Internet Engineering Task Force (IETF)
Request for Comments: 5753
Obsoletes: 3278
Category: Informational
ISSN: 2070-1721

S. Turner
IECA
D. Brown
Certicom

January 2010

Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)

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 NIST FIPS 186-3 for digital signature, NIST SP800-56A and SEC1 for key agreement, RFC 3370 and RFC 3565 for key wrap and content encryption, NIST FIPS 180-3 for message digest, SEC1 for key derivation, and RFC 2104 and RFC 4231 for message authentication code standards. This document obsoletes RFC 3278.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5753.

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

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

The document will obsolete [CMS-ECC]. The technical changes performed since RFC 3278 are detailed in Appendix B.

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].

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 [FIPS186-3]. The method is the elliptic curve analog of the Digital Signature Algorithm (DSA) [FIPS186-3]. ECDSA is used with the Secure Hash Algorithm (SHA) [FIPS180-3].

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 7.1.1), which MUST be one of the following: id-sha1, id-sha224, id-sha256, id-sha384, or id-sha512.

- signatureAlgorithm contains the signature algorithm identifier (see Section 7.1.3): ecdsa-with-SHA1, ecdsa-with-SHA224, ecdsa-with-SHA256, ecdsa-with-SHA384, or ecdsa-with-SHA512. The hash algorithm identified in the name of the signature algorithm MUST be the same as the digestAlgorithm (e.g., digestAlgorithm is id-sha256 therefore signatureAlgorithm is ecdsa-with-SHA256).

- signature MUST contain the DER encoding (as an octet string) of a value of the ASN.1 type ECDSA-Sig-Value (see Section 7.2).

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 5.

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 [FIPS186-3].

The sending agent encodes the resulting signature using the ECDSA-Sig-Value syntax (see Section 7.2) and places it in the SignerInfo signature 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 [FIPS186-3].

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.

This document does not specify the static-static ECDH, method C(0,2, ECC CDH) from [SP800-56A]. Static-static ECDH is analogous to static-static DH, which is specified in [CMS-ALG]. Ephemeral-static ECDH and 1-Pass ECMQV were specified because they provide better security due to the originator's ephemeral contribution to the key agreement scheme.

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. This algorithm has two variations:

- 'Standard' ECDH, described as the 'Elliptic Curve Diffie-Hellman Scheme' with the 'Elliptic Curve Diffie-Hellman Primitive' in [SEC1], and

- 'Co-factor' ECDH, described as the 'One-Pass Diffie-Hellman scheme' (method C(1, 1, ECC CDH)) in [SP800-56A].

Both variations of ephemeral-static ECDH are elliptic curve analogs of the ephemeral-static Diffie-Hellman key agreement algorithm specified jointly in the documents [CMS-ALG] and [CMS-DH].

If an implementation uses ECDH with CMS EnvelopedData, then the following techniques and formats MUST be used.

The fields of EnvelopedData are as in [CMS]; as ECDH is a key agreement algorithm, the RecipientInfo kari choice is used.

3.1.1. Fields of KeyAgreeRecipientInfo

When using ephemeral-static ECDH with EnvelopedData, the fields of KeyAgreeRecipientInfo are as follows:

- version MUST be 3.

- originator MUST be the alternative originatorKey. The originatorKey algorithm field MUST contain the id-ecPublicKey object identifier (see Section 7.1.2). The parameters associated with id-ecPublicKey MUST be absent, ECParameters, or NULL. The parameters associated with id-ecPublicKey SHOULD be absent or ECParameters, and NULL is allowed to support legacy implementations. The previous version of this document required NULL to be present. If the parameters are ECParameters, then they MUST be namedCurve. The originatorKey publicKey field MUST contain the DER encoding of the value of the ASN.1 type ECPoint (see Section 7.2), which represents the sending agent's ephemeral EC public key. The ECPoint in uncompressed form MUST be supported.

- ukm MAY be present or absent. However, message originators SHOULD include the ukm. As specified in RFC 3852 [CMS], implementations MUST support ukm message recipient processing, so interoperability is not a concern if the ukm is present or absent. The ukm is placed in the entityUInfo field of the ECC-CMS-SharedInfo structure. When present, the ukm is used to ensure that a different key-encryption key is generated, even when the ephemeral private key is improperly used more than once, by using the ECC-CMS-SharedInfo as an input to the key derivation function (see Section 7.2).

- keyEncryptionAlgorithm MUST contain the object identifier of the key-encryption algorithm, which in this case is a key agreement algorithm (see Section 7.1.4). 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) and any associated parameters (see Section 7.1.5). Algorithm requirements are found in Section 8.

- recipientEncryptedKeys contains an identifier and an encrypted key for each recipient. 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 ECDH public key.

RecipientEncryptedKey EncryptedKey MUST contain the content-encryption key encrypted with the ephemeral-static, ECDH-generated pairwise key-encryption key using the algorithm specified by the KeyWrapAlgorithm.

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 performs one of the two ECDH variations mentioned above:

- If the value of keyEncryptionAlgorithm indicates the use of 'standard' Diffie-Hellman, then the sending agent performs the 'Elliptic Curve Diffie-Hellman Scheme' with the 'Elliptic Curve Diffie-Hellman Primitive' in [SEC1].

- If the value of keyEncryptionAlgorithm indicates the use of 'co-factor' Diffie-Hellman, then the sending agent performs the 'One-Pass Diffie-Hellman scheme' (method C(1, 1, ECC CDH)) in [SP800-56A].

In both of these cases, the sending agent uses the KDF defined in Section 3.6.1 of [SEC1] with the hash algorithm identified by the value of keyEncryptionAlgorithm. As a result, the sending agent obtains:

- an ephemeral public key, which is represented as a value of the type ECPoint (see Section 7.2), encapsulated in a bit string and placed in the KeyAgreeRecipientInfo originator originatorKey publicKey field, and

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

In a single message, if there are multiple layers for a recipient, then the ephemeral public key can be reused by the originator for that recipient in each of the different layers.

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 7.2), 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 7.2) encapsulated as a bit string, and if

present, originally supplied additional user key material from the ukm field. The receiving agent then performs one of the two ECDH variations mentioned above:

- If the value of keyEncryptionAlgorithm indicates the use of 'standard' Diffie-Hellman, then the receiving agent performs the 'Elliptic Curve Diffie-Hellman Scheme' with the 'Elliptic Curve Diffie-Hellman Primitive' in [SEC1].

- If the value of keyEncryptionAlgorithm indicates the use of 'co-factor' Diffie-Hellman, then the receiving agent performs the 'One-Pass Diffie-Hellman scheme' (method C(1, 1, ECC CDH)) in [SP800-56A].

In both of these cases, the receiving agent uses the KDF defined in Section 3.6.1 of [SEC1] with the hash algorithm identified by the value of keyEncryptionAlgorithm. 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 Menezes-Qu-Vanstone (ECMQV) key agreement algorithm with EnvelopedData, method C(1, 2, ECC MQV) from [SP800-56A]. 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. Using an algorithm with the sender static key pair allows for knowledge of the message creator; this means that authentication can, in some circumstances, be obtained for AuthEnvelopedData and AuthenticatedData. This means that 1-Pass ECMQV can be a common algorithm for EnvelopedData, AuthenticatedData, and AuthEnvelopedData, while ECDH can only be used in EnvelopedData.

If an implementation uses 1-Pass ECMQV with CMS EnvelopedData, then the following techniques and formats MUST be used.

The fields of EnvelopedData are as in [CMS]; as 1-Pass ECMQV is a key agreement algorithm, the RecipientInfo kari choice is used. When using 1-Pass ECMQV, the EnvelopedData originatorInfo field MAY include the certificate(s) for the EC public key(s) used in the formation of the pairwise key. ECC certificates are discussed in Section 5.

3.2.1. Fields of KeyAgreeRecipientInfo

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

- version MUST be 3.

- 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 is an octet string that MUST contain the DER encoding of the type MQVuserKeyingMaterial (see Section 7.2). The MQVuserKeyingMaterial ephemeralPublicKey algorithm field MUST contain the id-ecPublicKey object identifier (see Section 7.1.2). The parameters associated with id-ecPublicKey MUST be absent, ECParameters, or NULL. The parameters associated with id-ecPublicKey SHOULD be absent or ECParameters, as NULL is allowed to support legacy implementations. The previous version of this document required NULL to be present. If the parameters are ECParameters, then they MUST be namedCurve. The MQVuserKeyingMaterial ephemeralPublicKey publicKey field MUST contain the DER encoding of the ASN.1 type ECPoint (see Section 7.2) representing the sending agent's ephemeral EC public key. The MQVuserKeyingMaterial addedukm field, if present, contains additional user keying material from the sending agent.

- keyEncryptionAlgorithm MUST contain the object identifier of the key-encryption algorithm, which in this case is a key agreement algorithm (see Section 7.1.4). The parameters field contains KeyWrapAlgorithm. The KeyWrapAlgorithm indicates the symmetric encryption algorithm used to encrypt the CEK with the KEK generated using the 1-Pass ECMQV algorithm and any associated parameters (see Section 7.1.5). Algorithm requirements are found in Section 8.

- recipientEncryptedKeys contains an identifier and an encrypted key for each recipient. 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 ECMQV public key. RecipientEncryptedKey EncryptedKey MUST contain the content-encryption key encrypted with the 1-Pass ECMQV-generated pairwise key-encryption key using the algorithm specified by the KeyWrapAlgorithm.

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 as the sender's domain parameters. 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 7.2). The sending agent then performs the key deployment and key agreement operations of the Elliptic Curve MQV Scheme specified in [SP800-56A], but uses the KDF defined in Section 3.6.1 of [SEC1]. As a result, the sending agent obtains:

- an ephemeral public key, which is represented as a value of type ECPoint (see Section 7.2), encapsulated in a bit string, placed in an MQVuserKeyingMaterial ephemeralPublicKey publicKey field (see Section 7.2), and

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

In a single message, if there are multiple layers for a recipient, then the ephemeral public key can be reused by the originator for that recipient in each of the different layers.

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 7.2), 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 Section 3.2.1, and its static EC public key identified in the rid field and checks that the originator's domain parameters are the same as the recipient's domain parameters. The receiving agent then performs the key agreement operation of the Elliptic Curve MQV Scheme [SP800-56A], but uses the KDF defined in Section 3.6.1 of [SEC1]. 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 and AuthEnvelopedData 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.)

This section also describes how to use ECC algorithms with the CMS AuthEnvelopedData format [CMS-AUTHENV]. AuthEnvelopedData supports authentication and encryption, and in some instances is preferable to signing and then encrypting data.

For both AuthenticatedData and AuthEnvelopedData, data origin authentication with 1-Pass ECMQV can only be provided when there is one and only one recipient. When 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 or AuthEnvelopedData for each recipient.

Using an algorithm with the sender static key pair allows for knowledge of the message creator; this means that authentication can, in some circumstances, be obtained for AuthEnvelopedData and AuthenticatedData. This means that 1-Pass ECMQV can be a common algorithm for EnvelopedData, AuthenticatedData, and AuthEnvelopedData while ECDH can only be used in EnvelopedData.

4.1. AuthenticatedData Using 1-Pass ECMQV

This section describes how to use the 1-Pass ECMQV key agreement algorithm with AuthenticatedData. ECMQV is method C(1, 2, ECC MQV) from [SP800-56A].

When using ECMQV with AuthenticatedData, the fields of AuthenticatedData are as in [CMS], but with the following restrictions:

- macAlgorithm MUST contain the algorithm identifier of the message authentication code (MAC) algorithm (see Section 7.1.7), which MUST be one of the following: hmac-SHA1, id-hmacWITHSHA224, id-hmacWITHSHA256, id-hmacWITHSHA384, or id-hmacWITHSHA512.

- digestAlgorithm MUST contain the algorithm identifier of the hash algorithm (see Section 7.1.1), which MUST be one of the following: id-sha1, id-sha224, id-sha256, id-sha384, or id-sha512.

As 1-Pass ECMQV is a key agreement algorithm, the RecipientInfo kari choice is used in the AuthenticatedData. When using 1-Pass ECMQV, the AuthenticatedData originatorInfo field MAY include the certificate(s) for the EC public key(s) used in the formation of the pairwise key. ECC certificates are discussed in Section 5.

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.

In a single message, if there are multiple layers for a recipient, then the ephemeral public key can be reused by the originator for that recipient in each of the different layers.

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.

4.2. AuthEnvelopedData Using 1-Pass ECMQV

This section describes how to use the 1-Pass ECMQV key agreement algorithm with AuthEnvelopedData. ECMQV is method C(1, 2, ECC MQV) from [SP800-56A].

When using ECMQV with AuthEnvelopedData, the fields of AuthEnvelopedData are as in [CMS-AUTHENV].

As 1-Pass ECMQV is a key agreement algorithm, the RecipientInfo kari choice is used. When using 1-Pass ECMQV, the AuthEnvelopedData originatorInfo field MAY include the certificate(s) for the EC public key used in the formation of the pairwise key. ECC certificates are discussed in Section 5.

4.2.1. Fields of the KeyAgreeRecipientInfo

The AuthEnvelopedData 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.

4.2.2. 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.

In a single message, if there are multiple layers for a recipient, then the ephemeral public key can be reused by the originator for that recipient in each of the different layers.

4.2.3. 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.

5. 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].

6. SMIMECapabilities Attribute and ECC

A sending agent MAY announce to receiving agents that it supports one or more of the ECC algorithms specified in this document by using the SMIMECapabilities signed attribute [MSG] in either a signed message or a certificate [CERTCAP].

The SMIMECapabilities attribute value indicates support for one of the ECDSA signature algorithms in a SEQUENCE with the capabilityID field containing the object identifier ecdsa-with-SHA1 with NULL parameters and ecdsa-with-SHA* (where * is 224, 256, 384, or 512) with absent 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 0a 06 08 2a 86 48 ce 3d 04 03 01

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

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

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

NOTE: The SMIMECapabilities attribute indicates that parameters for ECDSA with SHA-1 are NULL; however, the parameters are absent when used to generate a digital signature.

The SMIMECapabilities attribute value indicates 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 and

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.

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

KA=ECDH standard KDF=SHA-1 Wrap=Triple-DES

```
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=SHA-224 Wrap=Triple-DES

```
30 17 06 06 2b 81 04 01 0B 00 30 0d 06 0b 2a 86 48 86 f7 0d 01
09 10 03 06
```

KA=ECDH standard KDF=SHA-256 Wrap=Triple-DES

```
30 17 06 06 2b 81 04 01 0B 01 30 0d 06 0b 2a 86 48 86 f7 0d 01
09 10 03 06
```

KA=ECDH standard KDF=SHA-384 Wrap=Triple-DES

```
30 17 06 06 2b 81 04 01 0B 02 30 0d 06 0b 2a 86 48 86 f7 0d 01
09 10 03 06
```

KA=ECDH standard KDF=SHA-512 Wrap=Triple-DES

```
30 17 06 06 2b 81 04 01 0B 03 30 0d 06 0b 2a 86 48 86 f7 0d 01
09 10 03 06
```

KA=ECDH standard KDF=SHA-1 Wrap=AES-128

```
30 18 06 09 2b 81 05 10 86 48 3f 00 02 30 0b 06 09 60 86 48 01
65 03 04 01 05
```

KA=ECDH standard KDF=SHA-224 Wrap=AES-128

```
30 15 06 06 2b 81 04 01 0B 00 30 0b 06 09 60 86 48 01 65 03 04
01 05
```

KA=ECDH standard KDF=SHA-256 Wrap=AES-128

```
30 15 06 06 2b 81 04 01 0B 01 30 0b 06 09 60 86 48 01 65 03 04
01 05
```

KA=ECDH standard KDF=SHA-384 Wrap=AES-128

```
30 15 06 06 2b 81 04 01 0B 02 30 0b 06 09 60 86 48 01 65 03 04
01 05
```

KA=ECDH standard KDF=SHA-512 Wrap=AES-128

```
30 15 06 06 2b 81 04 01 0B 03 30 0b 06 09 60 86 48 01 65 03 04
01 05
```

KA=ECDH standard KDF=SHA-1 Wrap=AES-192

```
30 18 06 09 2b 81 05 10 86 48 3f 00 02 30 0b 06 09 60 86 48 01
65 03 04 01 19
```

KA=ECDH standard KDF=SHA-224 Wrap=AES-192

```
30 15 06 06 2b 81 04 01 0B 00 30 0b 06 09 60 86 48 01 65 03 04
01 19
```

KA=ECDH standard KDF=SHA-256 Wrap=AES-192

```
30 15 06 06 2b 81 04 01 0B 01 30 0b 06 09 60 86 48 01 65 03 04
01 19
```

KA=ECDH standard KDF=SHA-384 Wrap=AES-192

```
30 15 06 06 2b 81 04 01 0B 02 30 0b 06 09 60 86 48 01 65 03 04
01 19
```

KA=ECDH standard KDF=SHA-512 Wrap=AES-192

```
30 15 06 06 2b 81 04 01 0B 03 30 0b 06 09 60 86 48 01 65 03 04
01 19
```

KA=ECDH standard KDF=SHA-1 Wrap=AES-256

```
30 18 06 09 2b 81 05 10 86 48 3f 00 02 30 0b 06 09 60 86 48 01
65 03 04 01 2D
```

KA=ECDH standard KDF=SHA-224 Wrap=AES-256

```
30 15 06 06 2b 81 04 01 0B 00 30 0b 06 09 60 86 48 01 65 03 04
01 2D
```

KA=ECDH standard KDF=SHA-256 Wrap=AES-256

```
30 15 06 06 2b 81 04 01 0B 01 30 0b 06 09 60 86 48 01 65 03 04
01 2D
```

KA=ECDH standard KDF=SHA-384 Wrap=AES-256

```
30 15 06 06 2b 81 04 01 0B 02 30 0b 06 09 60 86 48 01 65 03 04
01 2D 05 00
```

KA=ECDH standard KDF=SHA-512 Wrap=AES-256

```
30 15 06 06 2b 81 04 01 0B 03 30 0b 06 09 60 86 48 01 65 03 04
01 2D
```

KA=ECDH cofactor KDF=SHA-1 Wrap=Triple-DES

```
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=SHA-224 Wrap=Triple-DES

```
30 17 06 06 2b 81 04 01 0E 00 30 0d 06 0b 2a 86 48 86 f7 0d 01
09 10 03 06
```

KA=ECDH cofactor KDF=SHA-256 Wrap=Triple-DES

```
30 17 06 06 2b 81 04 01 0E 01 30 0d 06 0b 2a 86 48 86 f7 0d 01
09 10 03 06
```

KA=ECDH cofactor KDF=SHA-384 Wrap=Triple-DES

```
30 17 06 06 2b 81 04 01 0E 02 30 0d 06 0b 2a 86 48 86 f7 0d 01
09 10 03 06
```

KA=ECDH cofactor KDF=SHA-512 Wrap=Triple-DES

```
30 17 06 06 2b 81 04 01 0E 03 30 0d 06 0b 2a 86 48 86 f7 0d 01
09 10 03 06
```

KA=ECDH cofactor KDF=SHA-1 Wrap=AES-128

```
30 18 06 09 2b 81 05 10 86 48 3f 00 03 30 0b 06 09 60 86 48 01
65 03 04 01 05
```

KA=ECDH cofactor KDF=SHA-224 Wrap=AES-128

```
30 15 06 06 2b 81 04 01 0E 00 30 0b 06 09 60 86 48 01 65 03 04
01 05
```

KA=ECDH cofactor KDF=SHA-256 Wrap=AES-128

```
30 15 06 06 2b 81 04 01 0E 01 30 0b 06 09 60 86 48 01 65 03 04
01 05
```

KA=ECDH cofactor KDF=SHA-384 Wrap=AES-128

```
30 15 06 06 2b 81 04 01 0E 02 30 0b 06 09 60 86 48 01 65 03 04
01 05
```

KA=ECDH cofactor KDF=SHA-512 Wrap=AES-128

```
30 17 06 06 2b 81 04 01 0E 03 30 0b 06 09 60 86 48 01 65 03 04
01 05
```

KA=ECDH cofactor KDF=SHA-1 Wrap=AES-192

```
30 18 06 09 2b 81 05 10 86 48 3f 00 03 30 0b 06 09 60 86 48 01
65 03 04 01 19
```

KA=ECDH cofactor KDF=SHA-224 Wrap=AES-192

```
30 15 06 06 2b 81 04 01 0E 00 30 0b 06 09 60 86 48 01 65 03 04
01 19
```

KA=ECDH cofactor KDF=SHA-256 Wrap=AES-192

```
30 15 06 06 2b 81 04 01 0E 01 30 0b 06 09 60 86 48 01 65 03 04
01 19
```

KA=ECDH cofactor KDF=SHA-384 Wrap=AES-192

```
30 15 06 06 2b 81 04 01 0E 02 30 0b 06 09 60 86 48 01 65 03 04
01 19
```

KA=ECDH cofactor KDF=SHA-512 Wrap=AES-192

```
30 15 06 06 2b 81 04 01 0E 03 30 0b 06 09 60 86 48 01 65 03 04
01 19
```

KA=ECDH cofactor KDF=SHA-1 Wrap=AES-256

```
30 15 06 09 2b 81 05 10 86 48 3f 00 03 30 0b 06 09 60 86 48 01
65 03 04 01 2D
```

KA=ECDH cofactor KDF=SHA-224 Wrap=AES-256

```
30 15 06 06 2b 81 04 01 0E 00 30 0b 06 09 60 86 48 01 65 03 04
01 2D
```

KA=ECDH cofactor KDF=SHA-256 Wrap=AES-256

```
30 15 06 06 2b 81 04 01 0E 01 30 0b 06 09 60 86 48 01 65 03 04
01 2D
```

KA=ECDH cofactor KDF=SHA-384 Wrap=AES-256

```
30 15 06 06 2b 81 04 01 0E 02 30 0b 06 09 60 86 48 01 65 03 04
01 2D
```

KA=ECDH cofactor KDF=SHA-512 Wrap=AES-256

```
30 15 06 06 2b 81 04 01 0E 03 30 0b 06 09 60 86 48 01 65 03 04
01 2D
```

KA=ECMQV 1-Pass KDF=SHA-1 Wrap=Triple-DES

```
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=SHA-224 Wrap=Triple-DES

```
30 17 06 06 2b 81 04 01 0F 00 30 0d 06 0b 2a 86 48 86 f7 0d 01
09 10 03 06
```

KA=ECMQV 1-Pass KDF=SHA-256 Wrap=Triple-DES

```
30 17 06 06 2b 81 04 01 0F 01 30 0d 06 0b 2a 86 48 86 f7 0d 01
09 10 03 06
```

KA=ECMQV 1-Pass KDF=SHA-384 Wrap=Triple-DES

```
30 17 06 06 2b 81 04 01 0F 02 30 0d 06 0b 2a 86 48 86 f7 0d 01
09 10 03 06
```

KA=ECMQV 1-Pass KDF=SHA-512 Wrap=Triple-DES

```
30 17 06 06 2b 81 04 01 0F 03 30 0d 06 0b 2a 86 48 86 f7 0d 01
09 10 03 06
```

KA=ECMQV 1-Pass KDF=SHA-1 Wrap=AES-128

```
30 18 06 09 2b 81 05 10 86 48 3f 00 10 30 0b 06 09 60 86 48 01
65 03 04 01 05
```

KA=ECMQV 1-Pass KDF=SHA-224 Wrap=AES-128

```
30 15 06 06 2b 81 04 01 0F 00 30 0b 06 09 60 86 48 01 65 03 04
01 05
```

KA=ECMQV 1-Pass KDF=SHA-256 Wrap=AES-128

```
30 15 06 06 2b 81 04 01 0F 01 30 0b 06 09 60 86 48 01 65 03 04
01 05
```

KA=ECMQV 1-Pass KDF=SHA-384 Wrap=AES-128

```
30 15 06 06 2b 81 04 01 0F 02 30 0b 06 09 60 86 48 01 65 03 04
01 05
```

KA=ECMQV 1-Pass KDF=SHA-512 Wrap=AES-128

```
30 15 06 06 2b 81 04 01 0F 03 30 0b 06 09 60 86 48 01 65 03 04
01 05
```

KA=ECMQV 1-Pass KDF=SHA-1 Wrap=AES-192

```
30 18 06 09 2b 81 05 10 86 48 3f 00 10 30 0b 06 09 60 86 48 01
65 03 04 01 19
```

KA=ECMQV 1-Pass KDF=SHA-224 Wrap=AES-192

```
30 15 06 06 2b 81 04 01 0F 00 30 0b 06 09 60 86 48 01 65 03 04
01 19
```

KA=ECMQV 1-Pass KDF=SHA-256 Wrap=AES-192

```
30 15 06 06 2b 81 04 01 0F 01 30 0b 06 09 60 86 48 01 65 03 04
01 19
```

KA=ECMQV 1-Pass KDF=SHA-384 Wrap=AES-192

```
30 15 06 06 2b 81 04 01 0F 02 30 0b 06 09 60 86 48 01 65 03 04
01 19
```

KA=ECMQV 1-Pass KDF=SHA-512 Wrap=AES-192

```
30 15 06 06 2b 81 04 01 0F 03 30 0b 06 09 60 86 48 01 65 03 04
01 19
```

KA=ECMQV 1-Pass KDF=SHA-1 Wrap=AES-256

```
30 18 06 09 2b 81 05 10 86 48 3f 00 10 30 0b 06 09 60 86 48 01
65 03 04 01 2D
```

KA=ECMQV 1-Pass KDF=SHA-224 Wrap=AES-256

```
30 15 06 06 2b 81 04 01 0F 00 30 0b 06 09 60 86 48 01 65 03 04
01 2D
```

KA=ECMQV 1-Pass KDF=SHA-256 Wrap=AES-256

```
30 15 06 06 2b 81 04 01 0F 01 30 0b 06 09 60 86 48 01 65 03 04
01 2D
```

KA=ECMQV 1-Pass KDF=SHA-384 Wrap=AES-256

```
30 15 06 06 2b 81 04 01 0F 02 30 0b 06 09 60 86 48 01 65 03 04
01 2D
```

KA=ECMQV 1-Pass KDF=SHA-512 Wrap=AES-256

30 15 06 06 2b 81 04 01 0F 03 30 0b 06 09 60 86 48 01 65 03 04 01 2D

NOTE: The S/MIME Capabilities for the supported AES content-encryption key sizes are defined in [CMS-AES].

NOTE: The S/MIME Capabilities for the supported MAC algorithms are defined in [CMS-ASN].

7. ASN.1 Syntax

The ASN.1 syntax [X.680], [X.681], [X.682], [X.683] used in this document is gathered in this section for reference purposes.

7.1. Algorithm Identifiers

This section provides the object identifiers for the algorithms used in this document along with any associated parameters.

7.1.1. Digest Algorithms

Digest algorithm object identifiers are used in the SignedData digestAlgorithms and digestAlgorithm fields and the AuthenticatedData digestAlgorithm field. The digest algorithms used in this document are SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. The object identifiers and parameters associated with these algorithms are found in [CMS-ALG] and [CMS-SHA2].

7.1.2. Originator Public Key

The KeyAgreeRecipientInfo originator field uses the following object identifier 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 MUST be either absent or ECParameters. Implementations MUST accept id-ecPublicKey with absent and ECParameters parameters. If ECParameters is present, its

value MUST match the recipient's ECParameters. Implementations SHOULD generate absent parameters for the id-ecPublicKey object identifier in the KeyAgreeRecipientInfo originator field.

[CMS-ECC] indicated the parameters were NULL. Support for this legacy form is OPTIONAL.

7.1.3. Signature Algorithms

Signature algorithm identifiers are used in the SignedData signatureAlgorithm and signature fields. The signature algorithms used in this document are ECDSA with SHA-1, ECDSA with SHA-224, ECDSA with SHA-256, ECDSA with SHA-384, and ECDSA with SHA-512. The object identifiers and parameters associated with these algorithms are found in [PKI-ALG].

[CMS-ECC] indicated the parameters were NULL. Support for this legacy form is OPTIONAL.

7.1.4. Key Agreement Algorithms

Key agreement algorithms are used in EnvelopedData, AuthenticatedData, and AuthEnvelopedData in the KeyAgreeRecipientInfo keyEncryptionAlgorithm field. The following object identifiers indicate the key agreement algorithms used in this document [SP800-56A], [SEC1]:

```
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 KeyWrapAlgorithm to indicate the key wrap algorithm and any associated parameters.

7.1.5. Key Wrap Algorithms

Key wrap algorithms are used as part of the parameters in the key agreement algorithm. The key wrap algorithms used in this document are Triple-DES, AES-128, AES-192, and AES-256. The object identifiers and parameters for these algorithms are found in [CMS-ALG] and [CMS-AES].

7.1.6. Content Encryption Algorithms

Content encryption algorithms are used in EnvelopedData and AuthEnvelopedData in the EncryptedContentInfo contentEncryptionAlgorithm field. The content encryption algorithms used with EnvelopedData in this document are 3-Key Triple DES in CBC mode, AES-128 in CBC mode, AES-192 in CBC mode, and AES-256 in CBC mode. The object identifiers and parameters associated with these algorithms are found in [CMS-ALG] and [CMS-AES]. The content encryption algorithms used with AuthEnvelopedData in this document are AES-128 in CCM mode, AES-192 in CCM mode, AES-256 in CCM mode, AES-128 in GCM mode, AES-192 in GCM mode, and AES-256 in GCM mode. The object identifiers and parameters associated with these algorithms are found in [CMS-AESCG].

7.1.7. Message Authentication Code Algorithms

Message authentication code algorithms are used in AuthenticatedData in the macAlgorithm field. The message authentication code algorithms used in this document are HMAC with SHA-1, HMAC with SHA-224, HMAC with SHA-256, HMAC with SHA-384, and HMAC with SHA-512. The object identifiers and parameters associated with these algorithms are found in [CMS-ALG] and [HMAC-SHA2].

NOTE: [HMAC-SHA2] defines the object identifiers for HMAC with SHA-224, HMAC with SHA-256, HMAC with SHA-384, and HMAC with SHA-512, but there is no ASN.1 module from which to import these object identifiers. Therefore, the object identifiers for these algorithms are included in the ASN.1 modules defined in Appendix A.

7.1.8. Key Derivation Algorithm

The KDF used in this document is as specified in Section 3.6.1 of [SEC1]. The hash algorithm is identified in the key agreement algorithm. For example, dhSinglePass-stdDH-sha256kdf-scheme uses the KDF from [SEC1] but uses SHA-256 instead of SHA-1.

7.2. Other Syntax

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 [PKI-ALG]. Within CMS, ECDSA-Sig-Value is DER-encoded and placed within a signature field of SignedData.

When using ECDH and ECMQV with EnvelopedData, AuthenticatedData, and AuthEnvelopedData, ephemeral and static public keys are encoded using the type ECPoint. Implementations MUST support uncompressed keys, MAY support compressed keys, and MUST NOT support hybrid keys.

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

When using ECMQV with EnvelopedData, AuthenticatedData, and AuthEnvelopedData, 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 is used to represent the ephemeral public key and is placed in the ephemeralPublicKey publicKey field. The additional user keying material is placed in the addedukm field. Then the MQVuserKeyingMaterial value is DER-encoded and placed within the ukm field of EnvelopedData, AuthenticatedData, or AuthEnvelopedData.

When using ECDH or ECMQV with EnvelopedData, AuthenticatedData, or AuthEnvelopedData, 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 associated parameters. In this specification, 3DES wrap has NULL parameters while the AES wraps have absent 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] and [CMS-AES]. (For example, for AES-256 it would be 00 00 01 00.)

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

NOTE: 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 Section 3.6.1 of [SEC1] ensures that sufficient keying data is provided.

8. Recommended Algorithms and Elliptic Curves

It is RECOMMENDED that implementations of this specification support SignedData and EnvelopedData. Support for AuthenticatedData and AuthEnvelopedData is OPTIONAL.

In order to encourage interoperability, implementations SHOULD use the elliptic curve domain parameters specified by [PKI-ALG].

Implementations that support SignedData with ECDSA:

- MUST support ECDSA with SHA-256; and
- MAY support ECDSA with SHA-1, ECDSA with SHA-224, ECDSA with SHA-384, and ECDSA with SHA-512; other digital signature algorithms MAY also be supported.

When using ECDSA, to promote interoperability it is RECOMMENDED that the P-192, P-224, and P-256 curves be used with SHA-256; the P-384 curve be used with SHA-384; and the P-521 curve be used with SHA-512.

If EnvelopedData is supported, then ephemeral-static ECDH standard primitive MUST be supported. Support for ephemeral-static ECDH co-factor is OPTIONAL, and support for 1-Pass ECMQV is also OPTIONAL.

Implementations that support EnvelopedData with the ephemeral-static ECDH standard primitive:

- MUST support the dhSinglePass-stdDH-sha256kdf-scheme key agreement algorithm, the id-aes128-wrap key wrap algorithm, and the id-aes128-cbc content encryption algorithm; and

- MAY support the dhSinglePass-stdDH-sha1kdf-scheme, dhSinglePass-stdDH-sha224kdf-scheme, dhSinglePass-stdDH-sha384kdf-scheme, and dhSinglePass-stdDH-sha512kdf-scheme key agreement algorithms; the id-alg-CMS3DESwrap, id-aes192-wrap, and id-aes256-wrap key wrap algorithms; and the des-ede3-cbc, id-aes192-cbc, and id-aes256-cbc content encryption algorithms; other algorithms MAY also be supported.

Implementations that support EnvelopedData with the ephemeral-static ECDH cofactor primitive:

- MUST support the dhSinglePass-cofactorDH-sha256kdf-scheme key agreement algorithm, the id-aes128-wrap key wrap algorithm, and the id-aes128-cbc content encryption algorithm; and

- MAY support the dhSinglePass-cofactorDH-sha1kdf-scheme, dhSinglePass-cofactorDH-sha224kdf-scheme, dhSinglePass-cofactorDH-sha384kdf-scheme, and dhSinglePass-cofactorDH-sha512kdf-scheme key agreement; the id-alg-CMS3DESwrap, id-aes192-wrap, and id-aes256-wrap key wrap algorithms; and the des-ede3-cbc, id-aes192-cbc, and id-aes256-cbc content encryption algorithms; other algorithms MAY also be supported.

Implementations that support EnvelopedData with 1-Pass ECMQV:

- MUST support the mqvSinglePass-sha256kdf-scheme key agreement algorithm, the id-aes128-wrap key wrap algorithm, and the id-aes128-cbc content encryption algorithm; and

- MAY support the mqvSinglePass-sha1kdf-scheme, mqvSinglePass-sha224kdf-scheme, mqvSinglePass-sha384kdf-scheme, and mqvSinglePass-sha512kdf-scheme key agreement algorithms; the id-alg-CMS3DESwrap, id-aes192-wrap, and id-aes256-wrap key wrap algorithms; and the des-ede3-cbc, id-aes192-cbc, and id-aes256-cbc content encryption algorithms; other algorithms MAY also be supported.

Implementations that support AuthenticatedData with 1-Pass ECMQV:

- MUST support the mqvSinglePass-sha256kdf-scheme key agreement, the id-aes128-wrap key wrap, the id-sha256 message digest, and id-hmacWithSHA256 message authentication code algorithms; and

- MAY support the mqvSinglePass-sha1kdf-scheme, mqvSinglePass-sha224kdf-scheme, mqvSinglePass-sha384kdf-scheme, mqvSinglePass-sha512kdf-scheme key agreement algorithms; the id-alg-CMS3DESwrap, id-aes192-wrap, and id-aes256-wrap key wrap algorithms; the id-sha1, id-sha224, id-sha384, and id-sha512,

message digest algorithms; and the hmac-SHA1, id-hmacWithSHA224, id-hmacWithSHA384, and id-hmacWithSHA512 message authentication code algorithms; other algorithms MAY also be supported.

Implementations that support AuthEnvelopedData with 1-Pass ECMQV:

- MUST support the mqvSinglePass-sha256kdf-scheme key agreement, the id-aes128-wrap key wrap, and the id-aes128-ccm authenticated-content encryption; and

- MAY support the mqvSinglePass-sha1kdf-scheme, mqvSinglePass-sha224kdf-scheme, mqvSinglePass-sha384kdf-scheme, and mqvSinglePass-sha512kdf-scheme key agreement algorithms; the id-alg-CMS3DESwrap, id-aes192-wrap, and id-aes256-wrap key wrap algorithms; and the id-aes192-ccm, id-aes256-ccm, id-aes128-gcm, id-aes192-gcm, and id-aes256-ccm authenticated-content encryption algorithms; other algorithms MAY also be supported.

9. Security Considerations

Cryptographic algorithms will be broken or weakened over time. Implementers and users need to check that the cryptographic algorithms listed in this document continue to provide the expected level of security. The IETF from time to time may issue documents dealing with the current state of the art.

Cryptographic algorithms rely on random numbers. See [RANDOM] for guidance on generation of random numbers.

Receiving agents that validate signatures and sending agents that encrypt messages need to be cautious of cryptographic processing usage when validating signatures and encrypting messages using keys larger than those mandated in this specification. An attacker could send keys and/or certificates with keys that would result in excessive cryptographic processing, for example, keys larger than those mandated in this specification, which could swamp the processing element. Agents that use such keys without first validating the certificate to a trust anchor are advised to have some sort of cryptographic resource management system to prevent such attacks.

Using secret keys of an appropriate size is crucial to the security of a Diffie-Hellman exchange. For elliptic curve groups, the size of the secret key must be equal to the size of n (the order of the group generated by the point g). Using larger secret keys provides absolutely no additional security, and using smaller secret keys is likely to result in dramatically less security. (See [SP800-56A] for more information on selecting secret keys.)

This specification is based on [CMS], [CMS-AES], [CMS-AESCG], [CMS-ALG], [CMS-AUTHENV], [CMS-DH], [CMS-SHA2], [FIPS180-3], [FIPS186-3], and [HMAC-SHA2], and the appropriate security considerations of those documents apply.

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

When implementing EnvelopedData, AuthenticatedData, and AuthEnvelopedData, 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 the strength of the security provided by each of these choices. Security algorithm strength is measured in bits, where bits is measured in equivalence to a symmetric cipher algorithm. Thus, a strong symmetric cipher algorithm with a key of X bits is said to provide X bits of security. For other algorithms, the key size is mapped to an equivalent symmetric cipher strength. It is recommended that the bits of security provided by each are roughly equivalent. The following table provides comparable minimum bits of security [SP800-57] 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 ECMQV Key Size	Key Derivation Function	Key Wrap Alg.	Content Encryption Alg.	Curves
80	160-223	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512	3DES AES-128 AES-192 AES-256	3DES CBC AES-128 CBC AES-192 CBC AES-256 CBC	sect163k1 secp163r2 secp192r1
112	224-255	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512	3DES AES-128 AES-192 AES-256	3DES CBC AES-128 CBC AES-192 CBC AES-256 CBC	secp224r1 sect233k1 sect233r1
128	256-383	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512	AES-128 AES-192 AES-256	AES-128 CBC AES-192 CBC AES-256 CBC	secp256r1 sect283k1 sect283r1
192	384-511	SHA-224 SHA-256 SHA-384 SHA-512	AES-192 AES-256	AES-192 CBC AES-256 CBC	secp384r1 sect409k1 sect409r1
256	512+	SHA-256 SHA-384 SHA-512	AES-256	AES-256 CBC	secp521r1 sect571k1 sect571r1

To promote interoperability, the following choices are RECOMMENDED:

Minimum Bits of Security	ECDH or ECMQV Key Size	Key Derivation Function	Key Wrap Alg.	Content Encryption Alg.	Curve
80	192	SHA-256	3DES	3DES CBC	secp192r1
112	224	SHA-256	3DES	3DES CBC	secp224r1
128	256	SHA-256	AES-128	AES-128 CBC	secp256r1
192	384	SHA-384	AES-256	AES-256 CBC	secp384r1
256	512+	SHA-512	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 [SP800-57] 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	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512	sect163k1 secp163r2 secp192r1
112	224-255	SHA-224 SHA-256 SHA-384 SHA-512	secp224r1 sect233k1 sect233r1
128	256-383	SHA-256 SHA-384 SHA-512	secp256r1 sect283k1 sect283r1
192	384-511	SHA-384 SHA-512	secp384r1 sect409k1 sect409r1
256	512+	SHA-512	secp521r1 sect571k1 sect571r1

To promote interoperability, the following choices are RECOMMENDED:

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

10. IANA Considerations

This document makes extensive use of object identifiers to register originator public key types and algorithms. The algorithm object identifiers are registered in the ANSI X9.62, ANSI X9.63, NIST, RSA, and SECG arcs. Additionally, object identifiers are used to identify the ASN.1 modules found in Appendix A (there are two). These are defined by the SMIME WG Registrar in an arc delegated by RSA to the SMIME Working Group: iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) modules(0). No action by IANA is necessary for this document or any anticipated updates.

11. References

11.1. Normative References

[CMS] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 5652, September 2009.

[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, December 2007.

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

[CMS-AUTHENV] Housley, R., "Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type", RFC 5083, November 2007.

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

[CMS-SHA2] Turner, S., "Using SHA2 Algorithms with Cryptographic Message Syntax", RFC 5754, January 2010.

[FIPS180-3] National Institute of Standards and Technology (NIST), FIPS Publication 180-3: Secure Hash Standard, October 2008.

[FIPS186-3] National Institute of Standards and Technology (NIST), FIPS Publication 186-3: Digital Signature Standard, June 2009.

[HMAC-SHA2] Nystrom, M., "Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512", RFC 4231, December 2005.

[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, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[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 T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.

[RANDOM] Eastlake, D., 3rd, Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RSAOAEP] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, June 2005.

[SEC1] Standards for Efficient Cryptography Group, "SEC 1: Elliptic Curve Cryptography", version 2.0, May 2009, available from www.secg.org.

[SP800-56A] National Institute of Standards and Technology (NIST), Special Publication 800-56A: Recommendation Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised), March 2007.

[X.680] ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1:2002. Information Technology - Abstract Syntax Notation One.

11.2. Informative References

[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

[CERTCAP] Santesson, S., "X.509 Certificate Extension for Secure/Multipurpose Internet Mail Extensions (S/MIME) Capabilities", RFC 4262, December 2005.

[CMS-ASN] Hoffman, P. and J. Schaad, "New ASN.1 Modules for CMS and S/MIME", Work in Progress, August 2009.

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

[CMS-KEA] Pawling, J., "Use of the KEA and SKIPJACK Algorithms in CMS", RFC 2876, July 2000.

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

[PKI-ASN] Hoffman, P. and J. Schaad, "New ASN.1 Modules for PKIX", Work in Progress, August 2009.

[SP800-57] National Institute of Standards and Technology (NIST), Special Publication 800-57: Recommendation for Key Management - Part 1 (Revised), March 2007.

[X.681] ITU-T Recommendation X.681 (2002) | ISO/IEC 8824-2:2002. Information Technology - Abstract Syntax Notation One: Information Object Specification.

[X.682] ITU-T Recommendation X.682 (2002) | ISO/IEC 8824-3:2002. Information Technology - Abstract Syntax Notation One: Constraint Specification.

[X.683] ITU-T Recommendation X.683 (2002) | ISO/IEC 8824-4:2002. Information Technology - Abstract Syntax Notation One: Parameterization of ASN.1 Specifications, 2002.

[X9.62] X9.62-2005, "Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Standard (ECDSA)", November, 2005.

Appendix 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.680] for compilers that support the 1988 ASN.1.

Appendix A.2 provides informative ASN.1 definitions for the structures described in this specification using ASN.1 as defined in [X.680], [X.681], [X.682], and [X.683] for compilers that support the 2002 ASN.1. 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.

A.1. 1988 ASN.1 Module

```
CMSECCAlgs-2009-88
  { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
    smime(16) modules(0) id-mod-cms-ecc-alg-2009-88(45) }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL

IMPORTS

-- From [PKI]

AlgorithmIdentifier
  FROM PKIX1Explicit88
    { iso(1) identified-organization(3) dod(6)
      internet(1) security(5) mechanisms(5) pkix(7) mod(0)
      pkix1-explicit(18) }

-- From [RSAOAEP]

id-sha224, id-sha256, id-sha384, id-sha512
  FROM PKIX1-PSS-OAEP-Algorithms
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-rsa-pkalgs(33) }
```

```
   -- From [PKI-ALG]

   id-sha1, ecdsa-with-SHA1, ecdsa-with-SHA224,
   ecdsa-with-SHA256, ecdsa-with-SHA384, ecdsa-with-SHA512,
   id-ecPublicKey, ECDSA-Sig-Value, ECPoint, ECParameters
     FROM PKIX1Algorithms2008
       { iso(1) identified-organization(3) dod(6) internet(1)
         security(5) mechanisms(5) pkix(7) id-mod(0) 45 }

   -- 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, des-ede3-cbc, 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) }

   -- From [CMS-AES]

   id-aes128-CBC, id-aes192-CBC, id-aes256-CBC, AES-IV,
   id-aes128-wrap, id-aes192-wrap, id-aes256-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) }

   ;

   --
   -- Message Digest Algorithms: Imported from [PKI-ALG] and [RSAOAEP]
   --

   -- id-sha1 Parameters are preferred absent
   -- id-sha224 Parameters are preferred absent
   -- id-sha256 Parameters are preferred absent
```

```
   -- id-sha384 Parameters are preferred absent
   -- id-sha512 Parameters are preferred absent

   --
   -- Signature Algorithms: Imported from [PKI-ALG]
   --

   -- ecdsa-with-SHA1 Parameters are NULL
   -- ecdsa-with-SHA224 Parameters are absent
   -- ecdsa-with-SHA256 Parameters are absent
   -- ecdsa-with-SHA384 Parameters are absent
   -- ecdsa-with-SHA512 Parameters are absent

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

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

   --
   -- Key Agreement Algorithms
   --

   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) }

   --
   -- Diffie-Hellman Single Pass, Standard, with KDFs
   --

   -- Parameters are always present and indicate the key wrap algorithm
   -- with KeyWrapAlgorithm.

   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 }

   --
   -- Diffie-Hellman Single Pass, Cofactor, with KDFs
   --

   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 }

   --
   -- MQV Single Pass, Cofactor, with KDFs
   --

   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 }

   --
   -- Key Wrap Algorithms: Imported from [CMS-ALG] and [CMS-AES]
   --
```

```
   KeyWrapAlgorithm ::= AlgorithmIdentifier

   -- id-alg-CMS3DESwrap Parameters are NULL
   -- id-aes128-wrap Parameters are absent
   -- id-aes192-wrap Parameters are absent
   -- id-aes256-wrap Parameters are absent

   --
   -- Content Encryption Algorithms: Imported from [CMS-ALG]
   -- and [CMS-AES]
   --

   -- des-ede3-cbc Parameters are CBCParameter
   -- id-aes128-CBC Parameters are AES-IV
   -- id-aes192-CBC Parameters are AES-IV
   -- id-aes256-CBC Parameters are AES-IV
   -- id-aes128-CCM Parameters are CCMParameters
   -- id-aes192-CCM Parameters are CCMParameters
   -- id-aes256-CCM Parameters are CCMParameters
   -- id-aes128-GCM Parameters are GCMParameters
   -- id-aes192-GCM Parameters are GCMParameters
   -- id-aes256-GCM Parameters are GCMParameters

   --
   -- Message Authentication Code Algorithms
   --

   -- hMAC-SHA1 Parameters are preferred absent

   -- HMAC with SHA-224, SHA-256, SHA_384, and SHA-512 Parameters are
   -- absent

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

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

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

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

```
   --
   -- Originator Public Key Algorithms: Imported from [PKI-ALG]
   --

   -- id-ecPublicKey Parameters are absent, NULL, or ECParameters

   -- Format for both ephemeral and static public keys: Imported from
   -- [PKI-ALG]

   -- ECPoint ::= OCTET STRING

   -- ECParameters ::= CHOICE {
   --   namedCurve      OBJECT IDENTIFIER
   --   commented out in [PKI-ALG]  implicitCurve   NULL
   --   commented out in [PKI-ALG]  specifiedCurve  SpecifiedECDomain
   --   commented out in [PKI-ALG]  ...
   -- }
       -- implicitCurve and specifiedCurve MUST NOT be used in PKIX.
       -- Details for SpecifiedECDomain can be found in [X9.62].
       -- Any future additions to this CHOICE should be coordinated
       -- with ANSI X9.

   -- Format of KeyAgreeRecipientInfo ukm field when used with
   -- ECMQV

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

   -- 'SharedInfo' for input to KDF when using ECDH and ECMQV with
   -- EnvelopedData, AuthenticatedData, or AuthEnvelopedData

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

   --
   -- S/MIME Capabilities
   -- An identifier followed by type.
   --
```

```
   --
   -- S/MIME Capabilities: Message Digest Algorithms
   --

   -- Found in [CMS-SHA2].

   --
   -- S/MIME Capabilities: Signature Algorithms
   --

   -- ecdsa-with-SHA1 Type NULL
   -- ecdsa-with-SHA224 Type absent
   -- ecdsa-with-SHA256 Type absent
   -- ecdsa-with-SHA384 Type absent
   -- ecdsa-with-SHA512 Type absent

   --
   -- S/MIME Capabilities: ECDH, Single Pass, Standard
   --

   -- dhSinglePass-stdDH-sha1kdf Type is the KeyWrapAlgorithm
   -- dhSinglePass-stdDH-sha224kdf Type is the KeyWrapAlgorithm
   -- dhSinglePass-stdDH-sha256kdf Type is the KeyWrapAlgorithm
   -- dhSinglePass-stdDH-sha384kdf Type is the KeyWrapAlgorithm
   -- dhSinglePass-stdDH-sha512kdf Type is the KeyWrapAlgorithm


   --
   -- S/MIME Capabilities: ECDH, Single Pass, Cofactor
   --

   -- dhSinglePass-cofactorDH-sha1kdf Type is the KeyWrapAlgorithm
   -- dhSinglePass-cofactorDH-sha224kdf Type is the KeyWrapAlgorithm
   -- dhSinglePass-cofactorDH-sha256kdf Type is the KeyWrapAlgorithm
   -- dhSinglePass-cofactorDH-sha384kdf Type is the KeyWrapAlgorithm
   -- dhSinglePass-cofactorDH-sha512kdf Type is the KeyWrapAlgorithm

   --
   -- S/MIME Capabilities: ECMQV, Single Pass, Standard
   --

   -- mqvSinglePass-sha1kdf Type is the KeyWrapAlgorithm
   -- mqvSinglePass-sha224kdf Type is the KeyWrapAlgorithm
   -- mqvSinglePass-sha256kdf Type is the KeyWrapAlgorithm
   -- mqvSinglePass-sha384kdf Type is the KeyWrapAlgorithm
   -- mqvSinglePass-sha512kdf Type is the KeyWrapAlgorithm
```

```
   --
   -- S/MIME Capabilities: Message Authentication Code Algorithms
   --

   -- hMACSHA1 Type is preferred absent
   -- id-hmacWithSHA224 Type is absent
   -- if-hmacWithSHA256 Type is absent
   -- id-hmacWithSHA384 Type is absent
   -- id-hmacWithSHA512 Type is absent

   END
```

A.2. 2004 ASN.1 Module

```
CMSECCAlgs-2009-02
  { 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) }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL

IMPORTS

-- From [PKI-ASN]

mda-sha1, sa-ecdsaWithSHA1, sa-ecdsaWithSHA224, sa-ecdsaWithSHA256,
sa-ecdsaWithSHA384, sa-ecdsaWithSHA512, id-ecPublicKey,
ECDSA-Sig-Value, ECPoint, ECParameters
  FROM PKIXAlgs-2009
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-algorithms2008-02(56) }

-- From [PKI-ASN]

mda-sha224, mda-sha256, mda-sha384, mda-sha512
  FROM PKIX1-PSS-OAEP-Algorithms-2009
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-rsa-pkalgs-02(54) }

-- FROM [CMS-ASN]

KEY-WRAP, SIGNATURE-ALGORITHM, DIGEST-ALGORITHM, ALGORITHM,
PUBLIC-KEY, MAC-ALGORITHM, CONTENT-ENCRYPTION, KEY-AGREE, SMIME-CAPS,
AlgorithmIdentifier{}
  FROM AlgorithmInformation-2009
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-algorithmInformation-02(58) }

-- From [CMS-ASN]

OriginatorPublicKey, UserKeyingMaterial
  FROM CryptographicMessageSyntax-2009
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) id-mod-cms-2004-02(41) }
```

```
-- From [CMS-ASN]

maca-hMAC-SHA1, cea-3DES-cbc, kwa-3DESWrap, CBCParameter
  FROM CryptographicMessageSyntaxAlgorithms-2009
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) id-mod-cmsalg-2001-02(37) }

-- From [CMS-ASN]

cea-aes128-cbc, cea-aes192-cbc, cea-aes256-cbc, kwa-aes128-wrap,
kwa-aes192-wrap, kwa-aes256-wrap
  FROM CMSAesRsaesOaep-2009
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) id-mod-cms-aes-02(38) }

-- From [CMS-ASN]

cea-aes128-CCM, cea-aes192-CCM, cea-aes256-CCM, cea-aes128-GCM,
cea-aes192-GCM, cea-aes256-GCM
  FROM CMS-AES-CCM-and-AES-GCM-2009
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
      smime(16) modules(0) id-mod-cms-aes-ccm-gcm-02(44) }

;

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

-- Message Digest Algorithms: Imported from [PKI-ASN]

-- MessageDigestAlgs DIGEST-ALGORITHM ::= {
--  mda-sha1   |
--  mda-sha224 |
--  mda-sha256 |
--  mda-sha384 |
--  mda-sha512,
--  ...
-- }

-- Constrains the SignedData SignerInfo signatureAlgorithm field

-- Signature Algorithms: Imported from [PKI-ASN]

-- SignatureAlgs SIGNATURE-ALGORITHM ::= {
--  sa-ecdsaWithSHA1   |
--  sa-ecdsaWithSHA224 |
--  sa-ecdsaWithSHA256 |
```

```
--  sa-ecdsaWithSHA384 |
--  sa-ecdsaWithSHA512,
--  ...
-- }

-- ECDSA Signature Value: Imported from [PKI-ALG]
-- Contents of SignatureValue OCTET STRING

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

--
-- Key Agreement Algorithms
--

-- 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

KeyAgreementAlgs KEY-AGREE ::= {
  kaa-dhSinglePass-stdDH-sha1kdf-scheme         |
  kaa-dhSinglePass-stdDH-sha224kdf-scheme       |
  kaa-dhSinglePass-stdDH-sha256kdf-scheme       |
  kaa-dhSinglePass-stdDH-sha384kdf-scheme       |
  kaa-dhSinglePass-stdDH-sha512kdf-scheme       |
  kaa-dhSinglePass-cofactorDH-sha1kdf-scheme    |
  kaa-dhSinglePass-cofactorDH-sha224kdf-scheme  |
  kaa-dhSinglePass-cofactorDH-sha256kdf-scheme  |
  kaa-dhSinglePass-cofactorDH-sha384kdf-scheme  |
  kaa-dhSinglePass-cofactorDH-sha512kdf-scheme  |
  kaa-mqvSinglePass-sha1kdf-scheme              |
  kaa-mqvSinglePass-sha224kdf-scheme            |
  kaa-mqvSinglePass-sha256kdf-scheme            |
  kaa-mqvSinglePass-sha384kdf-scheme            |
  kaa-mqvSinglePass-sha512kdf-scheme,
  ...
}
```

```
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) }

--
-- Diffie-Hellman Single Pass, Standard, with KDFs
--

-- Parameters are always present and indicate the Key Wrap Algorithm

kaa-dhSinglePass-stdDH-sha1kdf-scheme KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-stdDH-sha1kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-dhSinglePass-stdDH-sha1kdf-scheme
}

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

kaa-dhSinglePass-stdDH-sha224kdf-scheme KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-stdDH-sha224kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-dhSinglePass-stdDH-sha224kdf-scheme
}

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

kaa-dhSinglePass-stdDH-sha256kdf-scheme KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-stdDH-sha256kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-dhSinglePass-stdDH-sha256kdf-scheme
}

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

```
kaa-dhSinglePass-stdDH-sha384kdf-scheme KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-stdDH-sha384kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-dhSinglePass-stdDH-sha384kdf-scheme
}

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

kaa-dhSinglePass-stdDH-sha512kdf-scheme KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-stdDH-sha512kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-dhSinglePass-stdDH-sha512kdf-scheme
}

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

--
-- Diffie-Hellman Single Pass, Cofactor, with KDFs
--

kaa-dhSinglePass-cofactorDH-sha1kdf-scheme KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-cofactorDH-sha1kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-dhSinglePass-cofactorDH-sha1kdf-scheme
}

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

kaa-dhSinglePass-cofactorDH-sha224kdf-scheme KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-cofactorDH-sha224kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-dhSinglePass-cofactorDH-sha224kdf-scheme
}

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

```
kaa-dhSinglePass-cofactorDH-sha256kdf-scheme KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-cofactorDH-sha256kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-dhSinglePass-cofactorDH-sha256kdf-scheme
}

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

kaa-dhSinglePass-cofactorDH-sha384kdf-scheme KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-cofactorDH-sha384kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-dhSinglePass-cofactorDH-sha384kdf-scheme
}

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

kaa-dhSinglePass-cofactorDH-sha512kdf-scheme KEY-AGREE ::= {
  IDENTIFIER dhSinglePass-cofactorDH-sha512kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-dhSinglePass-cofactorDH-sha512kdf-scheme
}

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

--
-- MQV Single Pass, Cofactor, with KDFs
--

kaa-mqvSinglePass-sha1kdf-scheme KEY-AGREE ::= {
  IDENTIFIER mqvSinglePass-sha1kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-mqvSinglePass-sha1kdf-scheme
}

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

```
kaa-mqvSinglePass-sha224kdf-scheme KEY-AGREE ::= {
  IDENTIFIER mqvSinglePass-sha224kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-mqvSinglePass-sha224kdf-scheme
}

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

kaa-mqvSinglePass-sha256kdf-scheme KEY-AGREE ::= {
  IDENTIFIER mqvSinglePass-sha256kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-mqvSinglePass-sha256kdf-scheme
}

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

kaa-mqvSinglePass-sha384kdf-scheme KEY-AGREE ::= {
  IDENTIFIER mqvSinglePass-sha384kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-mqvSinglePass-sha384kdf-scheme
}

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

kaa-mqvSinglePass-sha512kdf-scheme KEY-AGREE ::= {
  IDENTIFIER mqvSinglePass-sha512kdf-scheme
  PARAMS TYPE KeyWrapAlgorithm ARE required
  UKM -- TYPE unencoded data -- ARE preferredPresent
  SMIME-CAPS cap-kaa-mqvSinglePass-sha512kdf-scheme
}

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

--
-- Key Wrap Algorithms: Imported from [CMS-ASN]
--
```

```
KeyWrapAlgorithm ::= AlgorithmIdentifier { KEY-WRAP, { KeyWrapAlgs } }

KeyWrapAlgs KEY-WRAP ::= {
  kwa-3DESWrap    |
  kwa-aes128-wrap |
  kwa-aes192-wrap |
  kwa-aes256-wrap,
  ...
}

--
-- Content Encryption Algorithms: Imported from [CMS-ASN]
--

-- Constrains the EnvelopedData EncryptedContentInfo encryptedContent
-- field and the AuthEnvelopedData EncryptedContentInfo
-- contentEncryptionAlgorithm field

-- ContentEncryptionAlgs CONTENT-ENCRYPTION ::= {
--   cea-3DES-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,
--   ...
--   }

-- des-ede3-cbc and aes*-cbc are used with EnvelopedData and
-- EncryptedData
-- aes*-ccm are used with AuthEnvelopedData
-- aes*-gcm are used with AuthEnvelopedData
-- (where * is 128, 192, and 256)

--
-- Message Authentication Code Algorithms
--

-- Constrains the AuthenticatedData
-- MessageAuthenticationCodeAlgorithm field
--
```

```
MessageAuthAlgs MAC-ALGORITHM ::= {
--  maca-hMAC-SHA1 |
  maca-hMAC-SHA224 |
  maca-hMAC-SHA256 |
  maca-hMAC-SHA384 |
  maca-hMAC-SHA512,
  ...
}

maca-hMAC-SHA224 MAC-ALGORITHM ::= {
  IDENTIFIER id-hmacWithSHA224
  PARAMS ARE absent
  IS-KEYED-MAC TRUE
  SMIME-CAPS cap-hMAC-SHA224
}

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

maca-hMAC-SHA256 MAC-ALGORITHM ::= {
  IDENTIFIER id-hmacWithSHA256
  PARAMS ARE absent
  IS-KEYED-MAC TRUE
  SMIME-CAPS cap-hMAC-SHA256
}

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

maca-hMAC-SHA384 MAC-ALGORITHM ::= {
  IDENTIFIER id-hmacWithSHA384
  PARAMS ARE absent
  IS-KEYED-MAC TRUE
  SMIME-CAPS cap-hMAC-SHA384
}

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

maca-hMAC-SHA512 MAC-ALGORITHM ::= {
  IDENTIFIER id-hmacWithSHA512
  PARAMS ARE absent
  IS-KEYED-MAC TRUE
  SMIME-CAPS cap-hMAC-SHA512
}
```

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

--
-- Originator Public Key Algorithms
--

-- Constraints on KeyAgreeRecipientInfo OriginatorIdentifierOrKey
-- OriginatorPublicKey algorithm field

OriginatorPKAlgorithms PUBLIC-KEY ::= {
  opka-ec,
  ...
}

opka-ec PUBLIC-KEY ::={
  IDENTIFIER id-ecPublicKey
  KEY ECPoint
  PARAMS TYPE CHOICE { n NULL, p ECParameters } ARE preferredAbsent
}

-- Format for both ephemeral and static public keys: Imported from
-- [PKI-ALG]

-- ECPoint ::= OCTET STRING

-- ECParameters ::= CHOICE {
--   namedCurve      CURVE.&id({NamedCurve})
--   commented out in [PKI-ALG] implicitCurve   NULL
--   commented out in [PKI-ALG] specifiedCurve  SpecifiedECDomain
--   commented out in [PKI-ALG] ...
-- }
  -- implicitCurve and specifiedCurve MUST NOT be used in PKIX.
  -- Details for SpecifiedECDomain can be found in [X9.62].
  -- Any future additions to this CHOICE should be coordinated
  -- with ANSI X.9.

-- Format of KeyAgreeRecipientInfo ukm field when used with
-- ECMQV

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

```
-- 'SharedInfo' for input to KDF when using ECDH and ECMQV with
-- EnvelopedData, AuthenticatedData, or AuthEnvelopedData

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

--
-- S/MIME CAPS for algorithms in this document
--
```

```
SMimeCAPS SMIME-CAPS ::= {
--  mda-sha1.&smimeCaps
--  mda-sha224.&smimeCaps
--  mda-sha256.&smimeCaps
--  mda-sha384.&smimeCaps
--  mda-sha512.&smimeCaps
--  sa-ecdsaWithSHA1.&smimeCaps
--  sa-ecdsaWithSHA224.&smimeCaps
--  sa-ecdsaWithSHA256.&smimeCaps
--  sa-ecdsaWithSHA384.&smimeCaps
--  sa-ecdsaWithSHA512.&smimeCaps
  kaa-dhSinglePass-stdDH-sha1kdf-scheme.&smimeCaps
  kaa-dhSinglePass-stdDH-sha224kdf-scheme.&smimeCaps
  kaa-dhSinglePass-stdDH-sha256kdf-scheme.&smimeCaps
  kaa-dhSinglePass-stdDH-sha384kdf-scheme.&smimeCaps
  kaa-dhSinglePass-stdDH-sha512kdf-scheme.&smimeCaps
  kaa-dhSinglePass-cofactorDH-sha1kdf-scheme.&smimeCaps
  kaa-dhSinglePass-cofactorDH-sha224kdf-scheme.&smimeCaps
  kaa-dhSinglePass-cofactorDH-sha256kdf-scheme.&smimeCaps
  kaa-dhSinglePass-cofactorDH-sha384kdf-scheme.&smimeCaps
  kaa-dhSinglePass-cofactorDH-sha512kdf-scheme.&smimeCaps
  kaa-mqvSinglePass-sha1kdf-scheme.&smimeCaps
  kaa-mqvSinglePass-sha224kdf-scheme.&smimeCaps
  kaa-mqvSinglePass-sha256kdf-scheme.&smimeCaps
  kaa-mqvSinglePass-sha384kdf-scheme.&smimeCaps
  kaa-mqvSinglePass-sha512kdf-scheme.&smimeCaps
--  kwa-3des.&smimeCaps
--  kwa-aes128.&smimeCaps
--  kwa-aes192.&smimeCaps
--  kwa-aes256.&smimeCaps
--  cea-3DES-cbc.&smimeCaps
--  cea-aes128-cbc.&smimeCaps
--  cea-aes192-cbc.&smimeCaps
--  cea-aes256-cbc.&smimeCaps
--  cea-aes128-ccm.&smimeCaps
--  cea-aes192-ccm.&smimeCaps
--  cea-aes256-ccm.&smimeCaps
--  cea-aes128-gcm.&smimeCaps
--  cea-aes192-gcm.&smimeCaps
--  cea-aes256-gcm.&smimeCaps
--  maca-hMAC-SHA1.&smimeCaps
  maca-hMAC-SHA224.&smimeCaps
  maca-hMAC-SHA256.&smimeCaps
  maca-hMAC-SHA384.&smimeCaps
  maca-hMAC-SHA512.&smimeCaps,
  ...
}
```

```
cap-kaa-dhSinglePass-stdDH-sha1kdf-scheme SMIME-CAPS ::= {
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY dhSinglePass-stdDH-sha1kdf-scheme
}

cap-kaa-dhSinglePass-stdDH-sha224kdf-scheme SMIME-CAPS ::= {
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY dhSinglePass-stdDH-sha224kdf-scheme
}

cap-kaa-dhSinglePass-stdDH-sha256kdf-scheme SMIME-CAPS ::= {
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY dhSinglePass-stdDH-sha256kdf-scheme
}

cap-kaa-dhSinglePass-stdDH-sha384kdf-scheme SMIME-CAPS ::= {
   TYPE KeyWrapAlgorithm
   IDENTIFIED BY dhSinglePass-stdDH-sha384kdf-scheme
}

cap-kaa-dhSinglePass-stdDH-sha512kdf-scheme SMIME-CAPS ::= {
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY dhSinglePass-stdDH-sha512kdf-scheme
}

cap-kaa-dhSinglePass-cofactorDH-sha1kdf-scheme SMIME-CAPS ::={
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY dhSinglePass-cofactorDH-sha1kdf-scheme
}

cap-kaa-dhSinglePass-cofactorDH-sha224kdf-scheme SMIME-CAPS ::={
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY dhSinglePass-cofactorDH-sha224kdf-scheme
}

cap-kaa-dhSinglePass-cofactorDH-sha256kdf-scheme SMIME-CAPS ::={
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY dhSinglePass-cofactorDH-sha256kdf-scheme
}

cap-kaa-dhSinglePass-cofactorDH-sha384kdf-scheme SMIME-CAPS ::={
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY dhSinglePass-cofactorDH-sha384kdf-scheme
}
```

```
cap-kaa-dhSinglePass-cofactorDH-sha512kdf-scheme SMIME-CAPS ::={
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY dhSinglePass-cofactorDH-sha512kdf-scheme
}

cap-kaa-mqvSinglePass-sha1kdf-scheme SMIME-CAPS ::={
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY mqvSinglePass-sha1kdf-scheme
}

cap-kaa-mqvSinglePass-sha224kdf-scheme SMIME-CAPS ::={
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY mqvSinglePass-sha224kdf-scheme
}

cap-kaa-mqvSinglePass-sha256kdf-scheme SMIME-CAPS ::={
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY mqvSinglePass-sha256kdf-scheme
}

cap-kaa-mqvSinglePass-sha384kdf-scheme SMIME-CAPS ::={
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY mqvSinglePass-sha384kdf-scheme
}

cap-kaa-mqvSinglePass-sha512kdf-scheme SMIME-CAPS ::={
  TYPE KeyWrapAlgorithm
  IDENTIFIED BY mqvSinglePass-sha512kdf-scheme
}

cap-hMAC-SHA224 SMIME-CAPS ::={ IDENTIFIED BY id-hmacWithSHA224 }

cap-hMAC-SHA256 SMIME-CAPS ::={ IDENTIFIED BY id-hmacWithSHA256 }

cap-hMAC-SHA384 SMIME-CAPS ::={ IDENTIFIED BY id-hmacWithSHA384 }

cap-hMAC-SHA512 SMIME-CAPS ::={ IDENTIFIED BY id-hmacWithSHA512 }

END
```

Appendix B. Changes since RFC 3278

The following summarizes the changes:

- Abstract: The basis of the document was changed to refer to NIST FIPS 186-3 and SP800-56A. However, to maintain backwards compatibility the Key Derivation Function from ANSI/SEC1 is retained.

- Section 1: A bullet was added to address AuthEnvelopedData.

- Section 2.1: A sentence was added to indicate FIPS180-3 is used with ECDSA. Replaced reference to ANSI X9.62 with FIPS186-3.

- Section 2.1.1: The permitted digest algorithms were expanded from SHA-1 to SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512.

- Section 2.1.2 and 2.1.3: The bullet addressing integer "e" was deleted.

- Section 3: Added explanation of why static-static ECDH is not included.

- Section 3.1: The reference for DH was changed from RFC 3852 to RFC 3370. Provided text to indicate fields of EnvelopedData are as in CMS.

- Section 3.1.1: The text was updated to include description of all KeyAgreeRecipientInfo fields. Parameters for id-ecPublicKey field changed from NULL to absent or ECParameter. Additional information about ukm was added.

- Section 3.2: The sentence describing the advantages of 1-Pass ECMQV was rewritten.

- Section 3.2.1: The text was updated to include description of all fields. Parameters for id-ecPublicKey field changed from NULL to absent or ECParameters.

- Sections 3.2.2 and 4.1.2: The re-use of ephemeral keys paragraph was reworded.

- Section 4.1: The sentences describing the advantages of 1-Pass ECMQV was moved to Section 4.

- Section 4.1.2: The note about the attack was moved to Section 4.

- Section 4.2: This section was added to address AuthEnvelopedData with ECMQV.

- Section 5: This section was moved to Section 8. The 1st paragraph was modified to recommend both SignedData and EnvelopedData. The requirements were updated for hash algorithms and recommendations for matching curves and hash algorithms. Also, the requirements were expanded to indicate which ECDH and ECMQV variants, key wrap algorithms, and content encryption algorithms are required for each of the content types used in this document. The permitted digest algorithms used in KDFs were expanded from SHA-1 to SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512.

- Section 6 (formerly 7): This section was updated to allow for SMIMECapabilities to be present in certificates. The S/MIME capabilities for ECDSA with SHA-224, SHA-256, SHA-384, and SHA-512 were added to the list of S/MIME Capabilities. Also, updated to include S/MIME capabilities for ECDH and ECMQV using the SHA-224, SHA-256, SHA-384, and SHA-512 algorithms as the KDF.

- Section 7.1 (formerly 8.1): Added sub-sections for digest, signature, originator public key, key agreement, content encryption, key wrap, and message authentication code algorithms. Pointed to algorithms and parameters in appropriate documents for: SHA-224, SHA-256, SHA-384, and SHA-512 as well as SHA-224, SHA-256, SHA-384, and SHA-512 with ECDSA. Also, added algorithm identifiers for ECDH std, ECDH cofactor, and ECMQV with SHA-224, SHA-256, SHA-384, and SHA-512 algorithms as the KDF. Changed id-ecPublicKey parameters to be absent, NULL, or ECParameters, and if present the originator's ECParameters must match the recipient's ECParameters.

- Section 7.2 (formerly 8.2): Updated to include AuthEnvelopedData. Also, added text to address support requirement for compressed, uncompressed, and hybrid keys; changed pointers from ANSI X9.61 to PKIX (where ECDSA-Sig-Value is imported); changed pointers from SECG to NIST specs; and updated example of suppPubInfo to be AES-256. keyInfo's parameters changed from NULL to any associated parameters (AES wraps have absent parameters).

- Section 9: Replaced text, which was a summary paragraph, with an updated security considerations section. Paragraph referring to definitions of SHA-224, SHA-256, SHA-384, and SHA-512 is deleted.

- Updated references.

- Added ASN.1 modules.

- Updated acknowledgements section.

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 and Paul Lambert. Without RFC 3278, this version wouldn't exist.

The authors also wish to thank Alfred Hoenes, Jonathan Herzog, Paul Hoffman, Russ Housley, and Jim Schaad for their valuable input.

Authors' 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