Stream:	Internet Engineering Task Force (IETF)
RFC:	9690
Obsoletes:	5990
Category:	Standards Track
Published:	January 2025
ISSN:	2070-1721

Authors:	R. Housley	S. Turner
	Vigil Security	*sn3rd*

RFC 9690
Use of the RSA-KEM Algorithm in the Cryptographic Message Syntax (CMS)

Abstract

The RSA Key Encapsulation Mechanism (RSA-KEM) algorithm is a one-pass (store-and-forward) cryptographic mechanism for an originator to securely send keying material to a recipient using the recipient's RSA public key. The RSA-KEM algorithm is specified in Clause 11.5 of ISO/IEC: 18033-2:2006. This document specifies the conventions for using the RSA-KEM algorithm as a standalone KEM algorithm and the conventions for using the RSA-KEM algorithm with the Cryptographic Message Syntax (CMS) using KEMRecipientInfo as specified in RFC 9629. This document obsoletes RFC 5990.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The RSA Key Encapsulation Mechanism (RSA-KEM) algorithm is a one-pass (store-and-forward) cryptographic mechanism for an originator to securely send keying material to a recipient using the recipient's RSA public key. The RSA-KEM algorithm is specified in Clause 11.5 of [ISO18033-2].

The RSA-KEM algorithm takes a different approach than other RSA key transport mechanisms [RFC8017] with the goal of providing higher security assurance while also satisfying the KEM interface. The RSA-KEM algorithm encrypts a random integer with the recipient's RSA public key and derives a shared secret from the random integer. The originator and recipient can derive a symmetric key from the shared secret. For example, a key-encryption key (KEK) can be derived from the shared secret to wrap a content-encryption key (CEK).

In the Cryptographic Message Syntax (CMS) [RFC5652] using KEMRecipientInfo [RFC9629], the shared-secret value is input to a key derivation function (KDF) to compute a key-encryption key and wrap a symmetric content-encryption key with the key-encryption key. In this way, the originator and the recipient end up with the same content-encryption key.

For completeness, a specification of the RSA-KEM algorithm is given in Appendix A of this document. ASN.1 syntax is given in Appendix B.

1.1. RSA-KEM Algorithm Rationale

The RSA-KEM algorithm provides higher security assurance than other variants of the RSA cryptosystem for two reasons. First, the input to the underlying RSA operation is a string-encoded 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. Second, the input is independent of the keying material, so the result of the RSA decryption operation is not directly available to an adversary. As a result, the RSA-KEM algorithm enjoys a "tight" security proof in the random oracle model. (In other padding schemes, such as PKCS #1 v1.5 [RFC8017], the input has structure and depends on the keying material. Additionally, 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 material. Another benefit is that the length of the keying material is determined by the symmetric algorithms, not the size of the RSA modulus.

1.2. RSA-KEM Algorithm Summary

All KEM algorithms provide three functions: KeyGen(), Encapsulate(), and Decapsulate().

The following summarizes these three functions for RSA-KEM:

KeyGen() -> (pk, sk):
: Generate the public key (pk) and a private key (sk) as described in Section 3 of [RFC8017].

Encapsulate(pk) -> (ct, SS):
: Given the recipient's public key (pk), produce a ciphertext (ct) to be passed to the recipient and a shared secret (SS) for use by the originator as follows:

1. Generate a random integer z between 0 and n-1.
2. Encrypt the integer z with the recipient's RSA public key to obtain the ciphertext:

```
        ct = z^e mod n
```

3. Derive a shared secret from the integer z using a Key Derivation Function (KDF):

```
        SS = KDF(Z, ssLen)
```

4. The ciphertext and the shared secret are returned by the function. The originator sends the ciphertext to the recipient.

Decapsulate(sk, ct) -> SS:
: Given the private key (sk) and the ciphertext (ct), produce the shared secret (SS) for the recipient as follows:

1. Decrypt the ciphertext with the recipient's RSA private key to obtain the random integer z:

```
        z = ct^d mod n
```

2. Derive a shared secret from the integer z:

```
        SS = KDF(Z, ssLen)
```

3. The shared secret is returned by the function.

1.3. CMS KEMRecipientInfo Processing Summary

To support the RSA-KEM algorithm, the CMS originator **MUST** implement Encapsulate().

Given a content-encryption key CEK, the RSA-KEM algorithm processing by the originator to produce the values that are carried in the CMS KEMRecipientInfo can be summarized as follows:

1. Obtain the shared secret using the Encapsulate() function of the RSA-KEM algorithm and the recipient's RSA public key:

```
         (ct, SS) = Encapsulate(pk)
```

2. Derive a key-encryption key KEK from the shared secret:

```
         KEK = KDF(SS, kekLength, otherInfo)
```

3. Wrap the CEK with the KEK to obtain wrapped keying material WK:

```
         WK = WRAP(KEK, CEK)
```

4. The originator sends the ciphertext and WK to the recipient in the CMS KEMRecipientInfo structure.

To support the RSA-KEM algorithm, the CMS recipient **MUST** implement Decapsulate().

The RSA-KEM algorithm recipient processing of the values obtained from the KEMRecipientInfo structure is summarized as follows:

1. Obtain the shared secret using the Decapsulate() function of the RSA-KEM algorithm and the recipient's RSA private key:

```
         SS = Decapsulate(sk, ct)
```

2. Derive a key-encryption key KEK from the shared secret:

```
         KEK = KDF(SS, kekLength, otherInfo)
```

3. Unwrap the WK with the KEK to obtain the content-encryption key CEK:

```
         CEK = UNWRAP(KEK, WK)
```

Note that the KDF used to process the KEMRecipientInfo structure **MAY** be different from the KDF used to derive the shared secret in the RSA-KEM algorithm.

1.4. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.5. ASN.1

CMS values are generated using ASN.1 [X.680], which uses the Basic Encoding Rules (BER) and the Distinguished Encoding Rules (DER) [X.690].

1.6. Changes Since RFC 5990

RFC 5990 [RFC5990] specified the conventions for using the RSA-KEM algorithm in CMS as a key transport algorithm. That is, it used KeyTransRecipientInfo [RFC5652] for each recipient. Since the publication of RFC 5990, a new KEMRecipientInfo structure [RFC9629] has been defined to support KEM algorithms. When the id-rsa-kem algorithm identifier appears in the SubjectPublicKeyInfo field of a certificate, the complex parameter structure defined in RFC 5990 can be omitted; however, the parameters are allowed for backward compatibility. Also, to avoid visual confusion with id-kem-rsa, id-rsa-kem-spki is introduced as an alias for id-rsa-kem.

RFC 5990 used EK as the EncryptedKey, which is the concatenation of the ciphertext C and the wrapped key WK, EK = (C || WK). The use of EK was necessary to align with the KeyTransRecipientInfo structure. In this document, the ciphertext and the wrapped key are sent in separate fields of the KEMRecipientInfo structure. In particular, the ciphertext is carried in the kemct field, and the wrapped key is carried in the encryptedKey field. See Appendix A for details about the computation of the ciphertext.

RFC 5990 included support for Camellia and Triple-DES block ciphers; discussion of these block ciphers does not appear in this document, but the algorithm identifiers remain in the ASN.1 module (see Appendix B.2).

RFC 5990 included support for SHA-1 hash function; discussion of this hash function does not appear this document, but the algorithm identifier remains in the ASN.1 module (see Appendix B.2).

RFC 5990 required support for the KDF3 key derivation function [ANS-X9.44]; this document continues to require support for the KDF3 key derivation function, but it requires support for SHA-256 [SHS] as the hash function.

RFC 5990 recommended support for alternatives to KDF3 and AES-Wrap-128; this document simply states that other key derivation functions and other key-encryption algorithms **MAY** be supported.

RFC 5990 supported the future definition of additional KEM algorithms that use RSA; this document supports only one, and it is identified by the id-kem-rsa object identifier.

RFC 5990 included an ASN.1 module; this document provides an alternative ASN.1 module that follows the conventions established in [RFC5911], [RFC5912], and [RFC6268]. The new ASN.1 module (Appendix B.2) produces the same bits-on-the-wire as the one in RFC 5990.

2. Use of the RSA-KEM Algorithm in CMS

The RSA-KEM algorithm **MAY** be employed for one or more recipients in the CMS enveloped-data content type [RFC5652], the CMS authenticated-data content type [RFC5652], or the CMS authenticated-enveloped-data content type [RFC5083]. In each case, the KEMRecipientInfo [RFC9629] is used with the RSA-KEM algorithm to securely transfer the content-encryption key from the originator to the recipient.

2.1. Mandatory To Implement

A CMS implementation that supports the RSA-KEM algorithm **MUST** support at least the following underlying components:

- For the key derivation function, an implementation **MUST** support KDF3 [ANS-X9.44] with SHA-256 [SHS].
- For key-wrapping, an implementation **MUST** support the AES-Wrap-128 [RFC3394] key-encryption algorithm.

An implementation **MAY** also support other key derivation functions and other key-encryption algorithms.

2.2. RecipientInfo Conventions

When the RSA-KEM algorithm is employed for a recipient, the RecipientInfo alternative for that recipient **MUST** be OtherRecipientInfo using the KEMRecipientInfo structure [RFC9629]. The fields of the KEMRecipientInfo **MUST** have the following values:

- version is the syntax version number; it **MUST** be 0.
- rid identifies the recipient's certificate or public key.
- kem identifies the KEM algorithm; it **MUST** contain id-kem-rsa.
- kemct is the ciphertext produced for this recipient; it contains C from steps 1 and 2 of Originator's Operations in Appendix A.
- kdf identifies the key derivation function (KDF). Note that the KDF used for CMS RecipientInfo process **MAY** be different than the KDF used within the RSA-KEM algorithm.
- kekLength is the size of the key-encryption key in octets.
- ukm is an optional random input to the key derivation function.
- wrap identifies a key-encryption algorithm used to encrypt the keying material.

- encryptedKey is the result of encrypting the keying material with the key-encryption key. When used with the CMS enveloped-data content type [RFC5652], the keying material is a content-encryption key. When used with the CMS authenticated-data content type [RFC5652], the keying material is a message-authentication key. When used with the CMS authenticated-enveloped-data content type [RFC5083], the keying material is a content-authenticated-encryption key (CAEK).

NOTE: For backward compatibility, implementations **MAY** also support the RSA-KEM Key Transport algorithm, identified by id-rsa-kem-spki, which uses KeyTransRecipientInfo as specified in [RFC5990].

2.3. Certificate Conventions

The conventions specified in this section augment RFC 5280 [RFC5280].

A recipient who employs the RSA-KEM algorithm **MAY** identify the public key in a certificate by the same AlgorithmIdentifier as for the PKCS #1 v1.5 algorithm, that is, using the rsaEncryption object identifier [RFC8017]. The fact that the recipient will accept RSA-KEM with this public key is not indicated by the use of this object identifier. The willingness to accept the RSA-KEM algorithm **MAY** be signaled by the use of the SMIMECapabilities Attribute as specified in Section 2.5.2 of [RFC8551] or the SMIMECapabilities certificate extension as specified in [RFC4262].

If the recipient wishes only to employ the RSA-KEM algorithm with a given public key, the recipient **MUST** identify the public key in the certificate using the id-rsa-kem-spki object identifier; see Appendix B. The use of the id-rsa-kem-spki object identifier allows certificates that were issued to be compatible with RSA-KEM Key Transport to also be used with this specification. When the id-rsa-kem-spki object identifier appears in the SubjectPublicKeyInfo algorithm field of the certificate, the parameters field from AlgorithmIdentifier **SHOULD** be absent. That is, the AlgorithmIdentifier **SHOULD** be a SEQUENCE of one component, the id-rsa-kem-spki object identifier. With absent parameters, the KDF3 key derivation function [ANS-X9.44] with SHA-256 [SHS] are used to derive the shared secret.

When the AlgorithmIdentifier parameters are present, the GenericHybridParameters **MUST** be used. Within the kem element, the algorithm identifier **MUST** be set to id-kem-rsa, and RsaKemParameters **MUST** be included. As described in Section 2.4, the GenericHybridParameters constrain the values that can be used with the RSA public key for the kdf, kekLength, and wrap fields of the KEMRecipientInfo structure.

Regardless of the AlgorithmIdentifier used, the RSA public key **MUST** be carried in the subjectPublicKey BIT STRING within the SubjectPublicKeyInfo field of the certificate using the RSAPublicKey type defined in [RFC8017].

The intended application for the public key **MAY** be indicated in the key usage certificate extension as specified in Section 4.2.1.3 of [RFC5280]. If the keyUsage extension is present in a certificate that conveys an RSA public key with the id-rsa-kem-spki object identifier as discussed above, then the key usage extension **MUST** contain only the following value:

  keyEncipherment

Other keyUsage extension values **MUST NOT** be present. That is, a public key intended to be employed only with the RSA-KEM algorithm **MUST NOT** also be employed for data encryption or for digital 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

Section 2.5.2 of [RFC8551] defines the SMIMECapabilities attribute to announce a partial list of algorithms that an S/MIME implementation can support. When constructing a CMS signed-data content type [RFC5652], a compliant implementation **MAY** include the SMIMECapabilities attribute that announces support for the RSA-KEM algorithm.

The SMIMECapability SEQUENCE representing the RSA-KEM algorithm **MUST** include the id-rsa-kem-spki object identifier in the capabilityID field; see Appendix B for the object identifier value and Appendix C for examples. When the id-rsa-kem-spki object identifier appears in the capabilityID field and the parameters are present, then the parameters field **MUST** use the GenericHybridParameters type.

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

The fields of the GenericHybridParameters type have the following meanings:

- kem is an AlgorithmIdentifer. The algorithm field **MUST** be set to id-kem-rsa, and the parameters field **MUST** be RsaKemParameters, which is a SEQUENCE of an AlgorithmIdentifier that identifies the supported key derivation function and a positive INTEGER that identifies the length of the key-encryption key in octets.
- dem is an AlgorithmIdentifier. The algorithm field **MUST** be present, and it identifies the key-encryption algorithm. The parameters are optional. If the GenericHybridParameters are present, then the provided dem value **MUST** be used in the wrap field of KEMRecipientInfo.

If the GenericHybridParameters are present, then the provided kem value **MUST** be used as the key derivation function in the kdf field of KEMRecipientInfo and the provided key length **MUST** be used in the kekLength of KEMRecipientInfo.

3. Security Considerations

The RSA-KEM algorithm should be considered as a replacement for the key transport portion of the widely implemented PKCS #1 v1.5 [RFC8017] for new applications that use CMS to avoid potential vulnerabilities to chosen-ciphertext attacks and gain a tighter security proof. However, the RSA-KEM algorithm has the disadvantage of slightly longer encrypted keying material. With PKCS #1 v1.5, the originator encrypts the key-encryption key directly with the recipient's RSA public key. With the RSA-KEM, the key-encryption key is encrypted separately.

The security of the RSA-KEM algorithm can be shown to be tightly related to the difficulty of either solving the RSA problem or breaking the underlying symmetric key-encryption algorithm if the underlying key derivation function is modeled as a random oracle, assuming that the symmetric key-encryption algorithm 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 need to be selected consistent with the desired security level. Several security levels have been identified in the NIST SP 800-57 Part 1 [NISTSP800-57pt1r5]. For example, one way to achieve 128-bit security, the RSA key size would be at least 3072 bits, the key derivation function would be SHA-256, and the symmetric key-encryption algorithm would be AES Key Wrap with a 128-bit key.

Implementations **MUST** protect the RSA private key, the key-encryption key, the content-encryption key, message-authentication key, and the content-authenticated-encryption key. Disclosure of the RSA private key could result in the compromise of all messages protected with that key. Disclosure of the key-encryption key, the content-encryption key, or the content-authenticated-encryption key could result in compromise of the associated encrypted content. Disclosure of the key-encryption key, the message-authentication key, or the content-authenticated-encryption key could allow modification of the associated authenticated content.

Additional considerations related to key management may be found in [NISTSP800-57pt1r5].

The security of the RSA-KEM algorithm depends on a quality random number generator. For further discussion on random number generation, see [RFC4086].

The RSA-KEM algorithm does not use an explicit padding scheme. Instead, an encoded random value (z) between zero and the RSA modulus minus one (n-1) is directly encrypted with the recipient's RSA public key. The IntegerToString(z, nLen) encoding produces a string that is the full length of the RSA modulus. In addition, the random value is passed through a KDF to reduce possible harm from a poorly implemented random number source or a maliciously chosen random value (z). Implementations **MUST NOT** use z directly for any purpose.

As long as a fresh random integer z is chosen as part of each invocation of the Encapsulate() function, RSA-KEM does not degrade as the number of ciphertexts increases. Since RSA encryption provides a bijective map, a collision in the KDF is the only way that RSA-KEM can produce more than one ciphertext that encapsulates the same shared secret.

The RSA-KEM algorithm provides a fixed-length ciphertext. The recipient **MUST** check that the received byte string is the expected length and the length corresponds to an integer in the expected range prior to attempting decryption with their RSA private key as described in Steps 1 and 2 of Appendix A.2.

Implementations **SHOULD NOT** reveal information about intermediate values or calculations, whether by timing or other "side channels"; 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 to an opponent.

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. RSA public keys have often been employed for multiple purposes such as key transport and digital signature without any known bad interactions; however, 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 algorithm **SHOULD NOT** also be used for digital signatures. Indeed, the Accredited Standards Committee X9 (ASC X9) requires such a separation between key pairs used for key establishment and key pairs used for digital signature [ANS-X9.44]. Continuing this principle of key separation, a key pair used for the RSA-KEM algorithm **SHOULD NOT** be used with other key establishment schemes, or for data encryption, or with more than one set of underlying algorithm components.

It is acceptable to use the same RSA key pair for RSA-KEM Key Transport as specified in [RFC5990] and this specification. This is acceptable because the operations involving the RSA public key and the RSA private key are identical in the two specifications.

Parties can gain assurance that implementations are correct through formal implementation validation, such as the NIST Cryptographic Module Validation Program (CMVP) [CMVP].

4. IANA Considerations

For the ASN.1 Module in Appendix B.2, IANA has assigned an object identifier (OID) for the module identifier. The OID for the module has been allocated in the "SMI Security for S/MIME Module Identifier" registry (1.2.840.113549.1.9.16.0), and the Description for the new OID has been set to "id-mod-cms-rsa-kem-2023".

IANA has updated the id-alg-rsa-kem entry in the "SMI Security for S/MIME Algorithms (1.2.840.113549.1.9.16.3)" repository to refer to this document. In addition, IANA has added the following note to the registry:

Value 14, "id-alg-rsa-kem," is also referred to as "id-rsa-kem-spki."

5. References

5.1. Normative References

[ANS-X9.44] American National Standards Institute, "Public Key Cryptography for the Financial Services Industry -- Key Establishment Using Integer Factorization Cryptography", ANSI X9.44-2007 (R2017), 2007, <https://webstore.ansi.org/standards/ascx9/ansix9442007r2017>.

[ISO18033-2] ISO/IEC, "Information technology -- Security techniques -- Encryption algorithms -- Part 2: Asymmetric ciphers", ISO/IEC 18033-2:2006, 2006, <https://www.iso.org/standard/37971.html>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, DOI 10.17487/RFC3394, September 2002, <https://www.rfc-editor.org/info/rfc3394>.

[RFC5083] Housley, R., "Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type", RFC 5083, DOI 10.17487/RFC5083, November 2007, <https://www.rfc-editor.org/info/rfc5083>.

[RFC5280] 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, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5911] Hoffman, P. and J. Schaad, "New ASN.1 Modules for Cryptographic Message Syntax (CMS) and S/MIME", RFC 5911, DOI 10.17487/RFC5911, June 2010, <https://www.rfc-editor.org/info/rfc5911>.

[RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <https://www.rfc-editor.org/info/rfc5912>.

[RFC6268] Schaad, J. and S. Turner, "Additional New ASN.1 Modules for the Cryptographic Message Syntax (CMS) and the Public Key Infrastructure Using X.509 (PKIX)", RFC 6268, DOI 10.17487/RFC6268, July 2011, <https://www.rfc-editor.org/info/rfc6268>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <https://www.rfc-editor.org/info/rfc8551>.

[RFC9629] Housley, R., Gray, J., and T. Okubo, "Using Key Encapsulation Mechanism (KEM) Algorithms in the Cryptographic Message Syntax (CMS)", RFC 9629, DOI 10.17487/RFC9629, August 2024, <https://www.rfc-editor.org/info/rfc9629>.

[SHS] National Institute of Standards and Technology, "Secure Hash Standard", NIST FIPS PUB 180-4, DOI 10.6028/NIST.FIPS.180-4, July 2015, <https://doi.org/10.6028/NIST.FIPS.180-4>.

[X.680] ITU-T, "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, ISO/IEC 8824-1:2021, February 2021, <https://www.itu.int/rec/T-REC-X.680>.

[X.690] ITU-T, "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1:2021, February 2021, <https://www.itu.int/rec/T-REC-X.690>.

5.2. Informative References

[CMVP] National Institute of Standards and Technology, "Cryptographic Module Validation Program", 2016, <https://csrc.nist.gov/projects/cryptographic-module-validation-program>.

[NISTSP800-57pt1r5] Barker, E., "Recommendation for Key Management: Part 1 - General", NIST SP 800-57, Part 1, Revision 5, DOI 10.6028/nist.sp.800-57pt1r5, May 2020, <https://doi.org/10.6028/nist.sp.800-57pt1r5>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC4262] Santesson, S., "X.509 Certificate Extension for Secure/Multipurpose Internet Mail Extensions (S/MIME) Capabilities", RFC 4262, DOI 10.17487/RFC4262, December 2005, <https://www.rfc-editor.org/info/rfc4262>.

[RFC5990] Randall, J., Kaliski, B., Brainard, J., and S. Turner, "Use of the RSA-KEM Key Transport Algorithm in the Cryptographic Message Syntax (CMS)", RFC 5990, DOI 10.17487/RFC5990, September 2010, <https://www.rfc-editor.org/info/rfc5990>.

[RFC6194] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <https://www.rfc-editor.org/info/rfc6194>.

[SHOUP] Shoup, V., "A Proposal for an ISO Standard for Public Key Encryption", Cryptology ePrint Archive Paper 2001/112, 2001, <https://eprint.iacr.org/2001/112>.

Appendix A. RSA-KEM Algorithm

The RSA-KEM algorithm is a one-pass (store-and-forward) cryptographic mechanism for an originator to securely send keying material to a recipient using the recipient's RSA public key.

With the RSA-KEM algorithm, an originator encrypts a random integer (z) with the recipient's RSA public key to produce a ciphertext (ct), and the originator derives a shared secret (SS) from the random integer (z). The originator then sends the ciphertext (ct) to the recipient. The recipient decrypts the ciphertext (ct) using their private key to recover the random integer (z), and the recipient derives a shared secret (SS) from the random integer (z). In this way, the originator and recipient obtain the same shared secret (SS).

The RSA-KEM algorithm depends on a key derivation function (KDF), which is used to derive the shared secret (SS). Many key derivation functions support the inclusion of other information in addition to the shared secret (SS) in the input to the function; however, no other information is included as an input to the KDF by the RSA-KEM algorithm.

A.1. Originator's Operations: RSA-KEM Encapsulate()

Let (n,e) be the recipient's RSA public key; see [RFC8017] for details.

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

The originator performs the following operations:

1. Generate a random integer z between 0 and n-1 (see NOTE below), 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 RSA 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

      ct = IntegerToString (c, nLen)
```

3. Derive a symmetric shared secret SS of length ssLen bytes (which **MUST** be the length of the key-encryption key) from the byte string Z using the underlying key derivation function:

```
      SS = KDF (Z, ssLen)
```

4. Output the shared secret SS and the ciphertext ct. Send the ciphertext ct to the recipient.

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

A.2. Recipient's Operations: RSA-KEM Decapsulate()

Let (n,d) be the recipient's RSA private key; see [RFC8017] for details, but other private key formats are allowed.

Let ct be the ciphertext received from the originator.

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

The recipient performs the following operations:

1. If the length of the encrypted keying material is less than nLen bytes, output "decryption error", and stop.
2. Convert the ciphertext ct to an integer c, most significant byte first (see NOTE below):

   ```
         c = StringToInteger (ct)
   ```

   If the integer c is not between 0 and n-1, output "decryption error", and stop.
3. Decrypt the integer c using the recipient's private key (n,d) to recover an integer z (see NOTE below):

   ```
         z = c^d mod n
   ```

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

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

5. Derive a shared secret SS of length ssLen bytes from the byte string Z using the key derivation function (see NOTE below):

   ```
         SS = KDF (Z, ssLen)
   ```

6. Output the shared secret SS.

NOTE: Implementations **SHOULD NOT** reveal information about the integer z, the string Z, or 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 algorithm is an extension of the syntax for the "generic hybrid cipher" in ANS X9.44 [ANS-X9.44].

The ASN.1 Module is unchanged from RFC 5990. The id-rsa-kem-spki object identifier is used in a backward compatible manner in certificates [RFC5280] and SMIMECapabilities [RFC8551]. Of course, the use of the id-kem-rsa object identifier in the new KEMRecipientInfo structure [RFC9629] was not yet defined at the time that RFC 5990 was written.

B.1. Underlying Components

Implementations that conform to this specification **MUST** support the KDF3 [ANS-X9.44] key derivation function using SHA-256 [SHS].

KDF2 [ANS-X9.44] and KDF3 are both key derivation functions based on a hash function. The only difference between KDF2 and KDF3 is the order of the components to be hashed.

```
   KDF2 calculates T as:   T = T || Hash (Z || D || otherInfo)

   KDF3 calculates T as:   T = T || Hash (D || Z || otherInfo)
```

The object identifier for KDF3 is:

```
   id-kdf-kdf3 OBJECT IDENTIFIER ::= { x9-44-components kdf3(2) }
```

The KDF3 parameters identify the underlying hash function. For alignment with ANS X9.44, the hash function **MUST** be an ASC X9-approved hash function. While the SHA-1 hash algorithm is included in the ASN.1 definitions, SHA-1 **MUST NOT** be used. SHA-1 is considered to be obsolete; see [RFC6194]. SHA-1 remains in the ASN.1 module for compatibility with RFC 5990. In addition, other hash functions **MAY** be used with CMS.

```
   kda-kdf3 KEY-DERIVATION ::= {
      IDENTIFIER id-kdf-kdf3
      PARAMS TYPE KDF3-HashFunction ARE required
      -- No S/MIME caps defined -- }

   KDF3-HashFunction ::=
      AlgorithmIdentifier { DIGEST-ALGORITHM, {KDF3-HashFunctions} }

   KDF3-HashFunctions DIGEST-ALGORITHM ::= { X9-HashFunctions, ... }

   X9-HashFunctions DIGEST-ALGORITHM ::= {
      mda-sha1 | mda-sha224 | mda-sha256 | mda-sha384 |
      mda-sha512, ... }
```

Implementations that conform to this specification **MUST** support the AES Key Wrap [RFC3394] key-encryption algorithm with a 128-bit key. There are three object identifiers for the AES Key Wrap, one for each permitted size of the key-encryption key. There are three object identifiers imported from [RFC5912], and none of these algorithm identifiers have associated parameters:

```
   kwa-aes128-wrap KEY-WRAP ::= {
       IDENTIFIER id-aes128-wrap
       PARAMS ARE absent
       SMIME-CAPS { IDENTIFIED BY id-aes128-wrap } }

   kwa-aes192-wrap KEY-WRAP ::= {
       IDENTIFIER id-aes192-wrap
       PARAMS ARE absent
       SMIME-CAPS { IDENTIFIED BY id-aes192-wrap } }

   kwa-aes256-wrap KEY-WRAP ::= {
       IDENTIFIER id-aes256-wrap
       PARAMS ARE absent
       SMIME-CAPS { IDENTIFIED BY id-aes256-wrap } }
```

B.2. ASN.1 Module

```
<CODE BEGINS>
CMS-RSA-KEM-2023
   { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
     pkcs-9(9) smime(16) modules(0) id-mod-cms-rsa-kem-2023(79) }

   DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS ALL

IMPORTS

  KEM-ALGORITHM
    FROM KEMAlgorithmInformation-2023  -- [I-D.ietf-lamps-cms-kemri]
       { iso(1) identified-organization(3) dod(6) internet(1)
         security(5) mechanisms(5) pkix(7) id-mod(0)
         id-mod-kemAlgorithmInformation-2023(109) }
  AlgorithmIdentifier{}, PUBLIC-KEY, DIGEST-ALGORITHM,
  KEY-DERIVATION, KEY-WRAP, SMIME-CAPS
    FROM AlgorithmInformation-2009  -- [RFC5912]
      { iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0)
        id-mod-algorithmInformation-02(58) }

  kwa-aes128-wrap, kwa-aes192-wrap, kwa-aes256-wrap
    FROM CMSAesRsaesOaep-2009  -- [RFC5911]
      { iso(1) member-body(2) us(840) rsadsi(113549)
        pkcs(1) pkcs-9(9) smime(16) modules(0)
        id-mod-cms-aes-02(38) }

  kwa-3DESWrap
    FROM CryptographicMessageSyntaxAlgorithms-2009  -- [RFC5911]
      { iso(1) member-body(2) us(840) rsadsi(113549)
        pkcs(1) pkcs-9(9) smime(16) modules(0)
        id-mod-cmsalg-2001-02(37) }

  id-camellia128-wrap, id-camellia192-wrap, id-camellia256-wrap
    FROM CamelliaEncryptionAlgorithmInCMS  -- [RFC3657]
      { iso(1) member-body(2) us(840) rsadsi(113549)
        pkcs(1) pkcs9(9) smime(16) modules(0)
        id-mod-cms-camellia(23) }

  mda-sha1, pk-rsa, RSAPublicKey
    FROM PKIXAlgs-2009  -- [RFC5912]
      { iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0)
        id-mod-pkix1-algorithms2008-02(56) }

  mda-sha224, mda-sha256, mda-sha384, mda-sha512
    FROM PKIX1-PSS-OAEP-Algorithms-2009  -- [RFC5912]
      { iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0)
        id-mod-pkix1-rsa-pkalgs-02(54) } ;


-- Useful types and definitions

OID ::= OBJECT IDENTIFIER  -- alias

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) }
```

```
-- X9.44 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) }

-- RSA-KEM Algorithm

id-rsa-kem OID ::= { iso(1) member-body(2) us(840) rsadsi(113549)
   pkcs(1) pkcs-9(9) smime(16) alg(3) 14 }

id-rsa-kem-spki OID ::= id-rsa-kem

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

KeyEncapsulationMechanism ::=
   AlgorithmIdentifier { KEM-ALGORITHM, {KEMAlgorithms} }

KEMAlgorithms KEM-ALGORITHM ::= { kema-kem-rsa | kema-rsa-kem, ... }

kema-rsa-kem KEM-ALGORITHM ::= {
   IDENTIFIER id-rsa-kem-spki
   PARAMS TYPE GenericHybridParameters ARE optional
   PUBLIC-KEYS { pk-rsa | pk-rsa-kem }
   UKM ARE optional
   SMIME-CAPS { TYPE GenericHybridParameters
      IDENTIFIED BY id-rsa-kem-spki } }

kema-kem-rsa KEM-ALGORITHM ::= {
   IDENTIFIER id-kem-rsa
   PARAMS TYPE RsaKemParameters ARE optional
   PUBLIC-KEYS { pk-rsa | pk-rsa-kem }
   UKM ARE optional
   SMIME-CAPS { TYPE GenericHybridParameters
      IDENTIFIED BY id-rsa-kem-spki } }

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

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

pk-rsa-kem PUBLIC-KEY ::= {
  IDENTIFIER id-rsa-kem-spki
  KEY RSAPublicKey
  PARAMS TYPE GenericHybridParameters ARE preferredAbsent
  -- Private key format is not specified here --
  CERT-KEY-USAGE {keyEncipherment} }

KeyDerivationFunction ::=
   AlgorithmIdentifier { KEY-DERIVATION, {KDFAlgorithms} }

KDFAlgorithms KEY-DERIVATION ::= { kda-kdf2 | kda-kdf3, ... }
```

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

DataEncapsulationMechanism ::=
   AlgorithmIdentifier { KEY-WRAP, {DEMAlgorithms} }

DEMAlgorithms KEY-WRAP ::= {
   X9-SymmetricKeyWrappingSchemes |
   Camellia-KeyWrappingSchemes, ... }

X9-SymmetricKeyWrappingSchemes KEY-WRAP ::= {
   kwa-aes128-wrap | kwa-aes192-wrap | kwa-aes256-wrap |
   kwa-3DESWrap, ... }

X9-SymmetricKeyWrappingScheme ::=
   AlgorithmIdentifier { KEY-WRAP, {X9-SymmetricKeyWrappingSchemes} }

Camellia-KeyWrappingSchemes KEY-WRAP ::= {
   kwa-camellia128-wrap | kwa-camellia192-wrap |
   kwa-camellia256-wrap, ... }

Camellia-KeyWrappingScheme ::=
   AlgorithmIdentifier { KEY-WRAP, {Camellia-KeyWrappingSchemes} }

kwa-camellia128-wrap KEY-WRAP ::= {
   IDENTIFIER id-camellia128-wrap
   PARAMS ARE absent
   SMIME-CAPS { IDENTIFIED BY id-camellia128-wrap } }

kwa-camellia192-wrap KEY-WRAP ::= {
   IDENTIFIER id-camellia192-wrap
   PARAMS ARE absent
   SMIME-CAPS { IDENTIFIED BY id-camellia192-wrap } }

kwa-camellia256-wrap KEY-WRAP ::= {
   IDENTIFIER id-camellia256-wrap
   PARAMS ARE absent
   SMIME-CAPS { IDENTIFIED BY id-camellia256-wrap } }

-- Key Derivation Functions

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

kda-kdf2 KEY-DERIVATION ::= {
   IDENTIFIER id-kdf-kdf2
   PARAMS TYPE KDF2-HashFunction ARE required
   -- No S/MIME caps defined -- }

KDF2-HashFunction ::=
   AlgorithmIdentifier { DIGEST-ALGORITHM, {KDF2-HashFunctions} }

KDF2-HashFunctions DIGEST-ALGORITHM ::= { X9-HashFunctions, ... }

id-kdf-kdf3 OID ::= { x9-44-components kdf3(2) }

kda-kdf3 KEY-DERIVATION ::= {
   IDENTIFIER id-kdf-kdf3
   PARAMS TYPE KDF3-HashFunction ARE required
   -- No S/MIME caps defined -- }

KDF3-HashFunction ::=
   AlgorithmIdentifier { DIGEST-ALGORITHM, {KDF3-HashFunctions} }

KDF3-HashFunctions DIGEST-ALGORITHM ::= { X9-HashFunctions, ... }

-- Hash Functions

X9-HashFunctions DIGEST-ALGORITHM ::= {
   mda-sha1 | mda-sha224 | mda-sha256 | mda-sha384 |
   mda-sha512, ... }

-- Updates for the SMIME-CAPS Set from RFC 5911

SMimeCapsSet SMIME-CAPS ::= {
   kema-kem-rsa.&smimeCaps |
   kwa-aes128-wrap |
   kwa-aes192-wrap |
   kwa-aes256-wrap |
   kwa-camellia128-wrap.&smimeCaps |
   kwa-camellia192-wrap.&smimeCaps |
   kwa-camellia256-wrap.&smimeCaps,
   ... }

END
<CODE ENDS>
```

Appendix C. SMIMECapabilities Examples

To indicate support for the RSA-KEM algorithm coupled with the KDF3 key derivation function with SHA-256 and the AES Key Wrap symmetric key-encryption algorithm 128-bit key-encryption key, the SMIMECapabilities will include the following entry:

```
SEQUENCE {
   id-rsa-kem-spki,                           -- RSA-KEM Algorithm
   SEQUENCE {                           -- GenericHybridParameters
      SEQUENCE {                    -- key encapsulation mechanism
         id-kem-rsa,                                    -- RSA-KEM
         SEQUENCE {                             -- RsaKemParameters
            SEQUENCE {                   -- key derivation function
               id-kdf-kdf3,                                -- KDF3
               SEQUENCE {                      -- KDF3-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 SMIMECapability value has the following DER encoding (in hexadecimal):

```
30 47
  06 0b 2a 86 48 86 f7 0d 01 09 10 03 0e          -- id-rsa-kem-spki
  30 38
     30 29
        06 07 28 81 8c 71 02 02 04                -- id-kem-rsa
        30 1e
           30 19
              06 0a 2b 81 05 10 86 48 09 2c 01 02 -- id-kdf-kdf3
              30 0b
                 06 09 60 86 48 01 65 03 04 02 01 -- id-sha256
                 02 01 10                         -- 16 bytes
      30 0b
         06 09 60 86 48 01 65 03 04 01 05         -- id-aes128-Wrap
```

To indicate support for the RSA-KEM algorithm coupled with the KDF3 key derivation function with SHA-384 and the AES Key Wrap symmetric key-encryption algorithm 192-bit key-encryption key, the SMIMECapabilities will include the following SMIMECapability value (in hexadecimal):

```
  30 47 06 0b 2a 86 48 86 f7 0d 01 09 10 03 0e 30
  38 30 29 06 07 28 81 8c 71 02 02 04 30 1e 30 19
  06 0a 2b 81 05 10 86 48 09 2c 01 02 30 0b 06 09
  60 86 48 01 65 03 04 02 02 02 01 18 30 0b 06 09
  60 86 48 01 65 03 04 01 19
```

To indicate support for the RSA-KEM algorithm coupled with the KDF3 key derivation function with SHA-512 and the AES Key Wrap symmetric key-encryption algorithm 256-bit key-encryption key, the SMIMECapabilities will include the following SMIMECapability value (in hexadecimal):

```
  30 47 06 0b 2a 86 48 86 f7 0d 01 09 10 03 0e 30
  38 30 29 06 07 28 81 8c 71 02 02 04 30 1e 30 19
  06 0a 2b 81 05 10 86 48 09 2c 01 02 30 0b 06 09
  60 86 48 01 65 03 04 02 03 02 01 20 30 0b 06 09
  60 86 48 01 65 03 04 01 2d
```

Appendix D. RSA-KEM CMS Enveloped-Data Example

This example shows the establishment of an AES-128 content-encryption key using:

- RSA-KEM with a 3072-bit key and KDF3 with SHA-256;
- KEMRecipientInfo key derivation using KDF3 with SHA-256; and
- KEMRecipientInfo Key Wrap using AES-128-KEYWRAP.

In real-world use, the originator would encrypt the content-encryption key in a manner that would allow decryption with their own private key as well as the recipient's private key. This is omitted in an attempt to simplify the example.

D.1. Originator RSA-KEM Encapsulate() Processing

Alice obtains Bob's public key:

```
-----BEGIN PUBLIC KEY-----
MIIBojANBgkqhkiG9w0BAQEFAAOCAY8AMIIBigKCAYEA3ocW14cxncPJ47fnEjBZ
AyfC2lqapL3ET4jvV6C7gGeVrRQxWPDwl+cFYBBR2ej3j3/0ecDmu+XuVi2+s5JH
Keeza+itfuhsz3yifgeEpeK8T+SusHhn20/NBLhYKbh3kiAcCgQ56dpDrDvDcLqq
vS3jg/VO+OPnZbofoHOOevt8Q/roahJe1PlIyQ4udWB8zZezJ4mLLfbOA9YVaYXx
2AHHZJevo3nmRnlgJXo6mE00E/6qkhjDHKSMdl2WG6mO9TCDZc9qY3cAJDU6Ir0v
SH7qUl8/vN13y4UOFkn8hM4kmZ6bJqbZt5NbjHtY4uQ0VMW3RyESzhrO02mrp39a
uLNnH3EXdXaV1tk75H3qC7zJaeGWMJyQfOE3YfEGRKn8fxubji716D8UecAxAzFy
FL6m1JiOyV5acAiOpxN14qRYZdHnXOM9DqGIGpoeY1UuD4Mo05osOqOUpBJHA9fS
whSZG7VNf+vgNWTLNYSYLI04KiMdulnvU6ds+QPz+KKtAgMBAAE=
-----END PUBLIC KEY-----
```

Bob's RSA public key has the following key identifier:

```
9eeb67c9b95a74d44d2f16396680e801b5cba49c
```

Alice randomly generates integer z between 0 and n-1:

```
9c126102a5c1c0354672a3c2f19fc9ddea988f815e1da812c7bd4f8eb082bdd1
4f85a7f7c2f1af11d5333e0d6bcb375bf855f208da72ba27e6fb0655f2825aa6
2b93b1f9bbd3491fed58f0380fa0de36430e3a144d569600bd362609be5b9481
0875990b614e406fa6dff500043cbca95968faba61f795096a7fb3687a51078c
4ca2cb663366b0bea0cd9cccac72a25f3f4ed03deb68b4453bba44b943f4367b
67d6cd10c8ace53f545aac50968fc3c6ecc80f3224b64e37038504e2d2c0e2b2
9d45e46c62826d96331360e4c17ea3ef89a9efc5fac99eda830e81450b6534dc
0bdf042b8f3b706649c631fe51fc2445cc8d447203ec2f41f79cdfea16de1ce6
abdfdc1e2ef2e5d5d8a65e645f397240ef5a26f5e4ff715de782e30ecf477293
e89e13171405909a8e04dd31d21d0c57935fc1ceea8e1033e31e1bc8c56da0f3
d79510f3f380ff58e5a61d361f2f18e99fbae5663172e8cd1f21deaddc5bbbea
060d55f1842b93d1a9c888d0bf85d0af9947fe51acf940c7e7577eb79cabecb3
```

Alice encrypts integer z using the Bob's RSA public key. The result is called ct:

```
   c071fc273af8e7bdb152e06bf73310361074154a43abcf3c93c13499d2065344
   3eed9ef5d3c0685e4aa76a6854815bb97691ff9f8dac15eea7d74f452bf350a6
   46163d68288e978cbf7a73089ee52712f9a4f49e06ace7bbc85ab14d4e336c97
   c5728a2654138c7b26e8835c6b0a9fbed26495c4eadf745a2933be283f6a88b1
   6695fc06666873cfb6d36718ef3376cefc100c3941f3c494944078325807a559
   186b95ccabf3714cfaf79f83bd30537fdd9aed5a4cdcbd8bd0486faed73e9d48
   6b3087d6c806546b6e2671575c98461e441f65542bd95de26d0f53a64e7848d7
   31d9608d053e8d345546602d86236ffe3704c98ad59144f3089e5e6d527b5497
   ba103c79d62e80d0235410b06f71a7d9bd1c38000f910d6312ea2f20a3557535
   ad01b3093fb5f7ee507080d0f77d48c9c3b3796f6b7dd3786085fb895123f04c
   a1f1c1be22c747a8dface32370fb0d570783e27dbb7e74fca94ee39676fde3d8
   a9553d878224736e37e191dab953c7e228c07ad5ca3122421c14debd072a9ab6
```

Alice derives the shared secret (SS) using KDF3 with SHA-256:

```
   3cf82ec41b54ed4d37402bbd8f805a52
```

D.2. Originator CMS Processing

Alice encodes the CMSORIforKEMOtherInfo structure with the algorithm identifier for AES-128-KEYWRAP and a key length of 16 octets. The DER encoding of CMSORIforKEMOtherInfo produces 18 octets:

```
   3010300b0609608648016503040105020110
```

The CMSORIforKEMOtherInfo structure contains:

```
  0  16: SEQUENCE {
  2  11:  SEQUENCE {
  4   9:   OBJECT IDENTIFIER aes128-wrap (2 16 840 1 101 3 4 1 5)
       :    }
 15   1:  INTEGER 16
       :   }
```

Alice derives the key-encryption key from shared secret produced by RSA-KEM Encapsulate() and the CMSORIforKEMOtherInfo structure with KDF3 and SHA-256. The KEK is:

```
   e6dc9d62ff2b469bef604c617b018718
```

Alice randomly generates a 128-bit content-encryption key:

```
   77f2a84640304be7bd42670a84a1258b
```

Alice uses AES-128-KEYWRAP to encrypt the 128-bit content-encryption key with the derived key-encryption key:

```
28782e5d3d794a7616b863fbcfc719b78f12de08cf286e09
```

Alice encrypts the padded content using AES-128-CBC with the content-encryption key. The 16-octet IV used is:

```
480ccafebabefacedbaddecaf8887781
```

The padded content plaintext is:

```
48656c6c6f2c20776f726c6421030303
```

The resulting ciphertext is:

```
c6ca65db7bdd76b0f37e2fab6264b66d
```

Alice encodes the EnvelopedData (using KEMRecipientInfo) and ContentInfo, and then sends the result to Bob. The Base64-encoded result is:

```
MIICXAYJKoZIhvcNAQcDoIICTTCCAkkCAQMxggIEpIICAAYLKoZIhvcNAQkQDQMw
ggHvAgEAgBSe62fJuVp01E0vFjlmgOgBtcuknDAJBgcogYxxAgIEBIIBgMBx/Cc6
+Oe9sVLga/czEDYQdBVKQ6vPPJPBNJnSBlNEPu2e9dPAaF5Kp2poVIFbuXaR/5+N
rBXup9dPRSvzUKZGFj1oKI6XjL96cwie5ScS+aT0ngas57vIWrFNTjNsl8VyiiZU
E4x7JuiDXGsKn77SZJXE6t90Wikzvig/aoixZpX8BmZoc8+202cY7zN2zvwQDDlB
88SUlEB4MlgHpVkYa5XMq/NxTPr3n4O9MFN/3ZrtWkzcvYvQSG+u1z6dSGswh9bI
BlRrbiZxV1yYRh5EH2VUK9ld4m0PU6ZOeEjXMdlgjQU+jTRVRmAthiNv/jcEyYrV
kUTzCJ5ebVJ7VJe6EDx51i6A0CNUELBvcafZvRw4AA+RDWMS6i8go1V1Na0Bswk/
tffuUHCA0Pd9SMnDs3lva33TeGCF+4lRI/BMofHBviLHR6jfrOMjcPsNVweD4n27
fnT8qU7jlnb949ipVT2HgiRzbjfhkdq5U8fiKMB61coxIkIcFN69ByqatjAbBgor
gQUQhkgJLAECMA0GCWCGSAFlAwQCAQUAAgEQMAsGCWCGSAFlAwQBBQQYKHguXT15
SnYWuGP7z8cZt48S3gjPKG4JMDwGCSqGSIb3DQEHATAdBglghkgBZQMEAQIEEEgM
yv66vvrO263eyviId4GAEMbKZdt73Xaw834vq2Jktm0=
```

This result decodes to:

```
  0 604: SEQUENCE {
  4   9:  OBJECT IDENTIFIER envelopedData (1 2 840 113549 1 7 3)
 15 589:  [0] {
 19 585:   SEQUENCE {
 23   1:    INTEGER 3
 26 516:    SET {
 30 512:     [4] {
 34  11:      OBJECT IDENTIFIER
       :       KEMRecipientInfo (1 2 840 113549 1 9 16 13 3)
 47 495:      SEQUENCE {
 51   1:       INTEGER 0
 54  20:       [0]
       :       9E EB 67 C9 B9 5A 74 D4 4D 2F 16 39 66 80 E8 01
       :       B5 CB A4 9C
 76   9:       SEQUENCE {
 78   7:        OBJECT IDENTIFIER kemRSA (1 0 18033 2 2 4)
       :         }
 87 384:       OCTET STRING
       :       C0 71 FC 27 3A F8 E7 BD B1 52 E0 6B F7 33 10 36
       :       10 74 15 4A 43 AB CF 3C 93 C1 34 99 D2 06 53 44
       :       3E ED 9E F5 D3 C0 68 5E 4A A7 6A 68 54 81 5B B9
       :       76 91 FF 9F 8D AC 15 EE A7 D7 4F 45 2B F3 50 A6
       :       46 16 3D 68 28 8E 97 8C BF 7A 73 08 9E E5 27 12
       :       F9 A4 F4 9E 06 AC E7 BB C8 5A B1 4D 4E 33 6C 97
       :       C5 72 8A 26 54 13 8C 7B 26 E8 83 5C 6B 0A 9F BE
       :       D2 64 95 C4 EA DF 74 5A 29 33 BE 28 3F 6A 88 B1
       :       66 95 FC 06 66 68 73 CF B6 D3 67 18 EF 33 76 CE
       :       FC 10 0C 39 41 F3 C4 94 94 40 78 32 58 07 A5 59
       :       18 6B 95 CC AB F3 71 4C FA F7 9F 83 BD 30 53 7F
       :       DD 9A ED 5A 4C DC BD 8B D0 48 6F AE D7 3E 9D 48
       :       6B 30 87 D6 C8 06 54 6B 6E 26 71 57 5C 98 46 1E
       :       44 1F 65 54 2B D9 5D E2 6D 0F 53 A6 4E 78 48 D7
       :       31 D9 60 8D 05 3E 8D 34 55 46 60 2D 86 23 6F FE
       :       37 04 C9 8A D5 91 44 F3 08 9E 5E 6D 52 7B 54 97
       :       BA 10 3C 79 D6 2E 80 D0 23 54 10 B0 6F 71 A7 D9
       :       BD 1C 38 00 0F 91 0D 63 12 EA 2F 20 A3 55 75 35
       :       AD 01 B3 09 3F B5 F7 EE 50 70 80 D0 F7 7D 48 C9
       :       C3 B3 79 6F 6B 7D D3 78 60 85 FB 89 51 23 F0 4C
       :       A1 F1 C1 BE 22 C7 47 A8 DF AC E3 23 70 FB 0D 57
       :       07 83 E2 7D BB 7E 74 FC A9 4E E3 96 76 FD E3 D8
       :       A9 55 3D 87 82 24 73 6E 37 E1 91 DA B9 53 C7 E2
       :       28 C0 7A D5 CA 31 22 42 1C 14 DE BD 07 2A 9A B6
475  27:       SEQUENCE {
477  10:        OBJECT IDENTIFIER
       :         kdf3 (1 3 133 16 840 9 44 1 2)
489  13:        SEQUENCE {
491   9:         OBJECT IDENTIFIER
       :          sha-256 (2 16 840 1 101 3 4 2 1)
502   0:         NULL
       :          }
       :         }
504   1:       INTEGER 16
507  11:       SEQUENCE {
509   9:        OBJECT IDENTIFIER
       :         aes128-wrap (2 16 840 1 101 3 4 1 5)
       :         }
520  24:       OCTET STRING
```

```
        :       28 78 2E 5D 3D 79 4A 76 16 B8 63 FB CF C7 19 B7
        :       8F 12 DE 08 CF 28 6E 09
        :        }
        :       }
        :      }
 546  60:    SEQUENCE {
 548   9:     OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
 559  29:     SEQUENCE {
 561   9:      OBJECT IDENTIFIER
        :       aes128-CBC (2 16 840 1 101 3 4 1 2)
 572  16:      OCTET STRING
        :      48 0C CA FE BA BE FA CE DB AD DE CA F8 88 77 81
        :       }
 590  16:     [0] C6 CA 65 DB 7B DD 76 B0 F3 7E 2F AB 62 64 B6 6D
        :      }
        :     }
        :    }
        :   }
```

D.3. Recipient RSA-KEM Decapsulate() Processing

Bob's private key:

```
-----BEGIN PRIVATE KEY-----
MIIG5AIBAAKCAYEA3ocW14cxncPJ47fnEjBZAyfC2lqapL3ET4jvV6C7gGeVrRQx
WPDwl+cFYBBR2ej3j3/0ecDmu+XuVi2+s5JHKeeza+itfuhsz3yifgeEpeK8T+Su
sHhn20/NBLhYKbh3kiAcCgQ56dpDrDvDcLqqvS3jg/VO+OPnZbofoHOOevt8Q/ro
ahJe1PlIyQ4udWB8zZezJ4mLLfbOA9YVaYXx2AHHZJevo3nmRnlgJXo6mE00E/6q
khjDHKSMdl2WG6mO9TCDZc9qY3cAJDU6Ir0vSH7qUl8/vN13y4UOFkn8hM4kmZ6b
JqbZt5NbjHtY4uQ0VMW3RyESzhrO02mrp39auLNnH3EXdXaV1tk75H3qC7zJaeGW
MJyQfOE3YfEGRKn8fxubji716D8UecAxAzFyFL6m1JiOyV5acAiOpxN14qRYZdHn
XOM9DqGIGpoeY1UuD4Mo05osOqOUpBJHA9fSwhSZG7VNf+vgNWTLNYSYLI04KiMd
ulnvU6ds+QPz+KKtAgMBAAECggGATFfkSkUjjJCjLvDk4aScpSx6+Rakf2hrdS3x
jwqhyUfAXgTTeUQQBs1HVtHCgxQd+qlXYn3/qu8TeZVwG4NPztyi/Z5yB1wOGJEV
3k8N/ytul6pJFFn6p48VM01bUdTrkMJbXERe6g/rr6dBQeeItCaOK7N5SIJH3Oqh
9xYuB5tH4rquCdYLmt17Tx8CaVqU9qPY3vOdQEOwIjjMV8uQUR8rHSO9KkSj8AGs
Lq9kcuPpvgJc2oqMRcNePS2WVh8xPFktRLLRazgLP8STHAtjT6SlJ2UzkUqfDHGK
q/BoXxBDu6L1VDwdnIS5HXtL54ElcXWsoOyKF8/ilmhRUIUWRZFmlS1ok8IC5IgX
UdL9rJVZFTRLyAwmcCEvRM1asbBrhyEyshSOuN5nHJi2WVJ+wSHijeKl1qeLlpMk
HrdIYBq4Nz7/zXmiQphpAy+yQeanhP8O4O6C8e7RwKdpxe44su4Z8fEgA5yQx0u7
8yR1EhGKydX5bhBLR5Cm1VM7rT2BAoHBAP/+e5gZLNf/ECtEBZjeiJ0VshszOoUq
haUQPA+9Bx9pytsoKm5oQhB7QDaxAvrn8/FUW2aAkaXsaj9F+/q30AYSQtExai9J
fdKKook3oimN8/yNRsKmhfjGOj8hd4+GjX0qoMSBCEVdT+bAjjry8wgQrqReuZnu
oXU85dmb3jvv0uIczIKvTIeyjXE5afjQIJLmZFXsBm09BG87Ia5EFUKly96BOMJh
/QWEzuYYXDqOFfzQtkAefXNFW21Kz4Hw2QKBwQDeiGh4lxCGTjECvG7fauMGlu+q
DSdYyMHif6t6mx57eS16EjvOrlXKItYhIyzW8Kw0rf/CSB2j8ig1GkMLTOgrGIJ1
0322o50FOr5oOmZPueeR4pOyAP0fgQ8DD1L3JBpY68/8MhYbsizVrR+Ar4jM0f96
W2bF5Xj3h+fQTDMkx6VrCCQ6miRmBUzH+ZPs5n/lYOzAYrqiKOanaiHy4mjRvlsy
mjZ6z5CG8sISqcLQ/k3Qli5pOY/v0rdBjgwAW/UCgcEAqGVYGjKdXCzuDvf9EpV4
mpTWB6yIV2ckaPOn/tZi5BgsmEPwvZYZt0vMbu28Px7sSpkqUuBKbzJ4pcy8uC3I
SuYiTAhMiHS4rxIBX3BYXSuDD2RD4vG1+XM0h6jVRHXHh0nOXdVfgnmigPGz3jVJ
B8oph/jD8O2YCk4YCTDOXPEi8Rjusxzro+whvRR+kG0gsGGcKSVNCPj1fNISEte4
gJId7O1mUAAzeDjn/VaS/PXQovEMolssPPKn9NocbKbpAoHBAJnFHJunl22W/lrr
ppmPnIzjI30YVcYOA5vlqLKyGaAsnfYqP1WUNgfVhq2jRsrHx9cnHQI9Hu442PvI
x+c5H30YFJ4ipE3eRRRmAUi4ghY5WgD+1hw8fqyUW7E7l5LbSbGEUVXtrkU5G64T
UR91LEyMF8OPATdiV/KD4PWYkgaqRm3tVEuCVACDTQkqNsOOi3YPQcm270w6gxfQ
SOEy/kdhCFexJFA8uZvmh6Cp2crczxyBilR/yCxqKOONqlFdOQKBwFbJk5eHPjJz
AYueKMQESPGYCrwIqxgZGCxaqeVArHvKsEDx5whI6JWoFYVkFA8F0MyhukoEb/2x
2qB5T88Dg3EbqjTiLg3qxrWJ2OxtUo8pBP2I2wbl2NOwzcbrlYhzEZ8bJyxZu5i1
sYILC8PJ4Qzw6jS4Qpm4y1WHz8e/ElW6VyfmljZYA7f9WMntdfeQVqCVzNTvKn6f
hg6GSpJTzp4LV3ougi9nQuWXZF2wInsXkLYpsiMbL6Fz34RwohJtYA==
-----END PRIVATE KEY-----
```

Bob checks that the length of the ciphertext is less than nLen bytes.

Bob checks that the ciphertext is greater than zero and is less than his RSA modulus.

Bob decrypts the ciphertext with his RSA private key to obtain the integer z:

```
   9c126102a5c1c0354672a3c2f19fc9ddea988f815e1da812c7bd4f8eb082bdd1
   4f85a7f7c2f1af11d5333e0d6bcb375bf855f208da72ba27e6fb0655f2825aa6
   2b93b1f9bbd3491fed58f0380fa0de36430e3a144d569600bd362609be5b9481
   0875990b614e406fa6dff500043cbca95968faba61f795096a7fb3687a51078c
   4ca2cb663366b0bea0cd9cccac72a25f3f4ed03deb68b4453bba44b943f4367b
   67d6cd10c8ace53f545aac50968fc3c6ecc80f3224b64e37038504e2d2c0e2b2
   9d45e46c62826d96331360e4c17ea3ef89a9efc5fac99eda830e81450b6534dc
   0bdf042b8f3b706649c631fe51fc2445cc8d447203ec2f41f79cdfea16de1ce6
   abdfdc1e2ef2e5d5d8a65e645f397240ef5a26f5e4ff715de782e30ecf477293
   e89e13171405909a8e04dd31d21d0c57935fc1ceea8e1033e31e1bc8c56da0f3
   d79510f3f380ff58e5a61d361f2f18e99fbae5663172e8cd1f21deaddc5bbbea
   060d55f1842b93d1a9c888d0bf85d0af9947fe51acf940c7e7577eb79cabecb3
```

Bob checks that the integer z is greater than zero and is less than his RSA modulus.

Bob derives the shared secret (SS) using KDF3 with SHA-256:

```
   3cf82ec41b54ed4d37402bbd8f805a52
```

D.4. Recipient CMS Processing

Bob encodes the CMSORIforKEMOtherInfo structure with the algorithm identifier for AES-128-KEYWRAP and a key length of 16 octets. The DER encoding of CMSORIforKEMOtherInfo is not repeated here.

Bob derives the key-encryption key from shared secret and the CMSORIforKEMOtherInfo structure with KDF3 and SHA-256, the KEK is:

```
   e6dc9d62ff2b469bef604c617b018718
```

Bob uses AES-KEY-WRAP to decrypt the content-encryption key with the key-encryption key. The content-encryption key is:

```
   77f2a84640304be7bd42670a84a1258b
```

Bob decrypts the content using AES-128-CBC with the content- encryption key. The 16-octet IV used is:

```
   480ccafebabefacedbaddecaf8887781
```

The received ciphertext content is:

```
   c6ca65db7bdd76b0f37e2fab6264b66d
```

The resulting padded plaintext content is:

```
   48656c6c6f2c20776f726c6421030303
```

After stripping the AES-CBC padding, the plaintext content is:

```
   Hello, world!
```

Acknowledgements

We thank James Randall, Burt Kaliski, and John Brainard as the original authors of [RFC5990]; this document is based on their work.

We thank the members of the ASC X9F1 working group for their contributions to drafts of ANS X9.44, which led to [RFC5990].

We thank Blake Ramsdell, Jim Schaad, Magnus Nystrom, Bob Griffin, and John Linn for helping bring [RFC5990] to fruition.

We thank Burt Kaliski, Alex Railean, Joe Mandel, Mike Ounsworth, Peter Campbell, Daniel Van Geest, and David Ireland for careful review and thoughtful comments that greatly improved this document.

Authors' Addresses

Russ Housley
Vigil Security, LLC
516 Dranesville Road
Herndon, VA 20170
United States of America
Email: housley@vigilsec.com

Sean Turner
sn3rd
Email: sean@sn3rd.com