RFC 9881 : Internet X.509 Public Key Infrastructure – Algorithm Identifiers for the Module-Lattice-Based Digital Signature Algorithm (ML-DSA)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 18 février 2026

Date de publication du RFC : Octobre 2025

https://www.bortzmeyer.org/9881.html

Vous reprendrez bien un peu de post-quantique ? Ce RFC normalise l'utilisation de signatures ML-DSA dans les certificats X.509.

ML-DSA est un des premiers gagnants du concours du NIST <https://www.bortzmeyer.org/nist-pq.html> (il se nommait Dilithium à l'époque). Il est normalisé dans FIPS-204 <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf>. Il utilise les réseaux euclidiens. C'est un algorithme de signature et il a donc toute sa place dans les certificats X.509, aux côtés des plus classiques RSA et ECDSA. Petit rappel de cryptographie post-quantique au passage : contrairement à RSA, ML-DSA ne sait faire que les signatures, pas l'échange de clés. Vous pouvez donc mettre des clés publiques ML-DSA dans un certificat, le signer avec ML-DSA mais, pour un protocole comme TLS, il faudra utiliser autre chose pour l'échange de clés (par exemple ML-KEM, autre vainqueur du concours NIST).

Mettre du post-quantique dans les certificats était moins urgent que dans l'échange de clés : ici, pas de risque qu'un attaquant prévoyant stocke des communications chiffrées, en attendant d'avoir un calculateur quantique pour les décrypter. Avec TLS, le risque est nul car la signature est au moment de la connexion, les calculateurs quantiques ne pourront usurper que les communications futures. Mais, bon, les certificats sont utilisés pour autre chose que TLS, et, comme il faut se préparer à un lent déploiement, il vaut mieux s'y prendre tout de suite.

Notez aussi qu'il n'y a pas que les certificats qui sont signés, les CRL le sont aussi, et ce RFC s'applique aussi à ces listes.

Des deux variantes normalisées dans FIPS-204, « pure » et « HashML-DSA », seule la première est utilisée dans notre RFC, pour les raisons qu'explique la section 8.3.

Le RFC utilise trois niveaux de sécurité différentes, et les identificateurs pour les trois niveaux sont `id-ml-dsa-44` (OID `2.16.840.1.101.3.4.3.17`, `id-ml-dsa-65` (OID `2.16.840.1.101.3.4.3.18` et `id-ml-dsa-87` (OID `2.16.840.1.101.3.4.3.19`. (Ces OID sont enregistrés par le NIST <`https://csrc.nist.gov/projects/computer-security-objects-register/algorithm-registration`>.) D'autre part, l'IANA a enregistré <`https://www.iana.org/assignments/smi-numbers/smi-numbers.xml#smi-numbers-1.3.6.1.5.5.7.0`> l'identificateur `id-mod-x509-ml-dsa-2025` (OID `1.3.6.1.5.5.7.0` pour le module ML-DSA (annexe A).

Un certificat X.509 contient des indications sur les utilisations permises (section 4.2.1.3 du RFC 5280 [1]). Comme ML-DSA sait faire des signatures mais pas des échanges de clés, un certificat avec ML-DSA aura les utilisations de signature, comme `keyCertSign` mais pas celles d'échange de clés comme `keyEncipherment`.

Le format de la clé privée (section 6) a été un des sujets de discussions chauds à l'IETF (et chez les implémenteurs <`https://openssl-library.org/post/2025-01-21-blog-positionandplans/`>). Dans la norme FIPS-204, on peut définir une clé privée de deux façons, une optimisée (section 4 de la norme) où on ne stocke que la graine (notée [Caractère Unicode non montré [2]]) ou bien une forme longue avec la graine, les éléments de la clé secrète, etc. On peut déduire la clé de la graine (section 6.1 de la norme), mais pas l'inverse donc si vous ne gardez pas la graine, vous pourrez toujours signer mais pas retrouver la graine. Le RFC permet de stocker la clé privée de trois façons, la graine seule (la méthode recommandée, cf. section 8.1), la clé seule (déconseillé) ou les deux (voir un exemple plus loin).

Je n'ai pas trouvé de certificat utilisant ML-DSA dans la nature. `crt.sh` ne permet pas de chercher par algorithme, même dans ses fonctions avancées. Il faudrait écrire son propre client d'examen des journaux "*Certificate Transparency*" (RFC 9162) et j'avais la flemme. Donc, on va utiliser OpenSSL 3.5 pour fabriquer des certificats (je n'ai pas testé pour voir s'ils étaient parfaitement conformes au RFC…).

```
% openssl req -new -newkey mldsa44 -keyout mldsa.key -out mldsa.csr -nodes -subj "/CN=test"
% openssl x509 -in mldsa.csr -out mldsa.crt -req -signkey mldsa.key -days 2001
Certificate request self-signature ok
subject=CN=test
```

Et hop, nous voilà avec un beau certificat :

```
% openssl x509 -text -in mldsa.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            54:5b:c0:5d:0e:85:83:43:90:77:93:4f:53:83:e1:38:a2:12:9f:61
        Signature Algorithm: ML-DSA-44
        Issuer: CN=test
        Validity
            Not Before: Feb 17 15:01:53 2026 GMT
```

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc5280.txt`
2. Car trop difficile à faire afficher par LaTeX

```
            Not After : Aug 11 15:01:53 2031 GMT
        Subject: CN=test
        Subject Public Key Info:
            Public Key Algorithm: ML-DSA-44
                ML-DSA-44 Public-Key:
                pub:
                    15:10:02:cc:d6:ec:53:12:bb:6d:34:23:82:65:ec:
                    ...
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                6E:F9:AD:56:9C:AA:60:EF:ED:B7:A1:7B:70:F6:71:55:74:B8:68:B9
    Signature Algorithm: ML-DSA-44
...
```

Autre solution, de plus bas niveau, en découpant les étapes :

```
# Créer la clé privée
openssl genpkey -algorithm ML-DSA-44 -out private.pem

# Extraire la clé publique (mais on ne s'en servira pas ici)
openssl pkey -in private.pem -pubout -out public.pem

# Créer la demande de signature du certificat
openssl req -new -subj /CN=Test -key private.pem -out cert.csr

# (Auto-)Signer
openssl x509 -req -in cert.csr -signkey private.pem -out cert.pem
Certificate request self-signature ok
subject=CN=Test

# Vérifier
openssl x509 -text -in cert.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            7c:65:cb:17:5b:ed:ee:08:c9:1a:6b:a5:97:da:ac:b8:0e:6e:df:cd
        Signature Algorithm: ML-DSA-44
        Issuer: CN=Test
        Validity
            Not Before: Feb 18 13:24:05 2026 GMT
            Not After : Mar 20 13:24:05 2026 GMT
        Subject: CN=Test
        Subject Public Key Info:
            Public Key Algorithm: ML-DSA-44
                ML-DSA-44 Public-Key:
   ...
   Signature Algorithm: ML-DSA-44
...
```

Et si vous voulez utiliser des extensions comme la restriction d'utilisation du certificat (ici, on se limite à la signature), remplacez la demande de signature et la signature par :

```
openssl req -new -subj /CN=Test -key private.pem -out cert.csr -addext "keyUsage=digitalSignature"
openssl x509 -req -in cert.csr -signkey private.pem -out cert.pem  -copy_extensions copy
```

Si vous voulez regarder ce qu'il y a dans la clé privée :

```
%  openssl pkey -text -in private.pem
...
ML-DSA-44 Private-Key:
seed:
    48:c8:e4:e3:63:16:c0:e4:57:da:22:31:94:36:98:
    ..
priv:
    d9:0b:5b:8d:18:4f:61:8d:c0:8f:85:6c:97:28:46:
    ...
```

Vous voyez que sont stockés graine et clé (et la clé publique, plus loin).

Pour wolfSSL, vous pouvez regarder ici <https://github.com/wolfSSL/wolfCLU/issues/189>.

Pour des « vrais » certificats, notez que Digicert en fait apparemment <https://docs.digicert.com/en/trust-lifecycle-manager/enroll-and-manage-certificates/post-quantum-cryptograph issue-pqc-mldsa-dilithium-certificates.html> ainsi que l'AC pour usage privé d'AWS. Vous connaissez d'autres AC qui permettraient ML-DSA ?

Sinon, l'annexe C du RFC comprend de nombreux exemples de clés et de signatures.