openpgp
Internet-Draft
Updates: 9580 (if approved)
Intended status: Standards Track
Expires: 19 September 2025

A. Gallagher, Ed.
PGPKeys.EU

18 March 2025

Code Point Exhaustion in OpenPGP
draft-gallagher-openpgp-code-point-exhaustion-00

Abstract

This document specifies variable-length encoding mechanisms to expand the current one-octet OpenPGP registries beyond 256 values.

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

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-code-point-exhaustion.

Status of This Memo

This Internet-Draft will expire on 19 September 2025.

Table of Contents

1. Introduction

We are motivated to define a variable-length encoding for OpenPGP code points so that we can have more than 256 of each. While there is no immediate prospect of any of the OpenPGP registries exceeding 256 entries, problems have already been encountered with the limited number of private and experimental use code points in the Public Key Algorithms registry [STRENZKE-2024]. It is also prudent to define and reserve an extension scheme significantly in advance of it becoming necessary.

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 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. Single-Octet Code Point Registries

The following OpenPGP registries currently define single-octet code points with up to 256 values:

Registry Name	Notes
OpenPGP String-to-Key (S2K) Types	
OpenPGP User Attribute Subpacket Types	
OpenPGP Image Attribute Encoding Format	
OpenPGP Signature Subpacket Types	(1)
OpenPGP Reason for Revocation (Revocation Octet)	
OpenPGP Public Key Algorithms	
OpenPGP Symmetric Key Algorithms	(2)
OpenPGP Hash Algorithms	
OpenPGP Compression Algorithms	
OpenPGP Secret Key Encryption (S2K Usage Octet)	(2)
OpenPGP Signature Types	(3)
OpenPGP Image Attribute Versions	(3)
OpenPGP AEAD Algorithms	
OpenPGP Key and Signature Versions	(3)

Table 1: OpenPGP Single-Octet Registries

1. The Signature Subpacket Types registry has only 128 usable code points in practice, due to the criticality bit (see Section 4.2).

2. The Secret Key Encryption registry automatically includes the code points from the Symmetric Key Algorithms registry, although their use is deprecated. The Secret Key Encryption registry is therefore the only one that assigns code points greater than 110 -- specifically the S2K algorithms 253, 254 and 255, to avoid code point clash.

3. These registries do not contain a private and experimental use area.

The OpenPGP Packet Type registry supports only code points in the range (0..63), however we consider it separately in Section 4.3.

Note that the following single-octet identifiers have no associated registry, and are not considered in this document:

* SEIPD packet version
* PKESK packet version
* SKESK packet version
* Literal data packet format (printable ASCII character)
* One-Pass Signature nesting octet

The most heavily populated single-octet registries are currently Signature Subpacket Types (51/128 or 102/256) and Public Key Algorithms (28/256). Other registries are unlikely to ever exceed their capacity, however we define a generic scheme here for future reference.

4. Extended Code Points

We specify three encoding methods for extended code points, one generic, one for subpacket types, and one for packet types.

4.1. UTF-8ish Encoding

For registries other than subpacket types or packet types, we use a similar encoding scheme to UTF-8, but with two- and four-octet encodings omitted, i.e.:

```
if the 1st octet <  128, then
    lengthOfEncoding = 1
    codePoint = 1st_octet

if the 1st octet >= 224 and < 240, then
   lengthOfEncoding = 3
    codePoint = ((1st_octet - 224) << 12) + ((2nd_octet - 128) << 6) + (3rd_octet - 128)

if the 1st octet >= 240, then
    lengthOfEncoding = 1
    codePoint = 1st_octet
```

The second and third octets MUST be >= 128 and < 192. This ensures that none of the octets in a multi-octet encoding can be incorrectly interpreted as a single-octet encoding of another code point. This means that strings of multi-octet code points are self-synchronising.

* single-octet encodings have an octet value < 128 and are fully backwards compatible

* octet values in the range 192..223 correspond to the first octet of two-octet UTF-8 encodings and MUST NOT be used

* three-octet encodings have a first octet in the range 224..239, and represent code points in the range 128..65535

* octet values in the range 240..247 correspond to the first octet of four-octet UTF-8 encodings and MUST NOT be used

* legacy single-octet encodings have an octet value in the range 248..255

* overlong encodings MUST NOT be used

We reserve initial octet values in the range 248..255, which are not used in UTF-8, for legacy single-octet encodings of code points with the same values. This ensures backwards compatibility of existing secret key encryption (S2K Usage) code points. Since Secret Key Encryption code points in this range are in current use, they MUST be represented by legacy single-octet encodings. Legacy single-octet encodings MUST NOT be used in any other context. The corresponding code points 248..255 will be reserved in the Symmetric Encryption Algorithms registry to avoid ambiguity.

The top half of the extended range (code points 32768..65535) is reserved for additional private and experimental use code points in registries that already contain a private and experimental use range.

Two- and four-octet encodings are not used, for the reasons outlined in Section 5.3.4 below.

4.2. Subpacket Type Encoding

The criticality bit means that the first octet of any multi-octet signature subpacket type encoding is effectively limited to values less than 128. We therefore cannot use UTF-8ish encoding for signature subpacket types, and cannot make use of its self-synchronisation properties. Luckily, subpacket types are only found as the second field of a subpacket, immediately after the packet length, so self-synchronisation is not required.

Code point 127 is reserved as a surrogate code point to indicate that the actual subpacket type is encoded in the following two octets.

The standard subpacket type encoding (including criticality bit) continues to represent code points 0..126 as usual, as well as the surrogate code point 127. Surrogate subpacket type encoding is used to represent code points 128..65535, and MUST NOT be used to represent code points 0..127.

The criticality bit is the most significant bit of the first octet when surrogate encoding is used, but applies to the specific subpacket type being encoded, not the use of 127 as a surrogate code point.

The top half of the extended range (code points 32768..65535) is reserved for additional private or experimental use code points.

For consistency, User Attribute Subpackets use the same surrogate encoding format as Signature Subpackets, even though they don't have a criticality bit. The most significant bit of the surrogate octet is therefore always 0.

4.3. Packet Type Encoding

It may eventually become necessary to also expand the Packet Type registry, which currently has 24 of 64 possible entries allocated. The encoding space is highly restricted due to the small number of free bits available in the OpenPGP packet framing.

Code point 16 was initially intended to indicate a "comment" packet but was never used, and is not encodable in Legacy packet framing. We therefore reserve it as a surrogate code point to indicate that the actual packet type is encoded in the two octets immediately following the length field.

The OpenPGP packet type encoding continues to represent code points 0..63 as usual, including the surrogate code point 16. Surrogate packet type encoding is used to represent code points 64..65535, and MUST NOT be used to represent code points 0..63.

Code points in the range 64..16383 represent non-critical packets. Code points in the range 16384..32767 represent critical packets. The top half of the extended range (code points 32768..65535) is reserved for additional private or experimental use code points.

5. Support and Compatibility

Implementations SHOULD support surrogate encodings of Signature Subpacket Types and User Attribute Subpacket Types.

Implementations MUST gracefully ignore UTF-8ish encodings of unknown code points, and MAY support known code points with UTF-8ish encodings, in the following registries:

* OpenPGP Public Key Algorithms
* OpenPGP Symmetric Key Algorithms
* OpenPGP Hash Algorithms
* OpenPGP Compression Algorithms
* OpenPGP AEAD Algorithms

These code points may be found in the following contexts:

* key material packets
* signature packets and subpackets
* OPS packets
* PKESK and SKESK packets
* compressed data packets

See the subsections below for further discussion of each of these contexts.

Implementations MAY support UTF-8ish encodings of code points from the following registries:

* OpenPGP String-to-Key (S2K) Types
* OpenPGP Image Attribute Encoding Format
* OpenPGP Reason for Revocation (Revocation Octet)
* OpenPGP Secret Key Encryption (S2K Usage Octet)
* OpenPGP Signature Types
* OpenPGP Image Attribute Versions

* OpenPGP Key and Signature Versions

Implementations MAY also support surrogate encodings of code points from the OpenPGP Packet Types registry.

UTF-8ish and surrogate encodings should be legacy-safe in these registries, as parsing is normally halted immediately if the code point in question is not supported. However, for safety with legacy code that might not safely ignore unknown code points, they MUST only appear in an OpenPGP packet sequence that was first specified in [RFC9580] or later, i.e.:

* a key material, signature, OPS, SKESK, or PKESK packet of version 6 or later

* a User Attribute packet attached to a primary key of version 6 or later

* an SEIPD packet of version 2 or later

* a compressed data packet signed by a signature of version 6 or later, and/or encrypted in an SEIPD packet of version 2 or later

5.1. v6 Key Material Packets

After the packet version identifier, a v6 key material packet contains the following fields:

* Creation time

* Public key algorithm

* Length of public key material

It is therefore theoretically possible for a continuation octet of a UTF-8ish encoding of the public key algorithm to be misinterpreted as the first octet of the following length parameter, resulting in a mis-parsing of the remaining packet data, and a possible out of bounds read. This introduces no new vulnerabilities however, as an attacker can already create a key material packet with an incorrect "length of public key material" parameter, and so a compliant implementation SHOULD already have protections against out of bounds read of the remaining packet.

A secret key material packet further contains a Secret Key Encryption (S2K Usage) parameter, followed by variable fields that cannot be parsed if the Secret Key Encryption parameter is not supported. These variable fields may include an S2K specifier, which always

begins with an S2K specifier type. The remainder of the S2K specifier cannot be parsed if the S2K specifier type is not supported.

UTF-8ish encodings are therefore safe in v6 key material packets.

5.2. v6 Signature Packets

After the packet version identifier, a v6 signature packet contains a string of three potentially UTF-8ish code point fields:

* Signature Type ID
* public key algorithm
* hash algorithm

The remainder of the packet cannot be parsed if the public key and hash algorithm code points are unsupported, and a signature digest cannot be constructed if the signature type is unsupported.

The trailer contains the same UTF-8ish fields, however the following features prevent malleability:

* The trailer contains a length field that covers the UTF-8ish fields.
* All octets used in UTF-8ish encoding map to currently unassigned octet values.

UTF-8ish encodings are therefore safe in v6 signature packets.

5.3. Signature Subpackets

Signature subpackets are prefixed by a length and a type parameter. The remainder of each subpacket cannot be parsed if the type parameter is not supported.

The same considerations apply to User Attribute Subpackets, as they are constructed identically.

UTF-8ish and surrogate encodings are generally safe in subpackets, subject to constraints:

5.3.1. Revocation Key

The Revocation Key subpacket contains a symmetric algorithm ID field, however it is only defined for v4 targets and is now deprecated. UTF-8ish encoded code points are therefore not valid in this subpacket and MUST NOT be used.

5.3.2. Signature Target

The Signature Target subpacket contains three fields:

* symmetric key algorithm
* hash algorithm
* hashed data

The hashed data cannot be parsed if the hash algorithm is unknown.

5.3.3. Reason for Revocation

A Reason for Revocation subpacket contains a (potentially UTF-8ish) reason identifier. The remainder of the subpacket contains an unparsed human-readable comment in UTF-8. A UTF-8ish encoding of the reason field might overflow into the human-readable UTF-8 comment, however a UTF-8 compliant client SHOULD convert the continuation bytes to replacement characters and display the rest of the comment unchanged.

5.3.4. Algorithm Preferences

Care should be taken when handling the following signature subpackets, which consist of arrays of code points in which unknown code points MUST be gracefully ignored.

* Preferred Symmetric Ciphers subpacket (array of Symmetric Key Algorithm code points)
* Preferred AEAD Ciphersuites subpacket (array of {Symmetric Key Algorithm, AEAD Algorithm} code point tuples)
* Preferred Hash Algorithms (array of Hash Algorithm code points)
* Preferred Compression Algorithms (array of Compression Algorithm code points)

Multi-octet encodings of code points in the Preferred AEAD Ciphersuites subpacket could potentially result in desynchronisation of the tuple parsing in legacy code that assumes all encoded tuples are two octets wide. Tuples where one or both code points are encoded in three octets will have an even number of octets overall (either four or six), and a legacy parser that correctly ignores unknown octet values should skip over the entire tuple.

UTF-8ish encodings are therefore safe in algorithm preference subpackets.

5.3.5. Issuer and Intended Recipient Fingerprints

Version-tagged fingerprints (fingerprints prefixed by a version field) are generally safe to use with UTF-8ish encodings of version numbers, as fingerprints are generally understood to have variable lengths and unknown fingerprint versions should be ignored.

5.4. v6 One-Pass Signature Packets

After the packet version identifier, a v6 OPS packet contains a string of three potentially UTF-8ish code point fields:

* Signature type ID

* Hash algorithm

* Public key algorithm

The remainder of the packet cannot be parsed if the hash algorithm code point is unsupported.

UTF-8ish encodings are therefore safe in v6 OPS packets.

5.5. v6 PKESK Packets

A v6 PKESK packet contains two potentially UTF-8ish code point fields, the fingerprint version number and the algorithm identifier. The fingerprint version is covered by a preceding length field that allows unsupported fingerprints to be safely skipped. The algorithm identifier is immediately followed by a variable-length field that cannot be parsed if the algorithm is unsupported.

UTF-8ish encodings are therefore safe in v6 PKESK packets.

5.6. v6 SKESK Packets

A v6 SKESK packet contains the following fields:

* Symmetric algorithm ID
* AEAD algorithm ID
* S2K specifier length
* S2K specifier
* An initialisation value

All five fields are covered by a preceding length field, and the string-to-key specifier is further covered by its own length field. The S2K specifier always begins with an S2K specifier type, and the remainder of the S2K specifier cannot be parsed if the S2K specifier type is not supported. The length of the initialisation value depends on the AEAD algorithm, and therefore cannot be parsed if the AEAD algorithm is not supported.

Secret Key Encryption (S2K Usage) parameters are not supplied in a v6 SKESK, as AEAD mode is always used.

UTF-8ish encodings are therefore safe in v6 SKESK packets.

5.7. Image Attribute Subpackets

Image Attribute Subpackets are prefixed by the following fields:

* Header length (2 octets)
* Image Attribute Version
* Image Attribute Type

The remainder of the subpacket cannot be parsed if the version and type are not supported.

UTF-8ish encodings are therefore safe in Image Attribute subpackets.

5.8. v2 SEIPD Packets

After the packet version identifier, a v2 SEIPD packet contains the following fields:

* Symmetric cipher algorithm ID

* AEAD algorithm identifier

* A 1-octet chunk size

* 32 octets of salt

* Encrypted data (variable length)

If the values of the symmetric cipher algorithm and the AEAD algorithm were not checked immediately, it might be possible for legacy code to misidentify the field boundaries and pass incorrectly bounded fields to the HKDF function. Even if this were the case, the decryption process would not be able to proceed any further without the symmetric and AEAD algorithms being supported, so it is highly unlikely that any information could be leaked to an attacker.

UTF-8ish encodings are therefore almost certainly safe in v2 SEIPD packets.

5.9. Compressed Data Packets

A compressed data packet consists of a compression algorithm ID followed by data that cannot be parsed if the compression algorithm is unsupported.

UTF-8ish encodings are therefore safe in compressed data packets.

6. PGP-GREASE

The following values MAY be used as additional PGP-GREASE code points (see [I-D.gallagher-openpgp-grease]) in any registry that supports both PGP-GREASE and UTF-8ish or surrogate encoding:

Sequence	Code Point (Decimal)
9	222
10	333
11	444
12	555
13	666
14	777
15	888
16	999

Table 2: Additional PGP-GREASE Code Points

Note that the division of the extended packet type registry into critical and non-critical ranges in Section 4.3 above ensures that the additional PGP-GREASE code points fall into the non-critical range.

7. Security Considerations

The individual security considerations listed in each subsection of Section 5 should be carefully examined.

8. IANA Considerations

IANA is requested to perform the following tasks:

* extend the OpenPGP Packet Types registry and each of the other registries listed in Table 1 to 65535 entries, and reserve the range (32768..65535) with the description "Private and Experimental Use" in each.

* allocate the code point 127 in the OpenPGP Signature Subpacket Types and OpenPGP User Attribute Subpacket Types registries, with the description "Surrogate code point" and a reference to this document.

* allocate the code point 16 in the OpenPGP Packet Types registry, with the description "Surrogate code point" and a reference to this document.

* allocate the additional code points in Table 2 in each registry listed in Table 1 that supports PGP-GREASE code points, with the description "Reserved (PGP-GREASE)" a reference to this document.

9. References

9.1. Normative References

[I-D.gallagher-openpgp-grease]
Gallagher, A., "GREASE Code Points in OpenPGP", Work in Progress, Internet-Draft, draft-gallagher-openpgp-grease-00, 3 February 2025, <https://datatracker.ietf.org/doc/html/draft-gallagher-openpgp-grease-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>.

9.2. Informative References

[STRENZKE-2024]
Strenzke, F., "PQC encryption algorithm selection", n.d., <https://mailarchive.ietf.org/arch/msg/openpgp/EhX8qElgHdV1-S75o_4lCIoKuc4/>.

Appendix A. Acknowledgments

The author would like to thank Daniel Huigens for discussions.

Author’s Address

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