RFC 9935 : Internet X.509 Public Key Infrastructure - Algorithm Identifiers for the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 7 mars 2026

Date de publication du RFC : Mars 2026

https://www.bortzmeyer.org/9935.html

Ce RFC spécifie comment mettre des clés ML-KEM dans un certificat X.509. Vous allez me dire « Mais ML-KEM est un algorithme d'échange de clés <https://www.bortzmeyer.org/echange-cles.html>, pas de signature, que fait-il dans un certificat ? » Et je vous répondrai que vous avez raison, d'ailleurs le RFC restreint sérieusement ce qu'on peut faire avec cet algorithme.

ML-KEM a l'intéressante propriété d'être résistant aux calculateurs quantiques. Il est normalisé dans FIPS 203 <https://doi.org/10.6028/nist.fips.203>. Ce RFC permet de l'utiliser dans les certificats normalisés dans le RFC 5280[1], et il décrit trois variantes, ML-KEM-512, ML-KEM-768 et ML-KEM-1024.

Comme indiqué plus haut, autant il était normal de permettre ML-DSA dans les certificats (RFC 9881), autant cela peut sembler surprenant pour ML-KEM. La section 1.1 de notre RFC explique les usages (limités) d'un algorithme d'échange de clés dans un certificat. C'est par exemple utile pour un protocole où vous avez besoin d'une clé publique, que vous allez tirer du certificat. Mais vous ne pourrez pas utiliser les certificats de ce RFC pour faire du TLS sur l'Internet public, aucune AC ne vous en produira, même si TLS le permet un jour. Pour la même raison, vous n'en trouverez pas dans les journaux "*Certificate Transparency*".

Les trois algorithmes sont identifiés en suivant le RFC 5912, par un OID via le NIST ; Ce sont `id-alg-ml-kem-512` (`2.16.840.1.101.3.4.4.1`), `id-alg-ml-kem-768` (`2.16.840.1.101.3.4.4.2`) et `id-alg-ml-kem-1024` (`2.16.840.1.101.3.4.4.3`).

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc5280.txt`

La section 5 du RFC précise que les bits indiquant les utilisations possibles de ces certificats (RFC 5280, section 4.2.1.3) doivent indiquer uniquement le chiffrement d'une clé (et pas l'authentification, comme avec les certificats habituels).

Le module ASN.1 figure dans l'annexe A, si vous voulez implémenter ce RFC (il faut aussi importer les modules des RFC 5912 et RFC 9629). Le module est identifié par `id-mod-x509-ml-kem-2025` et a l'OID `1.3.6.1.5.5.7.0.121`. Le gros du RFC est constitué d'exemples de clés et de certificats ML-KEM (annexe C), dans l'encodage en texte du RFC 7468.

Allez, un peu de pratique avec OpenSSL 3.5 :

```
% openssl genpkey -algorithm ML-KEM-512 -out ml-kem-key.pem -outform PEM

% cat ml-kem-key.pem
-----BEGIN PRIVATE KEY-----
MIIGvgIBADALBglghkgBZQMEBAEEggaqMIIGpgRAOwWCnP71SklfOIuwiLrIddsm
Up4AlOMwvwjqCy4PHr8fSq5jRjFoP6XYXnl8IQCR9HhzSRVffI09hBezOUrmxgSC
...

# Analysons cette clé :
% openssl pkey -text -in ml-kem-key.pem

# Demandons une signature du certificat :
% openssl req -new -key ml-kem-key.pem  -provider default -out req.csr
...
40E7CEC0027F0000:error:03000096:digital envelope routines:do_sigver_init:operation not supported for this ke
```

Ah, là, c'est raté, et c'est logique. OpenSSL n'accepte pas de fabriquer un CSR pour ces clés « *"operation not supported for this keytype"* » puisque ML-KEM ne permet pas de signer. La solution :

```
# Extraire la clé publique :
% openssl pkey -in ml-kem-key.pem -pubout   -out mlkem.pub

# Fabriquer une clé ECDSA pour signer :
% openssl ecparam -out ec_key.pem -name secp256r1 -genkey

# Signer :
% openssl x509 -new \
  -subj "/CN=test-mlkem" \
  -force_pubkey mlkem.pub \
  -signkey ec_key.pem \
  -days 365 \
    -out mlkem-cert.pem
Warning: Signature key and public key of cert do not match
```

L'avertissement est normal, puisqu'on a effectivement utilisé une autre clé, d'un autre algorithme, pour signer le certificat contenant notre clé publique ML-KEM. Mais tout s'est bien passé :

```
% openssl x509 -text -in mlkem-cert.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            69:33:f9:1e:e5:34:64:84:d9:f4:b2:15:3b:50:ec:36:db:0b:71:5c
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: CN=test-mlkem
        Validity
```

```
        Not Before: Feb 16 15:30:02 2026 GMT
        Not After : Feb 16 15:30:02 2027 GMT
    Subject: CN=test-mlkem
    Subject Public Key Info:
        Public Key Algorithm: ML-KEM-512
            ML-KEM-512 Public-Key:
            ek:
                d8:d2:6c:c4:9b:96:48:23:0d:ce:f7:0d:b3:0b:85:
                ...
    X509v3 extensions:
        X509v3 Subject Key Identifier:
            E7:B2:03:92:37:EC:C6:35:E4:F0:2E:AC:4E:1C:F2:81:67:F7:95:29
        X509v3 Authority Key Identifier:
            15:E5:2A:5E:B1:C3:50:69:7E:E4:48:EA:0F:DD:3C:5C:90:4B:7C:CC
Signature Algorithm: ecdsa-with-SHA256
```