RFC 9980 : Post-Quantum Cryptography in OpenPGP

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 juillet 2026

Date de publication du RFC : Juin 2026

https://www.bortzmeyer.org/9980.html

Vous le savez, le jour où des CRQC ("*Cryptographically Relevant Quantum Computer*", un calculateur quantique capable de calculs non triviaux, contrairement aux modèles d'aujourd'hui) seront disponibles, la cryptographie sera sérieusement secouée. Il est donc important de travailler dès maintenant sur des algorithmes pour l'après-quantique, et de les intégrer dans les protocoles et les formats utilisés sur l'Internet. Ce RFC documente l'utilisation des algorithmes normalisés par le NIST dans le format OpenPGP.

Le format OpenPGP, utilisé par de nombreux logiciels de cryptographie, est normalisé dans le RFC 9580 [1]. La liste des algorithmes de chiffrement n'est pas figée et de nouveaux algorithmes peuvent être disponibles pour ce format. C'est le cas de ceux pour la cryptographie post-quantique, un sujet d'actualité. Les messages chiffrés et/ou signés au format OpenPGP peuvent avoir besoin de résister à la décryption et/ou à l'usurpation pendant de nombreuses années. Aujourd'hui, ces messages utilisent typiquement RSA ou des algorithmes à courbes elliptiques, tous étant vulnérables aux calculateurs quantiques. C'est notamment en raison de cette nécessité de sécurité sur une longue période qu'il ne faut pas attendre que les CRQC soient disponibles pour intégrer les algorithmes post-quantiques à OpenPGP. Il y a peut-être des attaquants qui stockent aujourd'hui des messages OpenPGP, en attendant d'avoir un CRQC pour les lire (ou pour imiter des signatures).

(Rappelons au passage qu'OpenPGP n'est pas utilisé que dans le courrier électronique. Il sert, par exemple, à authentifier le code avec git, ou les paquetages compilés, avec apt et rpm.)

Il ne suffit pas d'ajouter les nouveaux algorithmes aux registres <https://www.iana.org/assignments/openpgp/openpgp.xml#openpgp-public-key-algorithms> IANA. Il y a des problèmes spécifiques, comme les clés hybrides (une PQ et une classique) et composées (hybrides, mais présentées d'une manière unifiée). En effet, il ne servirait à rien de déployer des algorithmes post-quantiques si ceux-ci

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc9580.txt

étaient cassables par de la cryptanalyse classique. Rien ne dit que ces « nouveaux » algorithmes soient incassables. Et comme ils sont relativement récents, on ne peut pas avoir le même degré de confiance qu'avec RSA ou ECDSA. L'approche la plus courante aujourd'hui, et que ce RFC suit, est d'utiliser une technique hybride : combinaison d'un algorithme traditionnel et d'un algorithme post-quantique. On n'abandonne donc pas RSA ou ECDSA, on les flanque d'un collègue, ce qu'on appelle le PQ/T (post-quantique/traditionnel, cf. section 1.1.1).

Plus précisément, notre RFC utilise des **composés**, des hybrides PQ/T mais où les deux clés, la post-quantique et la traditionnelle, sont gérées comme une seule. Le RFC 9794 est la bonne lecture, si vous voulez approfondir ces notions d'hybride et de composé et vous avez aussi intérêt à lire le RFC 9958, « "*Post-Quantum Cryptography for Engineers*" ».

Quels sont ces nouveaux algorithmes ? La section 1.2 les résume :

— ML-KEM, normalisé dans la norme FIPS-203 <https://csrc.nist.gov/pubs/fips/203/final>, pour l'échange de clés <https://www.bortzmeyer.org/echange-cles.html> préalable au chiffrement,
— ML-DSA, normalisé dans FIPS-204 <https://csrc.nist.gov/pubs/fips/204/final>, pour la signature,
— SLH-DSA, normalisé dans FIPS-205 <https://csrc.nist.gov/pubs/fips/205/final>, également pour la signature.

Notez que l'algorithme SLH-DSA, lui, est considéré suffisamment sûr pour se passer de l'assistance d'un algorithme traditionnel (il utilise des problèmes mathématiques complètement différents de ceux utilisés par ML-KEM ou ML-DSA). Les deux autres vont être utilisés par OpenPGP avec de la cryptographie traditionnelle, en l'occurrence ECDH avec les courbes X25519 et X448 (RFC 7748) et EdDSA (RFC 8032). Pour la vérification d'une signature, les deux (post-quantique et traditionnelle) signatures doivent être valides (cf. sections 3 et 5.2.3). Pour le chiffrement, les deux clés obtenues doivent être utilisées.

Le format OpenPGP permet d'avoir plusieurs signatures dans un message mais ces signatures parallèles sont différentes des clés composées utilisées pour le PQ/T car le succès d'une seule signature suffit à la validation. (Idem pour le chiffrement, cf. section 3.) Évidemment, le système n'est résistant aux CRQC que si toutes les signatures utilisent un algorithme PQ ou PQ/T. Si ces signatures multiples, incluant au moins une clé T (traditionnelle, sans post-quantique) sont moins sûres, elles ont par contre l'avantage d'assurer la compatibilité avec les vieilles versions des logiciels OpenPGP (section 5.2.5 du RFC 9580).

La section 2 du RFC donne la liste exhaustive des algorithmes qui viennent d'être officiellement ajoutés. À partir des trois cités plus haut, il y a quelques variantes, fondées sur la taille de certains paramètres ou sur la courbe elliptique utilisée dans le composé. Ainsi, SLH-DSA a trois variantes, SLH-DSA-SHAKE-128f (f pour "*fast*" car il optimise la vitesse), SLH-DSA-SHAKE-128s (s pour "*short*" car il optimise la taille) et SLH-DSA-SHAKE-256s. ML-KEM a deux variantes, ML-KEM-768+X25519 et ML-KEM-1024+X448, avec des courbes différentes.

Notez enfin que les clés PQ/T ne doivent être utilisées qu'avec des données OpenPGP des versions 4 ou 6 (et même uniquement version 6 pour ML-KEM-1024+X448 et ML-DSA). Ici, par exemple, GnuPG montre un paquet OpenPGP de version 3, trop vieux pour gérer le post-quantique :

```
% gpg --list-packets review.txt.gpg
...
:pubkey enc packet: version 3, algo 1, keyid XXXXXXXXX
 data: [4096 bits]
```

Les sections 4, 5 et 6 du RFC expliquent en détail le format des nouvelles clés et comment les utiliser. La section 7 donne des conseils sur les algorithmes de cryptographie symétrique, par exemple qu'il est nécessaire de mettre en œuvre AES-256 (la version à 128 bits est possiblement cassable grâce à l'algorithme de Grover).

Et la migration depuis les anciens algorithmes ? Tous les logiciels qui mettent en œuvre OpenPGP ne vont pas passer au post-quantique en même temps. On aura des messages qui vont passer d'un logiciel récent à un ancien, qui ne pourra pas les lire. La section 8 ajoute des conseils pour bien réussir sa migration. Déjà, un logiciel récent, qui pense que les récepteurs de ses messages seront pré-quantiques peut chiffrer ses messages avec une clé PQ (ou PQ/T) et une clé traditionnelle (chiffrement en parallèle, où une seule clé est nécessaire, et pas en série, comme c'est le cas ave les solutions hybrides citées plus haut, où les deux clés sont nécessaires). Bien sûr, s'il fait cela, le message sera déchiffrable par un calculateur quantique. Il faut donc choisir entre sécurité et interopérabilité (avec les vieux logiciels). PGP étant conçu pour des communications asynchrones (comme le courrier électronique), il n'est pas possible de savoir à l'avance les capacités du récepteur.

Le même problème se pose pour les signatures. Lors d'une vérification de signature, n'importe laquelle des deux signatures sera acceptée (là encore, on parle de signatures séparées, qui ont toujours existé dans OpenPGP, pas des hybrides du PQ/T). Le RFC permet toutefois à un vérificateur paranoïaque, ou simplement un vérificateur qui sait que l'émetteur a une clé PQ ou PQ/T, d'ignorer les signatures traditionnelles.

Enfin, la section 9 du RFC discute un certain nombre de questions de sécurité. Par exemple, elle explique comment les signatures composites du PQ/T ne sont pas vulnérables aux attaques par suppression d'une des signatures (les métadonnées indiquent l'identificateur de l'algorithme hybride).

Quelles sont les mises en œuvre de ces nouveaux algorithmes ? Malheureusement, il semble que GnuPG ne suive pas les récents RFC sur le format OpenPGP <https://lwn.net/Articles/1053089/> (sur cette affaire, lire le point de vue de Debian <https://alioth-lists.debian.net/pipermail/pkg-gnupg-maint/2026-January/010450.html> ou celui d'Arch Linux <https://wiki.archlinux.org/title/GnuPG>), entre autre (mais pas uniquement) sur le post-quantique. On va donc tester avec les autres (il existe une liste <https://sequoia-pgp.gitlab.io/openpgp-interoperability-test-suite/results.html?q=pqc>).

Si vous voulez écrire votre propre programme OpenPGP (ce que je ne conseillerai pas : la cryptographie, c'est difficile, et les bogues ne se voient pas forcément), l'annexe A du RFC, qui fait la grande majorité du RFC, est composée de vecteurs de test.

Les programmes testés sont conformes au projet de standard SOP <https://www.openpgp.org/about/sop/> et ont donc à peu près la même interface utilisateur (actuellement en projet, dans draft-dkg-openpgp-st
Souvent, le post-quantique n'est pas encore intégré dans les versions officielles et il va falloir changer de branche et compiler.

Commençons avec rsop <https://docs.rs/pgp/latest/pgp/>, écrit en Rust. Après le cargo install pgp :

```
% rsop list-profiles  generate-key
default: v4 key using Curve25519 (alias: draft-koch-eddsa-for-openpgp-00)
compatibility: v4 key using RSA (alias: rfc4880)
performance: v6 key using Ed25519/X25519 (alias: rfc9580)
security: v6 key using Ed448/X448 (alias: rfc9580-curve448)
```

Je vous l'avais bien dit : pas de post-quantique. Mais la FAQ dans le code nous le dit « *"### Is rPGP adding support for Post Quantum Cryptography (PQC)? Yes, rPGP implements the IETF draft [Post-Quantum Cryptography in OpenPGP](https ://datatracker.ietf.org/doc/draft-ietf-openpgp-pqc/), gated behind the feature [Caractère Unicode non montré [2]]draft-pqc[Caractère Unicode non montré]."* ». Ah, c'est planqué dans une "*feature*". Compilons :

```
% cargo install --features draft-pqc rsop

% rsop list-profiles  generate-key
default: v4 key using Curve25519 (alias: draft-koch-eddsa-for-openpgp-00)
compatibility: v4 key using RSA (alias: rfc4880)
performance: v6 key using Ed25519/X25519 (alias: rfc9580)
security: v6 key using Ed448/X448 (alias: rfc9580-curve448)
draft-ietf-openpgp-pqc-14-v4-ed25519-mlkem768x25519: TESTING ONLY
draft-ietf-openpgp-pqc-14-v6-ed25519-mlkem768x25519: TESTING ONLY
draft-ietf-openpgp-pqc-14-v6-mldsa65ed25519-mlkem768x25519: TESTING ONLY
draft-ietf-openpgp-pqc-14-v6-mldsa87ed448-mlkem1024x448: TESTING ONLY
draft-ietf-openpgp-pqc-14-v6-slhdsashake128s-mlkem768x25519: TESTING ONLY
draft-ietf-openpgp-pqc-14-v6-slhdsashake128f-mlkem768x25519: TESTING ONLY
draft-ietf-openpgp-pqc-14-v6-slhdsashake256s-mlkem1024x448: TESTING ONLY
```

Ça marche, on a du post-quantique, utilisons-le :

```
% rsop generate-key --profile draft-ietf-openpgp-pqc-14-v6-ed25519-mlkem768x25519 > key.asc
```

OK, on a une clé hybride (ML-KEM et Ed25519). Créeons la clé publique :

```
% cat key.asc | rsop extract-cert  > cert.asc
```

Et maintenant, on peut chiffrer un fichier avec la clé publique :

```
% cat hello.txt | rsop  encrypt cert.asc > hello.asc
```

Et le déchiffrer avec la clé privée :

```
% cat hello.asc | rsop decrypt key.asc
Hello, world
```

Bon, on a tout fait avec le même programme, mais le but d'OpenPGP est l'interopérabilité. Essayons avec un deuxième programme, Sequoia, pour lequel il y a des instructions détaillées pour le compiler avec gestion du post-quantique <https://sequoia-pgp.org/blog/2025/11/15/202511-post-quantum-cr> :

2. Car trop difficile à faire afficher par LaTeX

```
% sudo apt install libsqlite3-dev
% git clone https://gitlab.com/sequoia-pgp/sequoia-sq.git
% git checkout pqc
% cargo build --release --locked --no-default-features --features crypto-openssl
```

Et utilisons-le pour regarder les fichiers produits par rsop :

```
% sq  inspect key.asc
key.asc: Transferable Secret Key.

      Fingerprint: AC7FD6CBD09E90C928C2ED5796A69001B5F0535CE6BF09C227DFE71063006BD0
  Public-key algo: Ed25519
  Public-key size: 256 bits
       Secret key: Unencrypted
    Creation time: 2026-02-09 17:36:25 UTC
        Key flags: certification, signing

           Subkey: 91BED065B7E654656D3354CA16E901D8F8B192874385EB6C325421055AFB5B9E
  Public-key algo: ML-KEM-768+X25519
       Secret key: Unencrypted
    Creation time: 2026-02-09 17:36:25 UTC
        Key flags: transport encryption, data-at-rest encryption
```

Joli, non ? rsop avait bien fabriqué une clé hybride et sq arrive à la lire.

```
% cat hello.asc | sq  decrypt --recipient-file key.asc
Encrypted and protected using AES-256/OCB
Hello, world
Decrypted by AC7FD6CBD09E90C928C2ED5796A69001B5F0535CE6BF09C227DFE71063006BD0, unknown
0 authenticated signatures.
```

Et Sequoia peut déchiffrer les messages chiffrés par rsop.

```
% sq inspect --cert-file cert.asc < hello.asc
OpenPGP Certificate.

      Fingerprint: AC7FD6CBD09E90C928C2ED5796A69001B5F0535CE6BF09C227DFE71063006BD0
  Public-key algo: Ed25519
  Public-key size: 256 bits
    Creation time: 2026-02-09 17:36:25 UTC
        Key flags: certification, signing

           Subkey: 91BED065B7E654656D3354CA16E901D8F8B192874385EB6C325421055AFB5B9E
  Public-key algo: ML-KEM-768+X25519
    Creation time: 2026-02-09 17:36:25 UTC
        Key flags: transport encryption, data-at-rest encryption
```

Et examiner ses clés. rsop et sq étaient tous les deux écrits en Rust donc testons l'interopérabilité avec un programme en Go :

```
% git clone https://github.com/ProtonMail/gosop.git
% cd gosop
% git checkout gosop-gopenpgp-v3-pqc
% go build

% ./gosop list-profiles generate-key
default: Generate v4 keys using Curve25519
compatibility: Generate v4 keys using 3072-bit RSA (alias: rfc4880)
performance: Generate v6 keys using Ed25519/X25519 (alias: rfc9580)
security: Generate v6 keys using Ed448/X448
draft-ietf-openpgp-pqc-09: ML-KEM-768 and ML-DSA-65
draft-ietf-openpgp-pqc-09-high-security: ML-KEM-1024 and ML-DSA-87
draft-ietf-openpgp-persistent-symmetric-keys-00: AEAD and HMAC

% ./gosop/gosop decrypt key.asc < hello.asc
Hello, world
```

Et tout se passe bien.