Internet Engineering Task Force (IETF)
Request for Comments: 9936
Category: Standards Track
ISSN: 2070-1721
J. Prat
CryptoNext Security
M. Ounsworth
Entrust
D. Van Geest
CryptoNext Security
March 2026

Use of ML-KEM in the Cryptographic Message Syntax (CMS)

Abstract

Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) is a quantum-resistant Key Encapsulation Mechanism (KEM). Three parameter sets for the ML-KEM algorithm are specified by the US National Institute of Standards and Technology (NIST) in FIPS 203. In order of increasing security strength (and decreasing performance), these parameter sets are ML-KEM-512, ML-KEM-768, and ML-KEM-1024. This document specifies the conventions for using ML-KEM with the Cryptographic Message Syntax (CMS) using the KEMRecipientInfo structure defined in "Using Key Encapsulation Mechanism (KEM) Algorithms in the Cryptographic Message Syntax (CMS)" (RFC 9629).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) is an IND-CCA2-secure Key Encapsulation Mechanism (KEM) standardized in [FIPS203] by the NIST PQC Project [NIST-PQ]. ML-KEM is the name given to the final standardized version and is incompatible with pre-standards versions, often called "Kyber".

[RFC9629] defines the KEMRecipientInfo structure for the use of KEM algorithms for the CMS enveloped-data content type, the CMS authenticated-data content type, and the CMS authenticated-enveloped-data content type. This document specifies the direct use of ML-KEM in the KEMRecipientInfo structure using each of the three parameter sets from [FIPS203], namely ML-KEM-512, ML-KEM-768, and ML-KEM-1024. It does not address or preclude the use of ML-KEM as part of any hybrid scheme.

1.1. Conventions and Terminology

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.

1.2. ML-KEM

ML-KEM is a lattice-based KEM using Module Learning with Errors as its underlying primitive, which is a structured lattices variant that offers good performance and relatively small and balanced key and ciphertext sizes. ML-KEM was standardized with three parameter sets: ML-KEM-512, ML-KEM-768, and ML-KEM-1024. The parameters for each of the security levels were chosen to be at least as secure as a generic block cipher of 128, 192, or 256 bits, respectively. Appendix B provides more information on ML-KEM security levels and sizes.

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

The following summarizes these three functions for the ML-KEM algorithm, referencing corresponding functions in [FIPS203]:

KeyGen() -> (ek, dk): Generate the public encapsulation key (ek) and a private decapsulation key (dk). [FIPS203] specifies two formats for an ML-KEM private key: a 64-octet seed (d,z) and an (expanded) private decapsulation key (dk). Algorithm 19 (ML-KEM.KeyGen()) from [FIPS203] generates the public encapsulation key (ek) and the private decapsulation key (dk). As an alternative, when a seed (d,z) is generated first and then the seed is expanded to get the keys, algorithm 16 (ML-KEM.KeyGen_internal(d,z)) from [FIPS203] expands the seed to ek and dk. See Section 6 of [RFC9935] for private key encoding considerations.

Encapsulate(ek) -> (c, ss): Given the recipient's public key (ek), produce both a ciphertext (c) to be passed to the recipient and a shared secret (ss) for use by the originator. Algorithm 20 (ML-KEM.Encaps(ek)) from [FIPS203] is the encapsulation function for ML-KEM.

Decapsulate(dk, c) -> ss: Given the private key (dk) and the ciphertext (c), produce the shared secret (ss) for the recipient. Algorithm 21 (ML-KEM.Decaps(dk,c)) from [FIPS203] is the decapsulation function for ML-KEM. If the private key is stored in seed form, ML-KEM.KeyGen_internal(d,z) may be needed as a first step to compute dk. See Section 8 of [RFC9935] for consistency considerations if the private key was stored in both seed and expanded formats.

All security levels of ML-KEM use SHA3-256, SHA3-512, SHAKE128, and SHAKE256 internally.

2. Use of the ML-KEM Algorithm in the CMS

The ML-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 ML-KEM algorithm to securely transfer the content-encryption key from the originator to the recipient.

Processing ML-KEM with KEMRecipientInfo follows the same steps as Section 2 of [RFC9629]. To support the ML-KEM algorithm, a CMS originator MUST implement the Encapsulate() function and a CMS recipient MUST implement the Decapsulate() function.

2.1. RecipientInfo Conventions

When the ML-KEM algorithm is employed for a recipient, the RecipientInfo alternative for that recipient MUST be OtherRecipientInfo using the KEMRecipientInfo structure as defined in [RFC9629].

The fields of the KEMRecipientInfo have the following meanings:

version
    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 one of id-alg-ml-kem-512, id-alg-ml-kem-768, or id-alg-ml-kem-1024. These identifiers are reproduced in Section 3.

kemct
    The ciphertext produced for this recipient.

kdf
    Identifies the key derivation algorithm. Note that the Key Derivation Function (KDF) used for CMS RecipientInfo process MAY be different than the KDF used within the ML-KEM algorithm. Implementations MUST support the HMAC-based Key Derivation Function (HKDF) [RFC5869] with SHA-256 [FIPS180] using the id-alg-hkdf-with-sha256 KDF object identifier (OID) [RFC8619]. As specified in [RFC8619], the parameter field MUST be absent when

this OID appears within the ASN.1 type AlgorithmIdentifier. Implementations MAY support other KDFs as well.

kekLength
: The size of the key-encryption key in octets.

ukm
: Optional input to the KDF. The secure use of ML-KEM in CMS does not depend on the use of a ukm value, so this document does not place any requirements on this value. See Section 3 of [RFC9629] for more information about the ukm parameter.

wrap
: Identifies a key-encryption algorithm used to encrypt the content-encryption key. Implementations supporting ML-KEM-512 MUST support the AES-Wrap-128 [RFC3394] key-encryption algorithm using the id-aes128-wrap key-encryption algorithm OID [RFC3565]. Implementations supporting ML-KEM-768 or ML-KEM-1024 MUST support the AES-Wrap-256 [RFC3394] key-encryption algorithm using the id-aes256-wrap key-encryption algorithm OID [RFC3565]. Implementations MAY support other key-encryption algorithms as well.

Appendix C contains an example of establishing a content-encryption key using ML-KEM in the KEMRecipientInfo type.

2.2. Underlying Components

When ML-KEM is employed in the CMS, the underlying components used within the KEMRecipientInfo structure SHOULD be consistent with a minimum desired security level. Several security levels have been identified in [NIST.SP.800-57pt1r5].

If underlying components other than those specified in Section 2.1 are used, then the following table gives the minimum requirements on the components used with ML-KEM in the KEMRecipientInfo type in order to satisfy the KDF and key wrapping algorithm requirements from Section 7 of [RFC9629]:

Security Strength	Algorithm	KDF Preimage Strength	Symmetric Key-Encryption Strength
128-bit	ML-KEM-512	128-bit	128-bit
192-bit	ML-KEM-768	192-bit	192-bit (*)
256-bit	ML-KEM-1024	256-bit	256-bit

Table 1: ML-KEM KEMRecipientInfo Component Security Levels

(*) In the case of AES Key Wrap, a 256-bit key is typically used because AES-192 is not as commonly deployed.

2.2.1. Use of the HKDF-Based Key Derivation Function

The HKDF function is a composition of the HKDF-Extract and HKDF-Expand functions.

```
HKDF(salt, IKM, info, L)
  = HKDF-Expand(HKDF-Extract(salt, IKM), info, L)
```

When used with KEMRecipientInfo, the salt parameter is unused; that is, it is the zero-length string "". The IKM, info, and L parameters correspond to the same KDF inputs from Section 5 of [RFC9629]. The info parameter is independently generated by the originator and recipient. Implementations MUST confirm that L is consistent with the key size of the key-encryption algorithm.

2.3. Certificate Conventions

[RFC5280] specifies the profile for using X.509 certificates in Internet applications. A recipient static public key is needed for ML-KEM and the originator obtains that public key from the recipient's certificate. The conventions for carrying ML-KEM public keys are specified in [RFC9935].

2.4. SMIME Capabilities 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 one or more of the ML-KEM algorithm identifiers.

The SMIMECapability SEQUENCE representing the ML-KEM algorithm MUST include one of the ML-KEM OIDs in the capabilityID field. When one of the ML-KEM OIDs appears in the capabilityID field, the parameters MUST NOT be present.

3. Identifiers

All identifiers used to indicate ML-KEM within the CMS are defined in [CSOR] and [RFC8619]; they are reproduced here for convenience:

```
nistAlgorithms OBJECT IDENTIFIER ::= { joint-iso-ccitt(2)
    country(16) us(840) organization(1) gov(101) csor(3)
    nistAlgorithm(4) }
kems OBJECT IDENTIFIER ::= { nistAlgorithms 4 }

id-alg-ml-kem-512 OBJECT IDENTIFIER ::= { kems 1 }

id-alg-ml-kem-768 OBJECT IDENTIFIER ::= { kems 2 }

id-alg-ml-kem-1024 OBJECT IDENTIFIER ::= { kems 3 }

id-alg-hkdf-with-sha256 OBJECT IDENTIFIER ::= { iso(1)
    member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)
    smime(16) alg(3) 28 }

aes OBJECT IDENTIFIER ::= { joint-iso-itu-t(2) country(16) us(840)
    organization(1) gov(101) csor(3) nistAlgorithms(4) 1 }

id-aes128-wrap OBJECT IDENTIFIER ::= { aes 5 }
id-aes256-wrap OBJECT IDENTIFIER ::= { aes 45 }
```

4. Security Considerations

The Security Considerations sections of [RFC9935] and [RFC9629] apply to this specification as well.

For ongoing discussions of ML-KEM-specific security considerations, refer to [MLKEM-SEC-CONS].

Implementations MUST protect the ML-KEM private key, the key-encryption key, the content-encryption key, message-authentication key, and the content-authenticated-encryption key. Of these keys, all but the private key are ephemeral and MUST be wiped after use. Disclosure of the ML-KEM 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 the 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 [NIST.SP.800-57pt1r5].

The generation of private keys relies on random numbers, as does the encapsulation function of ML-KEM. The use of inadequate pseudorandom number generators (PRNGs) to generate these values can result in little or no security. In the case of key generation, a random 32-byte seed is used to deterministically derive the key (with an additional 32 bytes reserved as a rejection value). In the case of encapsulation, a KEM is derived from the underlying ML-KEM public key encryption algorithm by deterministically encrypting a random 32-byte message for the public key. If the random value is weakly chosen, then an attacker may find it much easier to reproduce the PRNG environment that produced the keys or ciphertext, searching the resulting small set of possibilities for a matching public key or ciphertext value, rather than performing a more complex algorithmic attack against ML-KEM. The generation of quality random numbers is difficult; see Section 3.3 of [FIPS203] for some additional information.

ML-KEM encapsulation and decapsulation only outputs a shared secret and ciphertext. Implementations MUST NOT use intermediate values directly for any purpose.

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 ML-KEM 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.

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

5. IANA Considerations

For the ASN.1 Module in Appendix A, IANA has assigned an OID for the module identifier (84) with a description of "id-mod-cms-ml-kem-2024" in the "SMI Security for S/MIME Module Identifier (1.2.840.113549.1.9.16.0)" registry.

6. References

6.1. Normative References

[CSOR] NIST, "Computer Security Objects Register (CSOR)", 13 June 2025, <https://csrc.nist.gov/projects/computer-security-objects-register/algorithm-registration>.

[FIPS180] NIST, "Secure Hash Standard", NIST FIPS 180-4, DOI 10.6028/NIST.FIPS.180-4, August 2015,

<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

[FIPS203] NIST, "Module-Lattice-Based Key-Encapsulation Mechanism Standard", NIST FIPS 203, DOI 10.6028/NIST.FIPS.203, August 2024, <https://doi.org/10.6028/NIST.FIPS.203>.

[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>.

[RFC3565] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, DOI 10.17487/RFC3565, July 2003, <https://www.rfc-editor.org/info/rfc3565>.

[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>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[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>.

[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>.

[RFC8411] Schaad, J. and R. Andrews, "IANA Registration for the Cryptographic Algorithm Object Identifier Range", RFC 8411, DOI 10.17487/RFC8411, August 2018, <https://www.rfc-editor.org/info/rfc8411>.

[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>.

[RFC8619] Housley, R., "Algorithm Identifiers for the HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 8619, DOI 10.17487/RFC8619, June 2019, <https://www.rfc-editor.org/info/rfc8619>.

[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>.

[RFC9935] Turner, S., Kampanakis, P., Massimo, J., and B. E. Westerbaan, "Internet X.509 Public Key Infrastructure - Algorithm Identifiers for the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM)", RFC 9935, DOI 10.17487/RFC9935, March 2026, <https://www.rfc-editor.org/info/rfc9935>.

[X680] 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>.

6.2. Informative References

[CMVP] NIST, "Cryptographic Module Validation Program (CMVP)", 3 September 2025, <https://csrc.nist.gov/projects/cryptographic-module-validation-program>.

[IKEv2-MLKEM] Kampanakis, P., "Post-quantum Key Exchange with ML-KEM in the Internet Key Exchange Protocol Version 2 (IKEv2)", Work in Progress, Internet-Draft, draft-ietf-ipsecme-ikev2-mlkem-04, 26 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-ipsecme-ikev2-mlkem-04>.

[MLKEM-SEC-CONS] Fluhrer, S., Dang, Q., Mattsson, J. P., Milner, K., and D. Shiu, "ML-KEM Security Considerations", Work in Progress, Internet-Draft, draft-sfluhrer-cfrg-ml-kem-security-considerations-04, 17 November 2025, <https://datatracker.ietf.org/doc/html/draft-sfluhrer-cfrg-ml-kem-security-considerations-04>.

[NIST-PQ] NIST, "Post-Quantum Cryptography (PQC)", 30 September 2025, <https://csrc.nist.gov/projects/post-quantum-cryptography>.

[NIST.SP.800-57pt1r5] Barker, E., "Recommendation for Key Management: Part 1 - General", NIST SP 800-57pt1r5, DOI 10.6028/NIST.SP.800-57pt1r5, May 2020, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf>.

[RFC9690] Housley, R. and S. Turner, "Use of the RSA-KEM Algorithm in the Cryptographic Message Syntax (CMS)", RFC 9690, DOI 10.17487/RFC9690, February 2025, <https://www.rfc-editor.org/info/rfc9690>.

Appendix A. ASN.1 Module

This appendix includes the ASN.1 module [X680] for ML-KEM. This module imports objects from [RFC5911], [RFC9629], [RFC8619], and [RFC9935].

```
<CODE BEGINS>
CMS-ML-KEM-2024
{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
  pkcs-9(9) smime(16) modules(0) id-mod-cms-ml-kem-2024(84) }
```

```
DEFINITIONS IMPLICIT TAGS ::= BEGIN

EXPORTS ALL;

IMPORTS
  SMIME-CAPS
    FROM AlgorithmInformation-2009  -- [RFC5911]
      { iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0)
        id-mod-algorithmInformation-02(58) }

  KEM-ALGORITHM
    FROM KEMAlgorithmInformation-2023  -- [RFC9629]
       { iso(1) identified-organization(3) dod(6) internet(1)
         security(5) mechanisms(5) pkix(7) id-mod(0)
         id-mod-kemAlgorithmInformation-2023(109) }

  kda-hkdf-with-sha256
    FROM HKDF-OID-2019  -- [RFC8619]
       { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
         pkcs-9(9) smime(16) modules(0) id-mod-hkdf-oid-2019(68) }

  kwa-aes128-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) }

  id-alg-ml-kem-512, id-alg-ml-kem-768, id-alg-ml-kem-1024,
  pk-ml-kem-512, pk-ml-kem-768, pk-ml-kem-1024
    FROM X509-ML-KEM-2024 -- [RFC9935]
       { iso(1) identified-organization(3) dod(6)
         internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
         id-mod-x509-ml-kem-2025(121) };

--
-- ML-KEM Key Encapsulation Mechanism Algorithms
--

kema-ml-kem-512 KEM-ALGORITHM ::= {
   IDENTIFIER id-alg-ml-kem-512
   PARAMS ARE absent
   PUBLIC-KEYS { pk-ml-kem-512 }
   UKM ARE optional
   SMIME-CAPS { IDENTIFIED BY id-alg-ml-kem-512 } }

kema-ml-kem-768 KEM-ALGORITHM ::= {
   IDENTIFIER id-alg-ml-kem-768
   PARAMS ARE absent
   PUBLIC-KEYS { pk-ml-kem-768 }
   UKM ARE optional
   SMIME-CAPS { IDENTIFIED BY id-alg-ml-kem-768 } }

kema-ml-kem-1024 KEM-ALGORITHM ::= {
   IDENTIFIER id-alg-ml-kem-1024
   PARAMS ARE absent
   PUBLIC-KEYS { pk-ml-kem-1024 }
   UKM ARE optional
   SMIME-CAPS { IDENTIFIED BY id-alg-ml-kem-1024 } }

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

SMimeCapsSet SMIME-CAPS ::=
   { kema-ml-kem-512.&smimeCaps |
     kema-ml-kem-768.&smimeCaps |
     kema-ml-kem-1024.&smimeCaps |
```

```
      kda-hkdf-with-sha256.&smimeCaps |
      kwa-aes128-wrap.&smimeCaps |
      kwa-aes256-wrap.&smimeCaps,
      ... }

END
<CODE ENDS>
```

Appendix B. Parameter Set Security and Sizes

Instead of defining the strength of a quantum algorithm using the imprecise notion of bits of security, NIST has defined security levels by picking a reference scheme, which is expected to offer notable levels of resistance to both quantum and classical attacks. To wit, a KEM algorithm that achieves NIST Post-Quantum Cryptography (PQC) security must require computational resources to break IND-CCA2 security comparable or greater than that required for key search on AES-128, AES-192, and AES-256 for Levels 1, 3, and 5, respectively. Levels 2 and 4 use collision search for SHA-256 and SHA-384 as reference.

Parameter Set	Level	Encap. Key Size	Decap. Key Size	Ciphertext Size	Shared Secret Size
ML-KEM-512	1	800	1632	768	32
ML-KEM-768	3	1184	2400	1088	32
ML-KEM-1024	5	1568	3168	1568	32

Table 2: ML-KEM parameter Sets, NIST Security Level, and Sizes in Bytes

Appendix C. ML-KEM CMS Authenticated-Enveloped-Data Example

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

* ML-KEM-512;

* KEMRecipientInfo key derivation using HKDF 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.

C.1. Originator CMS Processing

Alice obtains Bob’s ML-KEM-512 public key:


```
-----BEGIN PUBLIC KEY-----
MIIDMjALBglghkgBZQMEBAEDggMhADmVgV5ZfRBDVc8pqlMzyTJRhp1bzb5IcST2
Ari2pmwWxHYWSK12XPXYAGtRXpBafwrAdrDGLvoygVPnylcBaZ8TBfHmvG+QsOSb
aTUSts6ZKouAFt38GmYsfj+WGcvYad13GvMIlszVkYrGy3dGbF53mZbWf/mqvJdQ
Pyx7fi0ADYZFD7GAfKTKvaRlgloxx4mht6SRqzhydl0yDQtxkg+iE8lAk0Frg7gS
Tmn2XmLLUADcw3qpoP/3OXDEdy81fSQYnKb1MFVowOI3ajdipoxgXlY8XSCVcuD8
dTLKKUcpU1VntfxBPF6HktJGRTbMgI+YrddGZPFBVm+QFqkKVBgpqYoEZM5BqLtE
wtT6PCwglGByjvFKGnxMm5jRIgO0zDUpFgqasteDj3/2tTrgWqMafWRrevpsRZMl
JqPDdVYZvplMIRwqMcBbNEeDbLIVC+GCna5rBMVTXP9Ubjkrp5dBFyD5JPSQpaxU
lfITVtVQt4KmTBaItrZVvMeEIZekNML2Vjtbfwmni8xIgjJ4NWHRb0y6tnVUAAUH
```

gVcMZmBLgXrRJSKUc26LAYYaS1p0UZuLb+UUiaUHI5Llh2JscTd2V10zgGocjicy
r5fCaA9RZmMxxOuLvAQxxPloMtrxs8RVKPuhU/bHixwZhwKUfM0zdyekb7U7oR3l
y0GRNGhZUWy2rXJADzzyCbI2rvNaWArIfrPjD6/WaXPKin3SZ1r0H3oXthQzzRr4
D3cIhp9mVIhJeYCxrBCgzctjagDthoGzXkKRJMqANQcluF+DperDpKPMFgCQPmUp
NWC5szblrw1SnawaBIEZMCy3qbzBELlIUb8CEX8ZncSFqFK3Rz8JuDGmgx1bVMC3
kNIlz2u5LZRiomzbM92lEjx6rw4moLg2Ve6ii/OoB0clAY/WuuS2Ac9huqtxp6PT
UZejQ+dLSicsEl1UCJZCbYW3lY07OKa6mH7DciXHtEzbEt3kU5tKsII2NoPwS/eg
nMXEHf6DChsWLgsyQzQ2LwhKFEZ3IzRLrdAA+NjFN8SPmY8FMHzr0e3guBw7xZoG
WhttY7Js
-----END PUBLIC KEY-----

Bob’s ML-KEM-512 public key has the following key identifier:

599788C37AED400EE405D1B2A3366AB17D824A51

Alice generates a shared secret and ciphertext using Bob’s ML-KEM-512 public key:

Shared secret:

7DF12D412AE299A24FDE6D7C3BB8E3194C80AD3C733DCF2775E09FE8BEDB86D8

Ciphertext:

3EA40FC6CA090E2C8AF76E2727AB38E0652D9515986FE186827FE84E596E421B
85FD459CC78997372C9DE31D191B39C1D5A3EB6DDB56AADEDE765CC390FDBBC2
F88CB175681D4201B81CCDFCB24FEF13AF2F5A1ABCF8D8AF384F02A010A6E919
F1987A5E9B1C0E2D3F07F58A9FA539CE86CC149910A1692C0CA4CE0ECE4EEED2
E6699CB976332452DE4A2EB5CA61F7B081330C34798EF712A24E59C33CEA1F1F
9E6D4FBF3743A38467430011336F62D870792B866BEFCD1D1B365BED1952673D
3A5B0C20B386B4EFD1CF63FD376BD47CCC46AC4DD8EC66B047C4C95ACFF1CFD0
28A419B002FDA1B617CBA61D2E91CFE8FFFBCB8FFD4D5F6AD8B158C219E36DC5
1405DC0C0B234979AC658E72BDDF1B6773B96B2AE3E4D07BE86048040C016743
6FA839E7529B00CC9AB55A2F25DB63CC9F557594E691C11E553D4A3EBC760F5F
19E5FE144838B4C7D1591DA9B5D467494FD9CAC52CC5504060399DBDB72298EB
9A4C017B00786FDC7D9D7AA57ADBB8B61C34DE1E288B2AB728171DCE143CD169
53F984C1AED559E56BAA0CE658D32CCE42F4407504CD7A579AD0EF9B77135EAA
39B6F93A3A2E5997807F06361C83F4E67F8E3F9CF68316011514F5D85A181CEA
D714CD4940E4EBAC01D66528DA32F89CEA0428E8EBCADCF8AA188C9F62E85B19
57655B7FE2B8D7973B7A7226B66D93BF7B232F3DCF653C84B4ECF1A9920DB194
9AD750B546A5552A20E54909719B8C0C07056FCB7E574AD2A32EC95001DDE844
81BE77D039ED5BF74262ECF3981F1B00D3366A9C2E061C47E241A061C6249560
D2B8446A480C38C28BA989D9F68ADC4BBAF2A20B47E4923128C72342D597FDA2
59DE0B83C2056D6B77E799B319324AA50B1D659C2A56029B7453C5F3BA5243D9
FA749D917C40D9D101E453BC8B10E42A7C089323C026F783E100B9FA6E701442
4DA6FA3792BC957EE8219D016B773F28FEDCC962A485ABAFFEC023281971E29A
A689839ECFD2619E92287CD230DB26A2507CC500EB1C7A5293B5FE917AE29BF1
AD350124F8A311635214B411DB9F67D3B85BD715018537EA45B41F41B4C66051

Alice encodes the CMSORIforKEMOtherInfo:

3010300B0609608648016503040105020110

Alice derives the key-encryption key from the shared secret and CMSORIforKEMOtherInfo using HKDF with SHA-256:

CF453A3E2BAE0A78701B8206C185A008

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

C5153005588269A0A59F3C01943FDD56

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

C050E4392F9C14DD0AC2220203F317D701F94F9DD92778F5

Alice encrypts the padded content using AES-128-GCM with the content-encryption key and encodes the AuthEnvelopedData (using KEMRecipientInfo) and ContentInfo, and then sends the result to Bob.

The Base64-encoded result is:


```
-----BEGIN CMS-----
MIID4gYLKoZIhvcNAQkQAReggPRMIIDzQIBADGCA3ikggN0BgsqhkiG9w0BCRAN
AzCCA2MCAQCAFFmXiMN67UAO5AXRsqM2arF9gkpRMAsGCWCGSAFlAwQEAQSCAwA+
pA/GygkOLIr3bicnqzjgZS2VFZhv4YaCf+hOWW5CG4X9RZzHiZc3LJ3jHRkbOcHV
o+tt21aq3t52XMOQ/bvC+IyxdWgdQgG4HM38sk/vE68vWhq8+NivOE8CoBCm6Rnx
mHpemxwOLT8H9YqfpTnOhswUmRChaSwMpM4Ozk7u0uZpnLl2MyRS3koutcph97CB
Mww0eY73EqJOWcM86h8fnm1PvzdDo4RnQwARM29i2HB5K4Zr780dGzZb7RlSZz06
Wwwgs4a079HPY/03a9R8zEasTdjsZrBHxMlaz/HP0CikGbAC/aG2F8umHS6Rz+j/
+8uP/U1fatixWMIZ423FFAXcDAsjSXmsZY5yvd8bZ3O5ayrj5NB76GBIBAwBZ0Nv
qDnnUpsAzJq1Wi8l22PMn1V1lOaRwR5VPUo+vHYPXxnl/hRIOLTH0VkdqbXUZ0lP
2crFLMVQQGA5nb23IpjrmkwBewB4b9x9nXqletu4thw03h4oiyq3KBcdzhQ80WlT
+YTBrtVZ5WuqDOZY0yzOQvRAdQTNelea0O+bdxNeqjm2+To6LlmXgH8GNhyD9OZ/
jj+c9oMWARUU9dhaGBzq1xTNSUDk66wB1mUo2jL4nOoEKOjrytz4qhiMn2LoWxlX
ZVt/4rjXlzt6cia2bZO/eyMvPc9lPIS07PGpkg2xlJrXULVGpVUqIOVJCXGbjAwH
BW/LfldK0qMuyVAB3ehEgb530DntW/dCYuzzmB8bANM2apwuBhxH4kGgYcYklWDS
uERqSAw4woupidn2itxLuvKiC0fkkjEoxyNC1Zf9olneC4PCBW1rd+eZsxkySqUL
HWWcKlYCm3RTxfO6UkPZ+nSdkXxA2dEB5FO8ixDkKnwIkyPAJveD4QC5+m5wFEJN
pvo3kryVfughnQFrdz8o/tzJYqSFq6/+wCMoGXHimqaJg57P0mGekih80jDbJqJQ
fMUA6xx6UpO1/pF64pvxrTUBJPijEWNSFLQR259n07hb1xUBhTfqRbQfQbTGYFEw
DQYLKoZIhvcNAQkQAxwCARAwCwYJYIZIAWUDBAEFBBjAUOQ5L5wU3QrCIgID8xfX
AflPndknePUwOgYJKoZIhvcNAQcBMB4GCWCGSAFlAwQBBjARBAxcpXRouBvwO42n
GGwCARCADZTIaJqZ0sOOGS+muggEEFzxeGxXx0ArVPyTwwpKRTM=
-----END CMS-----
```


This result decodes to:

```
  0 994: SEQUENCE {
  4  11:  OBJECT IDENTIFIER
       :   authEnvelopedData (1 2 840 113549 1 9 16 1 23)
 17 977:  [0] {
 21 973:   SEQUENCE {
 25   1:    INTEGER 0
 28 888:    SET {
 32 884:     [4] {
 36  11:      OBJECT IDENTIFIER '1 2 840 113549 1 9 16 13 3'
 49 867:      SEQUENCE {
 53   1:       INTEGER 0
 56  20:       [0]
       :    59 97 88 C3 7A ED 40 0E E4 05 D1 B2 A3 36 6A B1
       :    7D 82 4A 51
 78  11:       SEQUENCE {
 80   9:        OBJECT IDENTIFIER '2 16 840 1 101 3 4 4 1'
       :         }
 91 768:       OCTET STRING
       :    3E A4 0F C6 CA 09 0E 2C 8A F7 6E 27 27 AB 38 E0
       :    65 2D 95 15 98 6F E1 86 82 7F E8 4E 59 6E 42 1B
       :    85 FD 45 9C C7 89 97 37 2C 9D E3 1D 19 1B 39 C1
       :    D5 A3 EB 6D DB 56 AA DE DE 76 5C C3 90 FD BB C2
       :    F8 8C B1 75 68 1D 42 01 B8 1C CD FC B2 4F EF 13
       :    AF 2F 5A 1A BC F8 D8 AF 38 4F 02 A0 10 A6 E9 19
       :    F1 98 7A 5E 9B 1C 0E 2D 3F 07 F5 8A 9F A5 39 CE
       :    86 CC 14 99 10 A1 69 2C 0C A4 CE 0E CE 4E EE D2
       :    E6 69 9C B9 76 33 24 52 DE 4A 2E B5 CA 61 F7 B0
       :    81 33 0C 34 79 8E F7 12 A2 4E 59 C3 3C EA 1F 1F
       :    9E 6D 4F BF 37 43 A3 84 67 43 00 11 33 6F 62 D8
       :    70 79 2B 86 6B EF CD 1D 1B 36 5B ED 19 52 67 3D
       :    3A 5B 0C 20 B3 86 B4 EF D1 CF 63 FD 37 6B D4 7C
       :    CC 46 AC 4D D8 EC 66 B0 47 C4 C9 5A CF F1 CF D0
       :    28 A4 19 B0 02 FD A1 B6 17 CB A6 1D 2E 91 CF E8
```

```
       :    FF FB CB 8F FD 4D 5F 6A D8 B1 58 C2 19 E3 6D C5
       :    14 05 DC 0C 0B 23 49 79 AC 65 8E 72 BD DF 1B 67
       :    73 B9 6B 2A E3 E4 D0 7B E8 60 48 04 0C 01 67 43
       :    6F A8 39 E7 52 9B 00 CC 9A B5 5A 2F 25 DB 63 CC
       :    9F 55 75 94 E6 91 C1 1E 55 3D 4A 3E BC 76 0F 5F
       :    19 E5 FE 14 48 38 B4 C7 D1 59 1D A9 B5 D4 67 49
       :    4F D9 CA C5 2C C5 50 40 60 39 9D BD B7 22 98 EB
       :    9A 4C 01 7B 00 78 6F DC 7D 9D 7A A5 7A DB B8 B6
       :    1C 34 DE 1E 28 8B 2A B7 28 17 1D CE 14 3C D1 69
       :    53 F9 84 C1 AE D5 59 E5 6B AA 0C E6 58 D3 2C CE
       :    42 F4 40 75 04 CD 7A 57 9A D0 EF 9B 77 13 5E AA
       :    39 B6 F9 3A 3A 2E 59 97 80 7F 06 36 1C 83 F4 E6
       :    7F 8E 3F 9C F6 83 16 01 15 14 F5 D8 5A 18 1C EA
       :    D7 14 CD 49 40 E4 EB AC 01 D6 65 28 DA 32 F8 9C
       :    EA 04 28 E8 EB CA DC F8 AA 18 8C 9F 62 E8 5B 19
       :    57 65 5B 7F E2 B8 D7 97 3B 7A 72 26 B6 6D 93 BF
       :    7B 23 2F 3D CF 65 3C 84 B4 EC F1 A9 92 0D B1 94
       :    9A D7 50 B5 46 A5 55 2A 20 E5 49 09 71 9B 8C 0C
       :    07 05 6F CB 7E 57 4A D2 A3 2E C9 50 01 DD E8 44
       :    81 BE 77 D0 39 ED 5B F7 42 62 EC F3 98 1F 1B 00
       :    D3 36 6A 9C 2E 06 1C 47 E2 41 A0 61 C6 24 95 60
       :    D2 B8 44 6A 48 0C 38 C2 8B A9 89 D9 F6 8A DC 4B
       :    BA F2 A2 0B 47 E4 92 31 28 C7 23 42 D5 97 FD A2
       :    59 DE 0B 83 C2 05 6D 6B 77 E7 99 B3 19 32 4A A5
       :    0B 1D 65 9C 2A 56 02 9B 74 53 C5 F3 BA 52 43 D9
       :    FA 74 9D 91 7C 40 D9 D1 01 E4 53 BC 8B 10 E4 2A
       :    7C 08 93 23 C0 26 F7 83 E1 00 B9 FA 6E 70 14 42
       :    4D A6 FA 37 92 BC 95 7E E8 21 9D 01 6B 77 3F 28
       :    FE DC C9 62 A4 85 AB AF FE C0 23 28 19 71 E2 9A
       :    A6 89 83 9E CF D2 61 9E 92 28 7C D2 30 DB 26 A2
       :    50 7C C5 00 EB 1C 7A 52 93 B5 FE 91 7A E2 9B F1
       :    AD 35 01 24 F8 A3 11 63 52 14 B4 11 DB 9F 67 D3
       :    B8 5B D7 15 01 85 37 EA 45 B4 1F 41 B4 C6 60 51
863  13:       SEQUENCE {
865  11:         OBJECT IDENTIFIER
       :           hkdfWithSha256 (1 2 840 113549 1 9 16 3 28)
       :           }
878   1:       INTEGER 16
881  11:       SEQUENCE {
883   9:         OBJECT IDENTIFIER
       :           aes128-wrap (2 16 840 1 101 3 4 1 5)
       :           }
894  24:       OCTET STRING
       :    C0 50 E4 39 2F 9C 14 DD 0A C2 22 02 03 F3 17 D7
       :    01 F9 4F 9D D9 27 78 F5
       :         }
       :       }
       :      }
920  58:    SEQUENCE {
922   9:     OBJECT IDENTIFIER data (1 2 840 113549 1 7 1)
933  30:     SEQUENCE {
935   9:       OBJECT IDENTIFIER
       :         aes128-GCM (2 16 840 1 101 3 4 1 6)
946  17:       SEQUENCE {
948  12:         OCTET STRING 5C A5 74 68 B8 1B F0 3B 8D A7 18 6C
962   1:         INTEGER 16
       :          }
       :         }
965  13:     [0] 94 C8 68 9A 99 D2 C3 8E 19 2F A6 BA 08
       :       }
980  16:    OCTET STRING
       :    5C F1 78 6C 57 C7 40 2B 54 FC 93 C3 0A 4A 45 33
       :     }
       :    }
       :   }
```

C.2. Recipient CMS Processing

Bob's ML-KEM-512 private key:


```
-----BEGIN PRIVATE KEY-----
MFQCAQAwCwYJYIZIAWUDBAQBBEKAQAABAgMEBQYHCAkKCwwNDg8QERITFBUWFxgZ
GhscHR4fICEiIyQlJicoKSorLC0uLzAxMjM0NTY3ODk6Ozw9Pj8=
-----END PRIVATE KEY-----
```


Bob decapsulates the ciphertext in the KEMRecipientInfo to get the ML-KEM-512 shared secret, encodes the CMSORIforKEMOtherInfo, derives the key-encryption key from the shared secret and the DER-encoded CMSORIforKEMOtherInfo using HKDF with SHA-256, uses AES-128-KEYWRAP to decrypt the content-encryption key with the key-encryption key, and decrypts the encrypted contents with the content-encryption key, revealing the plaintext content:

Hello, world!

Acknowledgements

This document borrows heavily from [RFC9690], [FIPS203], [RFC9935], and [IKEv2-MLKEM]. Thanks go to the authors of those documents. "Copying always makes things easier and less error prone." - [RFC8411].

Thanks to Carl Wallace, Jonathan Hammel, and Sean Turner for the detailed review and Carl Wallace and Philippe Cece for interoperability testing for the examples.

Authors' Addresses

Julien Prat
CryptoNext Security
16, Boulevard Saint-Germain
75005 Paris
France
Email: julien.prat@cryptonext-security.com

Mike Ounsworth
Entrust Limited
2500 Solandt Road -- Suite 100
Ottawa Ontario K2K 3G5
Canada
Email: mike.ounsworth@entrust.com

Daniel Van Geest
CryptoNext Security
16, Boulevard Saint-Germain
75005 Paris
France
Email: daniel.vangeest@cryptonext-security.com