S/MIME WG
Internet Draft
Intended Status: Informational
Updates: 3278 (once approved)
Expires: November 6, 2008

Sean Turner, IECA
Daniel Brown, Certicom

May 6, 2008

Update to Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS) draft-ietf-smime-rfc3278-update-04.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of 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/ietf/1id-abstracts.txt

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

This Internet-Draft will expire on November 6, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2008).

Abstract

RFC 3278 describes how to use Elliptic Curve Cryptography (ECC) public-key algorithms in the Cryptographic Message Syntax (CMS). This document updates RFC 3278 to add support for the SHA2 family of hash algorithms, Elliptic Curve Digital Signature Algorithm (ECDSA)

224-512, and Key Derivation Functions (KDFs) that utilize SHA2 algorithms.

Discussion

This draft is being discussed on the 'ietf-smime' mailing list. To subscribe, send a message to ietf-smime-request@imc.org with the single word subscribe in the body of the message. There is a Web site for the mailing list at <http://www.imc.org/ietf-smime/>.

Table of Contents

1. Introduction

RFC 3278 describes how to use Elliptic Curve Cryptography (ECC) public-key algorithms in the Cryptographic Message Syntax (CMS). This document updates RFC 3278 to add support for the SHA2 family of hash algorithms, Elliptic Curve Digital Signature Algorithm (ECDSA) 224-512, and Key Derivation Functions (KDFs) that utilize SHA2 algorithms.

1.1. Overview of Changes to RFC 3278

The following summarizes the changes:

- Paragraph 2.1.1 limited the digest algorithm to SHA-1. This document expands the allowed algorithms to SHA-224, SHA-256, SHA-384, and SHA-512.

- Paragraph 3.1.1 used SHA1 in the KDF with ECDH std and cofactor methods. This document expands the set of allowed algorithms by adding SHA-224, SHA-256, SHA-384, and SHA-512.

- Paragraph 3.1.2 used SHA1 in the KDF with ECMQV. This document expands the set of allowed allowed algorithms by adding SHA-224, SHA-256, SHA-384, and SHA-512.

- Paragraph 5 is updated to include requirements for hash algorithms and recommendations for matching curves and hash algorithms. It also was expanded to indicate which ECDH and ECMQV variants are required.

- Paragraph 7 is updated to include S/MIME capabilities for ECDSA with SHA-224, SHA-256, SHA-384, and SHA-512. It was also updated to include S/MIME capabilities for ECDH and ECMQV using SHA2 algorithms as the KDF.

- Paragraph 8.1 listed the algorithm identifiers for SHA-1 and SHA-1 with ECDSA. This document adds algorithm identifiers for SHA-224, SHA-256, SHA-384, and SHA-512 as well as SHA-224, SHA-256, SHA-384, and SHA-512 with ECDSA. This document also updates the list of algorithm identifiers for ECDH std, ECDH cofactor, and ECMQV with SHA2 algorithms as the KDF.

- Paragraph 9 references need to be updated.

- Added ASN.1 module.

- Security considerations paragraph referring to definitions of SHA-224, SHA-256, SHA-384, and SHA-512 needs to be deleted.

1.2. Conventions Used in this Document

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 [MUST].

2. Updates to Paragraph 2.1.1

Old:

> digestAlgorithm MUST contain the algorithm identifier sha-1 (see Section 8.1) which identifies the SHA-1 hash algorithm.

signatureAlgorithm contains the algorithm identifier ecdsa-with-SHA1 (see Section 8.1) which identifies the ECDSA signature algorithm.

New:

digestAlgorithm MUST contain the algorithm identifier of the hash algorithm (see Section 8.1) which MUST be one of the following: id-sha1 identifies the SHA-1 hash algorithm, id-sha224 identifies the SHA-224 hash algorithm, id-sha256 identifies the SHA-256 hash algorithm, id-sha384 identifies the SHA-384 algorithm, and id-sha512 identifies the SHA-512 algorithm.

signatureAlgorithm contains the signature algorithm identifier (see Section 8.1): ecdsa-with-SHA1, ecdsa-with-SHA224, ecdsa-with-SHA256, ecdsa-with-SHA384, or ecdsa-with-SHA512.

3. Updates to Paragraph 3.1.1

Old:

keyEncryptionAlgorithm MUST contain the dhSinglePass-stdDH-sha1kdf-scheme object identifier (see Section 8.1) if standard ECDH primitive is used, or the dhSinglePass-cofactorDH-sha1kdf-scheme object identifier (see Section 8.1) if the cofactor ECDH primitive is used. The parameters field contains KeyWrapAlgorithm. The KeyWrapAlgorithm is the algorithm identifier that indicates the symmetric encryption algorithm used to encrypt the content-encryption key (CEK) with the key-encryption key (KEK).

New:

keyEncryptionAlgorithm MUST contain the key encryption algorithm object identifier (see Section 8.1). The parameters field contains KeyWrapAlgorithm. The KeyWrapAlgorithm is the algorithm identifier that indicates the symmetric encryption algorithm used to encrypt the content-encryption key (CEK) with the key-encryption key (KEK). Algorithm requirements are found in paragraph 5.

4. Updates to Paragraph 3.2.1

Old:

keyEncryptionAlgorithm MUST be the mqvSinglePass-sha1kdf-scheme algorithm identifier (see Section 8.1), with the parameters field KeyWrapAlgorithm. The KeyWrapAlgorithm indicates the symmetric

encryption algorithm used to encrypt the CEK with the KEK generated using the 1-Pass ECMQV algorithm.

New:

keyEncryptionAlgorithm MUST be the key encryption algorithm identifier (see Section 8.1), with the parameters field KeyWrapAlgorithm. The KeyWrapAlgorithm indicates the symmetric encryption algorithm used to encrypt the CEK with the KEK generated using the 1-Pass ECMQV algorithm. Algorithm requirements are found in paragraph 5.

5. Updates to Paragraph 5

Add the following to the end of the section:

Implementations of this specification MUST implement the SHA-256 hash algorithm. The SHA-1, SHA-224, SHA-384, SHA-512 hash algorithms MAY be supported.

When ECDSA, ECDH, or ECMQV is used, it is RECOMMENDED that the P-256 curve be used with SHA-256, the P-384 curve be used with SHA-384, and the P-521 curve be used with SHA-512.

Implementations of this specification that support EnvelopedData with ephemeral-static ECDH standard primitive MUST support the dhSinglePass-stdDH-sha256kdf-scheme algorithm. They MUST also support the id-aes128-wrap algorithm. The dhSinglePass-stdDH-sha1kdf-scheme, dhSinglePass-stdDH-sha224kdf-scheme, dhSinglePass-stdDH-sha384kdf-scheme, and dhSinglePass-stdDH-sha512kdf-scheme algorithms MAY be supported. Likewise, the id-alg-CMS3DESwrap, id-aes198-wrap, and id-aes256wrap MAY be supported.

Implementations of this specification that support EnvelopedData with ephemeral-static ECDH cofactor primitive MUST support the dhSinglePass-cofactorDH-sha256kdf-scheme algorithm. They MUST also support the id-aes128-wrap algorithm. The dhSinglePass-cofactorDH-sha1kdf-scheme, dhSinglePass-cofactorDH-sha224kdf-scheme, dhSinglePass-cofactorDH-sha384kdf-scheme, and dhSinglePass-cofactorDH-sha512kdf-scheme algorithms MAY be supported. Likewise, the id-alg-CMS3DESwrap, id-aes198-wrap, and id-aes256wrap MAY be supported.

Implementations of this specification that support EnvelopedData with ECMQV MUST support the mqvSinglePass-sha256kdf-scheme algorithm. They MUST also support the id-aes128-wrap algorithm.

The mqvSinglePass-sha1kdf-scheme, mqvSinglePass-sha224kdf-scheme, mqvSinglePass-sha384kdf-scheme, and mqvSinglePass-sha512kdf-scheme algorithms MAY be supported. Likewise, the id-alg-CMS3DESwrap, id-aes198-wrap, and id-aes256wrap MAY be supported.

Implementations of this specification that support AuthenticatedData with ECMQV MUST support the mqvSinglePass-sha256kdf-scheme algorithm. They MUST also support the id-aes128-wrap algorithm. The mqvSinglePass-sha1kdf-scheme, mqvSinglePass-sha224kdf-scheme, mqvSinglePass-sha384kdf-scheme, and mqvSinglePass-sha512kdf-scheme algorithms MAY be supported. Likewise, the id-alg-CMS3DESwrap, id-aes198-wrap, and id-aes256wrap MAY be supported.

6. Updates to Paragraph 7

Old:

The SMIMECapability value to indicate support for the ECDSA signature algorithm is the SEQUENCE with the capabilityID field containing the object identifier ecdsa-with-SHA1 with NULL parameters. The DER encoding is:

30 0b 06 07 2a 86 48 ce 3d 04 01 05 00

New:

The SMIMECapability value to indicate support for the ECDSA signature algorithm is the SEQUENCE with the capabilityID field containing the object identifiers ecdsa-with-SHA* object identifiers (where * is 1, 224, 256, 384, or 512) all with NULL parameters. The DER encodings are:

ecdsa-with-SHA1: 30 0b 06 07 2a 86 48 ce 3d 04 01 05 00

ecdsa-with-SHA224: 30 0c 06 08 2a 86 48 ce 3d 04 03 01 05 00

ecdsa-with-SHA256: 30 0c 06 08 2a 86 48 ce 3d 04 03 02 05 00

ecdsa-with-SHA384: 30 0c 06 08 2a 86 48 ce 3d 04 03 03 05 00

ecdsa-with-SHA512: 30 0c 06 08 2a 86 48 ce 3d 04 03 04 05 00

Old:

The SMIMECapability capabilityID object identifiers for the supported key agreement algorithms in this document are

dhSinglePass-stdDH-sha1kdf-scheme, dhSinglePass-cofactorDH-sha1kdf-scheme, and mqvSinglePass-sha1kdf-scheme. For each of these SMIMECapability SEQUENCEs, the parameters field is present and indicates the supported key-encryption algorithm with the KeyWrapAlgorithm algorithm identifier. The DER encodings that indicate capability of the three key agreement algorithms with CMS Triple-DES key wrap are:

```
30 1c 06 09 2b 81 05 10 86 48 3f 00 02 30 0f 06
0b 2a 86 48 86 f7 0d 01 09 10 03 06 05 00
```

for ephemeral-static ECDH,

```
30 1c 06 09 2b 81 05 10 86 48 3f 00 03 30 0f 06
0b 2a 86 48 86 f7 0d 01 09 10 03 06 05 00
```

for ephemeral-static ECDH with cofactor method, and

```
30 1c 06 09 2b 81 05 10 86 48 3f 00 10 30 0f 06
0b 2a 86 48 86 f7 0d 01 09 10 03 06 05 00
```

for ECMQV.

New:

The SMIMECapability value to indicate support for
a) the standard ECDH key agreement algorithm,
b) the cofactor ECDH key agreement algorithm, or
c) the 1-Pass ECMQV key agreement algorithm
is a SEQUENCE with the capabilityID field containing the object identifier
a) dhSinglePass-stdDH-sha*kdf-scheme,
b) dhSinglePass-cofactorDH-sha*kdf-scheme, or
c) mqvSinglePass-sha*kdf-scheme
respectively (where * is 1, 224, 256, 384, or 512) with the parameters present. The parameters indicate the supported key-encryption algorithm with the KeyWrapAlgorithm algorithm identifier.

Example DER encodings that indicate some capabilities are as follows (KA is key agreement, KDF is key derivation function, and Wrap is key wrap algorithm):

```
KA=ECDH standard KDF=SHA1 Wrap=3DES

  30 1c
        06 09 2b 81 05 10 86 48 3f 00 02
        30 0f
              06 0b 2a 86 48 86 f7 0d 01 09 10 03 06
              05 00

KA=ECDH standard KDF=SHA256 Wrap=AES128

  30 1a
        06 09 2b 81 04 01 0B 01
        30 0f
              06 09 60 83 48 01 65 03 04 01 05
              05 00

KA=ECDH standard KDF=SHA384 Wrap=AES256

  30 1a
        06 06 2b 81 04 01 0B 02
        30 0f
              06 09 60 83 48 01 65 03 04 01 2D
              05 00

KA=ECDH cofactor KDF=SHA1 Wrap=3DES

  30 1c
        06 09 2b 81 05 10 86 48 3f 00 03
        30 0f
              06 0b 2a 86 48 86 f7 0d 01 09 10 03 06
              05 00

KA=ECDH cofactor KDF=SHA256 Wrap=AES128

  30 1a
        06 06 2b 81 04 01 0E 01
        30 0f
              06 09 60 83 48 01 65 03 04 01 05
              05 00

KA=ECDH cofactor KDF=SHA384 Wrap=AES256

  30 1a
        06 06 2b 81 04 01 0E 02
        30 0f
              06 09 60 83 48 01 65 03 04 01 2D
              05 00
```

```
KA=ECMQV 1-Pass KDF=SHA1 Wrap=3DES

   30 1c
         06 09 2b 81 05 10 86 48 3f 00 10
         30 0f
               06 0b 2a 86 48 86 f7 0d 01 09 10 03 06
               05 00

KA=ECMQV 1-Pass KDF=SHA256 Wrap=AES128

  30 1a
        06 06 2b 81 04 01 0F 01
        30 0f
              06 09 60 83 48 01 65 03 04 01 05
              05 00

KA=ECMQV 1-Pass KDF=SHA384 Wrap=AES256

  30 1a
        06 06 2b 81 04 01 0F 02
        30 0f
              06 09 60 83 48 01 65 03 04 01 2D
              05 00
```

7. Updates to Paragraph 8.1

Old:

> The algorithm identifiers used in this document are taken from [X9.62], [SEC1] and [SEC2].
>
> The following object identifier indicates the hash algorithm used in this document:
>
> ```
> sha-1 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
> oiw(14) secsig(3) algorithm(2) 26 }
> ```

New:

> The following object identifier indicates the hash algorithm used in this document:
>
> ```
> id-sha1 OBJECT IDENTIFIER ::= { iso(1) identified-
> organization(3) oiw(14) secsig(3) algorithm(2) 26 }
> ```

```
id-sha224 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
country(16) us(840) organization(1) gov(101) csor(3)
nistalgorithm(4) hashalgs(2) 4 }

id-sha256 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
country(16) us(840) organization(1) gov(101) csor(3)
nistalgorithm(4) hashalgs(2) 1 }

id-sha384 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
country(16) us(840) organization(1) gov(101) csor(3)
nistalgorithm(4) hashalgs(2) 2 }

id-sha512 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2)
country(16) us(840) organization(1) gov(101) csor(3)
nistalgorithm(4) hashalgs(2) 3 }
```

Old:

The following object identifier indicates the digital signature algorithm used in this document:

```
ecdsa-with-SHA1 OBJECT IDENTIFIER ::= { ansi-x9-62
signatures(4) 1 }
```

When the object identifier ecdsa-with-SHA1 is used within an algorithm identifier, the associated parameters field contains NULL.

New:

The following object identifier indicates the digital signature algorithm used in this document:

```
ecdsa-with-SHA1 OBJECT IDENTIFIER ::= { ansi-x9-62
signatures(4) 1 }

ecdsa-with-SHA224 OBJECT IDENTIFIER ::= { ansi-x9-62
signatures(4) ecdsa-with-SHA2(3) 1 }

ecdsa-with-SHA256 OBJECT IDENTIFIER ::= { ansi-x9-62
signatures(4) ecdsa-with-SHA2(3) 2 }

ecdsa-with-SHA384 OBJECT IDENTIFIER ::= { ansi-x9-62
signatures(4) ecdsa-with-SHA2(3) 3 }

ecdsa-with-SHA512 OBJECT IDENTIFIER ::= { ansi-x9-62
signatures(4) ecdsa-with-SHA2(3) 4 }
```

When the object identifiers ecdsa-with-SHA1, ecdsa-with-SHA224, ecdsa-with-SHA256, ecdsa-with-SHA384, or ecdsa-with-SHA512 are used within an algorithm identifier, the associated parameters field contains NULL.

Old:

The following object identifiers indicate the key agreement algorithms used in this document:

```
dhSinglePass-stdDH-sha1kdf-scheme OBJECT IDENTIFIER ::= {
  x9-63-scheme 2}

dhSinglePass-cofactorDH-sha1kdf-scheme OBJECT IDENTIFIER ::= {
  x9-63-scheme 3}

mqvSinglePass-sha1kdf-scheme OBJECT IDENTIFIER ::= {
  x9-63-scheme 16}
```

where

```
x9-63-scheme OBJECT IDENTIFIER ::= { iso(1)
  identified-organization(3) tc68(133) country(16) x9(840)
  x9-63(63) schemes(0) }
```

When the object identifiers are used here within an algorithm identifier, the associated parameters field contains the CMS KeyWrapAlgorithm algorithm identifier.

New:

The following object identifiers indicate the key agreement algorithms used in this document:

```
dhSinglePass-stdDH-sha1kdf-scheme OBJECT IDENTIFIER ::= {
  x9-63-scheme 2 }

dhSinglePass-stdDH-sha224kdf-scheme OBJECT IDENTIFIER ::= {
  secg-scheme 11 0 }

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 }

   dhSinglePass-cofactorDH-sha1kdf-scheme OBJECT IDENTIFIER ::= {
     x9-63-scheme 3 }

   dhSinglePass-cofactorDH-sha224kdf-scheme OBJECT IDENTIFIER ::=
   { secg-scheme 14 0 }

   dhSinglePass-cofactorDH-sha256kdf-scheme OBJECT IDENTIFIER ::=
   { secg-scheme 14 1 }

   dhSinglePass-cofactorDH-sha384kdf-scheme OBJECT IDENTIFIER ::=
   { secg-scheme 14 2 }

   dhSinglePass-cofactorDH-sha512kdf-scheme OBJECT IDENTIFIER ::=
   { secg-scheme 14 3 }

   mqvSinglePass-sha1kdf-scheme OBJECT IDENTIFIER ::= {
     x9-63-scheme 16 }

   mqvSinglePass-sha224kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 15 0 }

   mqvSinglePass-sha256kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 15 1 }

   mqvSinglePass-sha384kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 15 2 }

   mqvSinglePass-sha512kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 15 3 }

where

   x9-63-scheme OBJECT IDENTIFIER ::= {
     iso(1) identified-organization(3) tc68(133) country(16)
     x9(840) x9-63(63) schemes(0) }

and

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

When the object identifiers are used here within an algorithm identifier, the associated parameters field contains the CMS KeyWrapAlgorithm algorithm identifier.

8. Updates to Paragraph 9

Add the following reference:

> [SMIME-SHA2] Turner, S., "Using SHA2 Algorithms with Cryptographic Message Syntax", work-in-progress.

Update the following references:

Old:

> [PKI-ALG] Bassham, L., Housley R. and W. Polk, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 3279, April 2002.
>
> [FIPS-180] FIPS 180-1, "Secure Hash Standard", National Institute of Standards and Technology, April 17, 1995.

New:

> [PKI-ALG] Turner, S., Brown, D., Yiu, K., Housley, R., and W. Polk,"Elliptic Curve Cryptography Subject Public Key Information", work-in-progress.
>
> [FIPS] FIPS 180-2, "Secure Hash Standard", National Institute of Standards and Technology, August 1, 2002.

9. Changes to Security Considerations

Delete the following:

> When 256, 384, and 512 bit hash functions succeed SHA-1 in future revisions of [FIPS], [FIPS-186-2], [X9.62] and [SEC1], then they can similarly succeed SHA-1 in a future revision of this document.

10. Add Annex A: 2004 ASN.1 Module

Add the following section as Annex A ASN.1 Module.

This appendix provides the ASN.1 definitions for the structures described in this specification using ASN.1 as defined in [X.680,X.681].

```
SMIMEECCAlgs-2008
  { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
    smime(16) modules(0) TBD }

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL

IMPORTS

-- From [PKI-ALG]

ALGORITHM, algorithmIdentifier, MessageDigestAlgorithms,
SignatureAlgorithms
ow-sha1, ow-sha224, ow-sha256, ow-sha384, ow-sha512,
sa-ecdsaWithSHA1
  FROM PKIXAlgs-2008
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0) TBD }

-- From [RFC3565]

id-aes128-CBC, id-aes192-CBC, id-aes256-CBC, AES-IV
id-aes128-wrap, id-aes192-wrap, id-aes1256-wrap
  FROM CMSAesRsaesOaep
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) id-mod-cms-aes(19) }

-- From [RFC5084]

id-aes128-CCM, id-aes192-CCM, id-aes256-CCM, CCMParameters
id-aes128-GCM, id-aes192-GCM, id-aes256-GCM, GCMParameters
  FROM CMS-AES-CCM-and-AES-GCM
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) id-mod-cms-aes(32) }
```

```
   -- From [RFC3852]

   OriginatorPublicKey, UserKeyingMaterial
     FROM CryptographicMessageSyntax2004
       { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
         smime(16) modules(0) cms-2004(24) }

   -- From [RFC3370]

   hMAC-SHA1, id-alg-CMS3DESwrap, CBCParameter
     FROM CryptographicMessageSyntaxAlgorithms
       { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
         smime(16) modules(0) cmsalg-2001(16) }

   ;

   -- Constrains the SignedData digestAlgorithms field
   -- Constrains the SignedData SignerInfo digestAlgorithm field
   -- Constrains the AuthenticatedData digestAlgorithm field

   MessageDigestAlgorithms ALGORITHM ::= {
     ow-sha1   |
     ow-sha224 |
     ow-sha256 |
     ow-sha384 |
     ow-sha512,
     ... -- Extensible
   }

   -- Constrains the SignedData SignerInfo signatureAlgorithm field

   SignatureAlgorithms ALGORITHM ::= {
     sa-ecdsaWithSHA1   |
     sa-ecdsaWithSHA224 |
     sa-ecdsaWithSHA256 |
     sa-ecdsaWithSHA384 |
     sa-ecdsaWithSHA512 ,
     ... -- Extensible
   }

   sa-ecdsa-with-SHA224 ALGORITHM ::= {
     OID ecdsa-with-SHA224 PARMS NULL }

   ecdsa-with-SHA224 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) ansi-X9-62(10045) signatures(4)
     ecdsa-with-SHA2(3) 1 }
```

```
   sa-ecdsa-with-SHA256 ALGORITHM ::= {
     OID ecdsa-with-SHA256 PARMS NULL }

   ecdsa-with-SHA256 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840)ansi-X9-62(10045) signatures(4)
     ecdsa-with-SHA2(3) 2 }

   sa-ecdsa-with-SHA384 ALGORITHM ::= {
     OID ecdsa-with-SHA384 PARMS NULL }

   ecdsa-with-SHA384 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) ansi-X9-62(10045) signatures(4)
     ecdsa-with-SHA2(3) 3 }

   sa-ecdsa-with-SHA512 ALGORITHM ::= {
     OID ecdsa-with-SHA512 PARMS NULL }

   ecdsa-with-SHA512 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) ansi-X9-62(10045) signatures(4)
     ecdsa-with-SHA2(3) 4 }

   -- ECDSA Signature Value
   -- Contents of SignatureValue OCTET STRING

   ECDSA-Sig-Value ::= SEQUENCE {
     r  INTEGER,
     s  INTEGER
   }
```

```
   -- Constrains the EnvelopedData RecipientInfo KeyAgreeRecipientInfo
   -- keyEncryption Algorithm field
   -- Constrains the AuthenticatedData RecipientInfo
   -- KeyAgreeRecipientInfo keyEncryption Algorithm field
   -- Constrains the AuthEnvelopedData RecipientInfo
   -- KeyAgreeRecipientInfo keyEncryption Algorithm field

   -- DH variants are not used with AuthenticatedData or
   -- AuthEnvelopedData

   KeyAgreementAlgorithms ALGORITHM ::= {
     kaa-dhSinglePass-stdDH-sha1kdf        |
     kaa-dhSinglePass-stdDH-sha224kdf      |
     kaa-dhSinglePass-stdDH-sha256kdf      |
     kaa-dhSinglePass-stdDH-sha384kdf      |
     kaa-dhSinglePass-stdDH-sha512kdf      |
     kaa-dhSinglePass-cofactorDH-sha1kdf   |
     kaa-dhSinglePass-cofactorDH-sha224kdf |
     kaa-dhSinglePass-cofactorDH-sha256kdf |
     kaa-dhSinglePass-cofactorDH-sha384kdf |
     kaa-dhSinglePass-cofactorDH-sha512kdf |
     kaa-mqvSinglePass-sha1kdf             |
     kaa-mqvSinglePass-sha224kdf           |
     kaa-mqvSinglePass-sha256kdf           |
     kaa-mqvSinglePass-sha384kdf           |
     kaa-mqvSinglePass-sha512kdf,
     ... -- Extensible
   }

   x9-63-scheme OBJECT IDENTIFIER ::= {
     iso(1) identified-organization(3) tc68(133) country(16) x9(840)
     x9-63(63) schemes(0) }

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

   kaa-dhSinglePass-stdDH-sha1kdf ALGORITHM ::= {
    OID dhSinglePass-stdDH-sha1kdf-scheme PARMS KeyWrapAlgorithms }

   dhSinglePass-stdDH-sha1kdf-scheme OBJECT IDENTIFIER ::= {
     x9-63-scheme 2 }

   kaa-dhSinglePass-stdDH-sha224kdf ALGORITHM ::= {
    OID dhSinglePass-stdDH-sha224kdf-scheme PARMS KeyWrapAlgorithms }

   dhSinglePass-stdDH-sha224kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 11 0 }
```

```
   kaa-dhSinglePass-stdDH-sha256kdf ALGORITHM ::= {
    OID dhSinglePass-stdDH-sha256kdf-scheme PARMS KeyWrapAlgorithms }

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

   kaa-dhSinglePass-stdDH-sha384kdf ALGORITHM ::= {
    OID dhSinglePass-stdDH-sha384kdf-scheme PARMS KeyWrapAlgorithms }

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

   kaa-dhSinglePass-stdDH-sha512kdf ALGORITHM ::= {
    OID dhSinglePass-stdDH-sha512kdf-scheme PARMS KeyWrapAlgorithms }

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

   kaa-dhSinglePass-cofactorDH-sha1kdf ALGORITHM ::= {
    OID dhSinglePass-cofactorDH-sha1kdf-scheme PARMS KeyWrapAlgorithms }

   dhSinglePass-cofactorDH-sha1kdf-scheme OBJECT IDENTIFIER ::= {
     x9-63-scheme 3 }

   kaa-dhSinglePass-cofactorDH-sha224kdf ALGORITHM ::= {
    OID dhSinglePass-cofactorDH-sha224kdf-scheme
    PARMS KeyWrapAlgorithms }

   dhSinglePass-cofactorDH-sha224kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 14 0 }

   kaa-dhSinglePass-cofactorDH-sha256kdf ALGORITHM ::= {
    OID dhSinglePass-cofactorDH-sha256kdf-scheme
    PARMS KeyWrapAlgorithms }

   dhSinglePass-cofactorDH-sha256kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 14 1 }

   kaa-dhSinglePass-cofactorDH-sha384kdf ALGORITHM ::= {
    OID dhSinglePass-cofactorDH-sha384kdf-scheme
    PARMS KeyWrapAlgorithms }

   dhSinglePass-cofactorDH-sha384kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 14 2 }
```

```
   kaa-dhSinglePass-cofactorDH-sha512kdf ALGORITHM ::= {
    OID dhSinglePass-cofactorDH-sha512kdf-scheme
    PARMS KeyWrapAlgorithms }

   dhSinglePass-cofactorDH-sha512kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 14 3 }

   kaa-mqvSinglePass-sha1kdf ALGORITHM ::= {
    OID mqvSinglePass-sha1kdf-scheme PARMS KeyWrapAlgorithms }

   mqvSinglePass-sha1kdf-scheme OBJECT IDENTIFIER ::= {
     x9-63-scheme 16 }

   kaa-mqvSinglePass-sha224kdf ALGORITHM ::= {
    OID mqvSinglePass-sha224kdf-scheme PARMS KeyWrapAlgorithms }

   mqvSinglePass-sha224kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 15 0 }

   kaa-mqvSinglePass-sha256kdf ALGORITHM ::= {
    OID mqvSinglePass-sha256kdf-scheme PARMS KeyWrapAlgorithms }

   mqvSinglePass-sha256kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 15 1 }

   kaa-mqvSinglePass-sha384kdf ALGORITHM ::= {
    OID mqvSinglePass-sha384kdf-scheme PARMS KeyWrapAlgorithms }

   mqvSinglePass-sha384kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 15 2 }

   kaa-mqvSinglePass-sha512kdf ALGORITHM ::= {
    OID mqvSinglePass-sha512kdf-scheme PARMS KeyWrapAlgorithms }

   mqvSinglePass-sha512kdf-scheme OBJECT IDENTIFIER ::= {
     secg-scheme 15 3 }

   KeyWrapAlgorithms ALGORITHM ::= {
     kwa-3des   |
     kwa-aes128 |
     kwa-aes192 |
     kwa-aes256,
     ... -- Extensible
   }

   kwa-3des ALGORITHM :: = {
     OID id-alg-CMS3DESwrap PARMS NULL }
```

```
   kwa-aes128 ALGORITHM ::= {
     OID id-aes128-wrap PARMS ABSENT }

   kwa-aes192 ALGORITHM ::= {
     OID id-aes192-wrap PARMS ABSENT }

   kwa-aes256 ALGORITHM ::= {
     OID id-aes256-wrap PARMS ABSENT }

   -- Constrains the EnvelopedData EncryptedContentInfo encryptedContent
   -- field

   ContentEncryptionAlgorithms ALGORITHM ::= {
     cea-des-ede3-cbc |
     cea-aes128-cbc   |
     cea-aes192-cbc   |
     cea-aes256-cbc   |
     cea-aes128-ccm   |
     cea-aes192-ccm   |
     cea-aes256-ccm   |
     cea-aes128-gcm   |
     cea-aes128-gcm   |
     cea-aes128-gcm,
     ... -- Extensible
     }

   cea-des-ede3-cbc ALGORITHM ::= {
     OID des-ede3-cbc PARMS CBCParameter }

   cea-aes128-cbc ALGORITHM ::= {
     OID id-aes128-CBC PARMS AES-IV }

   cea-aes192-cbc ALGORITHM ::= {
     OID id-aes192-CBC PARMS AES-IV }

   cea-aes256-cbc ALGORITHM ::= {
     OID id-aes256-CBC PARMS AES-IV }

   cea-aes128-ccm ALGORITHM ::= {
     OID id-aes128-CCM PARMS CCMParameters }

   cea-aes192-ccm ALGORITHM ::= {
     OID id-aes192-CCM PARMS CCMParameters }

   cea-aes256-ccm ALGORITHM ::= {
     OID id-aes256-CCM PARMS CCMParameters }
```

```
   cea-aes128-gcm ALGORITHM ::= {
     OID id-aes128-GCM PARMS GCMParameters }

   cea-aes192-gcm ALGORITHM ::= {
     OID id-aes192-GCM PARMS GCMParameters }

   cea-aes256-gcm ALGORITHM ::= {
     OID id-aes256-GCM PARMS GCMParameters }

   -- Constrains the AuthenticatedData
   -- MessageAuthenticationCodeAlgorithm field
   -- Constrains the AuthEnvelopedData
   -- MessageAuthenticationCodeAlgorithm field

   MessageAuthenticationCodeAlgorithms ALGORITHM ::= {
     maca-sha1   |
     maca-sha224 |
     maca-sha256 |
     maca-sha384 |
     maca-sha512,
     ... -- Extensible
   }

   maca-sha1 ALGORITHM ::= {
     OID hMAC-SHA1 PARMS NULL }

   maca-sha224 ALGORITHM ::= {
     OID id-hmacWithSHA224 PARMS NULL }

   -- Would love to import the HMAC224-512 OIDS but they're not in a
   -- module (that I could find)

   id-hmacWithSHA224 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) rsadsi(2) 8 }

   maca-sha256 ALGORITHM ::= {
     OID id-hmacWithSHA256 PARMS NULL }

   id-hmacWithSHA256 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) rsadsi(2) 9 }

   maca-sha384 ALGORITHM ::= {
     OID id-hmacWithSHA384 PARMS NULL }

   id-hmacWithSHA384 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) rsadsi(2) 10 }
```

```
   maca-sha512 ALGORITHM ::= {
     OID id-hmacWithSHA512 PARMS NULL }

   id-hmacWithSHA512 OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) rsadsi(113549) rsadsi(2) 11 }

   -- Constraints on KeyAgreeRecipientInfo OriginatorIdentifierOrKey
   -- OriginatorPublicKey algorithm field

   -- PARMS are NULL

   OriginatorPKAlgorithms ALGORITHM ::= {
     opka-ec,
     ... -- Extensible
   }

   opka-ec AGLORITHM ::={
     OID id-ecPublicKey PARMS NULL }

   -- Format for both ephemeral and static public keys

   ECPoint ::= OCTET STRING

   -- Format of KeyAgreeRecipientInfo ukm field when used with
   -- ECDH or ECmqv

   MQVuserKeyingMaterial ::= SEQUENCE {
     ephemeralPublicKey       OriginatorPublicKey,
     addedukm             [0] EXPLICIT UserKeyingMaterial OPTIONAL
   }

   -- Format for ECDH and ECMQV key-encryption keys when using
   -- EnvelopedData or AuthenticatedData

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

```
   SMIME-CAPS ::= CLASS {
     &Type  OPTIONAL,
     &id    OBJECT IDENTIFIER UNIQUE
   }
   WITH SYNTAX {TYPE &Type IDENTIFIED BY &id }

   SMIMECapability ::= SEQUENCE {
     capabilityID  SMIME-CAPS.&id({SMimeCapsSet}),
     parameters    SMIME-CAPS.
                     &Type({SMimeCapsSet}{@capabilityID}) OPTIONAL
   }

   SMimeCapsSet SMIME-CAPS ::= {
     cap-ecdsa-with-SHA1                   |
     cap-ecdsa-with-SHA224                 |
     cap-ecdsa-with-SHA256                 |
     cap-ecdsa-with-SHA384                 |
     cap-ecdsa-with-SHA512                 |
     cap-dhSinglePass-stdDH-sha1kdf        |
     cap-dhSinglePass-stdDH-sha224kdf      |
     cap-dhSinglePass-stdDH-sha256kdf      |
     cap-dhSinglePass-stdDH-sha384kdf      |
     cap-dhSinglePass-stdDH-sha512kdf      |
     cap-dhSinglePass-cofactorDH-sha1kdf   |
     cap-dhSinglePass-cofactorDH-sha224kdf |
     cap-dhSinglePass-cofactorDH-sha256kdf |
     cap-dhSinglePass-cofactorDH-sha384kdf |
     cap-dhSinglePass-cofactorDH-sha512kdf |
     cap-mqvSinglePass-sha1kdf             |
     cap-mqvSinglePass-sha224kdf           |
     cap-mqvSinglePass-sha256kdf           |
     cap-mqvSinglePass-sha384kdf           |
     cap-mqvSinglePass-sha512kdf,
     ... -- Extensible
   }

   cap-ecdsa-with-SHA1 SMIME-CAPS ::= {
     TYPE NULL IDENTIFIED BY ecdsa-with-SHA1 }

   cap-ecdsa-with-SHA224 SMIME-CAPS ::= {
     TYPE NULL IDENTIFIED BY ecdsa-with-SHA224 }

   cap-ecdsa-with-SHA256 SMIME-CAPS ::= {
     TYPE NULL IDENTIFIED BY ecdsa-with-SHA256 }

   cap-ecdsa-with-SHA384 SMIME-CAPS ::= {
     TYPE NULL IDENTIFIED BY ecdsa-with-SHA384 }
```

```
   cap-ecdsa-with-SHA512 SMIME-CAPS ::= {
     TYPE NULL IDENTIFIED BY ecdsa-with-SHA512 }

   cap-dhSinglePass-stdDH-sha1kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms IDENTIFIED BY dhSinglePass-stdDH-sha1kdf }

   cap-dhSinglePass-stdDH-sha224kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms IDENTIFIED BY dhSinglePass-stdDH-sha224kdf }

   cap-dhSinglePass-stdDH-sha256kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms IDENTIFIED BY dhSinglePass-stdDH-sha256kdf }

   cap-dhSinglePass-stdDH-sha384kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms IDENTIFIED BY dhSinglePass-stdDH-sha384kdf }

   cap-dhSinglePass-stdDH-sha512kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms IDENTIFIED BY dhSinglePass-stdDH-sha512kdf }

   cap-dhSinglePass-cofactorDH-sha1kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms
     IDENTIFIED BY dhSinglePass-cofactorDH-sha1kdf }

   cap-dhSinglePass-cofactorDH-sha224kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms
     IDENTIFIED BY dhSinglePass-cofactorDH-sha224kdf }

   cap-dhSinglePass-cofactorDH-sha256kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms
     IDENTIFIED BY dhSinglePass-cofactorDH-sha256kdf }

   cap-dhSinglePass-cofactorDH-sha384kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms
     IDENTIFIED BY dhSinglePass-cofactorDH-sha384kdf }

   cap-dhSinglePass-cofactorDH-sha512kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms
     IDENTIFIED BY dhSinglePass-cofactorDH-sha512kdf }

   cap-mqvSinglePass-sha1kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms IDENTIFIED BY mqvSinglePass-sha1kdf }

   cap-mqvSinglePass-sha224kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms IDENTIFIED BY mqvSinglePass-sha224kdf }

   cap-mqvSinglePass-sha256kdf SMIME-CAPS ::= {
     TYPE KeyWrapAlgorithms IDENTIFIED BY mqvSinglePass-sha256kdf }
```

```
cap-mqvSinglePass-sha384kdf SMIME-CAPS ::= {
  TYPE KeyWrapAlgorithms IDENTIFIED BY mqvSinglePass-sha384kdf }

cap-mqvSinglePass-sha512kdf SMIME-CAPS ::= {
  TYPE KeyWrapAlgorithms IDENTIFIED BY mqvSinglePass-sha512kdf }

END
```

11. Security Considerations

No new security considerations to those already specified in [RFC3278], [SMIME-SHA2], and [PKI-ALG].

12. IANA Considerations

None: All identifiers are already registered. Please remove this section prior to publication as an RFC.

13. References

13.1. Normative References

[MUST] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, BCP 14, March 1997.

[PKI-ALG] Turner, S., Brown, D., Yiu, K., Housley, R., and W. Polk, "Elliptic Curve Cryptography Subject Public Key Information", work-in-progress.

[SMIME-SHA2] Turner, S., "Using SHA2 Algorithms with Cryptographic Message Syntax", work-in-progress.

[RFC3278] Blake-Wilson, S., Brown, D., and P. Lambert, "Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)", RFC 3278, April 2002.

[RFC3370] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, August 2002.

[RFC3565] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, July 2003.

[RFC3852] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 3852, July 2004.

[RFC5084] Housley, R., "Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)", RFC 5084, November 2007.

[X.680] ITU-T Recommendation X.680: Information Technology - Abstract Syntax Notation One, 1997.

[X.681] ITU-T Recommendation X.680: Information Technology - Abstract Syntax Notation One: Information Object Spcification, 1997.

13.2. Informative References

None.

Author's Addresses

Sean Turner

IECA, Inc.
3057 Nutley Street, Suite 106
Fairfax, VA 22031
USA

Email: turners@ieca.com

Daniel R. L. Brown

Certicom Corp
5520 Explorer Drive #400
Mississauga, ON L4W 5L1
CANADA

Email: dbrown@certicom.com

Full Copyright Statement

Copyright (C) The IETF Trust (2008).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).