

Internet-Draft  
Intended status: Standards Track  
Expires: 22 March 2017

R. Housley  
Vigil Security  
22 September 2016

Using ChaCha20-Poly1305 Authenticated Encryption  
in the Cryptographic Message Syntax (CMS)

<draft-ietf-curdle-cms-chacha20-poly1305-02.txt>

Abstract

This document describes the conventions for using ChaCha20-Poly1305 Authenticated Encryption in the Cryptographic Message Syntax (CMS). ChaCha20-Poly1305 is an authenticated encryption algorithm constructed of the ChaCha stream cipher and Poly1305 authenticator.

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 <http://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 22 March 2017.

Copyright Notice

Copyright (c) 2016 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 (<http://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.

## 1. Introduction

This document specifies the conventions for using the ChaCha20-Poly1305 Authenticated Encryption as the content-authenticated-encryption algorithm with the Cryptographic Message Syntax (CMS) [CMS] authenticated-enveloped-data content type [AUTHENV].

ChaCha [CHACHA] is a stream cipher developed by D. J. Bernstein in 2008. It is a refinement of Salsa20, which is one of the ciphers in the eSTREAM portfolio [ESTREAM].

ChaCha20 is the 20-round variant of ChaCha; it requires a 256-bit key and a 96-bit nonce. ChaCha20 is described in [FORIETF].

Poly1305 [POLY1305] is a Wegman-Carter, one-time authenticator designed by D. J. Bernstein. Poly1305 produces a 16-byte authentication tag; it requires a 256-bit, single-use key. Poly1305 is also described in [FORIETF].

ChaCha20 and Poly1305 have been designed for high performance software implementations. They can typically be implemented with few resources and inexpensive operations, making them suitable on a wide range of systems. They have also been designed to minimize leakage of information through side channels.

### 1.1. The ChaCha20 and Poly1305 AEAD Construction

ChaCha20 and Poly1305 have been combined to create an Authenticated Encryption with Associated Data (AEAD) algorithm [AEAD]. This AEAD algorithm is often referred to as AEAD\_CHACHA20\_POLY1305, and it is described in [FORIETF].

AEAD\_CHACHA20\_POLY1305 accepts four inputs: a 256-bit key, a 96-bit nonce, an arbitrary length plaintext, and an arbitrary length additional authenticated data (AAD). As the name implies, a nonce value cannot be used securely more than once with the same key.

A high-level summary of AEAD\_CHACHA20\_POLY1305 authenticated encryption processing is:

- 1) A Poly1305 one-time key is generated from the 256-bit key and and nonce using the procedure described in Section 2.6 of [FORIETF].
- 2) The ChaCha20 encryption function is used to encrypt the plaintext, using the same key and nonce, and with the initial counter set to 1.

- 3) The Poly1305 function is used with the Poly1305 key from step 1, and a buffer constructed as a concatenation of the AAD, padding1, the ciphertext, padding2, the length of the AAD in octets, and the length of the ciphertext in octets. The padding fields contain up to 15 octets, with all bits set to zero, and the padding brings the total length of the buffer so far to an integral multiple of 16. If the buffer length was already an integral multiple of 16 octets, then the padding field is zero octets. The length fields contain 64-bit little-endian integers.

AEAD\_CHACHA20\_POLY1305 produces ciphertext of the same length as the plaintext and a 128-bit authentication tag.

AEAD\_CHACHA20\_POLY1305 authenticated decryption processing is similar to the encryption processing. Of course, the roles of ciphertext and plaintext are reversed, so the ChaCha20 encryption function is applied to the ciphertext, producing the plaintext. The Poly1305 function is run over the AAD and the ciphertext, not the plaintext, and the resulting authentication tag is bitwise compared to the received authentication tag. The message is authenticated if and only if the calculated and received authentication tags match.

## 1.2. 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 RFC 2119 [STDWORDS].

## 1.3. ASN.1

CMS values are generated using ASN.1 [X680], which uses the Basic Encoding Rules (BER) and the Distinguished Encoding Rules (DER) [X690].

## 2. Automated Key Management

The reuse of an AEAD\_CHACHA20\_POLY1305 nonce value with the same key destroys the security guarantees. As a result, it can be extremely difficult to use AEAD\_CHACHA20\_POLY1305 securely when using statically configured keys. For safety's sake, implementations MUST use an automated key management system [KEYMGMT].

The CMS authenticated-enveloped-data content type supports four general key management techniques:

- Key Transport: the content-authenticated-encryption key is encrypted in the recipient's public key;
- Key Agreement: the recipient's public key and the sender's private key are used to generate a pairwise symmetric key, then the content-authenticated-encryption key is encrypted in the pairwise symmetric key;
- Symmetric Key-Encryption Keys: the content-authenticated-encryption key is encrypted in a previously distributed symmetric key-encryption key; and
- Passwords: the content-authenticated-encryption key is encrypted in a key-encryption key that is derived from a password or other shared secret value.

All of these key management techniques meet the automated key management system requirement as long as a fresh content-authenticated-encryption key is generated for the protection of each content. Note that some of these key management techniques use one key-encryption key to encrypt more than one content-authenticated-encryption key during the system life cycle. As long as fresh content-authenticated-encryption key is used each time, AEAD\_CHACHA20\_POLY1305 can be used safely with the CMS authenticated-enveloped-data content type.

In addition to these four general key management techniques, CMS supports other key management techniques. See Section 6.2.5 of [CMS]. Since the properties of these key management techniques are unknown, no statement can be made about whether these key management techniques meet the automated key management system requirement. Designers and implementers must perform their own analysis if one of these other key management techniques is supported.

### 3. Using the AEAD\_CHACHA20\_POLY1305 Algorithm with AuthEnvelopedData

This section specifies the conventions employed by CMS implementations that support content authenticated encryption using the AEAD\_CHACHA20\_POLY1305 algorithm.

Content authenticated encryption algorithm identifiers are located in the AuthEnvelopedData EncryptedContentInfo contentEncryptionAlgorithm field.

Content authenticated encryption algorithms are used to encipher the content located in the AuthEnvelopedData EncryptedContentInfo encryptedContent field and to provide the message authentication code for the AuthEnvelopedData mac field. Note that the message authentication code provides integrity protection for both the AuthEnvelopedData authAttrs and the AuthEnvelopedData EncryptedContentInfo encryptedContent.

Neither the plaintext content nor the optional AAD inputs need to be padded prior to invoking the AEAD\_CHACHA20\_POLY1305 algorithm.

There is one algorithm identifiers for the AEAD\_CHACHA20\_POLY1305 algorithm:

```
id-alg-AEADChaCha20Poly1305 OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
    pkcs9(9) smime(16) alg(3) TBD1 }
```

The AlgorithmIdentifier parameters field MUST be present, and the parameters field must contain a AEADChaCha20Poly1305Nonce:

```
AEADChaCha20Poly1305Nonce ::= OCTET STRING (SIZE(12))
```

The AEADChaCha20Poly1305Nonce contains a 12-octet nonce. With the CMS, the content-authenticated-encryption key is normally used for a single content. Within the scope of any content-authenticated-encryption key, the nonce value MUST be unique. That is, the set of nonce values used with any given key MUST NOT contain any duplicate values.

#### 4. S/MIME Capabilities

```
{{{ This can be written once the Object Identifier is assigned. }}}}
```

#### 5. IANA Considerations

IANA is requested to add the following entry in the SMI Security for S/MIME Algorithms (1.2.840.113549.1.9.16.3) registry:

```
TBD1    id-alg-AEADChaCha20Poly1305    [This Document]
```

IANA is requested to add the following entry in the SMI Security for S/MIME Module Identifier (1.2.840.113549.1.9.16.0) registry:

```
TBD2    id-mod-CMS-AEADChaCha20Poly1305    [This Document]
```

## 6. Security Considerations

The CMS AuthEnvelopedData provides all of the tools needed to avoid reuse of the same nonce value under the same key. Automated key management is discussed in Section 2.

When using AEAD\_CHACHA20\_POLY1305, the resulting ciphertext is always the same size as the original plaintext. Some other mechanism needs to be used in conjunction with AEAD\_CHACHA20\_POLY1305 if disclosure of the size of the plaintext is a concern.

The amount of encrypted data possible in a single invocation of AEAD\_CHACHA20\_POLY1305 is  $2^{32}-1$  blocks of 64 octets each, because of the size of the block counter field in the ChaCha20 block function. This gives a total of 247,877,906,880 octets, which likely to be sufficient to handle the size of any CMS content type. Note that ciphertext length field in the authentication buffer will accomodate  $2^{64}$  octets, which is much larger than necessary.

The AEAD\_CHACHA20\_POLY1305 construction is a novel composition of ChaCha20 and Poly1305. A security analysis of this composition is given in [PROCTER].

Implementations must randomly generate content-authenticated-encryption keys. The use of inadequate pseudo-random number generators (PRNGs) to generate cryptographic keys can result in little or no security. An attacker may find it much easier to reproduce the PRNG environment that produced the keys, searching the resulting small set of possibilities, rather than brute force searching the whole key space. The generation of quality random numbers is difficult. RFC 4086 [RANDOM] offers important guidance in this area.

## 7. Acknowledgements

Thanks to Jim Schaad for his review and insightful comments.

## 8. Normative References

- [AUTHENV] Housley, R., "Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type", RFC 5083, November 2007.
- [CMS] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 5652, September 2009.
- [FORIETF] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 7539, May 2015.

- [STDWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [X680] ITU-T, "Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, 2015.
- [X690] ITU-T, "Information technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, 2015.

## 9. Informative References

- [AEAD] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, January 2008.
- [CHACHA] Bernstein, D., "ChaCha, a variant of Salsa20", January 2008, <<http://cr.yp.to/chacha/chacha-20080128.pdf>>.
- [ESTREAM] Babbage, S., DeCanniere, C., Cantenaut, A., Cid, C., Gilbert, H., Johansson, T., Parker, M., Preneel, B., Rijmen, V., and M. Robshaw, "The eSTREAM Portfolio (rev. 1)", September 2008, <<http://www.ecrypt.eu.org/stream/finallist.html>>.
- [KEYMGMT] Bellare, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, June 2005.
- [POLY1305] Bernstein, D., "The Poly1305-AES message-authentication code.", March 2005, <<http://cr.yp.to/mac/poly1305-20050329.pdf>>.
- [PROCTER] Procter, G., "A Security Analysis of the Composition of ChaCha20 and Poly1305", August 2014, <<http://eprint.iacr.org/2014/613.pdf>>.
- [RANDOM] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Recommendations for Security", BCP 106, RFC 4086, June 2005.

## Appendix: ASN.1 Module

```
CMS-AEADChaCha20Poly1305
  { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
    pkcs-9(9) smime(16) modules(0) TBD2 }

DEFINITIONS IMPLICIT TAGS ::= BEGIN

IMPORTS
  CONTENT-ENCRYPTION
  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) };

-- EXPORTS All

AEADContentEncryptionAlgs CONTENT-ENCRYPTION ::=
  { cea-AEADChaCha20Poly1305, ... }

cea-AEADChaCha20Poly1305 CONTENT-ENCRYPTION ::= {
  IDENTIFIER id-alg-AEADChaCha20Poly1305
  PARAMS TYPE AEADChaCha20Poly1305Nonce ARE required
  SMIME-CAPS { IDENTIFIED BY id-alg-AEADChaCha20Poly1305 } }

id-alg-AEADChaCha20Poly1305 OBJECT IDENTIFIER ::=
  { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
    pkcs9(9) smime(16) alg(3) TBD1 }

AEADChaCha20Poly1305Nonce ::= OCTET STRING (SIZE(12))

END
```

## Author's Addresses

Russell Housley  
Vigil Security, LLC  
918 Spring Knoll Drive  
Herndon, VA 20170  
USA  
  
EMail: housley@vigilsec.com



Internet-Draft  
Intended status: Standards Track  
Expires: 8 March 2017

R. Housley  
Vigil Security  
8 September 2016

Use of the Elliptic Curve Diffie-Hellman Key Agreement Algorithm  
with X25519 and X448 in the Cryptographic Message Syntax (CMS)

<draft-ietf-curdle-cms-ecdh-new-curves-01.txt>

## Abstract

This document describes the conventions for using Elliptic Curve Diffie-Hellman (ECDH) key agreement algorithm using curve25519 and curve448 in the Cryptographic Message Syntax (CMS).

## 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 <http://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 8 March 2017.

## Copyright Notice

Copyright (c) 2016 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 (<http://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.

## 1. Introduction

This document describes the conventions for using Elliptic Curve Diffie-Hellman (ECDH) key agreement using curve25519 and curve448 [CURVE] in the Cryptographic Message Syntax (CMS) [CMS]. Key agreement is supported in three CMS content types: the enveloped-data content type [CMS], authenticated-data content type [CMS], and the authenticated-enveloped-data content type [AUTHENV].

The conventions for using some Elliptic Curve Cryptography (ECC) algorithms in CMS are described in [CMSECC]. These conventions cover the use of ECDH with some curves other than curve25519 and curve448 [CURVE]. Those other curves are not deprecated, but support for curve25519 and curve448 is encouraged.

When these two curves are used with with Diffie-Hellman key agreement, they are referred to as X25519 and X448.

### 1.1. 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 RFC 2119 [STDWORDS].

### 1.2. ASN.1

CMS values are generated using ASN.1 [X680], which uses the Basic Encoding Rules (BER) and the Distinguished Encoding Rules (DER) [X690].

## 2. Key Agreement

In 1976, Diffie and Hellman describe a means for two parties to agree upon a shared secret value in manner that prevents eavesdroppers from learning the shared secret value [DH1976]. This secret may then be converted into pairwise symmetric keying material for use with other cryptographic algorithms. Over the years, many variants of this fundamental technique have been developed. This document describes the conventions for using Ephemeral-Static Elliptic Curve Diffie-Hellman (ECDH) key agreement using X25519 and X448 [CURVE].

The originator uses an ephemeral public/private key pair that is generated on the same elliptic curve as the public key of the recipient. The ephemeral key pair is used for a single CMS protected content type, and then it is discarded. The originator obtains the recipient's static public key from the recipient's certificate [PROFILE].

X25519 is described in Section 6.1 of [CURVE], and X448 is described in Section 6.2 of [CURVE]. Since curve25519 and curve448 have cofactors of 8 and 4, respectively, an input point of small order will eliminate any contribution from the other party's private key. As described in Section 7 of [CURVE], implementations MAY detect this situation by checking for the all-zero output.

In [CURVE], the shared secret value that is produced by ECDH is called K. (In some other specifications, the shared secret value is called Z.) A key derivation function (KDF) is used to produce a pairwise key-encryption key from K, the length of the key-encryption key, and the DER-encoded ECC-CMS-SharedInfo structure [CMSECC].

The ECC-CMS-SharedInfo definition from [CMSECC] is repeated here for convenience.

```
ECC-CMS-SharedInfo ::= SEQUENCE {  
    keyInfo          AlgorithmIdentifier,  
    entityUInfo [0] EXPLICIT OCTET STRING OPTIONAL,  
    suppPubInfo [2] EXPLICIT OCTET STRING }
```

The ECC-CMS-SharedInfo keyInfo field contains the object identifier of the key-encryption algorithm and associated parameters. This algorithm will be used to wrap the content-encryption key. In this specification, the AES Key Wrap algorithm identifier has absent parameters.

The ECC-CMS-SharedInfo entityUInfo field optionally contains additional keying material supplied by the sending agent. Note that [CMS] requires implementations to accept a KeyAgreeRecipientInfo SEQUENCE that includes the ukm field. If the ukm field is present, the ukm is placed in the entityUInfo field. The ukm value need not be longer than the key-encryption key that will be produced by the KDF. When present, the ukm ensures that a different key-encryption key is generated, even when the originator ephemeral private key is improperly used more than once.

The ECC-CMS-SharedInfo suppPubInfo field contains the length of the generated key-encryption key, in bits, represented as a 32-bit number. For example, the key length for AES-256 would be 0x00000100.

## 2.1. ANSI-X9.63-KDF

The ANSI-X9.63-KDF key derivation function is a simple construct based on a one-way hash function described in ANSI X9.63 [X963]. This KDF is also described in Section 3.6.1 of [SEC1].

Three values are concatenated to produce the input string to the KDF:

1. The shared secret value generated by ECDH, K.
2. The iteration counter, starting with one, as described below.
3. The DER-encoded ECC-CMS-SharedInfo structure.

To generate a key-encryption key, generates one or more KM blocks, with the counter starting at 0x00000001, and incrementing the counter for each subsequent KM block until enough material has been generated. The KM blocks are concatenated left to right:

$$KM(i) = \text{Hash}(K \parallel \text{INT32}(\text{counter}=i) \parallel \text{DER}(\text{ECC-CMS-SharedInfo}))$$

$$KEK = KM(\text{counter}=1) \parallel KM(\text{counter}=2) \dots$$

KEK is the pairwise key-encryption key.

## 2.2. HKDF

The HKDF key derivation function is a robust construct based on a one-way hash function described in RFC 5869 [HMAC]. HKDF is comprised of two steps: HKDF-Extract followed by HKDF-Expand.

Three values are used as inputs to the HKDF:

1. The shared secret value generated by ECDH, K.
2. The length in octets of the keying data to be generated.
3. The DER-encoded ECC-CMS-SharedInfo structure.

The ECC-CMS-SharedInfo structure includes the ukm. This field is optional, and if it is present, the ukm is also used as the HKDF salt.

The length of the generated key-encryption key is used two places, once in bits, and once in octets. The ECC-CMS-SharedInfo structure includes the length of the generated key-encryption key in bits. The HKDF-Expand function takes an argument for the length of the generated key-encryption key in octets.

In summary:

$$\text{if ukm is provided, then salt} = \text{ukm, else salt} = \text{zero}$$

$$\text{PRK} = \text{HKDF-Extract}(\text{salt}, K)$$

$$\text{KEK} = \text{HKDF-Expand}(\text{PRK}, \text{DER}(\text{ECC-CMS-SharedInfo}), \text{SizeInOctets}(\text{KEK}))$$

KEK is the pairwise key-encryption key.

### 3. Enveloped-data Conventions

The CMS enveloped-data content type [CMS] consists of an encrypted content and wrapped content-encryption keys for one or more recipients. The ECDH key agreement algorithm is used to generate a pairwise key-encryption key between the originator and a particular recipient. Then, the key-encryption key is used to wrap the content-encryption key for that recipient. When there more than one recipient, the same content-encryption key is wrapped for each of them.

A compliant implementation MUST meet the requirements for constructing an enveloped-data content type in Section 6 of [CMS].

A content-encryption key MUST be randomly generated for each instance of an enveloped-data content type. The content-encryption key is used to encrypt the content.

#### 3.1. EnvelopedData Fields

The enveloped-data content type is ASN.1 encoded using the EnvelopedData syntax. The fields of the EnvelopedData syntax MUST be populated as described in [CMS]; for the recipients that use X25519 or X448 the RecipientInfo kari choice MUST be used.

#### 3.2. KeyAgreeRecipientInfo Fields

The fields of the KeyAgreeRecipientInfo syntax MUST be populated as described in this section when X25519 or X448 is employed for one or more recipients.

The KeyAgreeRecipientInfo version MUST be 3.

The KeyAgreeRecipientInfo originator provides three alternatives for identifying the originator's public key, and the originatorKey alternative MUST be used. The originatorKey MUST contain an ephemeral key for the originator. The originatorKey algorithm field MUST contain the id-ecPublicKey object identifier along with ECPParameters as specified in [PKIXECC]. The originator's ephemeral public key MUST be encoded using the type ECPoint as specified in [CMSECC]. As a convenience, the definitions are repeated here:

```
id-ecPublicKey OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) ansi-X9-62(10045) keyType(2) 1 }

ECPoint ::= OCTET STRING
```

```

ECParameters ::= CHOICE {
    namedCurve          OBJECT IDENTIFIER
    -- implicitCurve    NULL
    -- specifiedCurve   SpecifiedECDomain -- }

```

The object identifiers for X25519 and X448 have been assigned in [ID.curdle-pkix]. They are repeated below for convenience.

When using X25519, the ECPoint contains exactly 32 octets, and the ECParameters namedCurve MUST contain the following object identifier:

```
id-X25519 OBJECT IDENTIFIER ::= { 1.3.101.110 }
```

When using X448, the ECPoint contains exactly 56 octets, and the ECParameters namedCurve MUST contain the following object identifier:

```
id-X448 OBJECT IDENTIFIER ::= { 1.3.101.111 }
```

KeyAgreeRecipientInfo ukm is optional. Note that [CMS] requires implementations to accept a KeyAgreeRecipientInfo SEQUENCE that includes the ukm field. If present, the ukm is placed in the entityUIInfo field of the ECC-CMS-SharedInfo as input to the KDF. The ukm value need not be longer than the key-encryption key produced by the KDF.

KeyAgreeRecipientInfo keyEncryptionAlgorithm MUST contain the object identifier of the key-encryption algorithm that will be used to wrap the content-encryption key. The conventions for using AES-128, AES-192, and AES-256 in the key wrap mode are specified in [CMSAES].

KeyAgreeRecipientInfo recipientEncryptedKeys includes a recipient identifier and encrypted key for one or more recipients. The RecipientEncryptedKey KeyAgreeRecipientIdentifier MUST contain either the issuerAndSerialNumber identifying the recipient's certificate or the RecipientKeyIdentifier containing the subject key identifier from the recipient's certificate. In both cases, the recipient's certificate contains the recipient's static X25519 or X448 public key. RecipientEncryptedKey EncryptedKey MUST contain the content-encryption key encrypted with the pairwise key-encryption key using the algorithm specified by the KeyWrapAlgorithm.

#### 4. Authenticated-data Conventions

The CMS authenticated-data content type [CMS] consists an authenticated content, a message authentication code (MAC), and encrypted authentication keys for one or more recipients. The ECDH key agreement algorithm is used to generate a pairwise key-encryption key between the originator and a particular recipient. Then, the

key-encryption key is used to wrap the authentication key for that recipient. When there more than one recipient, the same authentication key is wrapped for each of them.

A compliant implementation MUST meet the requirements for constructing an authenticated-data content type in Section 9 of [CMS].

A authentication key MUST be randomly generated for each instance of an authenticated-data content type. The authentication key is used to compute the MAC over the content.

#### 4.1. AuthenticatedData Fields

The authenticated-data content type is ASN.1 encoded using the AuthenticatedData syntax. The fields of the AuthenticatedData syntax MUST be populated as described in [CMS]; for the recipients that use X25519 or X448 the RecipientInfo kari choice MUST be used.

#### 4.2. KeyAgreeRecipientInfo Fields

The fields of the KeyAgreeRecipientInfo syntax MUST be populated as described in Section 3.2 of this document.

### 5. Authenticated-Enveloped-data Conventions

The CMS authenticated-enveloped-data content type content type [AUTHENV] consists of an authenticated and encrypted content and encrypted content-authenticated-encryption keys for one or more recipients. The ECDH key agreement algorithm is used to generate a pairwise key-encryption key between the originator and a particular recipient. Then, the key-encryption key is used to wrap the content-authenticated-encryption key for that recipient. When there more than one recipient, the same content-authenticated-encryption key is wrapped for each of them.

A compliant implementation MUST meet the requirements for constructing an authenticated-data content type in Section 2 of [AUTHENV].

A content-authenticated-encryption key MUST be randomly generated for each instance of an authenticated-enveloped-data content type. The content-authenticated-encryption key key is used to authenticate and encrypt the content.

#### 5.1. AuthEnvelopedData Fields

The authenticated-enveloped-data content type is ASN.1 encoded using

the AuthEnvelopedData syntax. The fields of the AuthEnvelopedData syntax MUST be populated as described in [AUTHENV]; for the recipients that use X25519 or X448 the RecipientInfo kari choice MUST be used.

#### 5.2. KeyAgreeRecipientInfo Fields

The fields of the KeyAgreeRecipientInfo syntax MUST be populated as described in Section 3.2 of this document.

### 6. Certificate Conventions

RFC 5280 [PROFILE] specifies the profile for using X.509 Certificates in Internet applications. A recipient static public key is needed for X25519 or X448, and the originator obtains that public key from the recipient's certificate. The conventions in this section augment RFC 5280 [PROFILE].

The id-ecPublicKey object identifier continues to identify the static ECDH public key for the recipient. The associated EcpkParameters parameters structure is specified in [PKIXALG], and the namedCurve alternative MUST be used. The object identifiers from Section 3.2 of this document are used for X25519 and X448. The EcpkParameters parameters structure is repeated here for convenience:

```
EcpkParameters ::= CHOICE {  
    ecParameters    ECPParameters,  
    namedCurve      OBJECT IDENTIFIER,  
    implicitlyCA     NULL }
```

The certificate issuer MAY use indicate the intended usage for the certified public key by including the key usage certificate extension as specified in Section 4.2.1.3 of [PROFILE]. If the keyUsage extension is present in a certificate that conveys an ECDH static public key, then the key usage extension MUST set the keyAgreement bit.

### 7. Key Agreement Algorithm Identifiers

The following object identifiers are assigned to indicate ECDH with HKDF using various one-way hash functions. These are expected to be used as AlgorithmIdentifiers with a parameter that specifies the key-encryption algorithm.



```
dhSinglePass-stdDH-hkdf-sha256-scheme OBJECT IDENTIFIER ::= {  
    TBD0 }
```

```
dhSinglePass-stdDH-hkdf-sha384-scheme OBJECT IDENTIFIER ::= {  
    TBD1 }
```

```
dhSinglePass-stdDH-hkdf-sha512-scheme OBJECT IDENTIFIER ::= {  
    TBD2 }
```

## 8. SMIMECapabilities Attribute Conventions

A sending agent MAY announce to other agents that it supports ECDH key agreement using the SMIMECapabilities signed attribute in a signed message [SMIME] or a certificate [CERTCAP]. Following the pattern established in [CMSECC], the SMIMECapabilities associated with ECDH carries a DER-encoded object identifier that identifies support for ECDH in conjunction with a particular KDF, and it includes a parameter that names the key wrap algorithm.

The following SMIMECapabilities values (in hexadecimal) from [CMSECC] might be of interest to implementations that support X25519 and X448:

```
ECDF with ANSI-X9.63-KDF using SHA-256; uses AES-128 key wrap:  
    30 15 06 06 2B 81 04 01 0B 01 30 0B 06 09 60 86 48 01 65 03 04  
    01 05
```

```
ECDF with ANSI-X9.63-KDF using SHA-384; uses AES-128 key wrap:  
    30 15 06 06 2B 81 04 01 0B 02 30 0B 06 09 60 86 48 01 65 03 04  
    01 05
```

```
ECDF with ANSI-X9.63-KDF using SHA-512; uses AES-128 key wrap:  
    30 15 06 06 2B 81 04 01 0B 03 30 0B 06 09 60 86 48 01 65 03 04  
    01 05
```

```
ECDF with ANSI-X9.63-KDF using SHA-256; uses AES-256 key wrap:  
    30 15 06 06 2B 81 04 01 0B 01 30 0B 06 09 60 86 48 01 65 03 04  
    01 2D
```

```
ECDF with ANSI-X9.63-KDF using SHA-384; uses AES-256 key wrap:  
    30 15 06 06 2B 81 04 01 0B 02 30 0B 06 09 60 86 48 01 65 03 04  
    01 2D
```

```
ECDF with ANSI-X9.63-KDF using SHA-512; uses AES-256 key wrap:  
    30 15 06 06 2B 81 04 01 0B 03 30 0B 06 09 60 86 48 01 65 03 04  
    01 2D
```

The following SMIMECapabilities values (in hexadecimal) based on the algorithm identifiers in Section 7 of this document might be of interest to implementations that support X25519 and X448:

ECDH with HKDF using SHA-256; uses AES-128 key wrap:  
TBD

ECDH with HKDF using SHA-384; uses AES-128 key wrap:  
TBD

ECDH with HKDF using SHA-512; uses AES-128 key wrap:  
TBD

ECDH with HKDF using SHA-256; uses AES-256 key wrap:  
TBD

ECDH with HKDF using SHA-384; uses AES-256 key wrap:  
TBD

ECDH with HKDF using SHA-512; uses AES-256 key wrap:  
TBD

## 9. Security Considerations

Please consult the security considerations of [CMS] for security considerations related to the enveloped-data content type and the authenticated-data content type.

Please consult the security considerations of [AUTHENV] for security considerations related to the authenticated-enveloped-data content type.

Please consult the security considerations of [CURVES] for security considerations related to the use of X25519 and X448.

The originator uses an ephemeral public/private key pair that is generated on the same elliptic curve as the public key of the recipient. The ephemeral key pair is used for a single CMS protected content type, and then it is discarded. If the originator wants to be able to decrypt the content (for enveloped-data and authenticated-enveloped-data) or check the authentication (for authenticated-data), then the originator needs to treat themselves as a recipient.

As specified in [CMS], implementations MUST support processing of the KeyAgreeRecipientInfo ukm field, so interoperability is not a concern if the ukm is present or absent. The ukm is placed in the entityUInfo field of the ECC-CMS-SharedInfo structure. When present, the ukm ensures that a different key-encryption key is generated,

even when the originator ephemeral private key is improperly used more than once.

#### 10. IANA Considerations

Three object identifiers for the Key Agreement Algorithm Identifiers in Sections 7 are needed. Are they going to come from an IANA registry or from the registry that assigned the object identifiers in [ID.curdle-pkix]?

#### 11. Normative References

- [AUTHENV] Housley, R., "Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type", RFC 5083, November 2007.
- [CERTCAP] Santesson, S., "X.509 Certificate Extension for Secure/Multipurpose Internet Mail Extensions (S/MIME) Capabilities", RFC 4262, December 2005.
- [CMS] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 5652, September 2009.
- [CURVES] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, January 2016.
- [HKDF] Krawczyk, H., and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010.
- [ID.curdle-pkix] Josefsson, S., and J. Schaad, "Algorithm Identifiers for Ed25519, Ed25519ph, Ed448, Ed448ph, X25519 and X448 for use in the Internet X.509 Public Key Infrastructure", 15 August 2016, Work-in-progress.
- [PKIXALG] 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, April 2002.
- [PKIXECC] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.

- [PROFILE] 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, May 2008.
- [SEC1] Standards for Efficient Cryptography Group, "SEC 1: Elliptic Curve Cryptography", version 2.0, May 2009, <<http://www.secg.org/sec1-v2.pdf>>.
- [SMIME] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.
- [STDWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [X680] ITU-T, "Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, 2015.
- [X690] ITU-T, "Information technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, 2015.

## 12. Informative References

- [CMSECC] Turner, S., and D. Brown, "Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)", RFC 5753, January 2010.
- [CMSAES] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, July 2003.
- [DH1976] Diffie, W., and M. E. Hellman, "New Directions in Cryptography", IEEE Trans. on Info. Theory, Vol. IT-22, Nov. 1976, pp. 644-654.
- [X963] "Public-Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography", American National Standard X9.63-2001, 2001.

## 13. Acknowledgements

Thanks to Jim Schaad, Stefan Santesson, Sean Turner for their review and insightful suggestions.

Author Address

Russ Housley  
918 Spring Knoll Drive  
Herndon, VA 20170  
USA  
housley@vigilsec.com

Internet-Draft  
Intended status: Standards Track  
Expires: 8 March 2017

R. Housley  
Vigil Security  
8 September 2016

## Use of EdDSA Signatures in the Cryptographic Message Syntax (CMS)

<draft-ietf-curdle-cms-eddsa-signatures-00.txt>

### Abstract

This document describes the conventions for using Edwards-curve Digital Signature Algorithm (EdDSA) in the Cryptographic Message Syntax (CMS). The conventions for Ed25519 and Ed448 are described, but Ed25519ph and Ed448ph are not used with the CMS.

### 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 <http://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 8 March 2017.

### Copyright Notice

Copyright (c) 2016 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 (<http://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.

## 1. Introduction

This document specifies the conventions for using the Edwards-curve Digital Signature Algorithm (EdDSA) [EDDSA] with the Cryptographic Message Syntax [CMS] signed-data content type. For each curve, [EDDSA] defines two modes, the PureEdDSA mode without pre-hashing, and the HashEdDSA mode with pre-hashing. The CMS conventions for two PureEdDSA curves (Ed25519 and Ed448) are described in this document, but HashEdDSA is not used with the CMS.

### 1.1. 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 RFC 2119 [STDWORDS].

### 1.2. ASN.1

CMS values are generated using ASN.1 [X680], which uses the Basic Encoding Rules (BER) and the Distinguished Encoding Rules (DER) [X690].

## 2. EdDSA Signature Algorithm

The Edwards-curve Digital Signature Algorithm (EdDSA) [EDDSA] is a variant of Schnorr's signature system with (possibly twisted) Edwards curves. Ed25519 is intended to operate at around the 128-bit security level, and Ed448 at around the 224-bit security level.

One of the parameters of the EdDSA algorithm is the "prehash" function. This may be the identity function, resulting in an algorithm called PureEdDSA, or a collision-resistant hash function, resulting in an algorithm called HashEdDSA. In most situations the CMS SignedData includes signed attributes, including the message digest of the content. Since HashEdDSA offers no benefit when signed attributes are present, only PureEdDSA is used with the CMS.

A message digest is computed over the data to be signed using PureEdDSA, and then a private key operation is performed to generate the signature value. As described in Section 3.3 of [EDDSA], the signature value is the opaque value  $\text{ENC}(R) \parallel \text{ENC}(S)$ . As described in Section 5.3 of [CMS], the signature value is ASN.1 encoded as an OCTET STRING and included in the signature field of SignerInfo.

### 2.1. EdDSA Algorithm Identifiers

The EdDSA signature algorithm is defined in [EDDSA], and the conventions for encoding the public key are defined in [ID.curdle-pkix].

The id-Ed25519 and id-Ed448 object identifiers are used to identify EdDSA public keys in certificates. The object identifiers are specified in [ID.curdle-pkix], and they are repeated here for convenience:

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

### 2.2. EdDSA Signatures

The id-Ed25519 and id-Ed448 object identifiers are also used for signature values. When used to identify signature algorithms, the AlgorithmIdentifier parameters field MUST be absent.

An EdDSA private key operation produces the opaque signature value, ENC(R) || ENC(S), as described in Section 3.3 of [EDDSA]. The resulting octet string is carried in the signature field of SignerInfo.

## 3. Signed-data Conventions

The digestAlgorithms field SHOULD contain the one-way hash function used to compute the message digest on the eContent value.

If signedAttributes are present, the same one-way hash function SHOULD be used to compute the message digest on both the eContent and the signedAttributes.

The signatureAlgorithm field MUST contain either id-Ed25519 or id-Ed448, depending on the elliptic curve that was used by the signer. The algorithm parameters field MUST be absent.

The signature field contains the octet string resulting from the EdDSA private key signing operation.

## 4. Security Considerations

Implementations must protect the EdDSA private key. Compromise of the EdDSA private key may result in the ability to forge signatures.

The generation of EdDSA private key relies on random numbers. The use of inadequate pseudo-random number generators (PRNGs) to generate



these values can result in little or no security. An attacker may find it much easier to reproduce the PRNG environment that produced the keys, searching the resulting small set of possibilities, rather than brute force searching the whole key space. The generation of quality random numbers is difficult. RFC 4086 [RANDOM] offers important guidance in this area.

Using the same private key for different algorithms has the potential of allowing an attacker to get extra information about the private key. For this reason, the same private key SHOULD NOT be used with more than one EdDSA set of parameters. For example, do not use the same private key with PureEdDSA and HashEdDSA.

When computing signatures, the same hash function should be used for all operations. This reduces the number of failure points in the signature process.

## 5. Normative References

- [CMS] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 5652, September 2009.
- [EDDSA] Josefsson, S. and I. Liusvaara, "Edwards-curve Digital Signature Algorithm (EdDSA)", draft-irtf-cfrg-eddsa-00, (work in progress), October 2015.
- [ID.curdle-pkix] Josefsson, S., and J. Schaad, "Algorithm Identifiers for Ed25519, Ed25519ph, Ed448, Ed448ph, X25519 and X448 for use in the Internet X.509 Public Key Infrastructure", 15 August 2016, Work-in-progress.
- [STDWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [X680] ITU-T, "Information technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, 2015.
- [X690] ITU-T, "Information technology -- ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, 2015.

## 6. Informative References

- [RANDOM] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", RFC 4086, June 2005.

Author Address

Russ Housley  
918 Spring Knoll Drive  
Herndon, VA 20170  
USA  
housley@vigilsec.com

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: April 13, 2017

O. Sury  
CZ.NIC  
R. Edmonds  
Fastly  
October 10, 2016

EdDSA for DNSSEC  
draft-ietf-curdle-dnskey-eddsa-01

Abstract

This document describes how to specify EdDSA keys and signatures in DNS Security (DNSSEC). It uses the Edwards-curve Digital Security Algorithm (EdDSA) with the choice of two curves, Ed25519 and Ed448.

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 <http://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 April 13, 2017.

Copyright Notice

Copyright (c) 2016 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 (<http://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 . . . . .	2
2. Requirements Language . . . . .	2
3. DNSKEY Resource Records . . . . .	2
4. RRSIG Resource Records . . . . .	3
5. Algorithm Number for DS, DNSKEY and RRSIG Resource Records .	3
6. Examples . . . . .	3
6.1. Ed25519 Examples . . . . .	3
6.2. Ed448 Example . . . . .	4
7. Acknowledgements . . . . .	5
8. IANA Considerations . . . . .	5
9. Implementation Status . . . . .	6
10. Security Considerations . . . . .	6
11. References . . . . .	7
11.1. Normative References . . . . .	7
11.2. Informative References . . . . .	7
Authors' Addresses . . . . .	8

## 1. Introduction

DNSSEC, which is broadly defined in [RFC4033], [RFC4034], and [RFC4035], uses cryptographic keys and digital signatures to provide authentication of DNS data. Currently, the most popular signature algorithm in use is RSA. GOST ([RFC5933]) and NIST-specified elliptic curve cryptography ([RFC6605]) are also standardized.

[I-D.irtf-cfrg-eddsa] describes the elliptic curve signature system EdDSA and recommends two curves, Ed25519 and Ed448.

This document defines the use of DNSSEC's DS, DNSKEY, and RRSIG resource records (RRs) with a new signing algorithm, Edwards-curve Digital Signature Algorithm (EdDSA) using a choice of two curves, Ed25519 and Ed448.

## 2. Requirements Language

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. DNSKEY Resource Records

An Ed25519 public key consists of a 32-octet value, which is encoded into the Public Key field of a DNSKEY resource record as a simple bit string. The generation of a public key is defined in Chapter 5.1.5 in [I-D.irtf-cfrg-eddsa].

An Ed448 public key consists of a 57-octet value, which is encoded into the Public Key field of a DNSKEY resource record as a simple bit string. The generation of a public key is defined in Chapter 5.2.5 in [I-D.irtf-cfrg-eddsa].

#### 4. RRSIG Resource Records

In Chapter 10.3 in [I-D.irtf-cfrg-eddsa], the use of a context label is described. EdDSA signatures in this scheme use the 12-octet context label 'DNSSECRRSIG\0' (where \0 represents a zero-valued octet).

An Ed25519 signature consists of a 64-octet value, which is encoded into the Signature field of an RRSIG resource record as a simple bit string. The Ed25519 signature algorithm is described in Chapter 5.1.6 in [I-D.irtf-cfrg-eddsa].

An Ed448 signature consists of a 114-octet value, which is encoded into the Signature field of an RRSIG resource record as a simple bit string. The Ed448 signature algorithm is described in Chapter 5.2.6 and verification of the Ed448 signature is described in Chapter 5.2.7 in [I-D.irtf-cfrg-eddsa].

#### 5. Algorithm Number for DS, DNSKEY and RRSIG Resource Records

The algorithm number associated with the use of Ed25519 in DS, DNSKEY and RRSIG resource records is TBD. The algorithm number associated with the use of Ed448 in DS, DNSKEY and RRSIG resource records is TBD. This registration is fully defined in the IANA Considerations section.

#### 6. Examples

##### 6.1. Ed25519 Examples

This section needs an update after the algorithm for Ed25519 is assigned.

```
Private-key-format: v1.2
Algorithm: TBD (ED25519)
PrivateKey: ODIyNjAzODQ2MjgwODAxMjI2NDUxOTAyMDQxNDIyNjI=
# corresponding to 82260384628080122645190204142262 INT
```

```
example.com. 3600 IN DNSKEY 257 3 TBD (
    102Woi0iS8Aa25FQkUd9RMzZHJpBoRQwAQEX1SxZJA4= )
```

```
example.com. 3600 IN DS 3613 TBD 2 (
    3aa5ab37efce57f737fc1627013fee07bdf241bd10f3
    b1964ab55c78e79a304b )
```

```
www.example.com. 3600 IN A 192.0.2.1
www.example.com. 3600 IN RRSIG A TBD 3 3600 (
    20150820000000 20150730000000 3613 example.com.
    cvTRVrU7dwnemQuBq9/E4tlIiRpvWcEmYdzqs6SCQxw6
    qmczBBQGlDssMx1TCJnwsEs9ZuA2phPzuJNoon9BCA== )
```

```
Private-key-format: v1.2
Algorithm: TBD (ED25519)
PrivateKey: DSSF3o0s0f+ElWzj9E/OsXw8hLpk55chkmx0LYN5WiY=
```

```
example.com. 3600 IN DNSKEY 257 3 TBD (
    zPnZ/QwEe7S8C5SPz2OfS5RR40ATk2/rYnE9xHIEijs= )
```

```
example.com. 3600 IN DS 55648 TBD 2 (
    96401675bc7ecdd541ec0f70d69238c7b95d3bd4de1e
    231a068ceb214d02a4ed )
```

```
www.example.com. 3600 IN A 192.0.2.1
www.example.com. 3600 IN RRSIG A TBD 3 3600 (
    20150820000000 20150730000000 35452 example.com.
    yUGb9rCNiuhDaRJbuhYHj89Y/3Pi8KWUm7lOt00ivVRGvgulmVX8Dgpe
    AFyMP2MKXJrqYJr+ViICIDwcOIbPAQ==)
```

## 6.2. Ed448 Example

This section needs an update after the algorithm for Ed448 is assigned.

Private-key-format: v1.2  
Algorithm: TBD (ED448)  
PrivateKey: TBD

example.com. 3600 IN DNSKEY 257 3 TBD (  
TBD )

example.com. 3600 IN DS 3613 TBD 2 (  
TBD )

www.example.com. 3600 IN A 192.0.2.1  
www.example.com. 3600 IN RRSIG A TBD 3 3600 (  
20150820000000 20150730000000 3613 example.com.  
TBD )

Private-key-format: v1.2  
Algorithm: TBD (ED448)  
PrivateKey: TBD

example.com. 3600 IN DNSKEY 257 3 TBD (  
TBD )

example.com. 3600 IN DS 55648 TBD 2 (  
TBD )

www.example.com. 3600 IN A 192.0.2.1  
www.example.com. 3600 IN RRSIG A TBD 3 3600 (  
20150820000000 20150730000000 35452 example.com.  
TBD )

## 7. Acknowledgements

Some of the material in this document is copied liberally from [RFC6605].

The authors of this document wish to thank Jan Vcelak, Pieter Lexis and Kees Monshouwer for a review of this document.

## 8. IANA Considerations

This document updates the IANA registry "Domain Name System Security (DNSSEC) Algorithm Numbers". The following entry has been added to the registry:

Number	TBD	TBD
Description	Ed25519	Ed448
Mnemonic	ED25519	Ed448
Zone Signing	Y	Y
Trans. Sec.	*	*
Reference	This document	This document

\* There has been no determination of standardization of the use of this algorithm with Transaction Security.

## 9. Implementation Status

(Note to the RFC Editor: please remove this entire section as well as the reference to RFC 6982 before publication.)

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6982], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

## 10. Security Considerations

The security level of Ed25519 is slightly under the standard 128-bit level and the security level of Ed448 is slightly under the standard 224-bit level ([RFC7748]). Security considerations listed in [RFC7748] also apply to the usage of Ed25519 and Ed448 in DNSSEC. Such an assessment could, of course, change in the future if new attacks that work better than the ones known today are found.



## 11. References

### 11.1. Normative References

- [I-D.irtf-cfrg-eddsa]  
Josefsson, S. and I. Liusvaara, "Edwards-curve Digital Signature Algorithm (EdDSA)", draft-irtf-cfrg-eddsa-08 (work in progress), August 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <<http://www.rfc-editor.org/info/rfc4033>>.
- [RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <<http://www.rfc-editor.org/info/rfc4034>>.
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <<http://www.rfc-editor.org/info/rfc4035>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<http://www.rfc-editor.org/info/rfc7748>>.

### 11.2. Informative References

- [RFC5933] Dolmatov, V., Ed., Chuprina, A., and I. Ustinov, "Use of GOST Signature Algorithms in DNSKEY and RRSIG Resource Records for DNSSEC", RFC 5933, DOI 10.17487/RFC5933, July 2010, <<http://www.rfc-editor.org/info/rfc5933>>.
- [RFC6605] Hoffman, P. and W. Wijngaards, "Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC", RFC 6605, DOI 10.17487/RFC6605, April 2012, <<http://www.rfc-editor.org/info/rfc6605>>.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, DOI 10.17487/RFC6982, July 2013, <<http://www.rfc-editor.org/info/rfc6982>>.

Authors' Addresses

Ondrej Sury  
CZ.NIC  
Milesovska 1136/5  
Praha 130 00  
CZ

Email: [ondrej.sury@nic.cz](mailto:ondrej.sury@nic.cz)

Robert Edmonds  
Fastly  
Atlanta, Georgia  
US

Email: [edmonds@mycre.ws](mailto:edmonds@mycre.ws)

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: February 16, 2017

S. Josefsson  
SJD AB  
J. Schaad  
August Cellars  
August 15, 2016

Algorithm Identifiers for Ed25519, Ed25519ph, Ed448, Ed448ph, X25519 and  
X448 for use in the Internet X.509 Public Key Infrastructure  
draft-ietf-curdle-pkix-01

## Abstract

This document specifies algorithm identifiers and ASN.1 encoding formats for Elliptical Curve constructs using the Curve25519 and Curve448 curves. The signature algorithms covered are Ed25519, Ed25519ph, Ed448 and Ed448ph. 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 <http://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 February 16, 2017.

## Copyright Notice

Copyright (c) 2016 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 (<http://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 . . . . .	2
2. Requirements Terminology . . . . .	3
3. Curve25519 and Curve448 Algorithm Identifiers . . . . .	3
4. Subject Public Key Fields . . . . .	4
5. Key Usage Bits . . . . .	5
6. EdDSA Signatures . . . . .	5
7. Private Key Format . . . . .	6
8. Human Readable Algorithm Names . . . . .	7
9. ASN.1 Module . . . . .	8
10. Examples . . . . .	10
10.1. Example Ed25519 Public Key . . . . .	10
10.2. Example X25519 Certificate . . . . .	11
10.3. Example Ed25519 Private Key . . . . .	13
11. Acknowledgements . . . . .	13
12. IANA Considerations . . . . .	13
13. Security Considerations . . . . .	13
14. References . . . . .	13
14.1. Normative References . . . . .	14
14.2. Informative References . . . . .	14
Authors' Addresses . . . . .	15

## 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. A convention has developed that when these two curves are used with the Diffie-Hellman operation, they are referred to as X25519 and X448.

In [I-D.irtf-cfrg-eddsa] the elliptic curve signature system EdDSA is described and the recommended choice of curves Ed25519/Ed448 are chosen. EdDSA has defined two modes, the PureEdDSA mode without pre-hashing, and the HashEdDSA mode with pre-hashing. Unlike other digital signature algorithms, the Ed25519ph and Ed448ph algorithm definitions specify the one-way hash function that is used. Attacks have been described when the same key is used with and without pre-hashing for Ed25519, so a single key **MUST NOT** be used for both modes. The convention used for identifying the algorithm/curve combinations are to use the Ed25519 and Ed448 for the PureEdDSA mode and Ed25519ph and Ed448ph for the HashEdDSA mode.

This RFC defines ASN.1 object identifiers for EdDSA for use in the Internet X.509 PKI [RFC5280], and parameters for Ed25519, Ed25519ph, Ed448 and Ed448ph. This document serves a similar role as [RFC3279] does for RSA (and more), [RFC4055] for RSA-OAEP/PSS, and [RFC5758] for SHA2-based (EC)DSA. This document also specifies ASN.1 "named curve" object identifiers for Curve25519 and Curve448, similar to what is done in [RFC5639]. This allows the curves to be used and referenced in PKIX standards and software, in particular enabling re-use of existing constructs already defined for ECDSA/ECDH but for the new curves. Similar to [RFC5639], this document does not describe the cryptographic algorithms to be used with the specified parameters nor their application in other standards.

## 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:

- o algorithm identifies the cryptographic algorithm with an object identifier. This is one of the OIDs defined below.
- o 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 six new OIDs for identifying the different curve/algorithm pairs. The curves being Curve25519 and Curve448. The algorithms being ECDH, EdDSA in pure mode and EdDSA in pre-hash mode. For all of the OIDs, the parameters MUST be absent. Implementations SHOULD 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 four 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 }
id-Ed25519ph   OBJECT IDENTIFIER ::= { 1.3.101.114 }
id-Ed448ph     OBJECT IDENTIFIER ::= { 1.3.101.115 }
```

#### 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:

- o algorithm is the algorithm identifier and parameters for the public key (see above).
- o subjectPublicKey contains the byte stream of the public key. While the encoded public keys for the current algorithms are all an even number of octets, future curves could change that.

Both [RFC7748] and [I-D.irtf-cfrg-eddsa] 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-----
MCowBQYDK2VwAyEAGb9ECWmEzf6FQbrBZ9w7lshQhqowtrbLDFw4rXAxZmE=
-----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-X25119 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-EdDSA25519, id-EdDSA25519ph, id-EdDSA448 or id-EdDSA448ph, 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-EdDSA25519 or id-EdDSA448, then the keyUsage extension MUST contain one or more of the following values:

nonRepudiation;  
digitalSignature;  
keyCertSign; and  
cRLSign.

CAs MUST NOT use the pre-hash versions of the EdDSA algorithms for the creation of certificates or CRLs. This is implied by the fact that those algorithms are not listed in the previous paragraph. Additionally OCSF responders SHOULD NOT use the pre-hash versions of the EdDSA algorithms when generating OCSF responses. No restriction is placed on generation of OCSF requests.

AAs MUST NOT use the pre-hash versions of the EdDSA algorithms for the creation of attribute certificates or attribute CRLs [RFC5755].

## 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 [I-D.irtf-cfrg-eddsa]. 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.

When the pre-hash versions of the EdDSA signature algorithms are used, the hash function used for the pre-hash is defined by the algorithm. This means that the pre-hash function is implicitly included in the algorithm identifier rather than being explicit as done in [RFC3279].

## 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] Attributes OPTIONAL,
    ...,
    [[2: publicKey [1] PublicKey OPTIONAL ]],
    ...
}

```

```
PrivateKey ::= OCTET STRING
```

```
PublicKey ::= OCTET STRING
```

For the keys defined in this document, the private key is always an opaque byte sequence. The ASN.1 type `EdPrivateKey` is defined in this document to hold the byte sequence. Thus when encoding a `OneAsymmetricKey` object, the private key is wrapped in an `EdPrivateKey` object and then placed in the 'privateKey' field.

```
EdPrivateKey ::= 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 [I-D.irtf-cfrg-eddsa].

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

```

-----BEGIN PRIVATE KEY-----
MC4CAQAwBQYDK2VwBCIEINTuctv5ElhK1bbY8fdp+K06/nwoy/HU++CXqI9EdVhC
-----END PRIVATE 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-EdDSA25519 value use the string "Ed25519". For the id-EdDSA25519ph value use the string "Ed25519ph". For id-EdDSA448 use "Ed448". For id-EdDSA448ph use "Ed448ph".

## 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 {1 3 101 120}
```

```
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-EdDSA25519      OBJECT IDENTIFIER ::= { id-edwards-curve-algs 112 }
id-EdDSA25519-ph   OBJECT IDENTIFIER ::= { id-edwards-curve-algs 114 }
id-EdDSA448        OBJECT IDENTIFIER ::= { id-edwards-curve-algs 113 }
id-EdDSA448-ph     OBJECT IDENTIFIER ::= { id-edwards-curve-algs 115 }
```

```
sa-EdDSA25519 SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-EdDSA25519
  PARAMS ARE absent
  PUBLIC-KEYS {pk-EdDSA25519}
  SMIME-CAPS { IDENTIFIED BY id-EdDSA25519 }
}
```

```
pk-EdDSA25519 PUBLIC-KEY ::= {
  IDENTIFIER id-EdDSA25519
  -- KEY no ASN.1 wrapping --
  PARAMS ARE absent
  CERT-KEY-USAGE {digitalSignature, nonRepudiation,
                  keyCertSign, cRLSign}
  PRIVATE-KEY EdPrivateKey
}
```

```
sa-EdDSA25519-ph SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-EdDSA25519-ph
  PARAMS ARE absent
  HASHES { mda-sha512 }
  PUBLIC-KEYS {pk-EdDSA25519-ph}
  SMIME-CAPS { IDENTIFIED BY id-EdDSA25519-ph }
}
```

```
pk-EdDSA25519-ph PUBLIC-KEY ::= {
  IDENTIFIER id-EdDSA25519-ph
  -- KEY no ASN.1 wrapping --
  PARAMS ARE absent
  CERT-KEY-USAGE {digitalSignature, nonRepudiation}
  PRIVATE-KEY EdPrivateKey
}
```

```
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 EdPrivateKey
}

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

kaa-X488 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 EdPrivateKey
}

EdPrivateKey ::= 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: EdDSA25519

Algorithm Security Level: High

## Public Key Usage:

Public Key ID: 9b1f5eeded043385e4f7bc623c5975b90bc8bb3b

-----BEGIN PUBLIC KEY-----

MCowBQYDK2VwAyEAGb9ECWmEzf6FQbrBZ9w7lshQhqowtrbLDFw4rXAxZmE=

-----END PUBLIC KEY-----

## 10.2. Example X25519 Certificate

An example of a PKIX certificate using Ed25519 to sign X25519 would be:

```

0 30 300: SEQUENCE {
4 30 223: SEQUENCE {
7 A0 3: [0] {
9 02 1: INTEGER 2
: }
12 02 8: INTEGER
: 56 01 47 4A 2A 8D C3 30
22 30 5: SEQUENCE {
24 06 3: OBJECT IDENTIFIER
: EdDSA 25519 signature algorithm { 1 3 101 112 }
: }
29 30 25: SEQUENCE {
31 31 23: SET {
33 30 21: SEQUENCE {
35 06 3: OBJECT IDENTIFIER commonName (2 5 4 3)
40 0C 14: UTF8String (1997) 'IETF Test Demo'
: }
: }
: }
56 30 30: SEQUENCE {
58 17 13: UTCTime '160801121924Z'
73 17 13: UTCTime '401231235959Z'
: }
88 30 25: SEQUENCE {
90 31 23: SET {
92 30 21: SEQUENCE {
94 06 3: OBJECT IDENTIFIER commonName (2 5 4 3)
99 0C 14: UTF8String (1997) 'IETF Test Demo'
: }
: }
: }

```

```

115 30 42: SEQUENCE {
117 30 5: SEQUENCE {
119 06 3: OBJECT IDENTIFIER
: ECDH 25519 key agreement { 1 3 101 110 }
: }
124 03 33: BIT STRING 0 unused bits
: 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 A3 69: [3] {
161 30 67: SEQUENCE {
163 30 15: SEQUENCE {
165 06 3: OBJECT IDENTIFIER basicConstraints (2 5 29 19)
170 01 1: BOOLEAN TRUE
173 04 5: OCTET STRING, encapsulates {
175 30 3: SEQUENCE {
177 01 1: BOOLEAN FALSE
: }
: }
: }
180 30 14: SEQUENCE {
182 06 3: OBJECT IDENTIFIER keyUsage (2 5 29 15)
187 01 1: BOOLEAN FALSE
190 04 4: OCTET STRING, encapsulates {
192 03 2: BIT STRING 7 unused bits
: '1'B
: }
: }
196 30 32: SEQUENCE {
198 06 3: OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29 14)
203 01 1: BOOLEAN FALSE
206 04 22: OCTET STRING
: 04 14 9B 1F 5E ED ED 04 33 85 E4 F7 BC 62 3C 59
: 75 B9 0B C8 BB 3B
: }
: }
: }
230 30 5: SEQUENCE {
232 06 3: OBJECT IDENTIFIER
: EdDSA 25519 signature algorithm { 1 3 101 112 }
: }
237 03 65: BIT STRING 0 unused bits
: D1 EE DF 10 15 68 CA C2 4A C2 13 7F 45 C6 B7 6E
: 7C 11 E8 B3 AC D5 67 D3 1A 6E 90 EA 0F 8B F6 50
: 0F 91 66 BB EF BE 10 DE FA 37 7B 61 FC D7 C5 C6
: AB CF 3F 89 01 F9 BD 80 E8 1B 9D 21 DD 32 73 0A
: }

```

-----BEGIN CERTIFICATE-----

MIIBLDCB36ADAgECAGhWAUdKKo3DMDAFBgMrZXAwGTEXMBUGA1UEAwOSUVURiBUZXN0IERlbW8wHhcNMTYwODAxMTIxOTI0WhcNNDExMjMxMjM1OTU5WjAZMRcwFQYDVQQDDA5JRVRGIFRlc3QgRGVtbzAqMAUGAyt1bgMhAIUg8AmJMKdUdIt93LQ+91oNvzoNJjga9OukqY6qm05qo0UwQzAPBgNVHRMBAf8EBTADAQEA  
MA4GA1UdDwEBAAQEAwIHgDAgBgNVHQ4BAQAEFgQUmx9e7e0EM4Xk97xiPF11uQvIuzswBQYDK2VwA0EA  
0e7fEBVoysJKwhN/Rca3bnwR6LOslWfTGm6Q6g+L9lAPkWa7774Q3vo3e2H818XGq88/iQH5vYDoG50h  
3TJzCg==

-----END CERTIFICATE-----

### 10.3. Example Ed25519 Private Key

An example of an Ed25519 private key:

-----BEGIN PRIVATE KEY-----

MC4CAQAwBQYDK2VwBCIEINTuctv5ElhK1bbY8fdp+K06/nwoy/HU++CXqI9EdVhC

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

### 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, and Alex Wilson.

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

### 12. IANA Considerations

None.

### 13. Security Considerations

The security considerations of [RFC5280], [RFC7748], and [I-D.irtf-cfrg-eddsa] apply accordingly.

A common misconception may be that a Ed25519 public key can be used to create Ed25519ph signatures, or vice versa. This leads to cross-key attacks, and is not permitted.

### 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, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc5480>>.
- [RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <<http://www.rfc-editor.org/info/rfc7748>>.
- [I-D.irtf-cfrg-eddsa] Josefsson, S. and I. Liusvaara, "Edwards-curve Digital Signature Algorithm (EdDSA)", draft-irtf-cfrg-eddsa-06 (work in progress), August 2016.
- [RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <<http://www.rfc-editor.org/info/rfc5958>>.

## 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, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc5639>>.
- [RFC5755] Farrell, S., Housley, R., and S. Turner, "An Internet Attribute Certificate Profile for Authorization", RFC 5755, DOI 10.17487/RFC5755, January 2010, <<http://www.rfc-editor.org/info/rfc5755>>.
- [RFC5758] Dang, Q., Santesson, S., Moriarty, K., Brown, D., and T. Polk, "Internet X.509 Public Key Infrastructure: Additional Algorithms and Identifiers for DSA and ECDSA", RFC 5758, DOI 10.17487/RFC5758, January 2010, <<http://www.rfc-editor.org/info/rfc5758>>.
- [RFC5915] Turner, S. and D. Brown, "Elliptic Curve Private Key Structure", RFC 5915, DOI 10.17487/RFC5915, June 2010, <<http://www.rfc-editor.org/info/rfc5915>>.
- [RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <<http://www.rfc-editor.org/info/rfc7468>>.

#### Authors' Addresses

Simon Josefsson  
SJD AB

Email: [simon@josefsson.org](mailto:simon@josefsson.org)

Jim Schaad  
August Cellars

Email: [ietf@augustcellars.com](mailto:ietf@augustcellars.com)

Internet-Draft  
Updates: 4252, 4253 (if approved)  
Intended status: Standards Track  
Expires: February 12, 2017

D. Bider  
Bitwise Limited  
September 12, 2016

Use of RSA Keys with SHA-2 256 and 512 in Secure Shell (SSH)  
draft-ietf-curdle-rsa-sha2-02.txt

## Abstract

This memo defines an algorithm name, public key format, and signature format for use of RSA keys with SHA-2 512 for server and client authentication in SSH connections.

## Status

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

## Copyright

Copyright (c) 2016 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 (<http://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.

## 1. Overview and Rationale

Secure Shell (SSH) is a common protocol for secure communication on the Internet. In [RFC4253], SSH originally defined the signature methods "ssh-rsa" for server and client authentication using RSA with SHA-1, and "ssh-dss" using 1024-bit DSA and SHA-1.

A decade later, these signature methods are considered deficient. For US government use, NIST has disallowed 1024-bit RSA and DSA, and use of SHA-1 for signing [800-131A].

This memo defines a new algorithm name allowing for interoperable use of RSA keys with SHA-2 256 and SHA-2 512, and a mechanism for servers to inform SSH clients of signature algorithms they support and accept.

### 1.1. 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].

## 2. Public Key Algorithms

This memo adopts the style and conventions of [RFC4253] in specifying how use of a signature algorithm is indicated in SSH.

The following new signature algorithms are defined:

rsa-sha2-256	RECOMMENDED	sign	Raw RSA key
rsa-sha2-512	OPTIONAL	sign	Raw RSA key

These signature algorithms are suitable for use both in the SSH transport layer [RFC4253] for server authentication, and in the authentication layer [RFC4252] for client authentication.

Since RSA keys are not dependent on the choice of hash function, both new algorithms reuse the public key format of the existing "ssh-rsa" algorithm as defined in [RFC4253]:

```
string    "ssh-rsa"
mpint     e
mpint     n
```

All aspects of the "ssh-rsa" format are kept, including the encoded string "ssh-rsa", in order to allow users' existing RSA keys to be used with the new signature formats, without requiring re-encoding, or affecting already trusted key fingerprints.

Signing and verifying using these algorithms is performed according to the RSASSA-PKCS1-v1\_5 scheme in [RFC3447] using SHA-2 [FIPS-180-4] as hash; MGF1 as mask function; and salt length equal to hash size.

For the algorithm "rsa-sha2-256", the hash used is SHA-2 256.  
For the algorithm "rsa-sha2-512", the hash used is SHA-2 512.

The resulting signature is encoded as follows:

```
string      "rsa-sha2-256" / "rsa-sha2-512"  
string      rsa_signature_blob
```

The value for 'rsa\_signature\_blob' is encoded as a string containing S - an octet string which is the output of RSASSA-PKCS1-v1\_5, of length equal to the length in octets of the RSA modulus.

## 2.1. Use for server authentication

To express support and preference for one or both of these algorithms for server authentication, the SSH client or server includes one or both algorithm names, "rsa-sha2-256" and/or "rsa-sha2-512", in the name-list field "server\_host\_key\_algorithms" in the SSH\_MSG\_KEXINIT packet [RFC4253]. If one of the two host key algorithms is negotiated, the server sends an "ssh-rsa" public key as part of the negotiated key exchange method (e.g. in SSH\_MSG\_KEXDH\_REPLY), and encodes a signature with the appropriate signature algorithm name - either "rsa-sha2-256", or "rsa-sha2-512".

## 2.2. Use for client authentication

To use this algorithm for client authentication, the SSH client sends an SSH\_MSG\_USERAUTH\_REQUEST message [RFC4252] encoding the "publickey" method, and encoding the string field "public key algorithm name" with the value "rsa-sha2-256" or "rsa-sha2-512". The "public key blob" field encodes the RSA public key using the "ssh-rsa" algorithm name. The signature field, if present, encodes a signature using an algorithm name that MUST match the SSH authentication request - either "rsa-sha2-256", or "rsa-sha2-512".

For example, an SSH "publickey" authentication request using an "rsa-sha2-512" signature would be properly encoded as follows:

```
byte        SSH_MSG_USERAUTH_REQUEST  
string      user name  
string      service name  
string      "publickey"  
boolean     TRUE  
string      "rsa-sha2-512"  
string      public key blob:  
    string   "ssh-rsa"  
    mpint    e  
    mpint    n  
string      signature:  
    string   "rsa-sha2-512"  
    string   rsa_signature_blob
```

### 3. Discovery of signature algorithms supported by servers

Implementation experience has shown that there are servers which apply authentication penalties to clients attempting signature algorithms which the SSH server does not support.

Servers that accept `rsa-sha2-*` signatures for client authentication SHOULD implement the extension negotiation mechanism defined in [SSH-EXT-INFO], including especially the "server-sig-algs" extension.

When authenticating with an RSA key against a server that does not implement the "server-sig-algs" extension, clients MAY default to an `ssh-rsa` signature to avoid authentication penalties.

### 4. IANA Considerations

IANA is requested to update the "Secure Shell (SSH) Protocol Parameters" registry, to extend the table Public Key Algorithm Names:

- To the immediate right of the column Public Key Algorithm Name, a new column is to be added, titled Signature Algorithm Name. For existing entries, the column Signature Algorithm Name should be assigned the same value found under Public Key Algorithm Name.
- Immediately following the existing entry for "ssh-rsa", two sibling entries are to be added:

P. K. Alg. Name	Sig. Alg. Name	Reference	Note
ssh-rsa	rsa-sha2-256	[this document]	Section 2
ssh-rsa	rsa-sha2-512	[this document]	Section 2

### 5. Security Considerations

The security considerations of [RFC4253] apply to this document.

The National Institute of Standards and Technology (NIST) Special Publication 800-131A [800-131A] disallows the use of RSA and DSA keys shorter than 2048 bits for US government use after 2013. Keys of 2048 bits or larger are considered acceptable.

The same document disallows the SHA-1 hash function, as used in the "ssh-rsa" and "ssh-dss" algorithms, for digital signature generation after 2013. The SHA-2 family of hash functions is seen as acceptable.

### 6. Why no DSA?

A draft version of this memo also defined an algorithm name for use of 2048-bit and 3072-bit DSA keys with a 256-bit subgroup and SHA-2 256 hashing. It is possible to implement DSA securely by generating "k" deterministically as per [RFC6979]. However, a plurality of reviewers were concerned that implementers would continue to use libraries that

generate "k" randomly. This is vulnerable to biased "k" generation, and extremely vulnerable to "k" reuse.

This document therefore abstains from defining new algorithm names for DSA, and recommends RSA where this is preferred over elliptic curve cryptography.

## 7. References

### 7.1. Normative References

- [FIPS-180-4] National Institute of Standards and Technology (NIST), United States of America, "Secure Hash Standard (SHS)", FIPS Publication 180-4, August 2015, <<http://dx.doi.org/10.6028/NIST.FIPS.180-4>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.
- [RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, January 2006.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, January 2006.

### 7.2. Informative References

- [800-131A] National Institute of Standards and Technology (NIST), "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths", NIST Special Publication 800-131A, January 2011, <<http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf>>.
- [RFC4250] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, January 2006.
- [RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, August 2013.
- [SSH-EXT-INFO] Bider, D., "Extension Negotiation in Secure Shell (SSH)", draft-ietf-curdle-ssh-ext-info-01, September 2016, <<https://tools.ietf.org/html/draft-ietf-curdle-ssh-ext-info-01>>.

Author's Address

Denis Bider  
Bitvise Limited  
Suites 41/42, Victoria House  
26 Main Street  
GI

Phone: +506 8315 6519  
EMail: [ietf-ssh3@denisbider.com](mailto:ietf-ssh3@denisbider.com)  
URI: <https://www.bitvise.com/>

Acknowledgments

Thanks to Jon Bright, Niels Moeller, Stephen Farrell, Mark D. Baushke, Jeffrey Hutzelman, Hanno Boeck, Peter Gutmann, Damien Miller, and Mat Berchtold for comments and suggestions.





Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: November 4, 2016

B. Harris  
May 3, 2016

Ed25519 public key algorithm for the Secure Shell (SSH) protocol  
draft-ietf-curdle-ssh-ed25519-00

## Abstract

This document describes the use of the Ed25519 digital signature algorithm in the Secure Shell (SSH) protocol.

## 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 <http://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 November 4, 2016.

## Copyright Notice

Copyright (c) 2016 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 (<http://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.

## 1. Introduction

Secure Shell (SSH) [RFC4251] is a secure remote-login protocol. It provides for an extensible variety of public key algorithms for identifying servers and users to one another. Ed25519 [I-D.irtf-cfrg-eddsa] is a digital signature system. OpenSSH 6.5 [OpenSSH-6.5] introduced support for using Ed25519 for server and user authentication. Compatible support for Ed25519 has since been added to other SSH implementations.

This document describes the method implemented by OpenSSH and others, and formalizes its use of the name "ssh-ed25519".

Comments on this draft are welcomed and should be sent to the Curdle Working Group mailing list.

## 2. Conventions Used in This Document

The descriptions of key and signature formats use the notation introduced in [RFC4251], Section 3 and the string data type from [RFC4251], Section 5.

## 3. Public Key Algorithm

This document describes a public key algorithm for use with SSH in accordance with [RFC4253], Section 6.6. The name of the algorithm is "ssh-ed25519". This algorithm only supports signing and not encryption.

## 4. Public Key Format

The "ssh-ed25519" key format has the following encoding:

```
string    "ssh-ed25519"
string    key
```

Here 'key' is the 32-octet public key described by [I-D.irtf-cfrg-eddsa], Section 5.1.5.

## 5. Signature Algorithm

Signatures are generated according to the procedure in [I-D.irtf-cfrg-eddsa], Section 5.1.6.

## 6. Signature format

The corresponding signature format is:

```
string    "ssh-ed25519"  
string    signature
```

Here 'signature' is the 64-octet signature produced in accordance with [I-D.irtf-cfrg-eddsa], Section 5.1.6.

## 7. Verification Algorithm

Signatures are verified according to the procedure in [I-D.irtf-cfrg-eddsa], Section 5.1.7.

## 8. SSHFP DNS resource records

The generation of SSHFP resource records for "ssh-ed25519" keys is described in [RFC7479].

## 9. IANA Considerations

IANA is requested to assign the Public Key Algorithm name "ssh-ed25519" in accordance with [RFC4250], Section 4.6.2:

Public Key Algorithm Name	Reference
-----	-----
ssh-ed25519	[RFCXXXX]

[TO BE REMOVED: This registration should take place at the following location: <<http://www.iana.org/assignments/ssh-parameters/ssh-parameters.xhtml#ssh-parameters-19>>]

## 10. Security Considerations

The security considerations in [RFC4251], Section 9 apply to all SSH implementations, including those using Ed25519.

The security considerations in [I-D.irtf-cfrg-eddsa], Section 10 apply to all uses of Ed25519, including those in SSH.

## 11. Acknowledgements

The OpenSSH implementation of Ed25519 in SSH was written by Markus Friedl.

## 12. References

### 12.1. Normative References

- [RFC4250] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, DOI 10.17487/RFC4250, January 2006, <<http://www.rfc-editor.org/info/rfc4250>>.
- [RFC4251] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, DOI 10.17487/RFC4251, January 2006, <<http://www.rfc-editor.org/info/rfc4251>>.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <<http://www.rfc-editor.org/info/rfc4253>>.
- [I-D.irtf-cfrg-eddsa] Josefsson, S. and I. Liusvaara, "Edwards-curve Digital Signature Algorithm (EdDSA)", draft-irtf-cfrg-eddsa-05 (work in progress), March 2016.

### 12.2. Informative References

- [RFC7479] Moonesamy, S., "Using Ed25519 in SSHFP Resource Records", RFC 7479, DOI 10.17487/RFC7479, March 2015, <<http://www.rfc-editor.org/info/rfc7479>>.
- [OpenSSH-6.5] Friedl, M., Provos, N., de Raadt, T., Steves, K., Miller, D., Tucker, D., McIntyre, J., Rice, T., and B. Lindstrom, "[OpenSSH 6.5 release notes]", January 2014, <<http://www.openssh.com/txt/release-6.5>>.

### Author's Address

Ben Harris  
2A Eachard Road  
CAMBRIDGE CB3 0HY  
UNITED KINGDOM

Email: [bjh21@bjh21.me.uk](mailto:bjh21@bjh21.me.uk)

Internet-Draft  
Updates: 4252, 4253, 4254 (if approved)  
Intended status: Standards Track  
Expires: March 5, 2017

D. Bider  
Bitwise Limited  
September 5, 2016

Extension Negotiation in Secure Shell (SSH)  
draft-ietf-curdle-ssh-ext-info-01.txt

## Abstract

This memo defines a mechanism for SSH clients and servers to exchange information about supported protocol extensions confidentially after completed key exchange.

## Status

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

## Copyright

Copyright (c) 2015 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 (<http://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.

## 1. Overview and Rationale

Secure Shell (SSH) is a common protocol for secure communication on the Internet. The original design of the SSH transport layer [RFC4253] lacks proper extension negotiation. Meanwhile, diverse implementations take steps to ensure that known message types contain no unrecognized information. This makes it difficult for implementations to signal capabilities and negotiate extensions without risking disconnection.

This obstacle has been recognized in relationship with [SSH-RSA-SHA2], where the need arises for a client to discover signature algorithms a server accepts, to avoid authentication penalties and trial-and-error.

### 1.1. 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].

## 2. Extension Negotiation Mechanism

### 2.1. Signaling of Extension Negotiation in KEXINIT

Applications implementing this mechanism MUST add to the field "kex\_algorithms", in their KEXINIT packet sent for the first key exchange, one of the following indicator names:

- When acting as server: "ext-info-s"
- When acting as client: "ext-info-c"

The indicator name is added without quotes, and MAY be added at any position in the name-list, subject to proper separation from other names as per name-list conventions.

The names are added to the "kex\_algorithms" field because this is one of two name-list fields in KEXINIT that do not have a separate copy for each data direction.

The indicator names inserted by the client and server are different to ensure that these names will not produce a match, and will be neutral with respect to key exchange algorithm negotiation.

The inclusion of textual indicator names is intended to provide a clue for implementers to discover this mechanism.

### 2.2. Enabling Criteria

If a client or server offers "ext-info-c" or "ext-info-s" respectively, it must be prepared to accept an SSH\_MSG\_EXT\_INFO message from the peer.

Thus a server only needs to send "ext-info-s" if it intends to process SSH\_MSG\_EXT\_INFO from the client.

If a server receives an "ext-info-c", it MAY send an SSH\_MSG\_EXT\_INFO message, but is not required to do so.

If an SSH\_MSG\_EXT\_INFO message is sent, then it MUST be the first message after the initial SSH\_MSG\_NEWKEYS.

Implementations MUST NOT send an incorrect indicator name for their role. Implementations MAY disconnect if the counter-party sends an incorrect indicator. If "ext-info-c" or "ext-info-s" ends up being negotiated as a key exchange method, the parties MUST disconnect.

### 2.3. SSH\_MSG\_EXT\_INFO Message

A party that received the "ext-info-c" or "ext-info-s" indicator MAY send the following message:

```
byte      SSH_MSG_EXT_INFO (value 7)
uint32    nr-extensions
repeat "nr-extensions" times:
    string extension-name
    string extension-value
```

This message is sent without delay, and immediately after SSH\_MSG\_NEWKEYS.

### 2.4. Server's Secondary SSH\_MSG\_EXT\_INFO

If the client sent "ext-info-c", the server MAY send, but is not obligated to send, an SSH\_MSG\_EXT\_INFO message immediately before SSH\_MSG\_USERAUTH\_SUCCESS, as defined in [RFC4252]. The server MAY send this message whether or not it sent EXT\_INFO after SSH\_MSG\_NEWKEYS.

This allows a server to reveal support for additional extensions that it was unwilling to reveal to an unauthenticated client. If a server sends a subsequent SSH\_MSG\_EXT\_INFO, this replaces any initial one, and both the client and the server re-evaluate extensions in effect. The server's last EXT\_INFO is matched against the client's original.

### 2.5. Interpretation of Extension Names and Values

Each extension is identified by its extension-name, and defines the conditions under which the extension is considered to be in effect. Applications MUST ignore unrecognized extension-names.

In general, if an extension requires both the client and the server to include it in order for the extension to take effect, the relative position of the extension-name in each EXT\_INFO message is irrelevant.

Extension-value fields are interpreted as defined by their respective extension. An extension-value field MAY be empty if so permitted by the extension. Applications that do not implement or recognize a particular extension MUST ignore the associated extension-value field, regardless of its size or content.

The cumulative size of an SSH\_MSG\_EXT\_INFO message is limited only by the maximum packet length that an implementation may apply in accordance with [RFC4253]. Implementations MUST accept well-formed SSH\_MSG\_EXT\_INFO messages up to the maximum packet length they accept.

### 3. Initially Defined Extensions

#### 3.1. "server-sig-algs"

This extension is sent with the following extension name and value:

```
string      "server-sig-algs"
name-list   signature-algorithms-accepted
```

Note that the name-list type is a strict subset of the string type, and is thus permissible as an extension-value.

This extension is sent by the server only, and contains a list of signature algorithms that the server is able to process as part of a "publickey" request.

A client that wishes to proceed with public key authentication MAY wait for the server's SSH\_MSG\_EXT\_INFO so it can send a "publickey" authentication request with an appropriate signature algorithm, rather than resorting to trial and error.

Servers that implement public key authentication SHOULD implement this extension.

If a server does not send this extension, a client SHALL NOT make any assumptions about the server's signature algorithm support, and MAY proceed with authentication request trial and error.

#### 3.2. "no-flow-control"

This extension is sent with the following extension name and value:

```
string      "no-flow-control"
string      (empty)
```

This extension MUST be sent by both parties in order to take effect.

If included by both parties, the effect of this extension is that the "initial window size" fields in the messages SSH\_MSG\_CHANNEL\_OPEN and



SSH\_MSG\_CHANNEL\_OPEN\_CONFIRMATION, as defined in [RFC4254], become meaningless. The values of these fields MUST be ignored, and a channel behaves as if the window size in either direction is infinite. Neither side is required to send any SSH\_MSG\_CHANNEL\_WINDOW\_ADJUST messages, and if received, such messages MUST be ignored.

This extension is intended, but not limited to, use by file transfer applications that are only going to use one channel, and for which the flow control provided by SSH is an impediment, rather than a feature.

Implementations MUST refuse to open more than one simultaneous channel when this extension is in effect. Nevertheless, server implementations SHOULD support clients opening more than one non-simultaneous channel.

### 3.3. "accept-channels"

This extension is sent with the following extension name and value:

```
string      "accept-channels"
name-list   channel-types-accepted
```

An implementation MAY use this extension to signal to the other party a list of channel types it might accept. A server that adapts the list of available channel types based on authentication MAY defer sending this extension until a subsequent EXT\_INFO, just before sending the message USERAUTH\_SUCCESS.

An implementation is not obligated to unconditionally accept open requests for channel types advertised in this extension. An open request for a listed channel type MAY still fail for another reason.

### 3.4. "elevation"

This extension MAY be sent by the client as follows:

```
string      "elevation"
string      choice of: "y" | "n" | "d"
```

A client sends "y" to indicate its preference that the session should be elevated (as used by Windows); "n" to not be elevated; and "d" for the server to use its default behavior. If a client does not send the "elevation" extension, the server SHOULD act as if "d" was sent.

If a client has included this extension, then after authentication, a server that supports this extension SHOULD indicate to the client whether elevation was done by sending the following global request:

```
byte        SSH_MSG_GLOBAL_REQUEST
string      "elevation"
boolean     want_reply = false
boolean     elevation performed
```

### 3.5. "delay-compression"

This extension MAY be sent by both parties as follows:

```
string          "delay-compression"
string:
  name-list      compression_algorithms_client_to_server
  name-list      compression_algorithms_server_to_client
```

This extension allows the server and client to renegotiate compression algorithm support without having to conduct a key re-exchange, putting new algorithms into effect immediately upon successful authentication.

This extension takes effect only if both parties send it. Name-lists MAY include any compression algorithm that could have been negotiated in SSH\_MSG\_KEXINIT, except algorithms that define their own delayed compression semantics. This means "zlib,none" is a valid algorithm list in this context; but "zlib@openssh.com" is not.

If both parties send this extension, but the name-lists do not contain a common algorithm in either direction, the parties MUST disconnect in the same way as if negotiation failed as part of SSH\_MSG\_KEXINIT.

If this extension takes effect, the renegotiated compression algorithm is used as follows:

- By the server, starting with the very next SSH message after SSH\_MSG\_USERAUTH\_SUCCESS.
- By the client, after sending SSH\_MSG\_NEWCOMPRESS. If this extension takes effect, the client MUST send the following message immediately after receiving the server's SSH\_MSG\_USERAUTH\_SUCCESS:

```
byte            SSH_MSG_NEWCOMPRESS (value 8)
```

The purpose of this message is to avoid a race condition where the server cannot reliably know whether a message sent by the client was sent before or after receiving the server's USERAUTH\_SUCCESS.

As with all extensions, the server may delay including this extension until its secondary SSH\_MSG\_EXT\_INFO, sent before USERAUTH\_SUCCESS. This allows the server to avoid advertising compression support until the client has been authenticated.

In subsequent key re-exchange, the compression algorithms negotiated in re-exchange override the algorithms negotiated with this extension.

## 4. IANA Considerations

### 4.1. Additions to existing tables

IANA is requested to insert the following entries into the table Message Numbers under Secure Shell (SSH) Protocol Parameters [RFC4250]:

Value	Message ID	Reference
7	SSH_MSG_EXT_INFO	[this document]
8	SSH_MSG_NEWCOMPRESS	[this document]

IANA is requested to insert the following entries into the table Key Exchange Method Names:

Method Name	Reference	Note
ext-info-s	[this document]	Section 2.2
ext-info-c	[this document]	Section 2.2

### 4.2. New table: Extension Names

Also under Secure Shell (SSH) Protocol Parameters, IANA is requested to create a new table, Extension Names, with initial content:

Extension Name	Reference	Note
server-sig-algs	[this document]	Section 3.1
no-flow-control	[this document]	Section 3.2
accept-channels	[this document]	Section 3.3
elevation	[this document]	Section 3.4
delay-compression	[this document]	Section 3.5

#### 4.2.1. Future Assignments to Extension Names

Names in the Extension Names table MUST follow the Conventions for Names defined in [RFC4250], Section 4.6.1.

Requests for assignments of new non-local names in the Extension Names table (i.e. names not including the '@' character) MUST be done through the IETF CONSENSUS method, as described in [RFC5226].

## 6. References

### 6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4250] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, January 2006.
- [RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, January 2006.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, January 2006.
- [RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, January 2006.
- [RFC5226] Narten, T. and Alvestrand, H., "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

### 6.2. Informative References

- [SSH-RSA-SHA2] Bider, D., "Use of RSA Keys with SHA-2 256 and 512 in Secure Shell (SSH)", draft-ietf-curdle-rsa-sha2-01.txt, August 2016, <<https://tools.ietf.org/html/draft-ietf-curdle-rsa-sha2-01>>.

Author's Address

Denis Bider  
Bitvise Limited  
Suites 41/42, Victoria House  
26 Main Street  
GI

Phone: +506 8315 6519  
EMail: [ietf-ssh3@denisbider.com](mailto:ietf-ssh3@denisbider.com)  
URI: <https://www.bitvise.com/>

Acknowledgments

Thanks to Markus Friedl and Damien Miller for comments and initial implementation.



Internet Engineering Task Force  
Internet-Draft  
Updates: 4253, 4419, 4432, 4462, 5656  
(if approved)  
Intended status: Standards Track  
Expires: March 24, 2017

M. Baushke  
Juniper Networks, Inc.  
September 20, 2016

Key Exchange (KEX) Method Updates and Recommendations for Secure Shell  
(SSH)  
draft-ietf-curdle-ssh-kex-sha2-05

Abstract

This document adds recommendations for adoption of ssh-curves from the [I-D.ietf-curdle-ssh-curves] and new-modp from the [I-D.ietf-curdle-ssh-modp-dh-sha2], and deprecates some previously specified Key Exchange Method algorithm names for the Secure Shell (SSH) protocol. It also updates [RFC4253], [RFC4419], [RFC4462], and [RFC5656] by specifying the set key exchange algorithms that currently exist and which ones MUST, SHOULD, MAY, and SHOULD NOT be implemented. New key exchange methods use the SHA-2 family of hashes.

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 <http://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 24, 2017.

Copyright Notice

Copyright (c) 2016 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

(<http://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.

## 1. Overview and Rationale

Secure Shell (SSH) is a common protocol for secure communication on the Internet. In [RFC4253], SSH originally defined the Key Exchange Method Name `diffie-hellman-group1-sha1` which used [RFC2409] Oakley Group 2 (a 1024-bit MODP group) and SHA-1 [RFC3174]. Due to recent security concerns with SHA-1 [RFC6194] and with MODP groups with less than 2048 bits [NIST-SP-800-131Ar1] implementer and users request support for larger MODP group sizes with data integrity verification using the SHA-2 family of secure hash algorithms as well as MODP groups providing more security.

The United States Information Assurance Directorate (IAD) at the National Security Agency (NSA) has published a FAQ [MFQ-U-OO-815099-15] suggesting that the use of Elliptic Curve Diffie-Hellman (ECDH) using the `nistp256` curve and SHA-2 based hashes less than SHA2-384 are no longer sufficient for transport of Top Secret information. It is for this reason that this draft moves `ecdh-sha2-nistp256` from a REQUIRED to OPTIONAL as a key exchange method. This is the same reason that the stronger MODP groups being adopted. As the MODP group14 is already present in most SSH implementations and most implementations already have a SHA2-256 implementation, so `diffie-hellman-group14-sha256` is provided as an easy to implement and faster to use key exchange. Small embedded applications may find this KEX desirable to use.

The NSA Information Assurance Directorate (IAD) has also published the Commercial National Security Algorithm Suite (CNSA Suite) [CNSA-SUITE] in which the 3072-bit MODP Group 15 in RFC 3526 is explicitly mentioned as the minimum modulus to protect Top Secret communications.

It has been observed in [safe-curves] that the NIST recommended Elliptic Curve Prime Curves (P-256, P-384, and P-521) are perhaps not the best available for Elliptic Curve Cryptography (ECC) Security. For this reason, none of the [RFC5656] curves are marked as a MUST implement. However, the requirement that "every compliant SSH ECC implementation MUST implement ECDH key exchange" is now taken to mean that if `ecdsa-sha2-[identifier]` is implemented, then `ecdh-sha2-[identifier]` MUST be implemented.



Please send comments on this draft to [curdle@ietf.org](mailto:curdle@ietf.org).

## 2. Requirements Language

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. Key Exchange Algorithms

This memo adopts the style and conventions of [RFC4253] in specifying how the use of new data key exchange is indicated in SSH.

A new set of Elliptic Curve Diffie-Hellman ssh-curves exist. The curve25519-sha256 MUST be adopted where possible.

As a hedge against uncertainty raised by the NSA IAD FAQ publication, new MODP Diffie-Hellman based key exchanges are proposed for inclusion in the set of key exchange method names as well as the curve448-sha512 curve.

The following new key exchange algorithms are defined:

Key Exchange Method Name	Note
-----	-----
curve25519-sha256	MUST/REQUIRED
curve448-sha512	MAY/OPTIONAL
diffie-hellman-group14-sha256	MUST/REQUIRED
diffie-hellman-group15-sha512	MAY/OPTIONAL
diffie-hellman-group16-sha512	SHOULD/RECOMMENDED
diffie-hellman-group17-sha512	MAY/OPTIONAL
diffie-hellman-group18-sha512	MAY/OPTIONAL
gss-group14-sha256-*	SHOULD/RECOMMENDED
gss-group15-sha512-*	MAY/OPTIONAL
gss-group16-sha512-*	SHOULD/RECOMMENDED
gss-group17-sha512-*	MAY/OPTIONAL
gss-group18-sha512-*	MAY/OPTIONAL

The SHA-2 family of secure hash algorithms are defined in [FIPS-180-4].

## 4. IANA Considerations

This RFC augments the Key Exchange Method Names in [RFC4253]. It downgrades the use of SHA-1 hashing for key exchange methods in [RFC4419], [RFC4432], and [RFC4462]. It also moves from MUST to SHOULD the ecdh-sha2-nistp256 given in [RFC5656].

It adds a new set of named "gss-\*" methods to [RFC4462] with a MAY recommendation.

It is desirable to also include the new-modp from the [I-D.ietf-curdle-ssh-modp-dh-sha2] in this list.

It is desirable to also include the ssh-curves from the [I-D.ietf-curdle-ssh-curves] in this list. The "curve25519-sha256" is currently available in some Secure Shell implementations under the name "curve25519-sha256@libssh.org" and is the best candidate for a fast, safe, and secure key exchange method.

IANA is requested to update the SSH algorithm registry to ensure that all of the listed Key Exchange Method Name and References exist in the following table. However, the Implement column is just the current recommendations of this RFC.

Key Exchange Method Name	Reference	Implement
-----	-----	-----
curve25519-sha256	ssh-curves	MUST
curve448-sha512	ssh-curves	MAY
diffie-hellman-group-exchange-sha1	RFC4419	SHOULD NOT
diffie-hellman-group-exchange-sha256	RFC4419	MAY
diffie-hellman-group1-sha1	RFC4253	SHOULD NOT
diffie-hellman-group14-sha1	RFC4253	SHOULD
diffie-hellman-group14-sha256	new-modp	MUST
diffie-hellman-group15-sha512	new-modp	MAY
diffie-hellman-group16-sha512	new-modp	SHOULD
diffie-hellman-group17-sha512	new-modp	MAY
diffie-hellman-group18-sha512	new-modp	MAY
ecdh-sha2-nistp256	RFC5656	SHOULD
ecdh-sha2-nistp384	RFC5656	SHOULD
ecdh-sha2-nistp521	RFC5656	SHOULD
ecdh-sha2-*	RFC5656	MAY
ecmqv-sha2	RFC5656	SHOULD NOT
gss-gex-sha1-*	RFC4462	SHOULD NOT
gss-group1-sha1-*	RFC4462	SHOULD NOT
gss-group14-sha1-*	RFC4462	SHOULD
gss-group14-sha256-*	new-modp	SHOULD
gss-group15-sha512-*	new-modp	MAY
gss-group16-sha512-*	new-modp	SHOULD
gss-group17-sha512-*	new-modp	MAY
gss-group18-sha512-*	new-modp	MAY
gss-*	RFC4462	MAY
rsa1024-sha1	RFC4432	SHOULD NOT
rsa2048-sha256	RFC4432	MAY

The Implement column in the above table is a suggestion/recommendation for the listed key exchange method to be implemented in the default list of key exchange methods. It is up to the end-user as to what algorithms they choose to be able to negotiate, so the KEX algorithms should be configurable by the administrator of the server as well as the user of the client. This RFC is intended to provide IANA defined names for these groups for interoperability. The Note column of the IANA table should probably continue to point to the implementation detail sections of the Reference RFCs where appropriate.

The guidance of this RFC is that the SHA-1 algorithm hashing SHOULD NOT be used. If it is used, it should only be provided for backwards compatibility, should not be used in new designs, and should be phased out of existing key exchanges as quickly as possible because of its known weaknesses. Any key exchange using SHA-1 SHOULD NOT be in a default key exchange list if at all possible. If they are needed for backward compatibility, they SHOULD be listed after all of the SHA-2 based key exchanges.

The RFC4253 REQUIRED diffie-hellman-group14-sha1 method SHOULD be retained for compatibility with older Secure Shell implementations. It is intended that this key exchange method be phased out as soon as possible.

It is believed that all current SSH implementations should be able to achieve an implementation of the "diffie-hellman-group14-sha256" method. To that end, this is one method that MUST be implemented.

If GSS-API methods are available, then the RFC4462 REQUIRED gss-group14-sha1-\* method SHOULD be retained for compatibility with older Secure Shell implementations and the gss-groups14-sha256-\* method SHOULD be added as for "sha1".

[TO BE REMOVED: This registration should take place at the following location: <<http://www.iana.org/assignments/ssh-parameters/ssh-parameters.xhtml#ssh-parameters-16>>]

## 5. Acknowledgements

Thanks to the following people for review and comments: Denis Bider, Peter Gutmann, Damien Miller, Niels Moeller, Matt Johnston, Iwamoto Kouichi, Simon Josefsson, Dave Dugal, Daniel Migault.

Thanks to the following people for code to implement interoperable exchanges using some of these groups as found in this draft: Darren Tucker for OpenSSH and Matt Johnston for Dropbear. And thanks to Iwamoto Kouichi for information about RLogin, Tera Term (ttssh)

and Poderosa implementations also adopting new Diffie-Hellman groups based on this draft.

## 6. Security Considerations

The security considerations of [RFC4253] apply to this RFC.

The security considerations of [RFC3526] suggest that these MODP groups have security strengths given in this table. They are based on [RFC3766] Determining Strengths For Public Keys Used For Exchanging Symmetric Keys.

Group modulus security strength estimates (RFC3526)

Group	Modulus	Strength Estimate 1		Strength Estimate 2	
		in bits	exponent size	in bits	exponent size
14	2048-bit	110	220-	160	320-
15	3072-bit	130	260-	210	420-
16	4096-bit	150	300-	240	480-
17	6144-bit	170	340-	270	540-
18	8192-bit	190	380-	310	620-

Figure 1

Many users seem to be interested in the perceived safety of using larger MODP groups and hashing with SHA2-based algorithms.

## 7. References

### 7.1. Normative References

- [FIPS-180-4] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, August 2015, <<http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

- [RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, DOI 10.17487/RFC3526, May 2003, <<http://www.rfc-editor.org/info/rfc3526>>.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <<http://www.rfc-editor.org/info/rfc4253>>.

## 7.2. Informative References

- [CNSA-SUITE]  
"Information Assurance by the National Security Agency",  
"Commercial National Security Algorithm Suite", September  
2016, <<https://www.iad.gov/iad/programs/iad-initiatives/cnsa-suite.cfm>>.
- [I-D.ietf-curdle-ssh-curves]  
Adamantiadis, A. and S. Josefsson, "Secure Shell (SSH) Key  
Exchange Method using Curve25519 and Curve448", draft-  
ietf-curdle-ssh-curves-00 (work in progress), March 2016.
- [I-D.ietf-curdle-ssh-modp-dh-sha2]  
Baushke, M., "More Modular Exponential (MODP) Diffie-  
Hellman (DH) Key Exchange (KEX) Groups for Secure Shell  
(SSH)", draft-ietf-curdle-ssh-modp-dh-sha2-00 (work in  
progress), September 2016.
- [MFQ-U-OO-815099-15]  
"National Security Agency/Central Security Service", "CNSA  
Suite and Quantum Computing FAQ", January 2016,  
<<https://www.iad.gov/iad/library/ia-guidance/ia-solutions-for-classified/algorithm-guidance/cnsa-suite-and-quantum-computing-faq.cfm>>.
- [NIST-SP-800-131Ar1]  
Barker, and Roginsky, "Transitions: Recommendation for the  
Transitioning of the Use of Cryptographic Algorithms and  
Key Lengths", NIST Special Publication 800-131A Revision  
1, November 2015,  
<<http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf>>.
- [RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange  
(IKE)", RFC 2409, DOI 10.17487/RFC2409, November 1998,  
<<http://www.rfc-editor.org/info/rfc2409>>.

- [RFC3174] Eastlake 3rd, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, DOI 10.17487/RFC3174, September 2001, <<http://www.rfc-editor.org/info/rfc3174>>.
- [RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, DOI 10.17487/RFC3766, April 2004, <<http://www.rfc-editor.org/info/rfc3766>>.
- [RFC4419] Friedl, M., Provos, N., and W. Simpson, "Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol", RFC 4419, DOI 10.17487/RFC4419, March 2006, <<http://www.rfc-editor.org/info/rfc4419>>.
- [RFC4432] Harris, B., "RSA Key Exchange for the Secure Shell (SSH) Transport Layer Protocol", RFC 4432, DOI 10.17487/RFC4432, March 2006, <<http://www.rfc-editor.org/info/rfc4432>>.
- [RFC4462] Hutzelman, J., Salowey, J., Galbraith, J., and V. Welch, "Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol", RFC 4462, DOI 10.17487/RFC4462, May 2006, <<http://www.rfc-editor.org/info/rfc4462>>.
- [RFC5656] Stebila, D. and J. Green, "Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer", RFC 5656, DOI 10.17487/RFC5656, December 2009, <<http://www.rfc-editor.org/info/rfc5656>>.
- [RFC6194] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <<http://www.rfc-editor.org/info/rfc6194>>.
- [safe-curves]  
Bernstein, and Lange, "SafeCurves: choosing safe curves for elliptic-curve cryptography.", February 2016, <<https://safecurves.cr.yp.to/>>.

Author's Address

Mark D. Baushke  
Juniper Networks, Inc.  
1133 Innovation Way  
Sunnyvale, CA 94089-1228  
US

Phone: +1 408 745 2952  
Email: [mdb@juniper.net](mailto:mdb@juniper.net)  
URI: <http://www.juniper.net/>

Internet Engineering Task Force  
Internet-Draft  
Updates: 4253, 4432, 4462 (if approved)  
Intended status: Standards Track  
Expires: March 17, 2017

M. Baushke  
Juniper Networks, Inc.  
September 13, 2016

More Modular Exponential (MODP) Diffie-Hellman (DH) Key Exchange (KEX)  
Groups for Secure Shell (SSH)  
draft-ietf-curdle-ssh-modp-dh-sha2-01

## Abstract

This document defines added Modular Exponential (MODP) Groups for the Secure Shell (SSH) protocol using SHA-2 hashes.

## 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 <http://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 17, 2017.

## Copyright Notice

Copyright (c) 2016 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 (<http://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.



## 1. Overview and Rationale

Secure Shell (SSH) is a common protocol for secure communication on the Internet. Due to recent security concerns with SHA-1 [RFC6194] and with MODP groups with less than 2048 bits [NIST-SP-800-131Ar1] implementer and users request support for larger Diffie Hellman (DH) MODP group sizes with data integrity verification using the SHA-2 family of secure hash algorithms as well as MODP groups providing more security.

The United States Information Assurance Directorate at the National Security Agency has published a FAQ [MFQ-U-00-815099-15] suggesting both: a) DH groups using less than 3072-bits, and b) the use of SHA-2 based hashes less than SHA2-384, are no longer sufficient for transport of Top Secret information. For this reason, the new MODP groups are being introduced starting with the MODP 3072-bit group 15 are all using SHA2-512 as the hash algorithm.

The DH 2048-bit MODP group 14 is already present in most SSH implementations and most implementations already have a SHA2-256 implementation, so diffie-hellman-group14-sha256 is provided as an easy to implement and faster to use key exchange for small embedded applications.

In [RFC4462], there is another method for providing DH key exchange with MODP Groups using "Generic Security Service Application Program Interface (GSS-API)". This RFC extends the "gss-\*" MODP DH groups and provides for using SHA-2 based hashes for them as well.

Please send comments on this draft to [ietf-ssh@NetBSD.org](mailto:ietf-ssh@NetBSD.org) and [ietf-curdle@ietf.org](mailto:ietf-curdle@ietf.org).

## 2. Requirements Language

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. Key Exchange Algorithms

This memo adopts the style and conventions of [RFC4253] in specifying how the use of new data key exchange is indicated in SSH.

The following new key exchange algorithms are defined:

```
Key Exchange Method Name
diffie-hellman-group14-sha256
diffie-hellman-group15-sha512
diffie-hellman-group16-sha512
diffie-hellman-group17-sha512
diffie-hellman-group18-sha512
gss-group14-sha256-*
gss-group15-sha512-*
gss-group16-sha512-*
gss-group17-sha512-*
gss-group18-sha512-*
```

Figure 1

The SHA-2 family of secure hash algorithms are defined in [FIPS-180-4].

The method of key exchange used for the name "diffie-hellman-group14-sha256" is the same as that for "diffie-hellman-group14-sha1" except that the SHA2-256 hash algorithm is used.

The method of key exchange used for the name "gss-group14-sha256-\*" is the same as that for "gss-group14-sha1-\*" except that the SHA2-256 hash algorithm is used.

The group15 through group18 names are the same as those specified in [RFC3526] 3071-bit MODP Group 15, 4096-bit MODP Group 16, 6144-bit MODP Group 17, and 8192-bit MODP Group 18.

The SHA2-512 algorithm is to be used when "sha512" is specified as a part of the key exchange method name.

#### 4. IANA Considerations

This document augments the Key Exchange Method Names in [RFC4253].

IANA is requested to add to the Key Exchange Method Names algorithm registry with the following entries:

Key Exchange Method Name	Reference
-----	-----
diffie-hellman-group14-sha256	This Draft
diffie-hellman-group15-sha512	This Draft
diffie-hellman-group16-sha512	This Draft
diffie-hellman-group17-sha512	This Draft
diffie-hellman-group18-sha512	This Draft
gss-group14-sha256-*	This Draft
gss-group15-sha512-*	This Draft
gss-group16-sha512-*	This Draft
gss-group17-sha512-*	This Draft
gss-group18-sha512-*	This Draft

[TO BE REMOVED: This registration should take place at the following location: <<http://www.iana.org/assignments/ssh-parameters/ssh-parameters.xhtml#ssh-parameters-16>>]

## 5. Security Considerations

The security considerations of [RFC4253] apply to this document.

The security considerations of [RFC3526] suggest that these MODP groups have security strengths given in this table. They are based on [RFC3766] Determining Strengths For Public Keys Used For Exchanging Symmetric Keys.

Group modulus security strength estimates (RFC3526)

Group	Modulus	Strength Estimate 1		Strength Estimate 2	
		in bits	exponent size	in bits	exponent size
14	2048-bit	110	220-	160	320-
15	3072-bit	130	260-	210	420-
16	4096-bit	150	300-	240	480-
17	6144-bit	170	340-	270	540-
18	8192-bit	190	380-	310	620-

Figure 2

Many users seem to be interested in the perceived safety of using larger MODP groups and hashing with SHA2-based algorithms.

## 6. References

### 6.1. Normative References

- [FIPS-180-4]  
National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, August 2015, <<http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, DOI 10.17487/RFC3526, May 2003, <<http://www.rfc-editor.org/info/rfc3526>>.
- [RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, DOI 10.17487/RFC4253, January 2006, <<http://www.rfc-editor.org/info/rfc4253>>.

### 6.2. Informative References

- [MFQ-U-OO-815099-15]  
"National Security Agency/Central Security Service", "CNSA Suite and Quantum Computing FAQ", January 2016, <<https://www.iad.gov/iad/library/ia-guidance/ia-solutions-for-classified/algorithm-guidance/cnsa-suite-and-quantum-computing-faq.cfm>>.
- [NIST-SP-800-131Ar1]  
Barker, and Roginsky, "Transitions: Recommendation for the Transitioning of the Use of Cryptographic Algorithms and Key Lengths", NIST Special Publication 800-131A Revision 1, November 2015, <<http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-131Ar1.pdf>>.
- [RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, DOI 10.17487/RFC3766, April 2004, <<http://www.rfc-editor.org/info/rfc3766>>.

- [RFC4462] Hutzelman, J., Salowey, J., Galbraith, J., and V. Welch, "Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol", RFC 4462, DOI 10.17487/RFC4462, May 2006, <<http://www.rfc-editor.org/info/rfc4462>>.
- [RFC6194] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <<http://www.rfc-editor.org/info/rfc6194>>.

## Author's Address

Mark D. Baushke  
Juniper Networks, Inc.  
1133 Innovation Way  
Sunnyvale, CA 94089-1228  
US

Phone: +1 408 745 2952  
Email: [mdb@juniper.net](mailto:mdb@juniper.net)  
URI: <http://www.juniper.net/>