S/MIME WG
Internet Draft
Intended Status: Informational
Obsoletes: 3278 (once approved)
Expires: December 3, 2008

Sean Turner, IECA
Dan Brown, Certicom
June 3, 2008

Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)
draft-ietf-smime-3278bis-00.txt

Status of this Memo

Copyright Notice

Abstract

This document describes how to use Elliptic Curve Cryptography (ECC) public-key algorithms in the Cryptographic Message Syntax (CMS). The ECC algorithms support the creation of digital signatures and the exchange of keys to encrypt or authenticate content. The definition

of the algorithm processing is based on the ANSI X9.62 standard, developed by the ANSI X9F1 working group, the IEEE 1363 standard, and the SEC 1 standard.

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

The Cryptographic Message Syntax (CMS) is cryptographic algorithm independent. This specification defines a profile for the use of Elliptic Curve Cryptography (ECC) public key algorithms in the CMS. The ECC algorithms are incorporated into the following CMS content types:

- 'SignedData' to support ECC-based digital signature methods (ECDSA) to sign content

- 'EnvelopedData' to support ECC-based public-key agreement methods (ECDH and ECMQV) to generate pairwise key-encryption keys to encrypt content-encryption keys used for content encryption

- 'AuthenticatedData' to support ECC-based public-key agreement methods (ECMQV) to generate pairwise key-encryption keys to encrypt MAC keys used for content authentication and integrity.

Certification of EC public keys is also described to provide public-key distribution in support of the specified techniques.

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 [MUST].

1.2. Changes since RFC 3278

The following summarizes the changes:

- Paragraph 2.1 added sentence indicating SHA is used with EDSA.

- 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 3.2.1 used SHA1 in the KDF with ECMQV. This document expands the set of 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.

- Deleted summary paragraph.

- Updated references.

- Updated security considerations. Security considerations paragraph referring to definitions of SHA-224, SHA-256, SHA-384, and SHA-512 is deleted.

- Added ASN.1 modules.

- Updated acknowledgements section.

2. SignedData using ECC

This section describes how to use ECC algorithms with the CMS SignedData format to sign data.

2.1. SignedData using ECDSA

This section describes how to use the Elliptic Curve Digital Signature Algorithm (ECDSA) with SignedData. ECDSA is specified in [X9.62]. The method is the elliptic curve analog of the Digital Signature Algorithm (DSA) [DSS]. ECDSA is used with the Secure Hash Algorithm (SHA) [SHS].

In an implementation that uses ECDSA with CMS SignedData, the following techniques and formats MUST be used.

2.1.1. Fields of the SignedData

When using ECDSA with SignedData, the fields of SignerInfo are as in [CMS], but with the following restrictions:

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.

signature MUST contain the DER encoding (as an octet string) of a value of the ASN.1 type ECDSA-Sig-Value (see Section 8.2).

When using ECDSA, the SignedData certificates field MAY include the certificate(s) for the EC public key(s) used in the generation of the ECDSA signatures in SignedData. ECC certificates are discussed in Section 6.

2.1.2. Actions of the sending agent

When using ECDSA with SignedData, the sending agent uses the message digest calculation process and signature generation process for SignedData that are specified in [CMS]. To sign data, the sending agent uses the signature method specified in [X9.62, Section 5.3] with the following exceptions:

- In [X9.62, Section 5.3.1], the integer "e" is instead determined by converting the message digest generated according to [CMS, Section 5.4] to an integer using the data conversion method in [X9.62, Section 4.3.2].

The sending agent encodes the resulting signature using the ECDSA-Sig-Value syntax (see Section 8.2) and places it in the SignerInfosignature field.

2.1.3. Actions of the receiving agent

When using ECDSA with SignedData, the receiving agent uses the message digest calculation process and signature verification process for SignedData that are specified in [CMS]. To verify SignedData, the receiving agent uses the signature verification method specified in [X9.62, Section 5.4] with the following exceptions:

- In [X9.62, Section 5.4.1] the integer "e'" is instead determined by converting the message digest generated according to [CMS, Section 5.4] to an integer using the data conversion method in [X9.62, Section 4.3.2].

In order to verify the signature, the receiving agent retrieves the integers r and s from the SignerInfo signature field of the received message.

3. EnvelopedData using ECC Algorithms

This section describes how to use ECC algorithms with the CMS EnvelopedData format.

3.1. EnvelopedData using (ephemeral-static) ECDH

This section describes how to use the ephemeral-static Elliptic Curve Diffie-Hellman (ECDH) key agreement algorithm with EnvelopedData. Ephemeral-static ECDH is specified in [SEC1] and [IEEE1363]. Ephemeral-static ECDH is the the elliptic curve analog of the ephemeral-static Diffie-Hellman key agreement algorithm specified jointly in the documents [CMS, Section 12.3.1.1] and [CMS-DH].

In an implementation that uses ECDH with CMS EnvelopedData with key agreement, the following techniques and formats MUST be used.

3.1.1. Fields of KeyAgreeRecipientInfo

When using ephemeral-static ECDH with EnvelopedData, the fields of KeyAgreeRecipientInfo are as in [CMS], but with the following restrictions:

originator MUST be the alternative originatorKey. The originatorKey algorithm field MUST contain the id-ecPublicKey object identifier (see Section 8.1) with NULL parameters. The originatorKey publicKey field MUST contain the DER-encoding of a value of the ASN.1 type ECPoint (see Section 8.2), which represents the sending agent's ephemeral EC public key.

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.

3.1.2. Actions of the sending agent

When using ephemeral-static ECDH with EnvelopedData, the sending agent first obtains the recipient's EC public key and domain parameters (e.g. from the recipient's certificate). The sending agent then determines an integer "keydatalen", which is the KeyWrapAlgorithm symmetric key-size in bits, and also a bit string "SharedInfo", which is the DER encoding of ECC-CMS-SharedInfo (see Section 8.2). The sending agent then performs the key deployment and the key agreement operation of the Elliptic Curve Diffie-Hellman

Scheme specified in [SEC1, Section 6.1]. As a result the sending agent obtains:

- an ephemeral public key, which is represented as a value of the type ECPoint (see Section 8.2), encapsulated in a bit string and placed in the KeyAgreeRecipientInfo originator field, and

- a shared secret bit string "K", which is used as the pairwise key-encryption key for that recipient, as specified in [CMS].

3.1.3. Actions of the receiving agent

When using ephemeral-static ECDH with EnvelopedData, the receiving agent determines the bit string "SharedInfo", which is the DER encoding of ECC-CMS-SharedInfo (see Section 8.2), and the integer "keydatalen" from the key-size, in bits, of the KeyWrapAlgorithm. The receiving agent retrieves the ephemeral EC public key from the bit string KeyAgreeRecipientInfo originator, with a value of the type ECPoint (see Section 8.2) encapsulated as a bit string. The receiving agent performs the key agreement operation of the Elliptic Curve Diffie-Hellman Scheme specified in [SEC1, Section 6.1]. As a result, the receiving agent obtains a shared secret bit string "K", which is used as the pairwise key-encryption key to unwrap the CEK.

3.2. EnvelopedData using 1-Pass ECMQV

This section describes how to use the 1-Pass elliptic curve MQV (ECMQV) key agreement algorithm with EnvelopedData. ECMQV is specified in [SEC1] and [IEEE1363]. Like the KEA algorithm [CMS-

KEA], 1-Pass ECMQV uses three key pairs: an ephemeral key pair, a static key pair of the sending agent, and a static key pair of the receiving agent. An advantage of using 1-Pass ECMQV is that it can be used with both EnvelopedData and AuthenticatedData.

In an implementation that uses 1-Pass ECMQV with CMS EnvelopedData with key agreement, the following techniques and formats MUST be used.

3.2.1. Fields of KeyAgreeRecipientInfo

When using 1-Pass ECMQV with EnvelopedData, the fields of KeyAgreeRecipientInfo are:

originator identifies the static EC public key of the sender. It SHOULD be one of the alternatives, issuerAndSerialNumber or subjectKeyIdentifier, and point to one of the sending agent's certificates.

ukm MUST be present. The ukm field MUST contain an octet string which is the DER encoding of the type MQVuserKeyingMaterial (see Section 8.2). The MQVuserKeyingMaterial ephemeralPublicKey

algorithm field MUST contain the id-ecPublicKey object identifier (see Section 8.1) with NULL parameters field. The MQVuserKeyingMaterial ephemeralPublicKey publicKey field MUST contain the DER-encoding of the ASN.1 type ECPoint (see Section 8.2) representing sending agent's ephemeral EC public key. The MQVuserKeyingMaterial addedukm field, if present, SHOULD contain an octet string of additional user keying material of the sending agent.

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.

3.2.2. Actions of the sending agent

When using 1-Pass ECMQV with EnvelopedData, the sending agent first obtains the recipient's EC public key and domain parameters, (e.g. from the recipient's certificate) and checks that the domain parameters are the same. The sending agent then determines an integer "keydatalen", which is the KeyWrapAlgorithm symmetric key-size in bits, and also a bit string "SharedInfo", which is the DER

encoding of ECC-CMS-SharedInfo (see Section 8.2). The sending agent then performs the key deployment and key agreement operations of the Elliptic Curve MQV Scheme specified in [SEC1, Section 6.2]. As a result, the sending agent obtains:

- an ephemeral public key, which is represented as a value of type ECPoint (see Section 8.2), encapsulated in a bit string, placed in an MQVuserKeyingMaterial ephemeralPublicKey publicKey field

(see Section 8.2), and

- a shared secret bit string "K", which is used as the pairwise key-encryption key for that recipient, as specified in [CMS].

The ephemeral public key can be re-used with an AuthenticatedData for greater efficiency.

3.2.3. Actions of the receiving agent

When using 1-Pass ECMQV with EnvelopedData, the receiving agent determines the bit string "SharedInfo", which is the DER encoding of ECC-CMS-SharedInfo (see Section 8.2), and the integer "keydatalen" from the key-size, in bits, of the KeyWrapAlgorithm. The receiving agent then retrieves the static and ephemeral EC public keys of the originator, from the originator and ukm fields as described in field and checks that the domain parameters are the same. The receiving agent then performs the key agreement operation of the Elliptic Curve MQV Scheme [SEC1, Section 6.2]. As a result, the receiving agent obtains a shared secret bit string "K" which is used as the pairwise key-encryption key to unwrap the CEK.

4. AuthenticatedData using ECC

This section describes how to use ECC algorithms with the CMS AuthenticatedData format. AuthenticatedData lacks non-repudiation, and so in some instances is preferable to SignedData. (For example, the sending agent might not want the message to be authenticated when forwarded.)

4.1. AuthenticatedData using 1-pass ECMQV

This section describes how to use the 1-Pass elliptic curve MQV (ECMQV) key agreement algorithm with AuthenticatedData. ECMQV is specified in [SEC1]. An advantage of using 1-Pass ECMQV is that it can be used with both EnvelopedData and AuthenticatedData.

4.1.1. Fields of the KeyAgreeRecipientInfo

The AuthenticatedData KeyAgreeRecipientInfo fields are used in the

same manner as the fields for the corresponding EnvelopedData KeyAgreeRecipientInfo fields of Section 3.2.1 of this document.

4.1.2. Actions of the sending agent

The sending agent uses the same actions as for EnvelopedData with 1-Pass ECMQV, as specified in Section 3.2.2 of this document.

The ephemeral public key can be re-used with an EnvelopedData for greater efficiency.

Note: if there are multiple recipients, an attack is possible where one recipient modifies the content without other recipients noticing [BON]. A sending agent who is concerned with such an attack SHOULD use a separate AuthenticatedData for each recipient.

4.1.3. Actions of the receiving agent

The receiving agent uses the same actions as for EnvelopedData with 1-Pass ECMQV, as specified in Section 3.2.3 of this document.

Note: see Note in Section 4.1.2.

5. Recommended Algorithms and Elliptic Curves

Implementations of this specification MUST implement either SignedData with ECDSA or EnvelopedData with ephemeral-static ECDH. Implementations of this specification SHOULD implement both SignedData with ECDSA and EnvelopedData with ephemeral-static ECDH. Implementations MAY implement the other techniques specified, such as AuthenticatedData and 1-Pass ECMQV.

Furthermore, in order to encourage interoperability, implementations SHOULD use the elliptic curve domain parameters specified by ANSI [X9.62], NIST [DSS] and SECG [SEC2]. 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 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-aes192-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-aes192-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-aes192-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-aes192-wrap, and id-aes256wrap MAY be supported.

6. Certificates using ECC

Internet X.509 certificates [PKI] can be used in conjunction with this specification to distribute agents' public keys. The use of ECC algorithms and keys within X.509 certificates is specified in [PKI-ALG].

7. SMIMECapabilities Attribute and ECC

A sending agent MAY announce to receiving agents that it supports one or more of the ECC algorithms in this document by using the SMIMECapabilities signed attribute [MSG, Section 2.5.2].

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

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 17
              06 06 2b 81 04 01 0B 01
              30 0d
                    06 09 60 86 48 01 65 03 04 01 05
                    05 00

      KA=ECDH standard KDF=SHA384 Wrap=AES256

        30 17
              06 06 2b 81 04 01 0B 02
              30 0d
                    06 09 60 86 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 17
              06 06 2b 81 04 01 0E 01
              30 0d
                    06 09 60 86 48 01 65 03 04 01 05
                    05 00

      KA=ECDH cofactor KDF=SHA384 Wrap=AES256

        30 17
              06 06 2b 81 04 01 0E 02
              30 0d
                    06 09 60 86 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 17
              06 06 2b 81 04 01 0F 01
              30 0d
                    06 09 60 86 48 01 65 03 04 01 05
                    05 00

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

        30 17
              06 06 2b 81 04 01 0F 02
              30 0d
                    06 09 60 86 48 01 65 03 04 01 2D
                    05 00
```

8. ASN.1 Syntax

The ASN.1 syntax used in this document is gathered in this section for reference purposes.

8.1. Algorithm Identifiers

The following object identifier indicates the hash algorithm used in this document [SMIME-SHA2]:

```
   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 }
```


The following object identifier is used in this document to indicate an elliptic curve public key:

```
id-ecPublicKey OBJECT IDENTIFIER ::= { ansi-x9-62 keyType(2) 1 }
```

where

```
ansi-x9-62 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
10045 }
```

When the object identifier id-ecPublicKey is used here with an algorithm identifier, the associated parameters contain NULL.

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.

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.

8.2. Other Sytnax

The following additional syntax is used here.

When using ECDSA with SignedData, ECDSA signatures are encoded using the type:

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

ECDSA-Sig-Value is specified in [X9.62]. Within CMS, ECDSA-Sig-Value is DER-encoded and placed within a signature field of SignedData.

When using ECDH and ECMQV with EnvelopedData and AuthenticatedData, ephemeral and static public keys are encoded using the type ECPoint.

```
ECPoint ::= OCTET STRING
```

When using ECMQV with EnvelopedData and AuthenticatedData, the sending agent's ephemeral public key and additional keying material are encoded using the type:

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

The ECPoint syntax in used to represent the ephemeral public key and placed in the ephemeralPublicKey field. The additional user keying material is placed in the addedukm field. Then the MQVuserKeyingMaterial value is DER-encoded and placed within a ukm field of EnvelopedData or AuthenticatedData.

When using ECDH or ECMQV with EnvelopedData or AuthenticatedData, the key-encryption keys are derived by using the type:

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


The fields of ECC-CMS-SharedInfo are as follows:

keyInfo contains the object identifier of the key-encryption algorithm (used to wrap the CEK) and NULL parameters.

entityUInfo optionally contains additional keying material supplied by the sending agent. When used with ECDH and CMS, the entityUInfo field contains the octet string ukm. When used with ECMQV and CMS, the entityUInfo contains the octet string addedukm (encoded in MQVuserKeyingMaterial).

suppPubInfo contains the length of the generated KEK, in bits, represented as a 32 bit number, as in [CMS-DH]. (E.g. for 3DES

it would be 00 00 00 c0.)

Within CMS, ECC-CMS-SharedInfo is DER-encoded and used as input to the key derivation function, as specified in [SEC1, Section 3.6.1].

Note that ECC-CMS-SharedInfo differs from the OtherInfo specified in [CMS-DH]. Here, a counter value is not included in the keyInfo field because the key derivation function specified in [SEC1, Section 3.6.1] ensures that sufficient keying data is provided.

9. Security Considerations

This specification is based on [CMS], [X9.62] and [SEC1] and the appropriate security considerations of those documents apply.

In addition, implementors of AuthenticatedData should be aware of the concerns expressed in [BON] when using AuthenticatedData to send messages to more than one recipient. Also, users of MQV should be aware of the vulnerability in [K].

When implementing EnvelopedData or AuthenticatedData, there are five algorithm related choices that need to be made:

1) What is the public key size?
2) What is the KDF?
3) What is the key wrap algorithm?
4) What is the content encryption algorithm?
5) What is the curve?

Consideration must be given to strength of the security provided by each of these choices. Security is measured in bits, where a strong symmetric cipher with a key of X bits is said to provide X bits of security. It is recommended that the bits of security provided by each are roughly equivalent. The following table provides comparable

minimum bits of security [NISTSP800-57] for the ECDH/ECMQV key sizes, KDFs, key wrapping algorithms, and content encryption algorithms. It also lists curves [PKI-ALG] for the key sizes.

Minimum Bits of Security	ECDH or ECQMV Key Size	Key Derivation Function	Key Wrap Alg.	Content Encryption Alg.	Curves

80	160-223	SHA1	3DES	3DES CBC	sect163k1
		SHA224	AES-128	AES-128 CBC	secp163r2
		SHA256	AES-192	AES-192 CBC	secp192r1
		SHA384	AES-256	AES-256 CBC	
		SHA512			
112	224-255	SHA1	3DES	3DES CBC	secp224r1
		SHA224	AES-128	AES-128 CBC	sect233k1
		SHA256	AES-192	AES-192 CBC	sect233r1
		SHA384	AES-256	AES-256 CBC	
		SHA512			
128	256-383	SHA1	AES-128	AES-128 CBC	secp256r1
		SHA224	AES-192	AES-192 CBC	sect283k1
		SHA256	AES-256	AES-256 CBC	sect283r1
		SHA384			
		SHA512			
192	384-511	SHA224	AES-192	AES-192 CBC	secp384r1
		SHA256	AES-256	AES-256 CBC	sect409k1
		SHA384			sect409r1
		SHA512			
256	512+	SHA256	AES-256	AES-256 CBC	secp521r1
		SHA384			sect571k1
		SHA512			sect571r1

To promote interoperability, the following choices are REOMMENDED:

Minimum	ECDH or	Key	Key	Content	Curve

Bits of Security	ECQMV Key Size	Derivation Function	Wrap Alg.	Encryption Alg.	
80	192	SHA256	3DES	3DES CBC	secp192r1
112	224	SHA256	3DES	3DES CBC	secp224r1
128	256	SHA256	AES-128	AES-128 CBC	secp256r1
192	384	SHA384	AES-256	AES-256 CBC	secp384r1
256	512	SHA512	AES-256	AES-256 CBC	secp521r1

When implementing SignedData, there are three algorithm related choices that need to be made:

1) What is the public key size?
2) What is the hash algorithm?
3) What is the curve?

Consideration must be given to the bits of security provided by each of these choices. Security is measured in bits, where a strong symmetric cipher with a key of X bits is said to provide X bits of security. It is recommended that the bits of security provided by each choice are roughly equivalent. The following table provides comparable minimum bits of security [NISTSP800-57] for the ECDSA key sizes and message digest algorithms. It also lists curves [PKI-ALG] for the key sizes.

Minimum Bits of Security	ECDSA Key Size	Message Digest Algorithm	Curve
80	160-223	SHA1	sect163k1
		SHA224	secp163r2
		SHA256	secp192r1
		SHA384	
		SHA512	
112	224-255	SHA224	secp224r1
		SHA256	sect233k1
		SHA384	sect233r1
		SHA512	
128	256-383	SHA256	secp256r1
		SHA384	sect283k1
		SHA512	sect283r1
192	384-511	SHA384	secp384r1
		SHA512	sect409k1
			sect409r1
256	512+	SHA512	secp521r1
			sect571k1
			sect571r1

To promote interoperability, the following choices are RECOMMENDED:

Minimum Bits of Security	ECDSA Key Size	Message Digest Algorithm	Curve
80	192	SHA256	sect192r1
112	224	SHA256	secp224r1
128	256	SHA256	secp256r1
192	384	SHA384	secp384r1
256	512+	SHA512	secp521r1

10. IANA Considerations

None.

11. References

11.1. Normative

[CMS] Housley, R., "Cryptographic Message Syntax", RFC 3852, July.

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

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

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

[CMS-DH] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, June 1999.

[IEEE1363] IEEE P1363, "Standard Specifications for Public Key Cryptography", Institute of Electrical and Electronics Engineers, 2000.

[DSS] FIPS 186-2, "Digital Signature Standard", National Institute of Standards and Technology, January 2000.

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

[MSG] Ramsdell, B., and S. Turner, "S/MIME Version 3.2 Message Specification", work-in-progress.

[PKI] 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.

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

[SEC1] SECG, "Elliptic Curve Cryptography", Standards for Efficient Cryptography Group, 2000. Available from www.secg.org/collateral/sec1.pdf.

[SEC2] SECG, "Recommended Elliptic Curve Domain Parameters", Standards for Efficient Cryptography Group, 2000. Available from www.secg.org/collateral/sec2.pdf.

[SHS] National Institute of Standards and Technology (NIST), FIPS Publication 180-2: Secure Hash Standard, August 2002.

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

[X9.62] ANSI X9.62-2005, "Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", American National Standards Institute, 2005.

[X.208] CCITT Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1), 1988.

[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 Specification, 1997.

[X.682] ITU-T Recommendation X.682: Information Technology - Abstract Syntax Notation One: Constraint Specification, 2002.

[X.683] ITU-T Recommendation X.683: Information Technology - Abstract Syntax Notation One: Parameterization of ASN.1 Specifications, 2002.

11.2. Informative

[BON] D. Boneh, "The Security of Multicast MAC", Presentation at Selected Areas of Cryptography 2000, Center for Applied Cryptographic Research, University of Waterloo, 2000. Paper version available from http://crypto.stanford.edu/~dabo/papers/mmac.ps

[CMS-KEA] Pawling, J., "CMS KEA and SKIPJACK Conventions", RFC 2876, July 2000.

[K] B. Kaliski, "MQV Vulnerability", Posting to ANSI X9F1 and IEEE P1363 newsgroups, 1998.

Annex A ASN.1 Modules

Appendix A.1 provides the normative ASN.1 definitions for the structures described in this specification using ASN.1 as defined in [X.208].

Appendix A.2 provides an informative ASN.1 definitions for the structures described in this specification using ASN.1 as defined in [X.680], [X.681], [X.682], [X.683]. This appendix contains the same information as Appendix A.1 in a more recent (and precise) ASN.1 notation, however Appendix A.1 takes precedence in case of conflict.

Annex A.1 1988 ASN.1 Module

Annex A.2 2004 ASN.1 Module

```
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 [CMS-AES]

   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 [CMS-AESCG]

   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 [CMS]

   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 [CMS-ALG]

   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-aes192-gcm   |
  cea-aes256-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)
     digestAlgorithm(2) 8 }

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

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

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

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

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

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

Acknowledgements

The methods described in this document are based on work done by the ANSI X9F1 working group. The authors wish to extend their thanks to ANSI X9F1 for their assistance. The authors also wish to thank Peter de Rooij for his patient assistance. The technical comments of Francois Rousseau were valuable contributions.

Many thanks go out to the other authors of RFC 3278: Simon Blake-Wilson, Paul Lambert, and Dan Brown. Without the initial version of RFC3278 this version wouldn't exist.

The authors also wish to thank Alfred Hines, Jim Schaad, and Russ Housley for their valuable input.

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

Intellectual Property

Acknowledgment

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