S/MIME Working Group   James Randall, Randall Consulting
Internet Draft   Burt Kaliski, EMC
                                                     John Brainard, RSA
                                                      Sean Turner, IECA
Expires: January 7, 2010   Category: Standards
                                                          July 7, 2009

                 Use of the RSA-KEM Key Transport Algorithm in CMS
                      <draft-ietf-smime-cms-rsa-kem-07.txt>

        Status of this Memo

        Internet-Drafts are working documents of the Internet Engineering
        Task Force (IETF), its areas, and its working groups. Note that
        other groups may also distribute working documents as Internet-
        Drafts.

        Internet-Drafts are draft documents valid for a maximum of six months
        and may be updated, replaced, or obsoleted by other documents at any
        time. It is inappropriate to use Internet-Drafts as reference
        material or to cite them other than as "work in progress."

        The list of current Internet-Drafts can be accessed at
        http://www.ietf.org/ietf/1id-abstracts.txt

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

        Internet-Drafts are working documents of the Internet Engineering
        Task Force (IETF), its areas, and its working groups. Note that other
        groups may also distribute working documents as Internet-Drafts.

This Internet-Draft will expire on January 6, 2010.

Copyright Notice

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

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). The ASN.1 syntax is aligned with ANS X9.44 and
ISO/IEC 18033-2.

Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in 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:

$$c = m^e \bmod n$$

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:

$$c = z^e \bmod n$$

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 (a) the input to the underlying RSA operation is effectively a random integer between 0 and n-1, where n is the RSA modulus, so it does not have any structure that could be exploited by an adversary, and (b) the input is independent of the keying data so the result of the RSA decryption operation is not directly available to an adversary. As a result, the algorithm enjoys a "tight" security proof in the random oracle model. (In other padding schemes, such as PKCS #1 v1.5, the input has structure and/or depends on the keying data, and the provable security assurances are not as strong.) The approach 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 as well as in ANS-X9.44 and ISO/IEC 18033-2. It has also been recommended by the NESSIE project [NESSIE].

    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:

o For the key derivation function, KDF2 (see [ANS-X9.44]) (see [IEEE-P1363a]) based on SHA-1 (see [FIPS-180-2]) (this function is also specified as the key derivation function in [ANS-X9.63]), and KDF3 (see [IEEE-P1363a]) based on SHA-256 (see [FIPS-180-2]).

o 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 KDF3 based on SHA-1 and KDF2 based on SHA-256. The Camellia key wrap algorithm (see [CAMELLIA]) SHOULD be supported, and, if 3DES is supported as a content-encryption cipher, then the Triple-DES Key Wrap (see [3DES-WRAP]) SHOULD also be supported.

It MAY support other underlying components. When AES or Camellia are used the data block size is 128 bits while the key size can be 128, 192, or 256 bits while Triple DES requires a data block size of 64 bits and a key size of 112 or 168 bits.

2.2. RecipientInfo Conventions

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

o keyEncryptionAlgorithm.algorithm MUST be id-rsa-kem see Appendix
   B)

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

o encryptedKey MUST be the encrypted keying data output by the
   algorithm, where the keying data is the content-encryption
   key.(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]. The fact that the user
will accept RSA-KEM with this public key is not indicated by the use
of this identifier. This may be signed by the use of the appropriate
SMIME Capabilities either in a message or in the certificate.

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-rsa-kem object identifier
(see Appendix B). The parameters are absent.

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-rsa-kem 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 or for authentication such as in signatures. 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.

2.4. SMIMECapabilities Attribute Conventions

RFC 3851 [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-rsa-kem 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-FIPS PUB 800-57]. For brevity, the first three levels are mentioned here:

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

o 112-bit security. The RSA key size SHOULD be at least 2048 bits, the hash function underlying the KDF SHOULD be SHA-224 or above, and the symmetric key-wrapping scheme SHOULD be AES Key Wrap, Triple-DES Key Wrap, or Camellia Key Wrap.

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

Note that the AES Key Wrap or Camellia Key Wrap MAY be used at all three of these levels; the use of AES or Camellia 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 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.

    [CAMELLIA]   Kato, A., Moriai, S., and Kanda, M.: Use of the
                        Camellia Encryption Algorithm in Cryptographic
                        Message Syntax. RFC 3657. December 2005.

    [CMS]   Housley, R. Cryptographic Message Syntax. RFC
                        3852. July 2004.

    [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 3851. July 2004.

    [PROFILE]   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.

    [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. American National
                        Standard X9.44: Public Key Cryptography for the
                        Financial Services Industry -- Key Establishment
                        Using Integer Factorization Cryptography. 2007

    [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 Std 1363a-2004: Standard Specifications for
                       Public Key Cryptography: Additional Techniques.
                       IEEE, 2004.

   [ISO-IEC-18033-2]   ISO/IEC 18033-2:2005 Information technology --
                       Security techniques -- Encryption algorithms --
                       Part 2: Asymmetric Ciphers. ISO/IEC, 2005.

   [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.
                       August 2005. Available via:
                       http://csrc.nist.gov/publications/index.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
                       4086. June 2005.

   [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/.

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:

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

   o 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. (Both the AES and Camellia Key Wraps, for instance,
   require 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
   ISO/IEC 18033-2 [ISO-IEC-18033-2], and is the same as employed in 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 ANS X9.44.

B.1 RSA-KEM Key Transport Algorithm

   The object identifier for the RSA-KEM Key Transport Algorithm is id-
   rsa-kem, which is defined in the draft as:

       id-rsa-kem OID ::= {
         iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
         pkcs-9(9) smime(16) alg(3) TBA
       }

   When id-rsa-kem is used in an AlgorithmIdentifier, the parameters
   MUST employ the GenericHybridParameters syntax. The parameters MUST
   be absent when used in the subjectPublicKeyInfo field The syntax for
   GenericHybridParameters is as follows:

      GenericHybridParameters ::= {
         kem KeyEncapsulationMechanism,

```
      dem   DataEncapsulationMechanism
   }
```

The fields of type GenericHybridParameters have the following
meanings:

o   kem identifies the underlying key encapsulation mechanism. For the
    RSA-KEM Key Transport Algorithm, the scheme is RSA-KEM from
    ISO/IEC 18033-2.

    The object identifier for RSA-KEM (as a key encapsulation
    mechanism) is id-kem-rsa, which is defined in 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 ANS X9.44, it
             MUST be KDF2 or KDF3. However, other key derivation
             functions MAY be used with CMS. Please see B.2.1 for
             the syntax for KDF2 and KDF3.

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

                KDFAlgorithms ALGORITHM ::= {
                   kdf2 | kdf3,
                   ...  -- 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)
```

o dem identifies the underlying data encapsulation mechanism. For
   alignment with 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, Triple-DES, and Camellia Key Wraps.

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

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

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

    Camellia-KeyWrappingSchemes ALGORITHM ::= {
        Camellia128-Wrap | Camellia192-Wrap | Camellia256-Wrap
    }
```

   NOTE: The generic hybrid cipher in 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. ISO/IEC 18033-2
   allows only three specific 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

B.2.1 Key Derivation Functions

The object identifier for KDF2 (see [ANS X9.44]) is:

```
    id-kdf-kdf2 OID ::= { x9-44-components kdf2(1) }
```

The associated parameters identify the underlying hash function. For
alignment with ANS X9.44, the hash function MUST be an ASC X9-
approved hash function. 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-224, SHA-256, SHA-384 and SHA-512 are

```
        id-sha224 OID ::= { nistAlgorithm hashAlgs(2) sha224(4) }
        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 } -- ""
```

    The object identifier for KDF3 (see [ANS X9.44]) is:

```
        id-kdf-kdf3 OID ::= { x9-44-components kdf3(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.

```
kdf3 ALGORITHM ::= { OID id-kdf-kdf3  PARMS KDF3-HashFunction }

KDF3-HashFunction ::= AlgorithmIdentifier { KDF3-HashFunctions }

KDF3-HashFunctions ALGORITHM ::= {
   X9-HashFunctions,
   ...  -- implementations may define other methods
}
```

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.

The object identifiers for the Camellia Key Wrap depend on the size
of the key encrypting key. There are three object identifiers:

```
    id-camellia128-Wrap OBJECT IDENTIFIER ::=
       { iso(1) member-body(2) 392 200011 61 security(1)
         algorithm(1) key-wrap-algorithm(3)
         camellia128-wrap(2) }

    id-camellia192-Wrap OBJECT IDENTIFIER ::=
       { iso(1) member-body(2) 392 200011 61 security(1)
         algorithm(1) key-wrap-algorithm(3)
         camellia192-wrap(3) }

    id-camellia256-Wrap OBJECT IDENTIFIER ::=
       { iso(1) member-body(2) 392 200011 61 security(1)
         algorithm(1) key-wrap-algorithm(3)
         camellia256-wrap(4) }
```

  These object identifiers have no associated parameters.

```
    camellia128-Wrap ALGORITHM ::= { OID id-camellia128-Wrap }
    camellia192-Wrap ALGORITHM ::= { OID id-camellia192-Wrap }
    camellia256-Wrap ALGORITHM ::= { OID id-camellia256-Wrap }
```

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) }

  DEFINITIONS ::=

  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

 id-rsa-kem OID ::= {
   iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
   pkcs-9(9) smime(16) alg(3) TBA
 }

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

KeyEncapsulationMechanism ::= AlgorithmIdentifier {{KEMAlgorithms}}
```

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

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

KeyDerivationFunction ::= AlgorithmIdentifier {{KDFAlgorithms}}

KDFAlgorithms ALGORITHM ::= {
   kdf2 | kdf3,
   ...  -- implementations may define other methods
}

KeyLength ::= INTEGER (1..MAX)

DataEncapsulationMechanism ::=
   AlgorithmIdentifier {{DEMAlgorithms}}

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

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

X9-SymmetricKeyWrappingScheme ::=
   AlgorithmIdentifier {{ X9-SymmetricKeyWrappingSchemes }}

Camellia-KeyWrappingSchemes ALGORITHM ::= {
   camellia128-Wrap | camellia192-Wrap | camellia256-Wrap,
   ... -- allows for future expansion
}

Camellia-KeyWrappingScheme ::=
   AlgorithmIdentifier {{ Camellia-KeyWrappingSchemes }}

-- Key Derivation Functions

id-kdf-kdf2 OID ::= { x9-44-components kdf2(1) }
```

```
      -- Base arc

      x9-44 OID ::= {
         iso(1) identified-organization(3) tc68(133) country(16) x9(840)
         x9Standards(9) x9-44(44)
      }

      x9-44-components OID ::= { x9-44 components(1) }

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

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

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

      -- id-kdf-kdf3 OID ::= { x9-44-components kdf3(2) } kdf3 ALGORITHM
   ::= { OID id-kdf-kdf2  PARMS KDF3-HashFunction } KDF3-HashFunction
   ::= AlgorithmIdentifier {{ KDF3-HashFunctions }}

      KDF3-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-sha224 OID ::= { nistAlgorithm hashAlgs(2) sha256(4) }
      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 }

    id-camellia128-Wrap OBJECT IDENTIFIER ::=
       { iso(1) member-body(2) 392 200011 61 security(1)
         algorithm(1) key-wrap-algorithm(3)
         camellia128-wrap(2) }

    id-camellia192-Wrap OBJECT IDENTIFIER ::=
       { iso(1) member-body(2) 392 200011 61 security(1)
         algorithm(1) key-wrap-algorithm(3)
         camellia192-wrap(3) }

    id-camellia256-Wrap OBJECT IDENTIFIER ::=
       { iso(1) member-body(2) 392 200011 61 security(1)
         algorithm(1) key-wrap-algorithm(3)
         camellia256-wrap(4) }

    camellia128-Wrap ALGORITHM ::= { OID id-camellia128-Wrap }
    camellia192-Wrap ALGORITHM ::= { OID id-camellia192-Wrap }
    camellia256-Wrap ALGORITHM ::= { OID id-camellia256-Wrap }

  END
```

B.4 **Examples**

As an example, if the key derivation function is KDF2 based onSHA-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-rsa-kem,                                  -- RSA-KEM 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 (??
indicates the algorithm number which is to be assigned):

```
30 53
   06 0b 2a 86 48 86 f7 0d 01 09 10 03 ??          -- id-rsa-kem
   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:

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

       30 46 06 0b 2a 86 48 86 f7 0d 01 09 10 03 ?? 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

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

       30 46 06 0b 2a 86 48 86 f7 0d 01 09 10 03 ?? 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

o KDF2 based on SHA-1, Triple-DES Key Wrap with a 128-bit KEK (two-
   key triple-DES)

       30 46 06 0b 2a 86 48 86 f7 0d 01 09 10 03 ?? 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

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.

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. We 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.

   Our thanks to Russ Housley as well for his guidance and
   encouragement. We also appreciate the helpful direction we've
   received from Blake Ramsdell and Jim Schaad in bringing this document
   to fruition. A special thanks to Magnus Nystrom for his assistance on
   Appendix B. Thanks also to Bob Griffin and John Linn for both
   editorial direction and procedural guidance.

Author Information

   James Randall

   Randall Consulting
   55 Sandpiper Drive
   Dover, NH 03820
   USA

   Email: jdrandall@comcast.net

   Burt Kaliski

   EMC
   176 South Street
   Hopkinton, MA 01748
   USA

   Email: kaliski_burt@emc.com

   John Brainard

   RSA, The Security Division of EMC
   174 Middlesex Turnpike
   Bedford, MA 01730
   USA

Email: jbrainard@rsa.com

Sean Turner

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

Email: turners@ieca.com