Network Working Group
Request for Comments: 3370
Obsoletes: 2630, 3211
Category: Standards Track

R. Housley
RSA Laboratories

August 2002

Cryptographic Message Syntax (CMS) Algorithms

Status of this Memo

Copyright Notice

Abstract

This document describes the conventions for using several cryptographic algorithms with the Cryptographic Message Syntax (CMS). The CMS is used to digitally sign, digest, authenticate, or encrypt arbitrary message contents.

Table of Contents

1 Introduction

The Cryptographic Message Syntax (CMS) [CMS] is used to digitally sign, digest, authenticate, or encrypt arbitrary message contents. This companion specification describes the use of common cryptographic algorithms with the CMS. Implementations of the CMS may support these algorithms; implementations of the CMS may also support other algorithms as well. However, if an implementation chooses to support one of the algorithms discussed in this document, then the implementation MUST do so as described in this document.

The CMS values are generated using ASN.1 [X.208-88], using BER-encoding [X.209-88]. Algorithm identifiers (which include ASN.1 object identifiers) identify cryptographic algorithms, and some algorithms require additional parameters. When needed, parameters are specified with an ASN.1 structure. The algorithm identifier for each algorithm is specified, and when needed, the parameter structure is specified. The fields in the CMS employed by each algorithm are identified.

1.1 Changes Since RFC 2630

This document obsoletes section 12 of RFC 2630 [OLDCMS]. RFC 3369 [CMS] obsoletes the rest of RFC 2630. Separation of the protocol and algorithm specifications allows each one to be updated without impacting the other. However, the conventions for using additional algorithms with the CMS are likely to be specified in separate documents.

1.2 Terminology

In this document, the key words MUST, MUST NOT, REQUIRED, SHOULD, SHOULD NOT, RECOMMENDED, and MAY are to be interpreted as described in [STDWORDS].

2 Message Digest Algorithms

This section specifies the conventions employed by CMS implementations that support SHA-1 or MD5.

Digest algorithm identifiers are located in the SignedData digestAlgorithms field, the SignerInfo digestAlgorithm field, the DigestedData digestAlgorithm field, and the AuthenticatedData digestAlgorithm field.

Digest values are located in the DigestedData digest field and the Message Digest authenticated attribute. In addition, digest values are input to signature algorithms.

2.1 SHA-1

The SHA-1 message digest algorithm is defined in FIPS Pub 180-1 [SHA1]. The algorithm identifier for SHA-1 is:

```
sha-1 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
    oiw(14) secsig(3) algorithm(2) 26 }
```

There are two possible encodings for the SHA-1 AlgorithmIdentifier parameters field. The two alternatives arise from the fact that when the 1988 syntax for AlgorithmIdentifier was translated into the 1997 syntax, the OPTIONAL associated with the AlgorithmIdentifier parameters got lost. Later the OPTIONAL was recovered via a defect report, but by then many people thought that algorithm parameters were mandatory. Because of this history some implementations encode parameters as a NULL element and others omit them entirely. The correct encoding is to omit the parameters field; however, implementations MUST also handle a SHA-1 AlgorithmIdentifier parameters field which contains a NULL.

The AlgorithmIdentifier parameters field is OPTIONAL. If present, the parameters field MUST contain a NULL. Implementations MUST accept SHA-1 AlgorithmIdentifiers with absent parameters. Implementations MUST accept SHA-1 AlgorithmIdentifiers with NULL parameters. Implementations SHOULD generate SHA-1 AlgorithmIdentifiers with absent parameters.

2.2 MD5

The MD5 digest algorithm is defined in RFC 1321 [MD5]. The algorithm identifier for MD5 is:

```
md5 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
    rsadsi(113549) digestAlgorithm(2) 5 }
```

The AlgorithmIdentifier parameters field MUST be present, and the parameters field MUST contain NULL. Implementations MAY accept the MD5 AlgorithmIdentifiers with absent parameters as well as NULL parameters.

3 Signature Algorithms

This section specifies the conventions employed by CMS implementations that support DSA or RSA (PKCS #1 v1.5).

Signature algorithm identifiers are located in the SignerInfo signatureAlgorithm field of SignedData. Also, signature algorithm identifiers are located in the SignerInfo signatureAlgorithm field of countersignature attributes.

Signature values are located in the SignerInfo signature field of SignedData. Also, signature values are located in the SignerInfo signature field of countersignature attributes.

3.1 DSA

The DSA signature algorithm is defined in FIPS Pub 186 [DSS]. DSA MUST be used with the SHA-1 message digest algorithm.

The algorithm identifier for DSA subject public keys in certificates is:

```
id-dsa OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) x9-57 (10040) x9cm(4) 1 }
```

DSA signature validation requires three parameters, commonly called p, q, and g. When the id-dsa algorithm identifier is used, the AlgorithmIdentifier parameters field is optional. If present, the AlgorithmIdentifier parameters field MUST contain the three DSA parameter values encoded using the Dss-Parms type. If absent, the subject DSA public key uses the same DSA parameters as the certificate issuer.

```
Dss-Parms ::= SEQUENCE {
  p INTEGER,
  q INTEGER,
  g INTEGER  }
```

When the id-dsa algorithm identifier is used, the DSA public key, commonly called Y, MUST be encoded as an INTEGER. The output of this encoding is carried in the certificate subject public key.

```
Dss-Pub-Key ::= INTEGER  -- Y
```

The algorithm identifier for DSA with SHA-1 signature values is:

```
id-dsa-with-sha1 OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) x9-57 (10040) x9cm(4) 3 }
```

When the id-dsa-with-sha1 algorithm identifier is used, the AlgorithmIdentifier parameters field MUST be absent.

When signing, the DSA algorithm generates two values, commonly called r and s. To transfer these two values as one signature, they MUST be encoded using the Dss-Sig-Value type:

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

3.2 RSA

The RSA (PKCS #1 v1.5) signature algorithm is defined in RFC 2437 [NEWPKCS#1]. RFC 2437 specifies the use of the RSA signature algorithm with the SHA-1 and MD5 message digest algorithms.

The algorithm identifier for RSA subject public keys in certificates is:

```
rsaEncryption OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 1 }
```

When the rsaEncryption algorithm identifier is used, the AlgorithmIdentifier parameters field MUST contain NULL.

When the rsaEncryption algorithm identifier is used, the RSA public key, which is composed of a modulus and a public exponent, MUST be encoded using the RSAPublicKey type. The output of this encoding is carried in the certificate subject public key.

```
RSAPublicKey ::= SEQUENCE {
   modulus INTEGER, -- n
   publicExponent INTEGER } -- e
```

CMS implementations that include the RSA (PKCS #1 v1.5) signature algorithm MUST also implement the SHA-1 message digest algorithm. Such implementations SHOULD also support the MD5 message digest algorithm.

The rsaEncryption algorithm identifier is used to identify RSA (PKCS #1 v1.5) signature values regardless of the message digest algorithm employed. CMS implementations that include the RSA (PKCS #1 v1.5) signature algorithm MUST support the rsaEncryption signature value algorithm identifier, and CMS implementations MAY support RSA (PKCS #1 v1.5) signature value algorithm identifiers that specify both the RSA (PKCS #1 v1.5) signature algorithm and the message digest algorithm.

The algorithm identifier for RSA (PKCS #1 v1.5) with SHA-1 signature values is:

```
sha1WithRSAEncryption OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 5 }
```

The algorithm identifier for RSA (PKCS #1 v1.5) with MD5 signature values is:

```
md5WithRSAEncryption OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 4 }
```

When the rsaEncryption, sha1WithRSAEncryption, or md5WithRSAEncryption signature value algorithm identifiers are used, the AlgorithmIdentifier parameters field MUST be NULL.

When signing, the RSA algorithm generates a single value, and that value is used directly as the signature value.

4 Key Management Algorithms

CMS accommodates the following general key management techniques: key agreement, key transport, previously distributed symmetric key-encryption keys, and passwords.

4.1 Key Agreement Algorithms

This section specifies the conventions employed by CMS implementations that support key agreement using X9.42 Ephemeral-Static Diffie-Hellman (X9.42 E-S D-H) and X9.42 Static-Static Diffie-Hellman (X9.42 S-S D-H).

When a key agreement algorithm is used, a key-encryption algorithm is also needed. Therefore, when key agreement is supported, a key-encryption algorithm MUST be provided for each content-encryption algorithm. The key wrap algorithms for Triple-DES and RC2 are described in RFC 3217 [WRAP].

For key agreement of RC2 key-encryption keys, 128 bits MUST be generated as input to the key expansion process used to compute the RC2 effective key [RC2].

Key agreement algorithm identifiers are located in the EnvelopedData RecipientInfos KeyAgreeRecipientInfo keyEncryptionAlgorithm and AuthenticatedData RecipientInfos KeyAgreeRecipientInfo keyEncryptionAlgorithm fields.

Key wrap algorithm identifiers are located in the KeyWrapAlgorithm parameters within the EnvelopedData RecipientInfos KeyAgreeRecipientInfo keyEncryptionAlgorithm and AuthenticatedData RecipientInfos KeyAgreeRecipientInfo keyEncryptionAlgorithm fields.

Wrapped content-encryption keys are located in the EnvelopedData RecipientInfos KeyAgreeRecipientInfo RecipientEncryptedKeys encryptedKey field. Wrapped message-authentication keys are located in the AuthenticatedData RecipientInfos KeyAgreeRecipientInfo RecipientEncryptedKeys encryptedKey field.

4.1.1 X9.42 Ephemeral-Static Diffie-Hellman

Ephemeral-Static Diffie-Hellman key agreement is defined in RFC 2631 [DH-X9.42]. When using Ephemeral-Static Diffie-Hellman, the EnvelopedData RecipientInfos KeyAgreeRecipientInfo field is used as follows:

version MUST be 3.

originator MUST be the originatorKey alternative. The originatorKey algorithm field MUST contain the dh-public-number object identifier with absent parameters. The originatorKey publicKey field MUST contain the sender's ephemeral public key. The dh-public-number object identifier is:

```
dh-public-number OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) ansi-x942(10046) number-type(2) 1 }
```

ukm may be present or absent. CMS implementations MUST support ukm being absent, and CMS implementations SHOULD support ukm being present. When present, the ukm is used to ensure that a different key-encryption key is generated when the ephemeral private key might be used more than once.

keyEncryptionAlgorithm MUST be the id-alg-ESDH algorithm identifier. The algorithm identifier parameter field for id-alg-ESDH is KeyWrapAlgorithm, and this parameter MUST be present. The KeyWrapAlgorithm denotes the symmetric encryption algorithm used to encrypt the content-encryption key with the pairwise key-encryption key generated using the X9.42 Ephemeral-Static Diffie-Hellman key agreement algorithm. Triple-DES and RC2 key wrap algorithms are described in RFC 3217 [WRAP]. The id-alg-ESDH algorithm identifier and parameter syntax is:

```
id-alg-ESDH OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
    alg(3) 5 }

KeyWrapAlgorithm ::= AlgorithmIdentifier
```

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 public key. RecipientEncryptedKey EncryptedKey MUST contain the content-encryption key encrypted with the X9.42 Ephemeral-Static Diffie-Hellman generated pairwise key-encryption key using the algorithm specified by the KeyWrapAlgorithm.

4.1.2 X9.42 Static-Static Diffie-Hellman

Static-Static Diffie-Hellman key agreement is defined in RFC 2631 [DH-X9.42]. When using Static-Static Diffie-Hellman, the EnvelopedData RecipientInfos KeyAgreeRecipientInfo and AuthenticatedData RecipientInfos KeyAgreeRecipientInfo fields are used as follows:

version MUST be 3.

originator MUST be either the issuerAndSerialNumber or subjectKeyIdentifier alternative. In both cases, the originator's certificate contains the sender's static public key. RFC 3279 [CERTALGS] specifies the AlgorithmIdentifier parameters syntax and values that are included in the originator's certificate. The originator's certificate subject public key information field MUST contain the dh-public-number object identifier:

```
dh-public-number OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) ansi-x942(10046) number-type(2) 1 }
```

ukm MUST be present. The ukm ensures that a different key-encryption key is generated for each message between the same sender and recipient.

keyEncryptionAlgorithm MUST be the id-alg-SSDH algorithm identifier. The algorithm identifier parameter field for id-alg-SSDH is KeyWrapAlgorihtm, and this parameter MUST be present. The KeyWrapAlgorithm denotes the symmetric encryption algorithm used to encrypt the content-encryption key with the pairwise key-encryption key generated using the X9.42 Static-Static Diffie-Hellman key agreement algorithm. Triple-DES and RC2 key wrap algorithms are described in RFC 3217 [WRAP]. The id-alg-SSDH algorithm identifier and parameter syntax is:

```
id-alg-SSDH OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
    alg(3) 10 }

KeyWrapAlgorithm ::= AlgorithmIdentifier
```

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 public key. RecipientEncryptedKey EncryptedKey MUST contain the content-encryption key encrypted with the X9.42 Static-Static Diffie-Hellman generated pairwise key-encryption key using the algorithm specified by the KeyWrapAlgortihm.

4.2 Key Transport Algorithms

This section specifies the conventions employed by CMS implementations that support key transport using RSA (PKCS #1 v1.5).

Key transport algorithm identifiers are located in the EnvelopedData RecipientInfos KeyTransRecipientInfo keyEncryptionAlgorithm field.

Key transport encrypted content-encryption keys are located in the EnvelopedData RecipientInfos KeyTransRecipientInfo encryptedKey field.

4.2.1 RSA (PKCS #1 v1.5)

The RSA key transport algorithm is the RSA encryption scheme defined in RFC 2313 [PKCS#1], block type is 02, where the message to be encrypted is the content-encryption key. The algorithm identifier for RSA (PKCS #1 v1.5) is:

```
rsaEncryption OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 1 }
```

The AlgorithmIdentifier parameters field MUST be present, and the parameters field MUST contain NULL.

When using a Triple-DES content-encryption key, CMS implementations MUST adjust the parity bits for each DES key comprising the Triple-DES key prior to RSA encryption.

The use of RSA (PKCS #1 v1.5) encryption, as defined in RFC 2313 [PKCS#1], to provide confidentiality has a known vulnerability. The vulnerability is primarily relevant to usage in interactive applications rather than to store-and-forward environments. Further information and proposed countermeasures are discussed in the Security Considerations section of this document and RFC 3218 [MMA].

Note that the same RSA encryption scheme is also defined in RFC 2437 [NEWPKCS#1]. Within RFC 2437, this RSA encryption scheme is called RSAES-PKCS1-v1_5.

4.3 Symmetric Key-Encryption Key Algorithms

This section specifies the conventions employed by CMS implementations that support symmetric key-encryption key management using Triple-DES or RC2 key-encryption keys. When RC2 is supported, RC2 128-bit keys MUST be used as key-encryption keys, and they MUST be used with the RC2ParameterVersion parameter set to 58. A CMS implementation MAY support mixed key-encryption and content-encryptionalgorithms. For example, a 40-bit RC2 content-encryption key MAY be wrapped with a 168-bit Triple-DES key-encryption key or with a 128-bit RC2 key-encryption key.

Key wrap algorithm identifiers are located in the EnvelopedData RecipientInfos KEKRecipientInfo keyEncryptionAlgorithm and AuthenticatedData RecipientInfos KEKRecipientInfo keyEncryptionAlgorithm fields.

Wrapped content-encryption keys are located in the EnvelopedData RecipientInfos KEKRecipientInfo encryptedKey field. Wrapped message-authentication keys are located in the AuthenticatedData RecipientInfos KEKRecipientInfo encryptedKey field.

The output of a key agreement algorithm is a key-encryption key, and this key-encryption key is used to encrypt the content-encryption key. To support key agreement, key wrap algorithm identifiers are located in the KeyWrapAlgorithm parameter of the EnvelopedData RecipientInfos KeyAgreeRecipientInfo keyEncryptionAlgorithm and AuthenticatedData RecipientInfos KeyAgreeRecipientInfo keyEncryptionAlgorithm fields. However, only key agreement algorithms that inherently provide authentication ought to be used with AuthenticatedData. Wrapped content-encryption keys are located in the EnvelopedData RecipientInfos KeyAgreeRecipientInfo RecipientEncryptedKeys encryptedKey field, wrapped message-authentication keys are located in the AuthenticatedData RecipientInfos KeyAgreeRecipientInfo RecipientEncryptedKeys encryptedKey field.

4.3.1 Triple-DES Key Wrap

A CMS implementation MAY support mixed key-encryption and content-encryption algorithms. For example, a 128-bit RC2 content-encryption key MAY be wrapped with a 168-bit Triple-DES key-encryption key.

Triple-DES key encryption has the algorithm identifier:

```
id-alg-CMS3DESwrap OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) alg(3) 6 }
```

The AlgorithmIdentifier parameter field MUST be NULL.

The key wrap algorithm used to encrypt a Triple-DES content-encryption key with a Triple-DES key-encryption key is specified in section 3.1 of RFC 3217 [WRAP]. The corresponding key unwrap algorithm is specified in section 3.2 of RFC 3217 [WRAP].

Out-of-band distribution of the Triple-DES key-encryption key used to encrypt the Triple-DES content-encryption key is beyond the scope of this document.

4.3.2 RC2 Key Wrap

A CMS implementation MAY support mixed key-encryption and content-encryption algorithms. For example, a 128-bit RC2 content-encryption key MAY be wrapped with a 168-bit Triple-DES key-encryption key. Similarly, a 40-bit RC2 content-encryption key MAY be wrapped with a 128-bit RC2 key-encryption key.

RC2 key encryption has the algorithm identifier:

```
id-alg-CMSRC2wrap OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) alg(3) 7 }
```

The AlgorithmIdentifier parameter field MUST be RC2wrapParameter:

```
RC2wrapParameter ::= RC2ParameterVersion

RC2ParameterVersion ::= INTEGER
```

The RC2 effective-key-bits (key size) greater than 32 and less than 256 is encoded in the RC2ParameterVersion. For the effective-key-bits of 40, 64, and 128, the rc2ParameterVersion values are 160, 120, and 58 respectively. These values are not simply the RC2 key length. Note that the value 160 must be encoded as two octets (00 A0), because the one octet (A0) encoding represents a negative number.

RC2 128-bit keys MUST be used as key-encryption keys, and they MUST be used with the RC2ParameterVersion parameter set to 58.

The key wrap algorithm used to encrypt a RC2 content-encryption key with a RC2 key-encryption key is specified in section 4.1 of RFC 3217 [WRAP]. The corresponding key unwrap algorithm is specified 4.2 of RFC 3217 [WRAP].

Out-of-band distribution of the RC2 key-encryption key used to encrypt the RC2 content-encryption key is beyond of the scope of this document.

4.4 Key Derivation Algorithms

This section specifies the conventions employed by CMS implementations that support password-based key management using PBKDF2.

Key derivation algorithms are used to convert a password into a key-encryption key as part of the password-based key management technique.

Key derivation algorithm identifiers are located in the EnvelopedData RecipientInfos PasswordRecipientInfo keyDerivationAlgorithm and AuthenticatedData RecipientInfos PasswordRecipientInfo keyDerivationAlgorithm fields.

The key-encryption key that is derived from the password is used to encrypt the content-encryption key.

The content-encryption keys encrypted with password-derived key-encryption keys are located in the EnvelopedData RecipientInfos PasswordRecipientInfo encryptedKey field. The message-authentication keys encrypted with password-derived key-encryption keys are located in the AuthenticatedData RecipientInfos PasswordRecipientInfo encryptedKey field.

4.4.1 PBKDF2

The PBKDF2 key derivation algorithm is specified in RFC 2898 [PKCS#5]. The KeyDerivationAlgorithmIdentifer identifies the key-derivation algorithm, and any associated parameters used to derive the key-encryption key from the user-supplied password. The algorithm identifier for the PBKDF2 key derivation algorithm is:

```
id-PBKDF2 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
    rsadsi(113549) pkcs(1) pkcs-5(5) 12 }
```

The AlgorithmIdentifier parameter field MUST be PBKDF2-params:

```
PBKDF2-params ::= SEQUENCE {
  salt CHOICE {
    specified OCTET STRING,
    otherSource AlgorithmIdentifier },
  iterationCount INTEGER (1..MAX),
  keyLength INTEGER (1..MAX) OPTIONAL,
  prf AlgorithmIdentifier
    DEFAULT { algorithm hMAC-SHA1, parameters NULL } }
```

Within the PBKDF2-params, the salt MUST use the specified OCTET STRING.

5 Content Encryption Algorithms

This section specifies the conventions employed by CMS implementations that support content encryption using Three-Key Triple-DES in CBC mode, Two-Key Triple-DES in CBC mode, or RC2 in CBC mode.

Content encryption algorithm identifiers are located in the EnvelopedData EncryptedContentInfo contentEncryptionAlgorithm and the EncryptedData EncryptedContentInfo contentEncryptionAlgorithm fields.

Content encryption algorithms are used to encipher the content located in the EnvelopedData EncryptedContentInfo encryptedContent field and the EncryptedData EncryptedContentInfo encryptedContent field.

5.1 Triple-DES CBC

The Triple-DES algorithm is described in ANSI X9.52 [3DES]. The Triple-DES is composed from three sequential DES [DES] operations: encrypt, decrypt, and encrypt. Three-Key Triple-DES uses a different key for each DES operation. Two-Key Triple-DES uses one key for the two encrypt operations and a different key for the decrypt operation. The same algorithm identifiers are used for Three-Key Triple-DES and Two-Key Triple-DES. The algorithm identifier for Triple-DES in Cipher Block Chaining (CBC) mode is:

```
des-ede3-cbc OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) encryptionAlgorithm(3) 7 }
```

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

```
CBCParameter ::= IV

IV ::= OCTET STRING  -- exactly 8 octets
```

5.2 RC2 CBC

The RC2 algorithm is described in RFC 2268 [RC2]. The algorithm identifier for RC2 in CBC mode is:

```
rc2-cbc OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
    rsadsi(113549) encryptionAlgorithm(3) 2 }
```

The AlgorithmIdentifier parameters field MUST be present, and the parameters field MUST contain a RC2CBCParameter:

```
RC2CBCParameter ::= SEQUENCE {
  rc2ParameterVersion INTEGER,
  iv OCTET STRING  }  -- exactly 8 octets
```

The RC2 effective-key-bits (key size) greater than 32 and less than 256 is encoded in the rc2ParameterVersion. For the effective-key-bits of 40, 64, and 128, the rc2ParameterVersion values are 160, 120, and 58 respectively. These values are not simply the RC2 key length. Note that the value 160 must be encoded as two octets (00 A0), since the one octet (A0) encoding represents a negative number.

6 Message Authentication Code Algorithms

This section specifies the conventions employed by CMS implementations that support the HMAC with SHA-1 message authentication code (MAC).

MAC algorithm identifiers are located in the AuthenticatedData macAlgorithm field.

MAC values are located in the AuthenticatedData mac field.

6.1 HMAC with SHA-1

The HMAC with SHA-1 algorithm is described in RFC 2104 [HMAC]. The algorithm identifier for HMAC with SHA-1 is:

```
hMAC-SHA1 OBJECT IDENTIFIER ::= { iso(1)
   identified-organization(3) dod(6) internet(1) security(5)
   mechanisms(5) 8 1 2 }
```

There are two possible encodings for the HMAC with SHA-1 AlgorithmIdentifier parameters field. The two alternatives arise from the fact that when the 1988 syntax for the AlgorithmIdentifier type was translated into the 1997 syntax, the OPTIONAL associated with the AlgorithmIdentifier parameters got lost. Later the OPTIONAL was recovered via a defect report, but by then many people thought that algorithm parameters were mandatory. Because of this history some implementations may encode parameters as a NULL while others omit them entirely.

The AlgorithmIdentifier parameters field is OPTIONAL. If present, the parameters field MUST contain a NULL. Implementations MUST accept HMAC with SHA-1 AlgorithmIdentifiers with absent parameters. Implementations MUST accept HMAC with SHA-1 AlgorithmIdentifiers with NULL parameters. Implementations SHOULD generate HMAC with SHA-1 AlgorithmIdentifiers with absent parameters.

7 ASN.1 Module

```
CryptographicMessageSyntaxAlgorithms
    { iso(1) member-body(2) us(840) rsadsi(113549)
      pkcs(1) pkcs-9(9) smime(16) modules(0) cmsalg-2001(16) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS All
-- The types and values defined in this module are exported for use
-- in the other ASN.1 modules.  Other applications may use them for
-- their own purposes.

IMPORTS
  -- Imports from RFC 3280 [PROFILE], Appendix A.1
        AlgorithmIdentifier
           FROM PKIX1Explicit88 { iso(1)
                identified-organization(3) dod(6) internet(1)
                security(5) mechanisms(5) pkix(7) mod(0)
                pkix1-explicit(18) } ;

-- Algorithm Identifiers

sha-1 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
    oiw(14) secsig(3) algorithm(2) 26 }

md5 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
    rsadsi(113549) digestAlgorithm(2) 5 }

id-dsa OBJECT IDENTIFIER ::=  { iso(1) member-body(2) us(840)
    x9-57(10040) x9cm(4) 1 }

id-dsa-with-sha1 OBJECT IDENTIFIER ::=  { iso(1) member-body(2)
    us(840) x9-57(10040) x9cm(4) 3 }

rsaEncryption OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 1 }

md5WithRSAEncryption OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 4 }

sha1WithRSAEncryption OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 5 }

dh-public-number OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) ansi-x942(10046) number-type(2) 1 }
```

```
   id-alg-ESDH OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
       rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) alg(3) 5 }

   id-alg-SSDH OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
       rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) alg(3) 10 }

   id-alg-CMS3DESwrap OBJECT IDENTIFIER ::= { iso(1) member-body(2)
       us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) alg(3) 6 }

   id-alg-CMSRC2wrap OBJECT IDENTIFIER ::= { iso(1) member-body(2)
       us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) alg(3) 7 }

   des-ede3-cbc OBJECT IDENTIFIER ::= { iso(1) member-body(2)
       us(840) rsadsi(113549) encryptionAlgorithm(3) 7 }

   rc2-cbc OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
       rsadsi(113549) encryptionAlgorithm(3) 2 }

   hMAC-SHA1 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)
       dod(6) internet(1) security(5) mechanisms(5) 8 1 2 }

   id-PBKDF2 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
       rsadsi(113549) pkcs(1) pkcs-5(5) 12 }

   -- Public Key Types

   Dss-Pub-Key ::= INTEGER  -- Y

   RSAPublicKey ::= SEQUENCE {
     modulus INTEGER,  -- n
     publicExponent INTEGER }  -- e

   DHPublicKey ::= INTEGER  -- y = g^x mod p


   -- Signature Value Types

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

   -- Algorithm Identifier Parameter Types

   Dss-Parms ::= SEQUENCE {
     p INTEGER,
     q INTEGER,
     g INTEGER }
```

```
   DHDomainParameters ::= SEQUENCE {
     p INTEGER,  -- odd prime, p=jq +1
     g INTEGER,  -- generator, g
     q INTEGER,  -- factor of p-1
     j INTEGER OPTIONAL,  -- subgroup factor
     validationParms ValidationParms OPTIONAL }

   ValidationParms ::= SEQUENCE {
     seed BIT STRING,
     pgenCounter INTEGER }

   KeyWrapAlgorithm ::= AlgorithmIdentifier

   RC2wrapParameter ::= RC2ParameterVersion

   RC2ParameterVersion ::= INTEGER

   CBCParameter ::= IV

   IV ::= OCTET STRING  -- exactly 8 octets

   RC2CBCParameter ::= SEQUENCE {
     rc2ParameterVersion INTEGER,
     iv OCTET STRING  }  -- exactly 8 octets

   PBKDF2-params ::= SEQUENCE {
     salt CHOICE {
       specified OCTET STRING,
       otherSource AlgorithmIdentifier },
     iterationCount INTEGER (1..MAX),
     keyLength INTEGER (1..MAX) OPTIONAL,
     prf AlgorithmIdentifier
       DEFAULT { algorithm hMAC-SHA1, parameters NULL } }

   END -- of CryptographicMessageSyntaxAlgorithms
```

8 References

[3DES] American National Standards Institute. ANSI X9.52-1998, Triple Data Encryption Algorithm Modes of Operation. 1998.

[CERTALGS] Bassham, L., Housley, R. and W. Polk, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002.

[CMS] Housley, R., "Cryptographic Message Syntax", RFC 3269, August 2002.

[DES] American National Standards Institute. ANSI X3.106, "American National Standard for Information Systems - Data Link Encryption". 1983.

[DH-X9.42] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, June 1999.

[DSS] National Institute of Standards and Technology. FIPS Pub 186: Digital Signature Standard. 19 May 1994.

[HMAC] Krawczyk, H., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[MD5] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[MMA] Rescorla, E., "Preventing the Million Message Attack on CMS", RFC 3218, January 2002.

[MODES] National Institute of Standards and Technology. FIPS Pub 81: DES Modes of Operation. 2 December 1980.

[NEWPKCS#1] Kaliski, B. and J. Staddon, "PKCS #1: RSA Encryption, Version 2.0, RFC 2437, October 1998.

[OLDCMS] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[PKCS#1] Kaliski, B, "PKCS #1: RSA Encryption, Version 2.0", RFC 2437, October, 1998.

[PKCS#5] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification", RFC 2898, September 2000.

[PROFILE] Housley, R., Ford, W., Polk, W. and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.

[RANDOM] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security, RFC 1750, December 1994.

[RC2] Rivest, R., "A Description of the RC2 (r) Encryption Algorithm", RFC 2268, March 1998.

[SHA1] National Institute of Standards and Technology. FIPS Pub 180-1: Secure Hash Standard. 17 April 1995.

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

[WRAP] Housley, R., "Triple-DES and RC2 Key Wrapping", RFC 3217, December 2001.

[X.208-88] CCITT. Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1). 1988.

[X.209-88] CCITT. Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). 1988.

9 Security Considerations

The CMS provides a method for digitally signing data, digesting data, encrypting data, and authenticating data. This document identifies the conventions for using several cryptographic algorithms for use with the CMS.

Implementations must protect the signer’s private key. Compromise of the signer’s private key permits masquerade.

Implementations must protect the key management private key, the key-encryption key, and the content-encryption key. Compromise of the key management private key or the key-encryption key may result in the disclosure of all contents protected with that key. Similarly, compromise of the content-encryption key may result in disclosure of the associated encrypted content.

Implementations must protect the key management private key and the message-authentication key. Compromise of the key management private key permits masquerade of authenticated data. Similarly, compromise of the message-authentication key may result in undetectable modification of the authenticated content.

The key management technique employed to distribute message-authentication keys must itself provide authentication, otherwise the content is delivered with integrity from an unknown source. Neither RSA [PKCS#1, NEWPKCS#1] nor Ephemeral-Static Diffie-Hellman [DH-X9.42] provide the necessary data origin authentication. Static-Static Diffie-Hellman [DH-X9.42] does provide the necessary data origin authentication when both the originator and recipient public keys are bound to appropriate identities in X.509 certificates [PROFILE].

When more than two parties share the same message-authentication key, data origin authentication is not provided. Any party that knows the message-authentication key can compute a valid MAC, therefore the content could originate from any one of the parties.

Implementations must randomly generate content-encryption keys, message-authentication keys, initialization vectors (IVs), one-time values (such as the k value when generating a DSA signature), and padding. Also, the generation of public/private key pairs relies on a random numbers. The use of inadequate pseudo-random number generators (PRNGs) to generate cryptographic such values can result in little or no security. An attacker may find it much easier to reproduce the PRNG environment that produced the keys, searching the resulting small set of possibilities, rather than brute force searching the whole key space. The generation of quality random numbers is difficult. RFC 1750 [RANDOM] offers important guidance in this area, and Appendix 3 of FIPS Pub 186 [DSS] provides one quality PRNG technique.

When using key agreement algorithms or previously distributed symmetric key-encryption keys, a key-encryption key is used to encrypt the content-encryption key. If the key-encryption and content-encryption algorithms are different, the effective security is determined by the weaker of the two algorithms. If, for example, content is encrypted with 168-bit Triple-DES and the Triple-DES content-encryption key is wrapped with a 40-bit RC2 key, then at most 40 bits of protection is provided. A trivial search to determine the value of the 40-bit RC2 key can recover Triple-DES key, and then the Triple-DES key can be used to decrypt the content. Therefore, implementers must ensure that key-encryption algorithms are as strong or stronger than content-encryption algorithms.

RFC 3217 [WRAP] specifies key wrap algorithms used to encrypt a Triple-DES content-encryption key with a Triple-DES key-encryption key [3DES] or to encrypt a RC2 content-encryption key with a RC2 key-encryption key [RC2]. The key wrap algorithms makes use of CBC mode [MODES]. These key wrap algorithms have been reviewed for use with Triple-DES and RC2. They have not been reviewed for use with other cryptographic modes or other encryption algorithms. Therefore, if a CMS implementation wishes to support ciphers in addition to Triple-DES or RC2, then additional key wrap algorithms need to be defined to support the additional ciphers.

Implementers should be aware that cryptographic algorithms become weaker with time. As new cryptanalysis techniques are developed and computing performance improves, the work factor to break a particular cryptographic algorithm will reduce. Therefore, cryptographic

algorithm implementations should be modular allowing new algorithms to be readily inserted. That is, implementers should be prepared to regularly update the set of algorithms in their implementations.

Users of the CMS, particularly those employing the CMS to support interactive applications, should be aware that RSA (PKCS #1 v1.5), as specified in RFC 2313 [PKCS#1], is vulnerable to adaptive chosen ciphertext attacks when applied for encryption purposes. Exploitation of this identified vulnerability, revealing the result of a particular RSA decryption, requires access to an oracle which will respond to a large number of ciphertexts (based on currently available results, hundreds of thousands or more), which are constructed adaptively in response to previously-received replies providing information on the successes or failures of attempted decryption operations. As a result, the attack appears significantly less feasible to perpetrate for store-and-forward S/MIME environments than for directly interactive protocols. Where the CMS constructs are applied as an intermediate encryption layer within an interactive request-response communications environment, exploitation could be more feasible.

An updated version of PKCS #1 has been published, PKCS #1 Version 2.0 [NEWPKCS#1]. This updated document supersedes RFC 2313. PKCS #1 Version 2.0 preserves support for the encryption padding format defined in PKCS #1 Version 1.5 [PKCS#1], and it also defines a new alternative. To resolve the adaptive chosen ciphertext vulnerability, the PKCS #1 Version 2.0 specifies and recommends use of Optimal Asymmetric Encryption Padding (OAEP) when RSA encryption is used to provide confidentiality. Designers of protocols and systems employing CMS for interactive environments should either consider usage of OAEP, or should ensure that information which could reveal the success or failure of attempted PKCS #1 Version 1.5 decryption operations is not provided. Support for OAEP will likely be added to a future version of the CMS algorithm specification.

See RFC 3218 [MMA] for more information about thwarting the adaptive chosen ciphertext vulnerability in PKCS #1 Version 1.5 implementations.

10 Acknowledgments

This document is the result of contributions from many professionals. I appreciate the hard work of all members of the IETF S/MIME Working Group. I extend a special thanks to Rich Ankney, Simon Blake-Wilson, Tim Dean, Steve Dusse, Carl Ellison, Peter Gutmann, Bob Jueneman, Stephen Henson, Paul Hoffman, Scott Hollenbeck, Don Johnson, Burt Kaliski, John Linn, John Pawling, Blake Ramsdell, Francois Rousseau, Jim Schaad, and Dave Solo for their efforts and support.

11 Author Address

Russell Housley
RSA Laboratories
918 Spring Knoll Drive
Herndon, VA 20170
EMail: rhousley@rsasecurity.com

12. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.