Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 16, 2018

S. Josefsson
SJD AB
J. Schaad
August Cellars

September 12, 2017

Algorithm Identifiers for Ed25519, Ed448, X25519 and X448 for use in the Internet X.509 Public Key Infrastructure

draft-ietf-curdle-pkix-06

Abstract

This document specifies algorithm identifiers and ASN.1 encoding formats for Elliptic Curve constructs using the curve25519 and curve448 curves. The signature algorithms covered are Ed25519 and Ed448. The key agreement algorithm covered are X25519 and X448. The encoding for Public Key, Private Key and EdDSA digital signature structures is provided.

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 March 16, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

In [RFC7748], the elliptic curves curve25519 and curve448 are described. They are designed with performance and security in mind. The curves may be used for Diffie-Hellman and Digital Signature operations.

[RFC7748] describes the operations on these curves for the Diffie-Hellman operation. A convention has developed that when these two curves are used with the Diffie-Hellman operation, they are referred to as X25519 and X448. This RFC defines the ASN.1 Object Identifiers (OIDs) for the operations X25519 and X448 along with the parameters. The use of these OIDs is described for public and private keys.

In [RFC8032] the elliptic curve signature system Edwards-curve

Digital Signature Algorithm (EdDSA) is described along with a recommendation for the use of the curve25519 and curve448. EdDSA has defined two modes, the PureEdDSA mode without pre-hashing, and the HashEdDSA mode with pre-hashing. The convention used for identifying the algorithm/curve combinations are to use the Ed25519 and Ed448 for

the PureEdDSA mode. The document does not provide the conventions needed for the pre-hash versions of the signature algorithm. The use of the OIDs is described for public keys, private keys and signatures.

[RFC8032] additionally defined the concept of a context. Contexts can be used to differentiate signatures generated for different purposes with the same key. The use of contexts is not defined in this document for the following reasons:

- The current implementations of Ed25519 do not support the use of contexts, thus if specified it will potentially delay the use of these algorithms further.
- The EdDSA algorithms are the only IETF algorithms that currently support the use of contexts, however there is a possibility that there will be confusion between which algorithms need have separate keys and which do not. This may result in a decrease of security for those other algorithms.
- There are still on going discussions among the cryptographic community about how effective the use of contexts is for preventing attacks.
- There needs to be discussions about the correct way to identify when context strings are to be used. It is not clear if different OIDs should be used for different contexts, or the OID should merely not that a context string needs to be provided.

2. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Curve25519 and Curve448 Algorithm Identifiers

Certificates conforming to [RFC5280] can convey a public key for any public key algorithm. The certificate indicates the algorithm through an algorithm identifier. This algorithm identifier is an OID and optionally associated parameters.

The AlgorithmIdentifier type, which is included for convenience, is defined as follows:

```
AlgorithmIdentifier  ::=  SEQUENCE  {
    algorithm   OBJECT IDENTIFIER,
    parameters  ANY DEFINED BY algorithm OPTIONAL
}
```

The fields in AlgorithmIdentifier have the following meanings:

- algorithm identifies the cryptographic algorithm with an object identifier. This is one of the OIDs defined below.

- parameters, which are optional, are the associated parameters for the algorithm identifier in the algorithm field. When the 1997 syntax for AlgorithmIdentifier was initially defined, it omitted the OPTIONAL key word. The optionality of the parameters field was later recovered via a defect report, but by then many people thought that the field was mandatory. For this reason, a small number of implementations may still require the field to be present.

In this document we defined four new OIDs for identifying the different curve/algorithm pairs. The curves being curve25519 and curve448. The algorithms being ECDH and EdDSA in pure mode. For all of the OIDs, the parameters MUST be absent. Regardless of the defect in the original 1997 syntax, implementations MUST NOT accept a parameters value of NULL.

The same algorithm identifiers are used for identifying a public key, identifying a private key and identifying a signature (for the two EdDSA related OIDs). Additional encoding information is provided below for each of these locations.

```
id-X25519    OBJECT IDENTIFIER ::= { 1 3 101 110 }
id-X448      OBJECT IDENTIFIER ::= { 1 3 101 111 }
id-Ed25519   OBJECT IDENTIFIER ::= { 1 3 101 112 }
id-Ed448     OBJECT IDENTIFIER ::= { 1 3 101 113 }
```

4. Subject Public Key Fields

In the X.509 certificate, the subjectPublicKeyInfo field has the SubjectPublicKeyInfo type, which has the following ASN.1 syntax:

```
SubjectPublicKeyInfo  ::=  SEQUENCE  {
    algorithm         AlgorithmIdentifier,
    subjectPublicKey  BIT STRING
}
```

The fields in SubjectPublicKeyInfo have the following meanings:

- algorithm is the algorithm identifier and parameters for the public key (see above).

- subjectPublicKey contains the byte stream of the public key. The algorithms defined in this document always encode the public key as an exact multiple of 8-bits.

Both [RFC7748] and [RFC8032] define the public key value as being a byte string. It should be noted that the public key is computed differently for each of these documents, thus the same private key will not produce the same public key.

The following is an example of a public key encoded using the textual encoding defined in [RFC7468].

```
-----BEGIN PUBLIC KEY-----
MCowBQYDK2VwAyEAGb9ECWmEzf6FQbrBZ9w7lshQhqowtrbLDFw4rXAxZuE=
-----END PUBLIC KEY-----
```

5. Key Usage Bits

The intended application for the key is indicated in the keyUsage certificate extension.

If the keyUsage extension is present in a certificate that indicates id-X25519 or id-X448 in SubjectPublicKeyInfo, then the following MUST be present:

keyAgreement;

one of the following MAY also be present:

encipherOnly; or
decipherOnly.

If the keyUsage extension is present in an end-entity certificate that indicates id-Ed25519 or id-Ed448, then the keyUsage extension MUST contain one or both of the following values:

nonRepudiation; and
digitalSignature.

If the keyUsage extension is present in a certification authority certificate that indicates id-Ed25519 or id-Ed448, then the keyUsage extension MUST contain one or more of the following values:

nonRepudiation;
digitalSignature;
keyCertSign; and
cRLSign.

6. EdDSA Signatures

Signatures can be placed in a number of different ASN.1 structures. The top level structure for a certificate is given below as being illustrative of how signatures are frequently encoded with an algorithm identifier and a location for the signature.

```
   Certificate  ::=  SEQUENCE  {
        tbsCertificate       TBSCertificate,
        signatureAlgorithm   AlgorithmIdentifier,
```

```
        signatureValue       BIT STRING  }
```

The same algorithm identifiers are used for signatures as are used for public keys. When used to identify signature algorithms, the parameters MUST be absent.

The data to be signed is prepared for EdDSA. Then, a private key operation is performed to generate the signature value. This value is the opaque value ENC(R) || ENC(S) described in section 3.3 of [RFC8032]. The octet string representing the signature is encoded directly in the BIT STRING without adding any additional ASN.1 wrapping. For the Certificate structure, the signature value is wrapped in the "signatureValue" BIT STRING field.

7. Private Key Format

Asymmetric Key Packages [RFC5958] describes how encode a private key in a structure that both identifies what algorithm the private key is for, but allows for the public key and additional attributes about the key to be included as well. For illustration, the ASN.1 structure OneAsymmetricKey is replicated below. The algorithm specific details of how a private key is encoded is left for the document describing the algorithm itself.

```
   OneAsymmetricKey ::= SEQUENCE {
      version Version,
      privateKeyAlgorithm PrivateKeyAlgorithmIdentifier,
      privateKey PrivateKey,
      attributes [0] IMPLICIT Attributes OPTIONAL,
      ...,
      [[2: publicKey [1] IMPLICIT PublicKey OPTIONAL ]],
      ...
```

```
}

PrivateKey ::= OCTET STRING

PublicKey ::= BIT STRING
```

For the keys defined in this document, the private key is always an opaque byte sequence. The ASN.1 type CurvePrivateKey is defined in this document to hold the byte sequence. Thus when encoding a OneAsymmetricKey object, the private key is wrapped in an CurvePrivateKey object and wrapped by the OCTET STRING of the "privateKey" field.

```
CurvePrivateKey ::= OCTET STRING
```

To encode a EdDSA, X25519 or X448 private key, the "privateKey" field will hold the encoded private key. The "privateKeyAlgorithm" field uses the AlgorithmIdentifier structure. The structure is encoded as defined above. If present, the "publicKey" field will hold the encoded key as defined in [RFC7748] and [RFC8032].

The following is an example of a private key encoded using the textual encoding defined in [RFC7468].

```
-----BEGIN PRIVATE KEY-----
MC4CAQAwBQYDK2VwBCIEINTuctv5E1hK1bbY8fdp+K06/nwoy/HU++CXqI9EdVhC
-----END PRIVATE KEY-----
```

The following example, in addition to encoding the private key, additionally has an attribute included as well as the public key. As with the prior example, the textual encoding defined in [RFC7468] is used.

```
-----BEGIN PRIVATE KEY-----
MHICAQEwBQYDK2VwBCIEINTuctv5E1hK1bbY8fdp+K06/nwoy/HU++CXqI9EdVhC
oB8wHQYKKoZIhvcNAQkJFDEPDA1DdXJkbGUgQ2hhaXJzgSEAGb9ECWmEzf6FQbrB
Z9w7lshQhqowtrbLDFw4rXAxZuE=
-----END PRIVATE KEY------
```

NOTE: There exist some private key import functions that have not picked up the new ASN.1 structure OneAsymmetricKey that is defined in [RFC7748]. This means that they will not accept a private key structure which contains the public key field. This means a balancing act needs to be done between being able to do a consistency check on the key pair and widest ability to import the key.

8. Human Readable Algorithm Names

For the purpose of consistent cross-implementation naming this section establishes human readable names for the algorithms specified in this document. Implementations SHOULD use these names when referring to the algorithms. If there is a strong reason to deviate from these names -- for example, if the implementation has a different naming convention and wants to maintain internal consistency -- it is encouraged to deviate as little as possible from the names given here.

Use the string "ECDH" when referring to a public key of type X25519 or X448 when the curve is not known or relevant.

When the curve is known, use the more specific string of X25519 or X448.

Use the string "EdDSA" when referring to a signing public key or signature when the curve is not known or relevant.

When the curve is known, use a more specific string. For the id-Ed25519 value use the string "Ed25519". For id-Ed448 use "Ed448".

9. ASN.1 Module

For reference purposes, the ASN.1 syntax is presented as an ASN.1 module here.

```
-- ASN.1 Module

Safecurves-pkix-0 -- TBD - IANA assigned module OID

DEFINITIONS EXPLICIT TAGS ::=
BEGIN

IMPORTS
  SIGNATURE-ALGORITHM, KEY-AGREE, PUBLIC-KEY, KEY-WRAP,
  KeyUsage, AlgorithmIdentifier
  FROM AlgorithmInformation-2009
    {iso(1) identified-organization(3) dod(6) internet(1) security(5)
    mechanisms(5) pkix(7) id-mod(0)
```

```
       id-mod-algorithmInformation-02(58)}

      mda-sha512
      FROM PKIX1-PSS-OAEP-Algorithms-2009
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) id-mod(0)
          id-mod-pkix1-rsa-pkalgs-02(54) }

      kwa-aes128-wrap, kwa-aes256-wrap
      FROM CMSAesRsaesOaep-2009
        { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
          smime(16) modules(0) id-mod-cms-aes-02(38) }
      ;


   id-edwards-curve-algs OBJECT IDENTIFIER ::= { 1 3 101 }

   id-X25519        OBJECT IDENTIFIER ::= { id-edwards-curve-algs 110 }
   id-X448          OBJECT IDENTIFIER ::= { id-edwards-curve-algs 111 }
   id-Ed25519       OBJECT IDENTIFIER ::= { id-edwards-curve-algs 112 }
   id-Ed448         OBJECT IDENTIFIER ::= { id-edwards-curve-algs 113 }


    sa-Ed25519 SIGNATURE-ALGORITHM ::= {
       IDENTIFIER id-Ed25519
        PARAMS ARE absent
        PUBLIC-KEYS {pk-Ed25519}
        SMIME-CAPS { IDENTIFIED BY id-Ed25519 }
    }

    pk-Ed25519 PUBLIC-KEY ::= {
        IDENTIFIER id-Ed25519
        -- KEY no ASN.1 wrapping --
        PARAMS ARE absent
        CERT-KEY-USAGE {digitalSignature, nonRepudiation,
                        keyCertSign, cRLSign}
        PRIVATE-KEY CurvePrivateKey
    }

    kaa-X25519 KEY-AGREE ::= {
        IDENTIFIER id-X25519
        PARAMS ARE absent
        PUBLIC-KEYS {pk-X25519}
        UKM -- TYPE no ASN.1 wrapping -- ARE preferredPresent
        SMIME-CAPS {
           TYPE AlgorithmIdentifier{KEY-WRAP, {KeyWrapAlgorithms}}
```

```
           IDENTIFIED BY id-X25519 }
    }
```

```
    pk-X25519 PUBLIC-KEY ::= {
        IDENTIFIER id-X25519
        -- KEY no ASN.1 wrapping --
        PARAMS ARE absent
        CERT-KEY-USAGE { keyAgreement }
        PRIVATE-KEY CurvePrivateKey
    }

    KeyWrapAlgorithms KEY-WRAP ::= {
        kwa-aes128-wrap | kwa-aes256-wrap,
        ...
    }

    kaa-X448 KEY-AGREE ::= {
        IDENTIFIER id-X448
        PARAMS ARE absent
        PUBLIC-KEYS {pk-X448}
        UKM -- TYPE no ASN.1 wrapping  -- ARE preferredPresent
        SMIME-CAPS {
           TYPE AlgorithmIdentifier{KEY-WRAP, {KeyWrapAlgorithms}}
           IDENTIFIED BY id-X448 }
    }

    pk-X448 PUBLIC-KEY ::= {
        IDENTIFIER id-X448
        -- KEY no ASN.1 wrapping --
        PARAMS ARE absent
        CERT-KEY-USAGE { keyAgreement }
        PRIVATE-KEY CurvePrivateKey
    }

   CurvePrivateKey ::= OCTET STRING


   END
```

10. Examples

This section contains illustrations of EdDSA public keys and certificates, illustrating parameter choices.

10.1. Example Ed25519 Public Key

An example of a Ed25519 public key:

```
      Public Key Information:
          Public Key Algorithm: Ed25519
          Algorithm Security Level: High

      Public Key Usage:

      Public Key ID: 9b1f5eeded043385e4f7bc623c5975b90bc8bb3b

      -----BEGIN PUBLIC KEY-----
      MCowBQYDK2VwAyEAGb9ECWmEzf6FQbrBZ9w7lshQhqowtrbLDFw4rXAxZuE=
      -----END PUBLIC KEY-----
```

10.2. Example X25519 Certificate

An example of a self issued PKIX certificate using Ed25519 to sign a X25519 public key would be:

```
    0 300: SEQUENCE {
    4 223:   SEQUENCE {
    7   3:     [0] {
    9   1:       INTEGER 2
         :       }
   12   8:     INTEGER 56 01 47 4A 2A 8D C3 30
   22   5:     SEQUENCE {
   24   3:       OBJECT IDENTIFIER
         :         Ed 25519 signature algorithm { 1 3 101 112 }
         :       }
   29  25:     SEQUENCE {
   31  23:       SET {
   33  21:         SEQUENCE {
   35   3:           OBJECT IDENTIFIER commonName (2 5 4 3)
   40  14:           UTF8String 'IETF Test Demo'
```

```
          :           }
          :         }
          :       }
    56  30:     SEQUENCE {
    58  13:       UTCTime 01/08/2016 12:19:24 GMT
    73  13:       UTCTime 31/12/2040 23:59:59 GMT
          :       }
    88  25:     SEQUENCE {
    90  23:       SET {
    92  21:         SEQUENCE {
    94   3:           OBJECT IDENTIFIER commonName (2 5 4 3)
    99  14:           UTF8String 'IETF Test Demo'
          :           }
          :         }
          :       }
   115  42:     SEQUENCE {
```

```
   117   5:       SEQUENCE {
   119   3:         OBJECT IDENTIFIER
          :           ECDH 25519 key agreement { 1 3 101 110 }
          :         }
   124  33:       BIT STRING
          :         85 20 F0 09 89 30 A7 54 74 8B 7D DC B4 3E F7 5A
          :         0D BF 3A 0D 26 38 1A F4 EB A4 A9 8E AA 9B 4E 6A
          :       }
   159  69:     [3] {
   161  67:       SEQUENCE {
   163  15:         SEQUENCE {
   165   3:           OBJECT IDENTIFIER basicConstraints (2 5 29 19)
   170   1:           BOOLEAN TRUE
   173   5:           OCTET STRING, encapsulates {
   175   3:             SEQUENCE {
   177   1:               BOOLEAN FALSE
          :               }
          :             }
          :           }
   180  14:         SEQUENCE {
   182   3:           OBJECT IDENTIFIER keyUsage (2 5 29 15)
   187   1:           BOOLEAN FALSE
   190   4:           OCTET STRING, encapsulates {
   192   2:             BIT STRING 3 unused bits
          :               '10000'B (bit 4)
```

```
        :             }
        :           }
 196  32:         SEQUENCE {
 198   3:           OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29 14)
 203   1:           BOOLEAN FALSE
 206  22:           OCTET STRING, encapsulates {
 208  20:             OCTET STRING
        :               9B 1F 5E ED ED 04 33 85 E4 F7 BC 62 3C 59 75
        :               B9 0B C8 BB 3B
        :             }
        :           }
        :         }
        :       }
        :     }
 230   5:   SEQUENCE {
 232   3:     OBJECT IDENTIFIER
        :       Ed 25519 signature algorithm { 1 3 101 112 }
        :     }
 237  65:   BIT STRING
        :     AF 23 01 FE DD C9 E6 FF C1 CC A7 3D 74 D6 48 A4
        :     39 80 82 CD DB 69 B1 4E 4D 06 EC F8 1A 25 CE 50
        :     D4 C2 C3 EB 74 6C 4E DD 83 46 85 6E C8 6F 3D CE
        :     1A 18 65 C5 7A C2 7B 50 A0 C3 50 07 F5 E7 D9 07
```

```
        :   }
```

```
-----BEGIN CERTIFICATE-----
MIIBLDCB36ADAgECAghWAUdKKo3DMDAFBgMrZXAwGTEXMBUGA1UEAwwOSUVURiBUZX
N0IERlbW8wHhcNMTYwODAxMTIxOTI0WhcNNDAxMjMxMjM1OTU5WjAZMRcwFQYDVQQD
DA5JRVRGIFRlc3QgRGVtbzAqMAUGAytlbgMhAIUg8AmJMKdUdIt93LQ+91oNvzoNJj
ga9OukqY6qm05qo0UwQzAPBgNVHRMBAf8EBTADAQEAMA4GA1UdDwEBAAQEAwIDCDAg
BgNVHQ4BAQAEFgQUmx9e7e0EM4Xk97xiPFl1uQvIuzswBQYDK2VwA0EAryMB/t3J5v
/BzKc9dNZIpDmAgs3babFOTQbs+BolzlDUwsPrdGxO3YNGhW7Ibz3OGhhlxXrCe1Cg
w1AH9efZBw==
-----END CERTIFICATE-----
```

10.3. Examples of Ed25519 Private Key

An example of an Ed25519 private key without the public key:

```
-----BEGIN PRIVATE KEY-----
MC4CAQAwBQYDK2VwBCIEINTuctv5E1hK1bbY8fdp+K06/nwoy/HU++CXqI9EdVhC
```

```
-----END PRIVATE KEY-----
```

The same item dumped as asn1 yields:

```
 0 30   46: SEQUENCE {
 2 02    1:   INTEGER 0
 5 30    5:   SEQUENCE {
 7 06    3:     OBJECT IDENTIFIER
          :       Ed 25519 signature algorithm { 1 3 101 112 }
          :     }
12 04   34:   OCTET STRING
          :     04 20 D4 EE 72 DB F9 13 58 4A D5 B6 D8 F1 F7 69
          :     F8 AD 3A FE 7C 28 CB F1 D4 FB E0 97 A8 8F 44 75
          :     58 42
          :   }
```

Note that the value of the private key is:

```
D4 EE 72 DB F9 13 58 4A D5 B6 D8 F1 F7 69 F8 AD
3A FE 7C 28 CB F1 D4 FB E0 97 A8 8F 44 75 58 42
```

An example of the same Ed25519 private key encoded with an attribute and the public key:

```
-----BEGIN PRIVATE KEY-----
MHICAQEwBQYDK2VwBCIEINTuctv5E1hK1bbY8fdp+K06/nwoy/HU++CXqI9EdVhC
oB8wHQYKKoZIhvcNAQkJFDEPDA1DdXJkbGUgQ2hhaXJzgSEAGb9ECWmEzf6FQbrB
Z9w7lshQhqowtrbLDFw4rXAxZuE=
-----END PRIVATE KEY-----
```

The same item dumped as asn1 yields:

```
  0 114: SEQUENCE {
  2   1:   INTEGER 1
  5   5:   SEQUENCE {
  7   3:     OBJECT IDENTIFIER '1 3 101 112'
       :     }
 12  34:   OCTET STRING, encapsulates {
 14  32:     OCTET STRING D4 EE 72 DB F9 13 58 4A D5 B6 D8 F1 F7
                 69 F8 AD 3A FE 7C 28 CB F1 D4 FB E0 97 A8 8F 44
                 75 58 42
```

```
           :     }
    48   31:   [0] {
    50   29:     SEQUENCE {
    52   10:       OBJECT IDENTIFIER '1 2 840 113549 1 9 9 20'
    64   15:       SET {
    66   13:         UTF8String 'Curdle Chairs'
           :         }
           :       }
           :     }
   81   33:   [1] 00 19 BF 44 09 69 84 CD FE 85 41 BA C1 67 DC 3B
                  96 C8 50 86 AA 30 B6 B6 CB 0C 5C 38 AD 70 31 66
                  E1
           :   }
```

11. Acknowledgements

Text and/or inspiration were drawn from [RFC5280], [RFC3279], [RFC4055], [RFC5480], and [RFC5639].

The following people discussed the document and provided feedback: Klaus Hartke, Ilari Liusvaara, Erwann Abalea, Rick Andrews, Rob Stradling, James Manger, Nikos Mavrogiannopoulos, Russ Housley, David Benjamin, Brian Smith, and Alex Wilson.

A big thank you to Symantec for kindly donating the OIDs used in this draft.

12. IANA Considerations

IANA is requested to assign a module OID from the "SMI for PKIX Module Identifier" registry for the ASN.1 module in Section 9.

13. Security Considerations

The security considerations of [RFC5280], [RFC7748], and [RFC8032] apply accordingly.

The procedures for going from a private key to a public key are different for when used with Diffie-Hellman and when used with Edwards Signatures. This means that the same public key cannot be used for both ECDH and EdDSA.

14. References

14.1. 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/info/rfc2119>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <https://www.rfc-editor.org/info/rfc5480>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <https://www.rfc-editor.org/info/rfc5958>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

14.2. Informative References

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, DOI 10.17487/RFC3279, April 2002, <https://www.rfc-editor.org/info/rfc3279>.

[RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, DOI 10.17487/RFC4055, June 2005, <https://www.rfc-editor.org/info/rfc4055>.

[RFC5639] Lochter, M. and J. Merkle, "Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation", RFC 5639, DOI 10.17487/RFC5639, March 2010, <https://www.rfc-editor.org/info/rfc5639>.

[RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <https://www.rfc-editor.org/info/rfc7468>.

Appendix A. Invalid Encodings

There are a number of things that need to be dealt with when a new key part is decoded and imported into the system. A partial list of these includes:

- ASN.1 encoding errors: Two items are highlighted here. First, the use of an OCTET STRING rather than a BIT STRING for the public key. This was an incorrect copy of the structure from [RFC5958] which was corrected before publication. However, any early implementation may have this wrong. Second, the value of the version field is required to be 0 if the publicKey is absent and 1 if present. This is called out in [RFC5958] but is not duplicated in the main text.

- Key encoding errors: Both [RFC7748] and [RFC8032] have formatting requirements for keys that need to be enforced. In some cases the enforcement is done at the time of importing, for example doing masking or a mod p operation. In other cases the enforcement is done by rejecting the keys and having an import failure.

- Key mismatch errors: If a public key is provided, it may not agree with the private key either because it is wrong or the wrong algorithm was used.

Some systems are also going to be stricter on what they accept. As stated in [RFC5958], BER decoding of OneAsymmetricKey objects is a requirement for compliance. Despite this requirement, some acceptors will only decode DER formats. The following is a BER encoding of a private key, as such is is valid, but it may not be accepted by many systems.

```
-----BEGIN PRIVATE KEY-----
MIACAQAwgAYDK2VwAAAEIgQg1O5y2/kTWErVttjx92n4rTr+fCjL8dT74Jeoj0R1W
EIAAA==
-----END PRIVATE KEY-----
```

What follows here is a brief sampling of some incorrect keys.

In the following example, the private key does not match the masking requirements for X25519. For this example the top bits are set to zero and the bottom three bits are set to 001.

```
-----BEGIN PRIVATE KEY-----
MFMCAQEwBQYDK2VuBCIEIPj///////////////////////////////////////8/oS
MDIQCEfA0sN1I082XmYJVRh6NzWg92E9FgnTpqTYxTrqpaIg==
-----END PRIVATE KEY-----
```

In the following examples, the key is the wrong length because an all zero byte has been removed. In one case the first byte has been removed, in the other case the last byte has been removed.

```
-----BEGIN PRIVATE KEY-----
MFICAQEwBQYDK2VwBCIEIC3GfeUYbZGTAhwLEE2cbvJL7ivTlcy17VottfN6L8HwoS
IDIADBfk2Lv/J8H7YYwj/OmIcDx++jzVkKrKwS0/HjyQyM
-----END PRIVATE KEY-----
```

```
-----BEGIN PRIVATE KEY-----
MFICAQEwBQYDK2VwBCIEILJXn1VaLqvausjUaZexwI/ozmOFjfEk78KcYN+7hsNJoS
IDIACdQhJwzi/MCGcsQeQnIUh2JFybDxSrZxuLudJmpJLk
-----END PRIVATE KEY-----
```

Authors' Addresses

Simon Josefsson
SJD AB

Email: simon@josefsson.org

Jim Schaad
August Cellars

Email: ietf@augustcellars.com