S/MIME Working Group
Internet Draft
Document: draft-ietf-smime-cms-rsa-kem-01.txt
Category: Standards

B. Kaliski
RSA Laboratories

October 2003

Use of the RSA-KEM Key Transport Algorithm in CMS
<draft-ietf-smime-cms-rsa-kem-01.txt>

Status of this Memo

Comments or suggestions for improvement may be made on the "ietf-smime" mailing list, or directly to the author.

Abstract

The RSA-KEM Key Transport Algorithm is a one-pass (store-and-forward) mechanism for transporting keying data to a recipient using the recipient's RSA public key. This document specifies the conventions for using the RSA-KEM Key Transport Algorithm with the Cryptographic Message Syntax (CMS). This version (-01) updates the ASN.1 syntax to align with the latest drafts of ANS X9.44 and ISO/IEC 18033-2, and adds material on certificate conventions and S/MIME capabilities.

Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [STDWORDS].

1. Introduction

The RSA-KEM Key Transport Algorithm is a one-pass (store-and-forward) mechanism for transporting keying data to a recipient using the recipient's RSA public key.

Most previous key transport algorithms based on the RSA public-key cryptosystem (e.g., the popular PKCS #1 v1.5 algorithm [PKCS1]) have the following general form:

1. Format or "pad" the keying data to obtain an integer m.

2. Encrypt the integer m with the recipient's RSA public key:

   c = m^e mod n

3. Output c as the encrypted keying data.

The RSA-KEM Key Transport Algorithm takes a different approach that provides higher security assurance, by encrypting a _random_ integer with the recipient's public key, and using a symmetric key-wrapping scheme to encrypt the keying data. It has the following form:

1. Generate a random integer z between 0 and n-1.

2. Encrypt the integer z with the recipient's RSA public key:

   c = z^e mod n.

3. Derive a key-encrypting key KEK from the integer z.

4. Wrap the keying data using KEK to obtain wrapped keying data WK.

5. Output c and WK as the encrypted keying data.

This different approach provides higher security assurance because the input to the underlying RSA operation is random and independent of the message, and the key-encrypting key KEK is derived from it in a strong way. As a result, the algorithm enjoys a "tight" security proof in the random oracle model. It is also architecturally convenient because the public-key operations are separate from the symmetric operations on the keying data. One benefit is that the length of the keying data is bounded only by the symmetric key-wrapping scheme, not the size of the RSA modulus.

The RSA-KEM Key Transport Algorithm in various forms is being adopted in several draft standards including the draft ANS X9.44 [ANS-X9.44] and the draft ISO/IEC 18033-2 [ISO-IEC-18033-2]. It has also been recommended by the NESSIE project [NESSIE]. Although the other standards are still in development, the algorithm is stable across the drafts. For completeness, a specification of the algorithm is given in Appendix A of this document; ASN.1 syntax is given in

Appendix B.

NOTE: The term KEM stands for "key encapsulation mechanism" and refers to the first three steps of the process above. The formalization of key transport algorithms (or more generally, asymmetric encryption schemes) in terms of key encapsulation mechanisms is described further in research by Victor Shoup leading to the development of the ISO/IEC 18033-2 standard [SHOUP].

2. Use in CMS

The RSA-KEM Key Transport Algorithm MAY be employed for one or more recipients in the CMS enveloped-data content type (Section 6 of [CMS]), where the keying data processed by the algorithm is the CMS content-encryption key.

The RSA-KEM Key Transport Algorithm SHOULD be considered for new CMS-based applications as a replacement for the widely implemented RSA encryption algorithm specified originally in PKCS #1 v1.5 (see [PKCS1] and Section 4.2.1 of [CMSALGS]), which is vulnerable to chosen-ciphertext attacks. The RSAES-OAEP Key Transport Algorithm has also been proposed as a replacement (see [PKCS1] and [CMS-OAEP]). RSA-KEM has the advantage over RSAES-OAEP of a tighter security proof, but the disadvantage of slightly longer encrypted keying data.

2.1 Underlying Components

A CMS implementation that supports the RSA-KEM Key Transport Algorithm MUST support at least the following underlying components:

- For the key derivation function, KDF2 (see [ANS-X9.44][IEEE-P1363a]) based on SHA-1 (see [FIPS-180-2]) (this function is also specified as the key derivation function in [ANS-X9.63])

- For the key-wrapping scheme, AES-Wrap-128, i.e., the AES Key Wrap with a 128-bit key encrypting key (see [AES-WRAP])

An implementation SHOULD also support KDF2 based on SHA-256 (see [FIPS-180-2]), and the Triple-DES Key Wrap (see [3DES-WRAP]). It MAY support other underlying components.

2.2 RecipientInfo Conventions

When the RSA-KEM Key Transport Algorithm is employed for a recipient, recipient, the RecipientInfo alternative for that recipient MUST be KeyTransRecipientInfo. The algorithm-specific fields of the KeyTransRecipientInfo value MUST have the following values:

- keyEncryptionAlgorithm.algorithm MUST be id-ac-generic-hybrid (see Appendix B)

  - keyEncryptionAlgorithm.parameters MUST be a value of type GenericHybridParameters, identifying the RSA-KEM key encapsulation mechanism (see Appendix B)

  - encryptedKey MUST be the encrypted keying data output by the algorithm (see Appendix A)

2.3 Certificate Conventions

The conventions specified in this section augment RFC 3280 [PROFILE].

A recipient who employs the RSA-KEM Key Transport Algorithm MAY identify the public key in a certificate by the same AlgorithmIdentifier as for the PKCS #1 v1.5 algorithm, i.e., using the rsaEncryption object identifier [PKCS1].

If the recipient wishes only to employ the RSA-KEM Key Transport Algorithm with a given public key, the recipient MUST identify the public key in the certificate using the id-ac-generic-hybrid object identifier (see Appendix B) where the associated GenericHybridParameters value indicates the underlying components with which the algorithm is to be employed. The certificate user MUST perform the RSA-KEM Key Transport algorithm using only those components.

Regardless of the AlgorithmIdentifier used, the RSA public key is encoded in the same manner in the subject public key information. The RSA public key MUST be encoded using the type RSAPublicKey type:

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

Here, the modulus is the modulus n, and publicExponent is the public exponent e. The DER encoded RSAPublicKey is carried in the subjectPublicKey BIT STRING within the subject public key information.

The intended application for the key MAY be indicated in the key usage certificate extension (see [PROFILE], Section 4.2.1.3). If the keyUsage extension is present in a certificate that conveys an RSA public key with the id-ac-generic-hybrid object identifier as discussed above, then the key usage extension MUST contain the following value:

```
keyEncipherment.
```

dataEncipherment SHOULD NOT be present. That is, a key intended to be employed only with the RSA-KEM Key Transport Algorithm SHOULD NOT also be employed for data encryption.

2.4 SMIMECapabilities Attribute Conventions

RFC 2633 [MSG], Section 2.5.2 defines the SMIMECapabilities signed attribute (defined as a SEQUENCE of SMIMECapability SEQUENCEs) to be used to specify a partial list of algorithms that the software announcing the SMIMECapabilities can support. When constructing a signedData object, compliant software MAY include the SMIMECapabilities signed attribute announcing that it supports the RSA-KEM Key Transport algorithm.

The SMIMECapability SEQUENCE representing the RSA-KEM Key Transport Algorithm MUST include the id-ac-generic-hybrid object identifier (see Appendix B) in the capabilityID field and MUST include a GenericHybridParameters value in the parameters field identifying the components with which the algorithm is to be employed.

The DER encoding of a SMIMECapability SEQUENCE is the same as the DER encoding of an AlgorithmIdentifier. Example DER encodings for typical sets of components are given in Appendix B.4.

3. Security Considerations

The security of the RSA-KEM Key Transport Algorithm described in this document can be shown to be tightly related to the difficulty of either solving the RSA problem or breaking the underlying symmetric key-wrapping scheme, if the underlying key derivation function is modeled as a random oracle, and assuming that the symmetric key-wrapping scheme satisfies the properties of a data encapsulation mechanism [SHOUP]. While in practice a random-oracle result does not provide an actual security proof for any particular key derivation function, the result does provide assurance that the general construction is reasonable; a key derivation function would need to be particularly weak to lead to an attack that is not possible in the random oracle model.

The RSA key size and the underlying components should be selected consistent with the desired symmetric security level for an application. Several security levels have been identified in [NIST-GUIDELINE]. For brevity, the first three levels are mentioned here:

* 80-bit security. The RSA key size SHOULD be at least 1024 bits, the hash function underlying KDF2 SHOULD be SHA-1 or above, and the symmetric key-wrapping scheme SHOULD be AES Key Wrap or Triple-DES Key Wrap.

* 112-bit security. The RSA key size SHOULD be at least 2048 bits, the hash function underlying KDF2 SHOULD be SHA-224 or above, and the symmetric key-wrapping scheme SHOULD be AES Key

Wrap or Triple-DES Key Wrap.

- 128-bit security. The RSA key size SHOULD be at least 3072 bits, the hash function underlying KDF2 SHOULD be SHA-256 or above, and the symmetric key-wrapping scheme SHOULD be AES Key Wrap.

Note that the AES Key Wrap MAY be used at all three of these levels; the use of AES does not require a 128-bit security level for other components.

Implementations MUST protect the RSA private key and the content-encryption key. Compromise of the RSA private key may result in the disclosure of all messages protected with that key. Compromise of the content-encryption key may result in disclosure of the associated encrypted content.

Additional considerations related to key management may be found in [NIST-GUIDELINE].

The security of the algorithm also depends on the strength of the random number generator, which SHOULD have a comparable security level. For further discussion on random number generation, please see [RANDOM].

Implementations SHOULD NOT reveal information about intermediate values or calculations, whether by timing or other "side channels", or otherwise an opponent may be able to determine information about the keying data and/or the recipient's private key. Although not all intermediate information may be useful to an opponent, it is preferable to conceal as much information as is practical, unless analysis specifically indicates that the information would not be useful.

Generally, good cryptographic practice employs a given RSA key pair in only one scheme. This practice avoids the risk that vulnerability in one scheme may compromise the security of the other, and may be essential to maintain provable security. While RSA public keys have often been employed for multiple purposes such as key transport and digital signature without any known bad interactions, for increased security assurance, such combined use of an RSA key pair is NOT RECOMMENDED in the future (unless the different schemes are specifically designed to be used together).

Accordingly, an RSA key pair used for the RSA-KEM Key Transport Algorithm SHOULD NOT also be used for digital signatures. (Indeed, ASC X9 requires such a separation between key establishment key pairs and digital signature key pairs.) Continuing this principle of key separation, a key pair used for the RSA-KEM Key Transport Algorithm SHOULD NOT be used with other key establishment schemes, or for data encryption, or with more than one set of underlying algorithm components.

Parties MAY wish to formalize the assurance that one another's implementations are correct through implementation validation, e.g. NIST's Cryptographic Module Validation Program (CMVP).

4. References

4.1 Normative References

3DES-WRAP Housley, R. Triple-DES and RC2 Key Wrapping. RFC 3217. December 2001.

AES-WRAP Schaad, J. and R. Housley. Advanced Encryption Standard (AES) Key Wrap Algorithm. RFC 3394. September 2002.

ANS-X9.63 American National Standard X9.63-2002: Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography.

CMS Housley, R. Cryptographic Message Syntax. RFC 3369. August 2002.

CMSALGS Housley, R. Cryptographic Message Syntax (CMS) Algorithms. RFC 3370. August 2002.

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

MSG Ramsdell, B. S/MIME Version 3 Message Specification. RFC 2633. June 1999.

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

STDWORDS Bradner, S. Key Words for Use in RFCs to Indicate Requirement Levels. RFC 2119. March 1997.

4.2 Informative References

ANS-X9.44 ASC X9F1 Working Group. Draft American National Standard X9.44: Public Key Cryptography for the Financial Services Industry -- Key Establishment Using Integer Factorization Cryptography. Draft D6, October 15, 2003.

CMS-OAEP Housley, R. Use of the RSAES-OAEP Key Transport Algorithm in the Cryptographic Message Syntax (CMS). RFC 3560. July 2003.

IEEE-P1363a IEEE P1363 Working Group. IEEE P1363a: Standard Specifications for Public Key Cryptography: Additional Techniques. Draft D12, May 12, 2003. Available via http://grouper.ieee.org/groups/1363.

ISO-IEC-18033-2 ISO/IEC 18033-2: Information technology -- Security techniques -- Encryption algorithms   Part 2: Asymmetric Ciphers. 2nd Committee Draft, July 10, 2003.

NESSIE NESSIE Consortium. Portfolio of Recommended Cryptographic Primitives. February 27, 2003. Available via http://www.cryptonessie.org/.

NIST-GUIDELINE National Institute of Standards and Technology. Special Publication 800-57: Recommendation for Key Management. Part 1: General Guideline. Draft, January 2003. Available via http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html.

PKCS1 Jonsson, J. and B. Kaliski. PKCS #1: RSA Cryptography Specifications Version 2.1. RFC 3447. February 2003.

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

SHOUP Shoup, V. A Proposal for an ISO Standard for Public Key Encryption. Version 2.1, December 20, 2001. Available via http://www.shoup.net/papers/.

5. IANA Considerations

Within the CMS, algorithms are identified by object identifiers (OIDs). With one exception, all of the OIDs used in this document were assigned in other IETF documents, in ISO/IEC standards documents, by the National Institute of Standards and Technology (NIST), and in Public-Key Cryptography Standards (PKCS) documents. The one exception is that the ASN.1 module's identifier (see Appendix B.3) is assigned in this document. No further action by the IANA is necessary for this document or any anticipated updates.

6. Acknowledgments

This document is one part of a strategy to align algorithm standards produced by ASC X9, ISO/IEC JTC1 SC27, NIST, and the IETF. I would like to thank the members of the ASC X9F1 working group for their contributions to drafts of ANS X9.44 which led to this specification. My thanks as well to Russ Housley as well for his guidance and encouragement. I also appreciate the helpful direction I've received from Blake Ramsdell and Jim Schaad in bringing this document to

fruition.

7. Author's Address

Burt Kaliski
RSA Laboratories
174 Middlesex Turnpike
Bedford, MA 01730
USA
bkaliski@rsasecurity.com

Appendix A. RSA-KEM Key Transport Algorithm

The RSA-KEM Key Transport Algorithm is a one-pass (store-and-forward) mechanism for transporting keying data to a recipient using the recipient's RSA public key.

With this type of algorithm, a sender encrypts the keying data using the recipient's public key to obtain encrypted keying data. The recipient decrypts the encrypted keying data using the recipient's private key to recover the keying data.

A.1 Underlying Components

The algorithm has the following underlying components:

* KDF, a key derivation function, which derives keying data of a specified length from a shared secret value

* Wrap, a symmetric key-wrapping scheme, which encrypts keying data using a key-encrypting key

In the following, kekLen denotes the length in bytes of the key-encrypting key for the underlying symmetric key-wrapping scheme.

In this scheme, the length of the keying data to be transported MUST be among the lengths supported by the underlying symmetric key-wrapping scheme. (The AES Key Wrap, for instance, requires the length of the keying data to be a multiple of 8 bytes, and at least 16 bytes.) Usage and formatting of the keying data (e.g., parity adjustment for Triple-DES keys) is outside the scope of this algorithm.

With some key derivation functions, it is possible to include other information besides the shared secret value in the input to the function. Also, with some symmetric key-wrapping schemes, it is possible to associate a label with the keying data. Such uses are outside the scope of this document, as they are not directly supported by CMS.

A.2 Sender's Operations

Let (n,e) be the recipient's RSA public key (see [PKCS1] for details) and let K be the keying data to be transported.

Let nLen denote the length in bytes of the modulus n, i.e., the least integer such that 2^{8*nLen} > n.

The sender performs the following operations:

1. Generate a random integer z between 0 and n-1 (see Note), and convert z to a byte string Z of length nLen, most significant byte first:

```
 z = RandomInteger (0, n-1)
Z = IntegerToString (z, nLen)
```

2. Encrypt the random integer z using the recipient's public key (n,e) and convert the resulting integer c to a ciphertext C, a byte string of length nLen:

```
        c = z^e mod n
C = IntegerToString (c, nLen)
```

3. Derive a key-encrypting key KEK of length kekLen bytes from the byte string Z using the underlying key derivation function:

```
KEK = KDF (Z, kekLen)
```

4. Wrap the keying data K with the key-encrypting key KEK using the underlying key-wrapping scheme to obtain wrapped keying data WK:

```
WK = Wrap (KEK, K)
```

5. Concatenate the ciphertext C and the wrapped keying data WK to obtain the encrypted keying data EK:

```
EK = C || WK
```

6. Output the encrypted keying data EK.

NOTE: The random integer z MUST be generated independently at random for different encryption operations, whether for the same or different recipients.

A.3 Recipient's Operations

Let (n,d) be the recipient's RSA private key (see [PKCS1]; other private key formats are allowed) and let EK be the encrypted keying data.

Let nLen denote the length in bytes of the modulus n.

The recipient performs the following operations:

1. Separate the encrypted keying data EK into a ciphertext C of length nLen bytes and wrapped keying data WK:

```
C || WK = EK
```

If the length of the encrypted keying data is less than nLen bytes, output "decryption error" and stop.

2. Convert the ciphertext C to an integer c, most significant byte first. Decrypt the integer c using the recipient's private key (n,d) to recover an integer z (see Note):

```
c = StringToInteger (C)
     z = c^d mod n
```

If the integer c is not between 0 and n-1, output "decryption error" and stop.

3. Convert the integer z to a byte string Z of length nLen, most significant byte first (see Note):

```
Z = IntegerToString (z, nLen)
```

4. Derive a key-encrypting key KEK of length kekLen bytes from the byte string Z using the underlying key derivation function (see Note):

```
KEK = KDF (Z, kekLen)
```

5. Unwrap the wrapped keying data WK with the key-encrypting key KEK using the underlying key-wrapping scheme to recover the keying data K:

```
K = Unwrap (KEK, WK)
```

If the unwrapping operation outputs an error, output "decryption error" and stop.

6. Output the keying data K.

NOTE: Implementations SHOULD NOT reveal information about the integer z and the string Z, nor about the calculation of the exponentiation in Step 2, the conversion in Step 3, or the key derivation in Step 4, whether by timing or other "side channels". The observable behavior of the implementation SHOULD be the same at these steps for all ciphertexts C that are in range. (For example, IntegerToString conversion should take the same amount of time regardless of the actual value of the integer z.) The integer z, the string Z and other intermediate results MUST be securely deleted when they are no longer needed.

Appendix B. ASN.1 Syntax

The ASN.1 syntax for identifying the RSA-KEM Key Transport Algorithm is an extension of the syntax for the "generic hybrid cipher" in the draft ISO/IEC 18033-2 [ISO-IEC-18033-2], and is the same as employed in the draft ANS X9.44 [ANS-X9.44]. The syntax for the scheme is given in Section B.1. The syntax for selected underlying components including those mentioned above is given in B.2.

The following object identifier prefixes are used in the definitions below:

```
   is18033-2 OID ::= { iso(1) standard(0) is18033(18033) part2(2) }

   nistAlgorithm OID ::= {
      joint-iso-itu-t(2) country(16) us(840) organization(1)
      gov(101) csor(3) nistAlgorithm(4)
   }

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

NullParms is a more descriptive synonym for NULL when an algorithm identifier has null parameters:

```
   NullParms ::= NULL
```

The material in this Appendix is based on a draft standard and is SUBJECT TO CHANGE as that standard is developed.

B.1 RSA-KEM Key Transport Algorithm

The object identifier for the RSA-KEM Key Transport Algorithm is the same as for the "generic hybrid cipher" in the draft ANS ISO/IEC 18033-2, id-ac-generic-hybrid, which is defined in the draft as

```
   id-ac-generic-hybrid OID ::= {
      is18033-2 asymmetric-cipher(1) generic-hybrid(2)
   }
```

The associated parameters for id-ac-generic-hybrid have type GenericHybridParameters:

```
   GenericHybridParameters ::= {
      kem  KeyEncapsulationMechanism,
      dem  DataEncapsulationMechanism
   }
```

The fields of type GenericHybridParameters have the following meanings:

* kem identifies the underlying key encapsulation mechanism. For

the RSA-KEM Key Transport Algorithm, the scheme is RSA-KEM from the draft ISO/IEC 18033-2.

The object identifier for RSA-KEM (as a key encapsulation mechanism) is id-kem-rsa, which is defined in the draft ISO/IEC 18033-2 as

```
id-kem-rsa OID ::= {
   is18033-2 key-encapsulation-mechanism(2) rsa(4)
}
```

The associated parameters for id-kem-rsa have type RsaKemParameters:

```
RsaKemParameters ::= {
  keyDerivationFunction  KeyDerivationFunction,
  keyLength              KeyLength
}
```

The fields of type RsaKemParameters have the following meanings:

* keyDerivationFunction identifies the underlying key derivation function. For alignment with the draft ANS X9.44, it MUST be KDF2. However, other key derivation functions MAY be used with CMS. Please see B.2.1 for the syntax for KDF2.

  ```
  KeyDerivationFunction ::=
     AlgorithmIdentifier {{KDFAlgorithms}}

  KDFAlgorithms ALGORITHMS ::= {
     kdf2,
     ...  -- implementations may define other methods
  }
  ```

* keyLength is the length in bytes of the key-encrypting key, which depends on the underlying symmetric key-wrapping scheme.

  ```
  KeyLength ::= INTEGER (1..MAX)
  ```

* dem identifies the underlying data encapsulation mechanism. For alignment with the draft ANS X9.44, it MUST be an X9-approved symmetric key-wrapping scheme. (See Note.) However, other symmetric key-wrapping schemes MAY be used with CMS. Please see B.2.2 for the syntax for the AES and Triple-DES Key Wraps.

  ```
  DataEncapsulationMechanism ::=
     AlgorithmIdentifier {{DEMAlgorithms}}

  DEMAlgorithms ALGORITHM ::= {
  ```

```
               X9-SymmetricKeyWrappingSchemes,
               ...  -- implementations may define other methods
            }
            X9-SymmetricKeyWrappingSchemes ALGORITHM ::= {
               aes128-Wrap | aes192-Wrap | aes256-Wrap | tdes-Wrap,
               ...   -- allows for future expansion
            }
```

NOTE: The generic hybrid cipher in the draft ISO/IEC 18033-2 can encrypt arbitrary data, hence the term "data encapsulation mechanism". The symmetric key-wrapping schemes take the role of data encapsulation mechanisms in the RSA-KEM Key Transport Algorithm. The draft ISO/IEC 18033-2 currently allows only three particular data encapsulation mechanisms, not including any of these symmetric key-wrapping schemes. However, the ASN.1 syntax in that document expects that additional algorithms will be allowed.

B.2 Selected Underlying Components

B.2.1 Key Derivation Functions

The object identifier for KDF2 (see [ISO-IEC-18033-2]) is

```
      id-kdf-kdf2 OID ::= {
         is18033-2 key-derivation-functions(5) kdf2(2)
      }
```

The associated parameters identify the underlying hash function. For alignment with the draft ANS X9.44, the hash function MUST be an ASC X9-approved hash function. (See Note.) However, other hash functions MAY be used with CMS.

```
      kdf2 ALGORITHM ::= {{ OID id-kdf-kdf2  PARMS KDF2-HashFunction }}

      KDF2-HashFunction ::= AlgorithmIdentifier {{KDF2-HashFunctions}}

      KDF2-HashFunctions ALGORITHM ::= {
         X9-HashFunctions,
         ...  -- implementations may define other methods
      }

      X9-HashFunctions ALGORITHM ::= {
         sha1 | sha224 | sha256 | sha384 | sha512,
         ...  -- allows for future expansion
      }
```

The object identifier for SHA-1 is

```
      id-sha1 OID ::= {
         iso(1) identified-organization(3) oiw(14) secsig(3)
         algorithms(2) sha1(26)
     }
```

The object identifiers for SHA-256, SHA-384 and SHA-512 are

```
   id-sha256 OID ::= { nistAlgorithm hashAlgs(2) sha256(1) }
```

```
      id-sha384 OID ::= { nistAlgorithm hashAlgs(2) sha384(2) }
      id-sha512 OID ::= { nistAlgorithm hashAlgs(2) sha512(3) }
```

There has been some confusion over whether the various SHA object identifiers have a NULL parameter, or no associated parameters. As also discussed in [PKCS1], implementations SHOULD generate algorithm identifiers without parameters, and MUST accept algorithm identifiers either without parameters, or with NULL parameters.

```
      sha1   ALGORITHM ::= {{ OID id-sha1   }} -- NULLParms MUST be
      sha224 ALGORITHM ::= {{ OID id-sha224 }} -- accepted for these
      sha256 ALGORITHM ::= {{ OID id-sha256 }} -- OIDs
      sha384 ALGORITHM ::= {{ OID id-sha384 }}  - ""
      sha512 ALGORITHM ::= {{ OID id-sha512 }}  - ""
```

NOTE: As of this writing, only SHA-1 is an ASC X9-approved hash function; SHA-224 and above are in the process of being approved. The object identifier for SHA-224 has not yet been assigned.

B.2.2 Symmetric Key-Wrapping Schemes

The object identifiers for the AES Key Wrap depends on the size of the key encrypting key. There are three object identifiers (see [AES-WRAP]):

```
      id-aes128-Wrap OID ::= { nistAlgorithm aes(1) aes128-Wrap(5)  }
      id-aes192-Wrap OID ::= { nistAlgorithm aes(1) aes192-Wrap(25) }
      id-aes256-Wrap OID ::= { nistAlgorithm aes(1) aes256-Wrap(45) }
```

These object identifiers have no associated parameters.

```
      aes128-Wrap ALGORITHM ::= {{ OID id-aes128-wrap }}
      aes192-Wrap ALGORITHM ::= {{ OID id-aes192-wrap }}
      aes256-Wrap ALGORITHM ::= {{ OID id-aes256-wrap }}
```

The object identifier for the Triple-DES Key Wrap (see [3DES-WRAP]) is

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

This object identifier has a NULL parameter.

```
      tdes-Wrap ALGORITHM ::=
         {{ OID id-alg-CMS3DESwrap  PARMS NullParms }}
```

NOTE: As of this writing, the AES Key Wrap and the Triple-DES Key

Wrap are in the process of being approved by ASC X9.

B.3 ASN.1 module

```
   CMS-RSA-KEM
      { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
        pkcs-9(9) smime(16) modules(0) cms-rsa-kem(21) } [[check]]

   BEGIN

   -- EXPORTS ALL

   -- IMPORTS None

   -- Useful types and definitions

   OID ::= OBJECT IDENTIFIER  -- alias

   -- Unless otherwise stated, if an object identifier has associated
   -- parameters (i.e., the PARMS element is specified), the parameters
   -- field shall be included in algorithm identifier values. The
   -- parameters field shall be omitted if and only if the object
   -- identifier does not have associated parameters (i.e., the PARMS
   -- element is omitted), unless otherwise stated.

   ALGORITHM ::= CLASS {
      &id    OBJECT IDENTIFIER  UNIQUE,
      &Type  OPTIONAL
   }
   WITH SYNTAX { OID &id [PARMS &Type] }

   AlgorithmIdentifier { ALGORITHM:IOSet } ::= SEQUENCE {
      algorithm   ALGORITHM.&id( {IOSet} ),
      parameters  ALGORITHM.&Type( {IOSet}{@algorithm} )  OPTIONAL
   }

   NullParms ::= NULL

   -- ISO/IEC 18033-2 arc

   is18033-2 OID ::= { iso(1) standard(0) is18033(18033) part2(2) }

   -- NIST algorithm arc

   nistAlgorithm OID ::= {
      joint-iso-itu-t(2) country(16) us(840) organization(1)
      gov(101) csor(3) nistAlgorithm(4)
   }

   -- PKCS #1 arc
   pkcs-1 OID ::= {
      iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1)
   }
   -- RSA-KEM Key Transport Algorithm, based on Generic Hybrid Cipher

   id-ac-generic-hybrid OID ::= {
      is18033-2 asymmetric-cipher(1) generic-hybrid(2)
   }

   GenericHybridParameters ::= {
      kem  KeyEncapsulationMechanism,
      dem  DataEncapsulationMechanism
   }

   id-kem-rsa OID ::= {
      is18033-2 key-encapsulation-mechanism(2) rsa(4)
   }

   RsaKemParameters ::= {
      keyDerivationFunction  KeyDerivationFunction,
      keyLength              KeyLength
   }

   KeyDerivationFunction ::= AlgorithmIdentifier {{KDFAlgorithms}}

   KDFAlgorithms ALGORITHMS ::= {
      kdf2,
      ...  -- implementations may define other methods
   }

   KeyLength ::= INTEGER (1..MAX)

   DataEncapsulationMechanism ::= AlgorithmIdentifier {{DEMAlgorithms}}

   DEMAlgorithms ALGORITHM ::= {
      X9-SymmetricKeyWrappingSchemes,
      ...  -- implementations may define other methods
   }

   X9-SymmetricKeyWrappingSchemes ALGORITHM ::= {
      aes128-Wrap | aes192-Wrap | aes256-Wrap | tdes-Wrap,
      ...   -- allows for future expansion
   }

   -- Key Derivation Functions

   id-kdf-kdf2 OID ::= { is18033-2 key-derivation-functions(5) kdf2(2) }

   kdf2 ALGORITHM ::= {{ OID id-kdf-kdf2  PARMS KDF2-HashFunction }}

   KDF2-HashFunction ::= AlgorithmIdentifier {{KDF2-HashFunctions}}
   KDF2-HashFunctions ALGORITHM ::= {
      X9-HashFunctions,
      ...  -- implementations may define other methods
   }
```

```
-- Hash Functions

X9-HashFunctions ALGORITHM ::= {
   sha1 | sha224 | sha256 | sha384 | sha512,
   ...  -- allows for future expansion
}

id-sha1 OID ::= {
   iso(1) identified-organization(3) oiw(14) secsig(3)
   algorithms(2) sha1(26)
}

id-sha256 OID ::= { nistAlgorithm hashAlgs(2) sha256(1) }
id-sha384 OID ::= { nistAlgorithm hashAlgs(2) sha384(2) }
id-sha512 OID ::= { nistAlgorithm hashAlgs(2) sha512(3) }

sha1   ALGORITHM ::= {{ OID id-sha1    }} -- NullParms MUST be
sha224 ALGORITHM ::= {{ OID id-sha224  }} -- accepted for these
sha256 ALGORITHM ::= {{ OID id-sha256  }} -- OIDs
sha384 ALGORITHM ::= {{ OID id-sha384  }}  - ""
sha512 ALGORITHM ::= {{ OID id-sha512  }}  - ""

-- Symmetric Key-Wrapping Schemes

id-aes128-Wrap OID ::= { nistAlgorithm aes(1) aes128-Wrap(5)  }
id-aes192-Wrap OID ::= { nistAlgorithm aes(1) aes192-Wrap(25) }
id-aes256-Wrap OID ::= { nistAlgorithm aes(1) aes256-Wrap(45) }

aes128-Wrap ALGORITHM ::= {{ OID id-aes128-wrap }}
aes192-Wrap ALGORITHM ::= {{ OID id-aes192-wrap }}
aes256-Wrap ALGORITHM ::= {{ OID id-aes256-wrap }}

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

tdes-Wrap ALGORITHM ::= {{ OID id-alg-CMS3DESwrap  PARMS NullParms }}
```

B.4 Examples

As an example, if the key derivation function is KDF2 based on SHA-256 and the symmetric key-wrapping scheme is the AES Key Wrap with a 128-bit KEK, the AlgorithmIdentifier for the RSA-KEM Key Transport Algorithm will have the following value:

```
   SEQUENCE {
      id-ac-generic-hybrid,                         -- generic cipher
      SEQUENCE {                           -- GenericHybridParameters
```

```
            SEQUENCE {                    -- key encapsulation mechanism
               id-kem-rsa,                                     -- RSA-KEM
               SEQUENCE {                            -- RsaKemParameters
                  SEQUENCE {                  -- key derivation function
```

```
                     id-kdf-kdf2,                                -- KDF2
                     SEQUENCE {                     -- KDF2-HashFunction
                       id-sha256   -- SHA-256; no parameters (preferred)
                     },
                  16                              -- KEK length in bytes
                  },
            SEQUENCE {                   -- data encapsulation mechanism
               id-aes128-Wrap             -- AES-128 Wrap; no parameters
            }
         }
      }
```

This AlgorithmIdentifier value has the following DER encoding:

```
      30 4f
         06 07 28 81 8c 71 02 01 02             -- id-ac-generic-hybrid
         30 44
            30 25
               06 07 28 81 8c 71 02 02 04                 -- id-kem-rsa
               30 1a
                  30 16
                     06 07 28 81 8c 71 02 05 02          -- id-kdf-kdf2
                     30 0b
                        06 09 60 86 48 01 65 03 04 02 01   -- id-sha256
                  02 10                                     -- 16 bytes
            30 0b
               06 09 60 86 48 01 65 03 04 01 05       -- id-aes128-Wrap
```

The DER encodings for other typical sets of underlying components are as follows:

* KDF2 based on SHA-384, AES Key Wrap with a 192-bit KEK

```
            30 4f 06 07 28 81 8c 71 02 01 02 30 44 30 25 06
            07 28 81 8c 71 02 02 04 30 1a 30 16 06 07 28 81
            8c 71 02 05 02 30 0b 06 09 60 86 48 01 65 03 04
            02 02 02 18 30 0b 06 09 60 86 48 01 65 03 04 01
            19
```

* KDF2 based on SHA-512, AES Key Wrap with a 256-bit KEK

```
            30 4f 06 07 28 81 8c 71 02 01 02 30 44 30 25 06
            07 28 81 8c 71 02 02 04 30 1a 30 16 06 07 28 81
            8c 71 02 05 02 30 0b 06 09 60 86 48 01 65 03 04
            02 03 02 20 30 0b 06 09 60 86 48 01 65 03 04 01
            2d
```

* KDF2 based on SHA-1, Triple-DES Key Wrap with a 128-bit KEK

(two-key triple-DES)

```
30 4f 06 07 28 81 8c 71 02 01 02 30 44 30 21 06
07 28 81 8c 71 02 02 04 30 16 30 12 06 07 28 81
```

```
            8c 71 02 05 02 30 07 06 05 2b 0e 03 02 1a 02 10
            30 0f 06 0b 2a 86 48 86 f7 0d 01 09 10 03 06 05
            00
```

* KDF2 based on SHA-224, Triple-DES Key Wrap with a 192-bit KEK (three-key triple-DES)

  [[to be defined, awaiting OID for SHA-224]]

Full Copyright Statement