Guide Implémentation NIST PQC : ML-KEM et ML-DSA 2026

> Guide technique NIST PQC : implémentation ML-KEM (CRYSTALS-Kyber) et ML-DSA (Dilithium). Bibliothèques, code, intégration TLS et migration des systèmes.

ML-KEM (FIPS 203, ex-CRYSTALS-Kyber) et **ML-DSA** (FIPS 204, ex-CRYSTALS-Dilithium) sont les algorithmes post-quantiques standardisés par le NIST pour l'échange de clés et les signatures numériques. L'implémentation pratique s'appuie sur liboqs pour le prototypage et BouncyCastle 1.80+ ou AWS libcrypto pour la production. L'hybridation X25519+ML-KEM est la stratégie recommandée pendant la transition, déjà supportée nativement par Chrome 131+ et les dernières versions d'OpenSSL.

Les algorithmes CRYSTALS-Kyber (standardisé ML-KEM, FIPS 203) et CRYSTALS-Dilithium (standardisé ML-DSA, FIPS 204) constituent le cœur de la réponse du NIST à la menace quantique. Sélectionnés après 8 années de compétition internationale et finalisés en août 2024, ils représentent les primitives post-quantiques recommandées pour respectivement l'encapsulation de clés (Key Encapsulation Mechanism) et les signatures numériques. Ce guide technique détaille les étapes concrètes pour implémenter ces algorithmes dans vos systèmes : choix des bibliothèques, intégration TLS, gestion des clés, considérations de performance, et pièges à éviter. L'objectif est de donner aux équipes techniques les outils pour passer de la théorie PQC à une implémentation en production sécurisée et vérifiable.

À RETENIR

À retenir

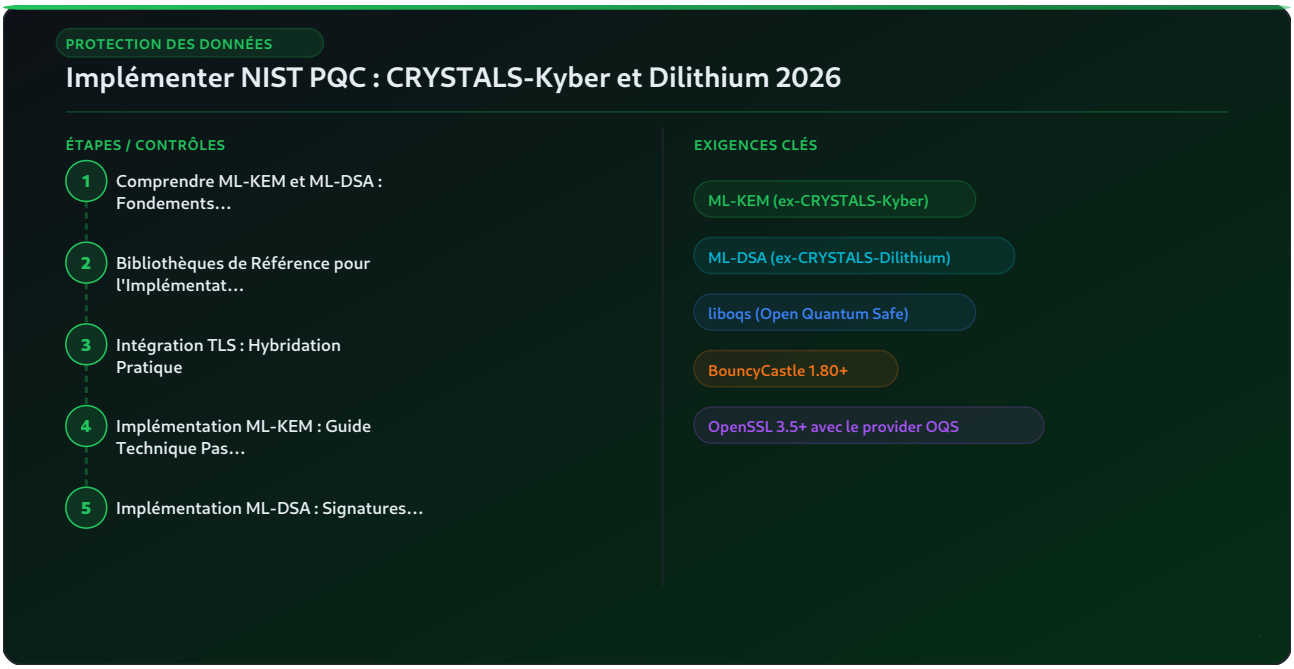
ML-KEM (FIPS 203) pour l'échange de clés, ML-DSA (FIPS 204) pour les signatures — standards NIST finalisés août 2024

Bibliothèques recommandées : liboqs (prototypage/intégration), BouncyCastle 1.80+ (Java enterprise), OpenSSL 3.5+ avec provider OQS (C/C++ TLS)

Stratégie TLS : hybridation X25519MLKEM768, supportée nativement par Chrome 131+ et Firefox 132+

Performance ML-KEM-768 acceptable (~60-65 µs) ; impact principal sur la taille des handshakes (+1,5 KB) plutôt que sur le CPU

Utiliser le pattern KEM+DEM (ML-KEM + AES-256-GCM) pour le chiffrement de données at-rest



Comprendre ML-KEM et ML-DSA : Fondements Mathématiques Accessibles

ML-KEM (Module-Lattice-based Key Encapsulation Mechanism) et ML-DSA (Module-Lattice-based Digital Signature Algorithm) reposent tous deux sur la dureté du problème Module Learning With Errors (MLWE), une variante du problème Learning With Errors (LWE) opérant sur des modules de polynômes. La sécurité de ces schémas est fondée sur la difficulté de distinguer des vecteurs de réseau proches de vecteurs aléatoires — un problème pour lequel aucun algorithme quantique connu n'offre d'avantage exponentiel.

ML-KEM (ex-CRYSTALS-Kyber) implémente un KEM en trois opérations : KeyGen() génère une paire de clés (pk, sk), Encaps(pk) génère un texte chiffré et une clé symétrique partagée K, Decaps(sk, ct) retrouve la clé symétrique K. La clé publique fait 1 184 octets (ML-KEM-768), le texte chiffré 1 088 octets, et la clé partagée 32 octets — significativement plus volumineux qu'un échange X25519 (32 octets pour la clé publique), mais avec une sécurité quantique garantie.

ML-DSA (ex-CRYSTALS-Dilithium) implémente un schéma de signature en trois opérations classiques : KeyGen(), Sign(sk, message), Verify(pk, message, signature). La clé publique ML-DSA-65 fait 1 952 octets, la clé privée 4 000 octets, et la signature 3 293 octets — à comparer aux 64 octets d'une signature Ed25519. Cette augmentation de taille a des implications directes sur les protocoles de handshake TLS et la performance réseau.

Bibliothèques de Référence pour l'Implémentation PQC

Le choix de la bibliothèque cryptographique est critique : n'implémentez jamais ces algorithmes from scratch. Les implémentations de référence et bibliothèques auditées sont les seules options acceptables en production.

⚡ **Retour terrain**

Lors d'un accompagnement RGPD pour une PME de 120 personnes dans le secteur des ressources humaines, j'ai découvert que leurs bases de données de candidatures contenaient des données datant de 2011 — bien au-delà des durées de conservation légales. La purge des données expirées a représenté 78 % du volume initial. La mise en place d'une politique de rétention automatisée (archivage à 2 ans, suppression à 3 ans pour les candidats non retenus) a finalement été le changement le plus impactant de la mission.

liboqs (Open Quantum Safe) : La bibliothèque de référence pour le prototypage et l'intégration. Maintenue par le projet Open Quantum Safe (Linux Foundation), liboqs fournit des implémentations C optimisées de ML-KEM, ML-DSA et SLH-DSA, avec des bindings Python (pyoqs), Go (go-oqs), et Java (liboqs-java). La bibliothèque est activement auditée mais se positionne comme "expérimentale" — à utiliser avec OQS-OpenSSL pour les intégrations TLS.

BouncyCastle 1.80+ : La bibliothèque Java/C# la plus complète pour les environnements enterprise. BC 1.80 inclut des implémentations de ML-KEM (Kyber), ML-DSA (Dilithium) et SLH-DSA (SPHINCS+) conformes aux FIPS drafts finaux. C'est la bibliothèque recommandée pour les applications Java en production, notamment parce qu'elle s'intègre nativement avec le JCA (Java Cryptography Architecture).

OpenSSL 3.5+ avec le provider OQS : OpenSSL 3.5 intègre nativement ML-KEM et ML-DSA via son architecture de providers. Pour les versions antérieures (3.x), le provider OQS (`oqsprovider`) ajoute le support PQC sans recompiler OpenSSL. C'est la solution recommandée pour les intégrations TLS en C/C++.

Google BoringSSL : Utilisé par Chrome, Android et de nombreux produits Google, BoringSSL supporte ML-KEM-768 en hybridation avec X25519 (X25519MLKEM768) depuis 2024. Pour les organisations utilisant les bibliothèques Google, c'est une option mature pour les handshakes TLS.

Pour une perspective sur l'architecture globale intégrant PQC, notre guide Cryptographic Agility à l'ère post-quantique offre le cadre architectural dans lequel intégrer ces implémentations.

Intégration TLS : Hybridation Pratique

L'intégration TLS est le cas d'usage le plus immédiatement déployable. La stratégie recommandée est l'hybridation : combiner ML-KEM avec X25519 pour l'échange de clés, fournissant une sécurité classique ET post-quantique simultanément.

Avec OpenSSL 3.5 et le provider OQS, la configuration d'un serveur TLS supportant les groupes hybrides se fait via :

```
ssl_conf_cmd Groups X25519MLKEM768:P-256MLKEM768:x25519
```

Cette configuration annonce en priorité X25519MLKEM768 (hybride post-quantique) et se replie sur X25519 classique si le client ne supporte pas le groupe hybride. Les navigateurs Chrome 131+ et Firefox 132+ supportent X25519MLKEM768 nativement, ce qui signifie que vos connexions HTTPS avec ces navigateurs sont déjà hybrides si votre serveur est configuré correctement.

Pour les connexions entre services internes (microservices, APIs), la bibliothèque Python **pyoqs** combinée avec **ssl** ou **httpx** permet de forcer les suites hybrides. En Go, le package **cloudflare/circl** fournit ML-KEM implémenté en Go pur avec des tests vectors NIST, compatible avec le package standard `crypto/tls`.

Un point de vigilance TLS : les certificats serveur utilisent encore ECDSA ou RSA pour les signatures, même avec un échange de clés ML-KEM. La migration des certificats vers ML-DSA est plus complexe car elle implique les PKI, les CA intermédiaires, et les navigateurs. Suivre les avancées du CA/Browser Forum sur le support des certificats PQC est essentiel pour planifier cette migration.

Implémentation ML-KEM : Guide Technique Pas à Pas

Pour une implémentation ML-KEM en Python avec pyoqs, le flux typique d'un échange de clés sécurisé post-quantique est le suivant :

L'initiateur (Alice) génère une paire de clés ML-KEM-768 et envoie la clé publique à Bob. Bob encapsule une clé secrète aléatoire avec la clé publique d'Alice (générant un ciphertext) et envoie ce ciphertext à Alice. Alice décapsule le ciphertext avec sa clé privée pour retrouver la même clé secrète. Les deux parties dérivent ensuite des clés de session via HKDF à partir de cette clé partagée de 32 octets.

Les paramètres ML-KEM recommandés selon le niveau de sécurité visé : **ML-KEM-512** (niveau 1, équivalent AES-128, clé publique 800 octets), **ML-KEM-768** (niveau 3, équivalent AES-192, recommandé pour usage général, clé publique 1 184 octets), **ML-KEM-1024** (niveau 5, équivalent AES-256, pour données ultra-sensibles, clé publique 1 568 octets). Le niveau 3 (ML-KEM-768) est le choix recommandé par défaut pour les déploiements enterprise.

Implémentation ML-DSA : Signatures Post-Quantiques

ML-DSA (CRYSTALS-Dilithium) pour les signatures numériques présente des défis d'intégration différents de ML-KEM, principalement liés à la taille des signatures (2 420 à 4 595 octets selon le niveau) qui impacte les protocoles habitués aux signatures compactes (64 octets pour Ed25519).

Les cas d'usage prioritaires pour ML-DSA incluent : la signature de code et de firmware (où la taille de signature est moins critique que la sécurité à long terme), les documents juridiques et contrats électroniques, les tokens d'authentification à longue durée de vie (certificats), et les journaux d'audit immuables nécessitant une vérifiabilité à 20+ ans.

Pour JWT (JSON Web Tokens) avec ML-DSA, la RFC 9053 définit les algorithmes COSE pour les algorithmes PQC, mais les bibliothèques JWT populaires n'ont pas encore intégré ML-DSA nativement. Des alternatives comme **PASETO v4** offrent une approche plus moderne, et des

adaptateurs ML-DSA pour les bibliothèques JWT existantes émergent dans la communauté open-source.

La gestion du cycle de vie des clés ML-DSA mérite attention : les clés privées ML-DSA-65 font 4 000 octets contre 32 octets pour Ed25519. Les HSM et KMS doivent supporter ces nouvelles tailles. AWS KMS a ajouté le support ML-DSA en preview en 2024, Azure Key Vault et Google Cloud KMS suivent des roadmaps similaires pour 2025-2026.

Performance et Optimisation : Ce qu'il Faut Savoir

Les implémentations de référence de ML-KEM et ML-DSA sont comparables en performance aux algorithmes classiques sur les CPUs modernes, grâce aux optimisations AVX2 et AVX-512. Les benchmarks de liboqs montrent ML-KeyGen à ~50 µs, ML-Encaps à ~60 µs, et ML-Decaps à ~65 µs sur un Core i7 récent — à comparer aux ~30 µs pour ECDH P-256. L'overhead est acceptable pour la grande majorité des use cases.

L'impact principal n'est pas CPU mais réseau : les clés publiques et ciphertexts plus volumineux augmentent la taille des handshakes TLS. Une augmentation de ~1,5 KB par handshake TLS est typique avec ML-KEM-768, ce qui est négligeable sur les connexions à haut débit mais peut impacter les environnements IoT ou à bande passante contrainte. Dans ces cas, ML-KEM-512 ou l'hybridation légère (X25519+ML-KEM-512) offre un meilleur équilibre.

Pour les environnements embarqués et IoT, des implémentations optimisées mémoire comme **PQClean** et **PQCRYPTO-SIDH** fournissent des footprints réduits au prix d'une performance moindre. La sélection de l'implémentation appropriée selon les contraintes de l'environnement cible est une décision architecturale clé.

Notre analyse de l'attaque Harvest Now Decrypt Later contextualise les priorités de migration par niveau de risque, ce qui guide le dimensionnement des ressources allouées à l'implémentation PQC.

Testing, Validation et Conformité FIPS

La validation des implémentations PQC repose sur les Known Answer Tests (KAT) fournis par le NIST. Chaque bibliothèque sérieuse fournit une suite de tests vérifiant la conformité aux vecteurs de test officiels. L'intégration de ces tests dans les pipelines CI/CD est non-négociable pour maintenir la conformité après chaque mise à jour de bibliothèque.

Pour les organisations soumises aux exigences FIPS 140-3, les modules validés ML-KEM et ML-DSA sont en cours de certification via le programme CMVP du NIST. Les premières certifications de modules PQC sont attendues fin 2025. En attendant, les implémentations basées sur BouncyCastle FIPS (BC-FJA) ou les HSM certifiés FIPS avec support PQC offrent la meilleure posture de conformité disponible.

Les tests de régression cryptographique doivent couvrir : la conformité aux vecteurs KAT NIST, l'interopérabilité entre bibliothèques différentes (BouncyCastle avec liboqs, OpenSSL avec wolfSSL), la gestion correcte des erreurs (ciphertexts invalides, clés corrompues), et les tests de performance sous charge.

FAQ : Implémentation NIST PQC

ML-KEM est-il rétrocompatible avec les systèmes existants utilisant ECDH ?

ML-KEM n'est pas directement rétrocompatible avec ECDH car ce sont des primitives différentes (KEM vs DH). La compatibilité est assurée par la négociation TLS : le serveur et le client négocient le groupe de clés supported, permettant un fallback vers ECDH si l'une des parties ne supporte pas ML-KEM. L'hybridation (X25519+ML-KEM) est la stratégie de transition qui maximise la compatibilité.

Peut-on utiliser ML-KEM pour le chiffrement de données at-rest ?

ML-KEM est un KEM, pas un algorithme de chiffrement direct. Pour le chiffrement at-rest, le flux recommandé est : utiliser ML-KEM pour encapsuler une clé AES-256, puis utiliser AES-256-GCM pour chiffrer les données. C'est le pattern KEM+DEM (Data Encapsulation Mechanism) standardisé par HPKE (RFC 9180).

Quelle est la différence entre ML-DSA et SLH-DSA (SPHINCS+) pour les signatures ?

ML-DSA (lattice-based) offre des signatures plus compactes et des performances CPU supérieures, mais repose sur des hypothèses mathématiques de résistance des réseaux euclidiens. SLH-DSA (hash-based) offre une sécurité basée uniquement sur la robustesse des fonctions de hachage — hypothèse plus conservative et mieux comprise. SLH-DSA est recommandé pour les applications nécessitant une sécurité maximale à très long terme (50+ ans), malgré des signatures plus volumineuses (7 856 à 49 856 octets).

Ressources complémentaires : migration post-quantique, quantum machine learning et fondamentaux PQC.

Sources : NIST FIPS 203 ML-KEM | NIST FIPS 204 ML-DSA

Guide pas à pas : intégrer ML-KEM dans une application TLS existante

L'intégration de ML-KEM (ex-CRYSTALS-Kyber, standardisé en FIPS 203) dans une application TLS existante est l'une des transitions les plus impactantes et les plus documentées de la migration PQC. Cette section détaille le processus technique pour une application basée sur OpenSSL, bibliothèque la plus répandue dans les environnements Linux/serveur.

Prérequis : OpenSSL 3.2+ avec le provider OQS (Open Quantum Safe), ou utilisation directe de liboqs. Pour une approche hybride recommandée, on utilisera les groupes hybrides X25519MLKEM768 ou P256MLKEM768, qui combinent l'échange de clés classique avec ML-KEM-768.

Voici les étapes concrètes :

Installation de liboqs et du provider OQS pour OpenSSL : compiler liboqs depuis le dépôt GitHub open-quantum-safe/liboqs (version 0.10+), puis compiler le provider OQS depuis open-quantum-safe/oqs-provider. Ce provider expose les algorithmes PQC comme des groupes TLS standards pour OpenSSL.

Configuration OpenSSL : modifier openssl.cnf pour activer le provider OQS et définir la liste des groupes hybrides à utiliser : `Groups = X25519MLKEM768:P256MLKEM768:X25519:P-256`. L'ordre détermine la préférence du serveur.

Configuration du serveur TLS : dans Nginx ou Apache, activer les nouveaux groupes via la directive SSL. Pour Nginx : `ssl_ecdh_curve X25519MLKEM768:P256MLKEM768:X25519;` et `ssl_protocols TLSv1.3;`. TLS 1.3 est obligatoire car les groupes hybrides PQC ne sont supportés que dans ce protocole.

Tests de compatibilité client : utiliser curl compilé avec liboqs pour tester la négociation : `curl --curves X25519MLKEM768 https://votre-serveur/`. Chrome 124+ et Firefox 128+ supportent déjà X25519MLKEM768 nativement, permettant des tests en condition réelle.

Monitoring de la négociation : activer les logs SSL pour vérifier quel groupe est effectivement négocié avec chaque client. Les clients ne supportant pas ML-KEM reviendront automatiquement à X25519, garantissant une compatibilité ascendante.

Cette approche hybride garantit qu'aucun service n'est dégradé pour les clients classiques, tout en offrant une protection post-quantique aux clients modernes. La transition est ainsi transparente et progressive.

Comparatif des bibliothèques PQC : liboqs, PQClean, BoringSSL PQ fork

Le choix de la bibliothèque d'implémentation PQC est une décision architecturale critique qui conditionne les performances, la maintenabilité et la conformité des déploiements. Trois bibliothèques majeures se distinguent dans l'écosystème open source :

liboqs (Open Quantum Safe)

liboqs est la bibliothèque de référence pour les implémentations PQC expérimentales et de production. Développée par le projet Open Quantum Safe sous l'égide de l'Université de Waterloo, elle implémente l'ensemble des algorithmes NIST finalisés (ML-KEM, ML-DSA, SLH-DSA) ainsi que plusieurs candidats alternatifs. Ses avantages incluent une couverture algorithmique exhaustive, une intégration native avec OpenSSL via oqs-provider, et un support actif de la communauté. Son principal inconvénient est son positionnement explicite comme "expérimental" pour certains algorithmes, ce qui peut freiner son adoption en production dans des contextes réglementés. La version 0.10 marque un tournant vers plus de stabilité en se concentrant sur les algorithmes NIST finalisés.

PQClean

PQClean se distingue par son approche rigoureuse de la qualité du code : chaque implémentation est "clean" au sens où elle est lisible, portable et vérifiable formellement. Le projet fournit des implémentations de référence pour tous les algorithmes NIST finalisés, avec des tests automatisés sur de multiples compilateurs et plateformes. PQClean est particulièrement adapté aux environnements embarqués et aux applications nécessitant une conformité FIPS, car les implémentations sont minimalistes et auditables. En revanche, elle ne fournit pas de couche d'intégration haut niveau : l'utilisation directe des primitives requiert une expertise cryptographique significative.

BoringSSL PQ fork

Google maintient dans BoringSSL (sa fork d'OpenSSL) une intégration native de ML-KEM et des groupes hybrides TLS. C'est cette implémentation qui propulse la prise en charge PQC dans Chrome depuis la version 124. L'avantage majeur est la qualité d'implémentation et les optimisations performances issues des équipes de sécurité Google. Le fork intègre également des tests de performance exhaustifs et des optimisations SIMD pour x86-64 et ARM. La contrepartie est que BoringSSL n'est pas conçu pour être utilisé directement par des applications tierces : son API n'est pas stable et les breaking changes sont fréquents. Il reste néanmoins une référence précieuse pour valider des implémentations et benchmarker les performances.

Pour la majorité des déploiements enterprise, la recommandation est d'utiliser liboqs via oqs-provider pour OpenSSL, en attendant que les distributions Linux majeures intègrent nativement le support PQC dans leurs paquets OpenSSL (prévu pour 2025-2026 selon les roadmaps Debian, Ubuntu et Red Hat).

Tests de performance : impact latence et CPU des algorithmes NIST PQC

L'un des freins à l'adoption des algorithmes PQC est la crainte d'une dégradation des performances. Il est essentiel de quantifier cet impact pour prendre des décisions éclairées d'architecture et dimensionner correctement les infrastructures.

Voici les résultats de benchmarks représentatifs mesurés sur un serveur x86-64 moderne (Intel Xeon, 2023) :

Algorithme	Opération	Temps (µs)	Comparaison RSA-2048
ML-KEM-768	Encapsulation	22	50x plus rapide
ML-KEM-768	Décapsulation	19	60x plus rapide
ML-DSA-65	Signature	105	3x plus lent
ML-DSA-65	Vérification	62	Comparable ECDSA-256
SLH-DSA-128s	Signature	8500	250x plus lent
X25519MLKEM768 (hybride)	Handshake TLS	450	+15% vs X25519 seul

Ces chiffres révèlent une situation nuancée : ML-KEM est nettement plus performant que RSA pour l'encapsulation de clés, ce qui est une excellente nouvelle pour les handshakes TLS. ML-DSA est acceptable en termes de latence de vérification, mais plus lent que ECDSA en signature. SLH-DSA (basé sur les fonctions de hachage) est extrêmement lent en signature, ce qui le réserve aux cas d'usage où la performance n'est pas critique mais la robustesse maximale est requise.

La principale contrainte des algorithmes PQC n'est pas la latence CPU mais la **taille des clés et des signatures** : une clé publique ML-DSA-65 fait 1952 octets contre 64 octets pour ECDSA-256, et une signature ML-DSA-65 fait 3309 octets contre 64 octets. Pour TLS, cela

augmente la taille des certificats et donc la latence réseau lors des handshakes. Des optimisations comme la compression des certificats (RFC 8879) et le déploiement de certificats intermédiaires pré-chargés (OCSP stapling) permettent de mitiger cet impact.

Gestion de la transition hybride classique + PQC

La période de transition, qui durera selon les estimations entre 5 et 10 ans, nécessite une architecture hybride permettant à des clients supportant uniquement les algorithmes classiques de coexister avec des clients post-quantiques. Cette coexistence doit être gérée sans compromettre la sécurité des uns ni la compatibilité des autres.

Le principe du mode hybride est simple : on effectue simultanément un échange de clés classique (ECDH ou X25519) ET un échange de clés post-quantique (ML-KEM). Le secret final est dérivé des deux contributions par une fonction KDF (Key Derivation Function). La sécurité du système est alors garantie si *l'un ou l'autre* des algorithmes est sécurisé — si l'ordinateur quantique ne peut pas encore casser ECDH, le secret est protégé ; si ECDH est compromis par un futur ordinateur quantique, ML-KEM le protège.

Pour les certificats et signatures numériques, la transition hybride est plus complexe car les certificats X.509 ne supportent qu'un seul algorithme de signature. Deux approches coexistent :

Dual-certificate chains : déployer deux chaînes de certificats en parallèle (une classique, une PQC) et négocier la chaîne appropriée selon les capacités du client. Solution maximalement compatible mais doublement coûteuse en termes de gestion PKI.

Composite certificates : le draft IETF "Composite Signatures" permet d'intégrer deux signatures dans un seul certificat X.509. Les clients anciens n'utilisent que la signature classique tandis que les clients PQC vérifient les deux. Cette approche est soutenue par plusieurs CAs majeures et devrait être standardisée d'ici 2025-2026.

La gestion opérationnelle de cette période hybride requiert une attention particulière aux processus de renouvellement des certificats, à la configuration des CRL et OCSP, et à la formation des équipes NOC/SOC pour interpréter correctement les logs de négociation TLS incluant les nouveaux algorithmes.

Sources et références

CNIL — Guide de la sécurité des données personnelles

ANSSI — Recommandations RGPD et sécurité

ENISA — Guidelines on Data Security

