Internet-Draft
Intended status: Standards Track
Expires: 22 February 2018

R. Housley
Vigil Security

22 August 2017

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-10.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 22 February 2017.

Copyright Notice

Table of Contents

1. Introduction

This document describes the conventions for using Elliptic Curve Diffie-Hellman (ECDH) key agreement using curve25519 and curve448 [CURVES] 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 [CURVES]. Those other curves are not deprecated.

Using curve25519 with Diffie-Hellman key agreement is referred to as X25519. Using curve448 with Diffie-Hellman key agreement is referred to as 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 described 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 [CURVES].

The originator MUST use 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 MUST be used for a single CMS protected content type, and then it MUST be discarded. The originator obtains the recipient's static public key from the recipient's certificate [PROFILE].

X25519 is described in Section 6.1 of [CURVES], and X448 is described in Section 6.2 of [CURVES]. Conforming implementations MUST check whether the computed Diffie-Hellman shared secret is the all-zero value, and abort if so, as described in Section 6 of [CURVES]. If an alternative implementation of these elliptic curves to that documented in Section 6 of [CURVES] is employed, then the additional checks specified in Section 7 of [CURVES] SHOULD be performed.

In [CURVES], 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 (KEK) from the shared secret value (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. For example, the AES Key Wrap algorithm [AESKW] does not need parameters, so the algorithm identifier parameters are absent.

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. By including the ukm, a different key-encryption key is generated even when the originator ephemeral private key is improperly used more than once. Therefore, if the ukm field is present, it MUST be selected in a manner that provides with very high probability a unique value; however, there is no security benefit to using a ukm value that is longer than the key-encryption key that will be produced by the KDF.

The ECC-CMS-SharedInfo suppPubInfo field contains the length of the generated key-encryption key, in bits, represented as a 32-bit number in network byte order. For example, the key length for AES-256 [AES] 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 American National Standard 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 (KEK), the KDF 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 32-bit counter is represented in network byte order. The KM blocks are concatenated left to right, and then the

leftmost portion of the result is used as the pairwise key-encryption key, KEK:

```
KM(i) = Hash(K || INT32(counter=i) || DER(ECC-CMS-SharedInfo))

KEK = KM(counter=1) || KM(counter=2) ...
```

2.2. HKDF

The HMAC-based Extract-and-Expand Key Derivation Function (HKDF) is a robust construct based on a one-way hash function described in RFC 5869 [HKDF]. 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 optionally includes the ukm. If the ukm is present, the ukm is also used as the HKDF salt. HKDF uses an appropriate number of zero octets when no salt is provided.

The length of the generated key-encryption key is used in 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, to produce the pairwise key-encryption key, KEK:

```
if ukm is provided, then salt = ukm, else salt is not provided
PRK = HKDF-Extract(salt, K)

KEK = HKDF-Expand(PRK, DER(ECC-CMS-SharedInfo), SizeInOctets(KEK))
```

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 is more than one recipient, the same content-encryption key MUST be 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 Section 6 of [CMS]. The RecipientInfo choice is described in Section 6.2 of [CMS], and repeated here for convenience.

```
RecipientInfo ::= CHOICE {
  ktri KeyTransRecipientInfo,
  kari [1] KeyAgreeRecipientInfo,
  kekri [2] KEKRecipientInfo,
  pwri [3] PasswordRecipientinfo,
  ori [4] OtherRecipientInfo }
```

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-X25519 or the id-X448 object identifier. The originator's ephemeral public key MUST be encoded as an OCTET STRING.

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

When using X25519, the public key contains exactly 32 octets, and the id-X25519 object identifier is used:

```
id-X25519 OBJECT IDENTIFIER ::= { 1 3 101 110 }
```

When using X448, the public key contains exactly 56 octets, and the id-X448 object identifier is used:

```
id-X448 OBJECT IDENTIFIER ::= { 1 3 101 111 }
```

KeyAgreeRecipientInfo ukm is optional. The processing of the ukm with The ANSI-X9.63-KDF key derivation function is described in Section 2.1, and the processing of the ukm with the HKDF key derivation function is described in Section 2.2.

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 is more than one recipient, the same authentication key MUST be 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 [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 is more than one recipient, the same content-authenticated-encryption key MUST be 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 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 for carrying X25519 and X448 public keys are specified in [ID.curdle-pkix].

7. Key Agreement Algorithm Identifiers

The following object identifiers are assigned in [CMSECC] to indicate ECDH with ANSI-X9.63-KDF using various one-way hash functions. These are expected to be used as AlgorithmIdentifiers with a parameter that specifies the key-encryption algorithm. These are repeated here for convenience.

```
secg-scheme OBJECT IDENTIFIER ::= {
  iso(1) identified-organization(3) certicom(132) schemes(1) }

dhSinglePass-stdDH-sha256kdf-scheme OBJECT IDENTIFIER ::= {
  secg-scheme 11 1 }

dhSinglePass-stdDH-sha384kdf-scheme OBJECT IDENTIFIER ::= {
  secg-scheme 11 2 }

dhSinglePass-stdDH-sha512kdf-scheme OBJECT IDENTIFIER ::= {
  secg-scheme 11 3 }
```

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.

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

dhSinglePass-stdDH-hkdf-sha256-scheme OBJECT IDENTIFIER ::= {
   smime-alg 19 }

dhSinglePass-stdDH-hkdf-sha384-scheme OBJECT IDENTIFIER ::= {
   smime-alg 20 }
```

```
dhSinglePass-stdDH-hkdf-sha512-scheme OBJECT IDENTIFIER ::= {
   smime-alg 21 }
```

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 hexidecimal) from [CMSECC] might be of interest to implementations that support X25519 and X448:

ECDH 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
```

ECDH 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
```

ECDH 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
```

ECDH 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
```

ECDH 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
```

ECDH 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 hexidecimal) 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:

```
30 1A 06 0B 2A 86 48 86 F7 0D 01 09 10 03 13 30 0B 06 09 60 86
48 01 65 03 04 01 05
```

```
ECDH with HKDF using SHA-384; uses AES-128 key wrap:
   30 1A 06 0B 2A 86 48 86 F7 0D 01 09 10 03 14 30 0B 06 09 60 86
   48 01 65 03 04 01 05

ECDH with HKDF using SHA-512; uses AES-128 key wrap:
   30 1A 06 0B 2A 86 48 86 F7 0D 01 09 10 03 15 30 0B 06 09 60 86
   48 01 65 03 04 01 05

ECDH with HKDF using SHA-256; uses AES-256 key wrap:
   30 1A 06 0B 2A 86 48 86 F7 0D 01 09 10 03 13 30 0B 06 09 60 86
   48 01 65 03 04 01 2D

ECDH with HKDF using SHA-384; uses AES-256 key wrap:
   30 1A 06 0B 2A 86 48 86 F7 0D 01 09 10 03 14 30 0B 06 09 60 86
   48 01 65 03 04 01 2D

ECDH with HKDF using SHA-512; uses AES-256 key wrap:
   30 1A 06 0B 2A 86 48 86 F7 0D 01 09 10 03 15 30 0B 06 09 60 86
   48 01 65 03 04 01 2D
```

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; this ensures that interoperability is not a concern whether 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

One object identifier for the ASN.1 module in the Appendix was assigned in the SMI Security for S/MIME Module Identifiers (1.2.840.113549.1.9.16.0) [IANA-MOD] registry:

```
id-mod-cms-ecdh-alg-2017 OBJECT IDENTIFIER ::= {
   iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
   pkcs-9(9) smime(16) mod(0) 67 }
```

Three object identifiers for the Key Agreement Algorithm Identifiers in Sections 7 were assigned in the SMI Security for S/MIME Algorithms (1.2.840.113549.1.9.16.3) [IANA-ALG] registry:

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

dhSinglePass-stdDH-hkdf-sha256-scheme OBJECT IDENTIFIER ::= {
   smime-alg 19 }

dhSinglePass-stdDH-hkdf-sha384-scheme OBJECT IDENTIFIER ::= {
   smime-alg 20 }

dhSinglePass-stdDH-hkdf-sha512-scheme OBJECT IDENTIFIER ::= {
   smime-alg 21 }
```

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.

[CMSASN1] Hoffman, P., and J. Schaad, "New ASN.1 Modules for Cryptographic Message Syntax (CMS) and S/MIME", RFC 5911, June 2010.

[CMSECC] Turner, S., and D. Brown, "Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)", RFC 5753, January 2010.

[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.

[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

[AES] National Institute of Standards and Technology. FIPS Pub 197: Advanced Encryption Standard (AES). 26 November 2001.

[AESKW] Schaad, J., and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, September 2002.

[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.

[IANA-ALG] https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#security-smime-3.

[IANA-MOD] https://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml#security-smime-0.

[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.

Appendix: ASN.1 Module

```
   CMSECDHAlgs-2017
     { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
       smime(16) modules(0) id-mod-cms-ecdh-alg-2017(67) }

   DEFINITIONS IMPLICIT TAGS ::=
   BEGIN

   -- EXPORTS ALL

   IMPORTS

     KeyWrapAlgorithm
       FROM CryptographicMessageSyntaxAlgorithms-2009  -- in [CMSASN1]
         { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
           pkcs-9(9) smime(16) modules(0) id-mod-cmsalg-2001-02(37) }

     KEY-AGREE, SMIME-CAPS
       FROM AlgorithmInformation-2009  -- in [CMSASN1]
         { iso(1) identified-organization(3) dod(6) internet(1)
           security(5) mechanisms(5) pkix(7) id-mod(0)
           id-mod-algorithmInformation-02(58) }

     dhSinglePass-stdDH-sha256kdf-scheme,
     dhSinglePass-stdDH-sha384kdf-scheme,
     dhSinglePass-stdDH-sha512kdf-scheme,
     kaa-dhSinglePass-stdDH-sha256kdf-scheme,
     kaa-dhSinglePass-stdDH-sha384kdf-scheme,
     kaa-dhSinglePass-stdDH-sha512kdf-scheme,
     cap-kaa-dhSinglePass-stdDH-sha256kdf-scheme,
     cap-kaa-dhSinglePass-stdDH-sha384kdf-scheme,
     cap-kaa-dhSinglePass-stdDH-sha512kdf-scheme
       FROM CMSECCAlgs-2009-02  -- in [CMSECC]
         { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
           pkcs-9(9) smime(16) modules(0)
           id-mod-cms-ecc-alg-2009-02(46) }
     ;
```

```
   --
   -- Object Identifiers
   --

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

   dhSinglePass-stdDH-hkdf-sha256-scheme OBJECT IDENTIFIER ::= {
      smime-alg 19 }

   dhSinglePass-stdDH-hkdf-sha384-scheme OBJECT IDENTIFIER ::= {
      smime-alg 20 }

   dhSinglePass-stdDH-hkdf-sha512-scheme OBJECT IDENTIFIER ::= {
      smime-alg 21 }

   --
   -- Extend the Key Agreement Algorithms in [CMSECC]
   --

   KeyAgreementAlgs KEY-AGREE ::= { ...,
     kaa-dhSinglePass-stdDH-sha256kdf-scheme   |
     kaa-dhSinglePass-stdDH-sha384kdf-scheme   |
     kaa-dhSinglePass-stdDH-sha512kdf-scheme   |
     kaa-dhSinglePass-stdDH-hkdf-sha256-scheme |
     kaa-dhSinglePass-stdDH-hkdf-sha384-scheme |
     kaa-dhSinglePass-stdDH-hkdf-sha512-scheme }

   kaa-dhSinglePass-stdDH-hkdf-sha256-scheme KEY-AGREE ::= {
     IDENTIFIER dhSinglePass-stdDH-hkdf-sha256-scheme
     PARAMS TYPE KeyWrapAlgorithm ARE required
     UKM -- TYPE unencoded data -- ARE preferredPresent
     SMIME-CAPS cap-kaa-dhSinglePass-stdDH-hkdf-sha256-scheme }

   kaa-dhSinglePass-stdDH-hkdf-sha384-scheme KEY-AGREE ::= {
     IDENTIFIER dhSinglePass-stdDH-hkdf-sha384-scheme
     PARAMS TYPE KeyWrapAlgorithm ARE required
     UKM -- TYPE unencoded data -- ARE preferredPresent
     SMIME-CAPS cap-kaa-dhSinglePass-stdDH-hkdf-sha384-scheme }

   kaa-dhSinglePass-stdDH-hkdf-sha512-scheme KEY-AGREE ::= {
     IDENTIFIER dhSinglePass-stdDH-hkdf-sha512-scheme
     PARAMS TYPE KeyWrapAlgorithm ARE required
     UKM -- TYPE unencoded data -- ARE preferredPresent
     SMIME-CAPS cap-kaa-dhSinglePass-stdDH-hkdf-sha512-scheme }
```

```
   --
   -- Extend the S/MIME CAPS in [CMSECC]
   --

   SMimeCAPS SMIME-CAPS ::= { ...,
     kaa-dhSinglePass-stdDH-sha256kdf-scheme.&smimeCaps   |
     kaa-dhSinglePass-stdDH-sha384kdf-scheme.&smimeCaps   |
     kaa-dhSinglePass-stdDH-sha512kdf-scheme.&smimeCaps   |
     kaa-dhSinglePass-stdDH-hkdf-sha256-scheme.&smimeCaps |
     kaa-dhSinglePass-stdDH-hkdf-sha384-scheme.&smimeCaps |
     kaa-dhSinglePass-stdDH-hkdf-sha512-scheme.&smimeCaps }

   cap-kaa-dhSinglePass-stdDH-hkdf-sha256-scheme SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithm
     IDENTIFIED BY dhSinglePass-stdDH-hkdf-sha256-scheme }

   cap-kaa-dhSinglePass-stdDH-hkdf-sha384-scheme SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithm
     IDENTIFIED BY dhSinglePass-stdDH-hkdf-sha384-scheme}

   cap-kaa-dhSinglePass-stdDH-hkdf-sha512-scheme SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithm
     IDENTIFIED BY dhSinglePass-stdDH-hkdf-sha512-scheme }

   END
```

Acknowledgements

Many thanks to Roni Even, Daniel Migault, Eric Rescorla, Jim Schaad, Stefan Santesson, and Sean Turner for their review and insightful suggestions.

Author's Address

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