semantics.

In addition, the Signature Creation Time subpacket has an overloaded meaning in both Key Binding and Certification signatures:

1. It is used as the "valid from" timestamp of the object being signed over

2. It is used to order multiple similar signatures to determine which is valid

If this is interpreted strictly, it means that it is not possible to create a new Key Binding signature that reliably leaves the starting date of the key's validity unchanged. Some implementations have worked around this by generating signatures with creation dates backdated to one second after that of the previous signature.

The ability to create a new signature with an unchanged valid-from date allows historical signatures to be losslessly cleaned from a TPK, saving space. It is also more compatible with the historical interpretation favoured by PGP and GnuPG.

Finally, both Expiration Time subpackets use zero to mean "the infinite future", which requires explicit handling of the special case.

9.3. Key Binding Temporal Validity

To clean up the ambiguity in Key Binding signatures, we specify the following:

1. Key Binding signatures other than self-certifications over v4 Primary User IDs (Subkey Binding signatures, Primary Key Binding signatures, and Direct Key signatures) SHOULD NOT contain Signature Expiration Time subpackets, and any such subpackets MUST be ignored.

2. The validity of a component key extends from its creation time until its revocation or key expiration time.

3. If the most recent Key Binding signature has no Key Expiration Time subpacket, then the key does not expire.

4. Key Binding signatures cannot be directly revoked; the corresponding revocation signatures affect the key, not the binding.

5. A Key Binding signature is temporally valid if its creation time is later than the creation time of the primary key that made it.

6. A Key Binding signature is temporally valid even if the primary has been hard-revoked (so that we can still associate the primary key with its subkeys).

7. The creation time of the Key Binding signature is used only for ordering, not for calculation of signature validity.

8. Key Expiration Time subpackets are only meaningful in Key Binding signatures (including self-certifications over v4 Primary User IDs); an implementation MUST ignore a Key Expiration Time subpacket in any other signature.

A signature other than a Key Binding signature is temporally valid if it was made by a component key during its validity period.

(See also [RFC4880BIS-71], [OPENPGPJS-1800]).

9.4. Certification Temporal Validity

To clean up the ambiguity in Certification signatures, we specify the following:

1. Certification signatures other than self-certifications over v4 Primary User IDs SHOULD NOT contain Key Expiration Time subpackets, and any such subpackets MUST be ignored.

2. The validity of a User ID or User Attribute extends from the Primary Key’s creation time until the User ID or User Attribute’s revocation or signature expiration time.

3. If the most recent Certification signature has no Signature Expiration Time subpacket, then the ID or attribute does not expire.

4. A Certification signature is NOT valid if the primary has been hard-revoked.

5. The creation time of the Certification signature is used only for ordering, not for calculation of signature validity.

9.4.1. Conflicting Expiration Times in v4 Self-Certifications

The above rules permit a v4 self-certification over a Primary User ID to contain both Key Expiration Time and Signature Expiration Time subpackets, both of which are semantically meaningful. If the calculated expiry times differ, it is RECOMMENDED that a receiving implementation interprets them as follows:

1. The Key Expiration Time applies to the Primary Key, while the Signature Expiration Time applies only to the identity link between the Primary User ID and the Primary Key.

2. If the Key Expiration Time is earlier then the Signature Expiration Time is not meaningful, since the whole certificate becomes unusable after the Primary Key expires.

3. If the Key Expiration Time is later or absent then the Primary Key remains usable in the interim, but is no longer linked to the identity in the Primary User ID.

The above rules ensure that a Key Expiration Time subpacket in a v4 self-certification over a Primary User ID has the same effect as if it had been contained in a Direct Key signature.

9.4.2. Issues with Temporary Identities

When making a third-party certification signature over a User ID, the third party may not wish to validate the User ID retrospectively. This may arise when a keyholder adds a User ID to their existing certificate on a temporary basis, for example if they assume a role-based identity such as "chairperson@example.com". It would be possible for such a keyholder to backdate a signature to a time when someone else held the identity, and thereby attempt to impersonate them. It is therefore desirable to allow a third-party certifier to indicate a custom initial validity date for the User ID they certify.

There are possible approaches that do not require new wire formats:

* A minimal approach would specify that third-party certification signatures only validate the User ID from the signature creation time.

* Type 10 certification signatures only validate the User ID from the signature creation time. This would allow both third parties and keyholders to choose whether to make retrospective or time-limited certifications.

There are common issues with the above proposals:

* Current client behaviour is not consistent, so we cannot reliably enforce non-retrospective certifications if legacy clients are in use.

* If the signature creation time acts as the start of validity, we cannot losslessly clean up those certifications.

It would appear that the only way to reliably enforce a novel temporal validity interpretation would require new wire formats. For example, we could define a new "Subject Valid From" subpacket that contains a timestamp field, by analogy with the Signature Creation

Time subpacket. If the critical bit were always set on this subpacket, a legacy client MUST automatically invalidate the certification. This would also allow lossless clean up of all certifications.

((TODO: is this a reasonable trade-off? See #9))

An alternative solution would be to define an identity format with intrinsic creation dates, for example as a novel User Attribute subpacket.

9.5. Cumulation of Signatures

A cryptographically valid Key Binding, Certification or Literal Data signature automatically and permanently supersedes any earlier signature of the same Signature Category, by the same key pair, over the same subject. If a later such signature expires before an earlier one, the earlier signature does not become valid again.

For the purposes of the above:

* "same key pair" refers to the public key packet as identified by the Issuer KeyID or Issuer Fingerprint subpacket.

* "same subject" refers only to the packets being signed over, and not to the metadata contained in the Signature packets (including subpackets) or any corresponding OPS packet.

Note however that this does not apply to revocation signatures, which have their own cumulation rules (Section 8).

(See also [SCHAUB2021])

10. Security Considerations

The OPS nesting octet is not signed over and is malleable in principle. An intermediary could swap an outer OPS with its inner OPS by also swapping the nesting octets. The order of OPS nesting therefore MUST NOT be considered meaningful.

In addition, the normalization applied during Literal Data signature calculation may result in semantic collisions. It is possible to construct distinct sequences of packets that map to the same sequence of octets after Literal Data normalization is applied. It is not known whether such a pair of colliding packet sequences might also have different semantics.

11. IANA Considerations

11.1. OpenPGP Signature Types Registry

IANA is requested to add a column to the OpenPGP Signature Types registry, called "Embeddable". This column should be empty by default.

IANA is requested to register the following new entry in the registry:

ID	Name	Embeddable	Reference
0x60-0x6F	Private or Experimental Use		Section 6

Table 2: OpenPGP Signature Types (new)

IANA is requested to update the following existing entries in the registry:

ID	Name	Embeddable	Reference
0x10	Generic Certification Signature		[RFC9580], Section 3.1
0x11	Persona Certification Signature		[RFC9580], Section 3.1
0x12	Casual Certification Signature (Deprecated)		[RFC9580], Section 3.1
0x13	Positive Certification Signature		[RFC9580], Section 3.1
0x19	Primary Key Binding Signature	Yes	[RFC9580], Section 3.2, Section 5.1
0x20	Primary Key Revocation Signature		[RFC9580], Section 3.3, Section 8.1
0x28	Subkey Revocation Signature		[RFC9580], Section 8.2
0x30	Certification Revocation Signature		[RFC9580], Section 8.3
0x40	Timestamp Signature		[RFC9580], Section 3.6
0x50	Third-Party Confirmation Signature	Yes	[RFC9580], Section 3.7, Section 5.1

Table 3: OpenPGP Signature Types (updated)

11.2. OpenPGP Key Flags Registry

IANA is requested to add a column to the OpenPGP Key Flags registry, called "Primary Key Signature Required". This column should be empty by default.

IANA is requested to register the following new entry in the OpenPGP Key Flags registry:

Flag	Definition	Primary Key Signature Required	Reference
((TBC))	This key may be used to make signatures in the Countersignature Category (0x50..0x57)	Yes	Section 6, Section 3.2

Table 4: OpenPGP Key Flags (new)

IANA is requested to update the following existing entries in the registry:

Flag	Definition	Primary Key Signature Required	Reference
0x01..	This key may be used to make signatures over other keys, in the Certification and Certification Revocation Categories (0x10..0x17 and 0x30..0x37)		Section 6
0x02...	This key may be used to make signatures in the Literal Data Signature Category (0x00..0x07)	Yes	Section 6, Section 3.2
0x0008...	This key may be used to make signatures in the Timestamping Category (0x40..0x47)	Yes	Section 6, Section 3.2

Table 5: OpenPGP Key Flags (update)

11.3. OpenPGP Signature Subpacket Types Registry

IANA is requested to add columns for "Category", "Critical", and "Self-Verifying" to the OpenPGP Signature Subpacket Types registry, and populate them with initial values as listed in Table 1.

IANA is requested to mark the "Revocable" subpacket entry as "deprecated", referencing this document, Section 8.11.

The "Reason for Revocation Code" entry in the "OpenPGP Signature Subpacket Types" registry should have its References column updated to point to this document.

11.4. OpenPGP Reason for Revocation Code Registry

The "OpenPGP Reason for Revocation Code" registry should add a column to indicate "Hard/Soft". Only "Key is Superseded" and "Key is retired and no longer used" are marked "Soft". All other values should be treated as "Hard".

12. References

12.1. Normative References

[I-D.gallagher-openpgp-user-attributes]
Gallagher, A., "User Attributes in OpenPGP", Work in Progress, Internet-Draft, draft-gallagher-openpgp-user-attributes-00, 11 February 2025, <https://datatracker.ietf.org/doc/html/draft-gallagher-openpgp-user-attributes-00>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC9580] Wouters, P., Ed., Huigens, D., Winter, J., and Y. Niibe, "OpenPGP", RFC 9580, DOI 10.17487/RFC9580, July 2024, <https://www.rfc-editor.org/rfc/rfc9580>.

12.2. Informative References

[ASKCERTLEVEL]
Gillmor, D. K., "'gpg --ask-cert-level' Considered Harmful", 20 May 2013, <https://dkg.fifthhorseman.net/blog/gpg-ask-cert-level-considered-harmful.html>.

[FINNEY1998]
Finney, H., "Re: More spec-ulations - update", 26 March 1998, <https://mailarchive.ietf.org/arch/msg/openpgp/U4Qg3Z9bj-RDgpwW5nmRNetOZKY/>.

[I-D.dkg-openpgp-1pa3pc]
Gillmor, D. K., "First-Party Approved Third-Party Certifications in OpenPGP", Work in Progress, Internet-Draft, draft-dkg-openpgp-1pa3pc-02, 6 September 2024, <https://datatracker.ietf.org/doc/html/draft-dkg-openpgp-1pa3pc-02>.

[I-D.dkg-openpgp-abuse-resistant-keystore]
Gillmor, D. K., "Abuse-Resistant OpenPGP Keystores", Work in Progress, Internet-Draft, draft-dkg-openpgp-abuse-resistant-keystore-06, 18 August 2023, <https://datatracker.ietf.org/doc/html/draft-dkg-openpgp-abuse-resistant-keystore-06>.

[I-D.dkg-openpgp-revocation]
Gillmor, D. K. and A. Gallagher, "Revocation in OpenPGP", Work in Progress, Internet-Draft, draft-dkg-openpgp-revocation-02, 28 March 2025, <https://datatracker.ietf.org/doc/html/draft-dkg-openpgp-revocation-02>.

[I-D.gallagher-openpgp-hkp]
Shaw, D., Gallagher, A., and D. Huigens, "OpenPGP HTTP Keyserver Protocol", Work in Progress, Internet-Draft, draft-gallagher-openpgp-hkp-09, 3 November 2025, <https://datatracker.ietf.org/doc/html/draft-gallagher-openpgp-hkp-09>.

[I-D.gallagher-openpgp-literal-metadata]
Gallagher, A., "OpenPGP Literal Data Metadata Integrity", Work in Progress, Internet-Draft, draft-gallagher-openpgp-literal-metadata-00, 1 January 2024, <https://datatracker.ietf.org/doc/html/draft-gallagher-openpgp-literal-metadata-00>.

[I-D.ietf-openpgp-replacementkey]
Shaw, D. and A. Gallagher, "OpenPGP Key Replacement", Work in Progress, Internet-Draft, draft-ietf-openpgp-replacementkey-06, 13 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-openpgp-replacementkey-06>.

[OPENPGPDEVBOOK]
"OpenPGP for Application Developers", 6 May 2024, <https://openpgp.dev/book/>.

[OPENPGPJS-1800]
Hell, I., "'Signature creation time is in the future' error for apparently valid signature", 28 October 2024, <https://github.com/openpgpjs/openpgpjs/issues/1800>.

[RFC1991] Atkins, D., Stallings, W., and P. Zimmermann, "PGP Message Exchange Formats", RFC 1991, DOI 10.17487/RFC1991, August 1996, <https://www.rfc-editor.org/rfc/rfc1991>.

[RFC2440] Callas, J., Donnerhacke, L., Finney, H., and R. Thayer, "OpenPGP Message Format", RFC 2440, DOI 10.17487/RFC2440, November 1998, <https://www.rfc-editor.org/rfc/rfc2440>.

[RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, DOI 10.17487/RFC4880, November 2007, <https://www.rfc-editor.org/rfc/rfc4880>.

[RFC4880BIS-71]
Gallagher, A., "Deprecate the use of 'Key Expiration Time' packets (type 9) in V5 sigs", 8 January 2022, <https://gitlab.com/openpgp-wg/rfc4880bis/-/issues/71>.

[SCHAUB2021]
Schaub, P., "[openpgp] Question on Signature Expiration", 13 December 2021, <https://mailarchive.ietf.org/arch/msg/openpgp/C0P4MxwqJBbxS6H0YoXFF3oEJ3A/>.

[SCHAUB2022]
Schaub, P., "[openpgp] Proposing a Simplification of Message Syntax", 7 October 2022, <https://mailarchive.ietf.org/arch/msg/openpgp/uepOF6XpSegMO4c59tt9e5H1i4g/>.

[SQ-WOT] Walfield, N., "OpenPGP Web of Trust", 3 February 2022, <https://sequoia-pgp.gitlab.io/sequoia-wot/>.

Appendix A. Acknowledgments

This document would not have been possible without the extensive work of the authors of [OPENPGPDEVBOOK].

The author would also like to thank Daniel Huigens, Daniel Kahn Gillmor, Heiko Sch辰fer, Neal Walfield, Justus Winter and Paul Schaub for additional discussions and suggestions.

Appendix B. Document History

Note to RFC Editor: this section should be removed before publication.

B.1. Changes Between draft-gallagher-openpgp-signatures-01 and draft-gallagher-openpgp-signatures-02

* Merged in half of draft-dkg-openpgp-revocation and added DKG as co-author.
* Adapted key temporal validity rules for certification signatures.
* Specified use of Persona Certifications for non-trust statements.
* Added section for Primary Key Binding signatures.
* Added sections for conflicting subpackets and conflicting requirements.
* Specified line ending normalization of bare LF only.
* Deprecated revocation of Direct Key signatures.
* Deprecated Signature Target subpackets.
* Added section for issues with temporary identities.
* Refactored and constrained message grammar.
* Fixed some crufty terminology.

B.2. Changes Between draft-gallagher-openpgp-signatures-00 and draft-gallagher-openpgp-signatures-01

* Expanded temporal validity.
* Renamed "Document" and "Data Type" Signature Categories to "Literal Data" and "Attribute Value" respectively.
* Expanded experimental range to cover 0x60..0x6F (96..111).
* Add explicit category ranges to the Key Flags registry.

* Add explicit note about when to ignore Direct and Key Binding subpackets.

* Distinguish between signature subject and signature type-specific data.

* Deprecated the nesting octet.

* Minor errata.

Authors' Addresses

Andrew Gallagher (editor)
PGPKeys.EU
Email: andrewg@andrewg.com

Daniel Kahn Gillmor
ACLU
Email: dkg@fifthhorseman.net