openpgp
Internet-Draft
Updates: 9580 (if approved)
Intended status: Standards Track
Expires: 6 December 2026

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

4 June 2026

OpenPGP Certificate Grammar
draft-gallagher-openpgp-certificates-00

Abstract

This document specifies several updates and clarifications to the grammar and semantics of OpenPGP certificates.

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-certificates. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-gallagher-openpgp-certificates/.

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-certificates.

Status of This Memo

This Internet-Draft will expire on 6 December 2026.

Table of Contents

1. Introduction

OpenPGP certificates have a complex grammar and sometimes poorly-understood semantics. 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.

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 used in certificates 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 historically many implementations did 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 by modifying the above text to read:

> A 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, if the Subkey Binding signature contains a Key Flags subpacket permitting that subkey to create OpenPGP signatures. A receiving implementation MUST reject any Subkey Binding signature that permits the creation of OpenPGP signatures by that subkey and does not contain a valid Subkey Binding signature. A Primary Key Binding signature is OPTIONAL otherwise.
>
> 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.

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.

We therefore update the definition to remove the reference to Direct Key Signatures:

> This signature revokes an earlier User ID certification signature (Type IDs 0x10 through 0x13). 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.

4. 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. Structural Validation (the signature packet is placed in the correct context)
3. Cryptographic Validation (the signature data was calculated correctly)
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.

4.1. 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_.

4.2. 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.

4.3. 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]).

4.4. 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.

4.4.1. 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.

4.4.2. 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:

* We could specify that third-party certification signatures only validate the User ID from the signature creation time.

* We could specify that type 0x10 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 draft-signatures#9))

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

4.5. 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 5).

(See also [SCHAUB2021])

FIXME: what about signatures with the same creation time? (issue #9)

5. Revoking Signatures and Keys

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.

5.1. 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.

5.1.1. 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 5.8) and publishing it directly.

5.1.2. 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.

5.1.3. 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.

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 5.3 below for possible alternative methods in a controlled environment.

5.2. 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.

5.3. 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 (see #8)

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.

5.4. 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.

5.5. Revocation Signature Subpackets

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.

5.6. 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.

5.7. 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 5.8) 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.

5.8. 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.

5.8.1. Use of Soft Revocations

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.

5.8.2. Reasons for Revocation Mismatch

FIXME: (all of this, see issue #3)

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?

5.9. 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)

5.9.1. Dealing With 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.

6. Revocation Guidance

This section describes a number of outstanding challenges with implementing OpenPGP revocation.

6.1. Freshness Considerations

6.1.1. Publishing a Revocation Certificate

FIXME: talk about interactions with HKP, VKS, WKD, OPENPGPKEY (DANE), or other key discovery methods? (issue #7)

6.1.2. Publication 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]).

6.1.3. 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? (issue #7)

6.1.4. 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.

6.2. Revocations Using Weak Cryptography

What if we find a Key Revocation signature made using SHA1 or MD5? Should we consider the indicated key revoked? (issue #2)

6.3. Shared 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.

6.4. Escrowed Revocation Certificates

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? (issue #6)

6.4.1. 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.

6.4.2. Escrowed Soft Revocation Workflow

Do regular updates of the escrowed revocation (e.g. after each signing). Store them somewhere safe? (issue #6)

7. Security Considerations

((TO BE COMPLETED))

8. IANA Considerations

8.1. OpenPGP Signature Types Registry

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, ((TBC))
0x20	Primary Key Revocation Signature		[RFC9580], Section 3.3, Section 5.1
0x28	Subkey Revocation Signature		[RFC9580], Section 5.2
0x30	Certification Revocation Signature		[RFC9580], Section 5.3

Table 1: OpenPGP Signature Types (updated)

((TODO: avoid clash between the updates to 0x19 here and in draft-certificates))

8.2. OpenPGP Signature Subpacket Types Registry

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

8.3. 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".

9. References

9.1. Normative References

[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>.

9.2. 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>.

[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.gallagher-openpgp-user-attributes]
Gallagher, A., "User Attributes in OpenPGP", Work in Progress, Internet-Draft, draft-gallagher-openpgp-user-attributes-01, 3 November 2025, <https://datatracker.ietf.org/doc/html/draft-gallagher-openpgp-user-attributes-01>.

[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>.

[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/>.

Appendix A. Acknowledgments

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

Authors' Addresses

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

Daniel Kahn Gillmor
ACLU
Email: dkg@fifthhorseman.net