Analyse fonctionnelle du QSA

Lockheed LLC

Novembre 2026

Auteur : Guillaume Oneill & Christophe Zeilas

Version : 1.0



POST-QUANTUM CRYPTOGRAPHY



Building long-term digital trust

LOCKHEED LLC | IN DEVELOPMENT



QSA / TLSH

Préparer la sécurité des communications à l'ère post-quantique

Aujourd'hui
Cryptographie classique (X25519)

+

Demain
Post-quantique (ML-KEM-768)

=

TLSH
TLS Hybride Post-Quantique

QSA
Socle cryptographique (clés, encapsulation, signatures, vérifications)

TLSH
Moteur d'intégration et d'exploitation (TLS 1.3 + OpenSSL)

- **Résistance post-quantique**
- **Compatibilité TLS existant**
- **Performances et scalabilité**
- **Déploiement industriel** (Linux / Windows)
- **Qualifié et vérifiable**


TLS
TLS
Transport Layer Security
Le protocole qui sécurise les communications sur Internet
Confidentialité
(chiffrement)
Authentification
(certificats)
Intégrité
(données fiables)
https://

TLS — Transport Layer Security

Le **TLS (Transport Layer Security)** est le protocole de sécurité qui protège les communications sur Internet. Il constitue notamment le fondement du **HTTPS**, utilisé pour sécuriser les sites web, les API, les échanges entre serveurs et les transactions numériques.

TLS repose sur trois principes fondamentaux :

- **Confidentialité :** chiffrer les communications pour empêcher leur interception et leur lecture.
- **Authentification :** vérifier l'identité des correspondants, notamment grâce aux certificats numériques.
- **Intégrité :** garantir que les informations échangées n'ont pas été modifiées pendant leur transmission.

La version **TLS 1.3** utilise des mécanismes cryptographiques modernes pour établir une connexion sécurisée, négocier les clés de session et protéger les données échangées.

L'enjeu de la cryptographie post-quantique

L'émergence future d'ordinateurs quantiques suffisamment puissants pourrait compromettre certains mécanismes classiques d'établissement des clés.

C'est précisément l'objectif du projet **TLSH (TLS Hybride Post-Quantique)** : associer la cryptographie classique (X25519) à la cryptographie post-quantique (ML-KEM-768), afin de préparer les communications sécurisées aux menaces de demain.

TLS protège les communications d'aujourd'hui. TLSH vise à préserver leur confidentialité face aux capacités quantiques de demain.

Une nouvelle génération de communications sécurisées

Aujourd'hui, une grande partie des communications sur Internet repose sur le protocole TLS (**Transport Layer Security**), qui protège notamment les connexions HTTPS, les échanges entre serveurs, les API et de nombreuses infrastructures critiques.

Cette sécurité s'appuie sur des mécanismes cryptographiques classiques, dont certains pourraient devenir vulnérables face à de futurs ordinateurs quantiques suffisamment puissants. Le risque existe déjà : des communications chiffrées interceptées aujourd'hui pourraient être conservées, puis déchiffrées dans plusieurs années. C'est le principe du *Harvest Now, Decrypt Later*.

Notre projet TLSH (TLS Hybride Post-Quantique), associé à la technologie QSA (Quantum Security Architecture), vise à préparer les infrastructures informatiques à cette nouvelle génération de menaces, sans renoncer à la compatibilité avec les technologies existantes.

Le principe : associer deux générations de cryptographie

L'innovation repose sur une approche hybride combinant deux mécanismes complémentaires :

- **X25519**, un mécanisme classique d'établissement de secret partagé, largement utilisé dans les communications sécurisées.
- **ML-KEM-768**, un mécanisme post-quantique standardisé par le NIST, conçu pour résister aux attaques connues utilisant des ordinateurs quantiques.

Ces deux mécanismes contribuent à établir un secret commun entre le client et le serveur. Le protocole TLS 1.3 utilise ensuite ce secret pour dériver les clés qui protégeront les communications.

L'intérêt de cette hybridation est fondamental : **conserver la sécurité de l'établissement des clés lorsqu'au moins une des deux composantes demeure sûre**, sous réserve d'une construction cryptographique et d'une intégration correctes.

Les applications continuent ainsi d'utiliser un chiffrement symétrique performant, notamment AES-GCM, sans devoir réinventer toute leur architecture.


QSA / TLSH — Synthèse de l'analyse fonctionnelle
Développement d'un noyau cryptographique pour les communications TLS hybrides post-quantiques

Développement d'un noyau cryptographique pour les communications TLS hybrides post-quantiques

1. Objectif du projet

Le projet **QSA/TLSH** vise à développer un composant cryptographique autonome permettant de préparer les communications TLS 1.3 aux menaces liées à l'émergence de l'informatique quantique.

Son objectif est de combiner la robustesse des mécanismes cryptographiques traditionnels avec celle des nouveaux algorithmes post-quantiques, tout en préservant la compatibilité, les performances et la fiabilité des infrastructures existantes.

L'enjeu majeur est notamment de contrer les attaques dites *Harvest Now, Decrypt Later*, consistant à intercepter aujourd'hui des communications chiffrées pour tenter de les déchiffrer ultérieurement.

2. Les technologies cryptographiques retenues

L'architecture repose sur trois familles d'algorithmes complémentaires :

- **X25519 :** échange de clés classique, performant et largement éprouvé.
- **ML-KEM-768 (FIPS 203) :** mécanisme post-quantique d'encapsulation permettant l'établissement d'un secret partagé.
- **ML-DSA-44 (FIPS 204) :** signatures numériques post-quantiques destinées à l'authentification et à la vérification de l'intégrité.

Une quatrième technologie, **FN-DSA-512 (Falcon)**, est également prévue à titre expérimental, avec un statut de normalisation et de conformité distinct.

Le principe fondamental est l'**hybridation cryptographique** : combiner X25519 et ML-KEM-768 afin que la confidentialité de l'établissement des clés puisse être préservée si l'une des deux composantes reste sûre, sous réserve d'une intégration correcte.

Architecture logicielle

Noyau QSA/TLSH en langage C, fondé sur l'interface EVP d'OpenSSL

Protocole TLS

Utilisation dans les connexions sécurisées

Interface stable QSA/TLSH

Exposition contrôlée des fonctions

Opérations cryptographiques

- X25519
- ML-KEM-768
- Signatures
- Secrets

OpenSSL EVP

Base d'accès aux primitives (OpenSSL 3.5+)

Exigences clés

- Temps constant
- Linux & Windows
- Erreurs explicites
- Maîtrise des clés et secrets
- Aucun succès simulé

Une architecture modulaire, claire et vérifiable pour TLS hybride post-quantique

3. Architecture logicielle

Le noyau QSA/TLSH est développé en **langage C**, en s'appuyant sur l'interface cryptographique générique EVP d'OpenSSL.

L'architecture privilégie une séparation claire entre les opérations cryptographiques, leur exposition par une interface stable et leur utilisation dans le protocole TLS.

Les exigences principales sont :

- Exécution cryptographique à temps constant pour les opérations sensibles.
- Compatibilité Linux et Windows.
- Gestion explicite des erreurs et des algorithmes indisponibles.
- Maîtrise des clés, secrets et objets cryptographiques.
- Absence de succès simulé lorsqu'un mécanisme n'est pas disponible.

OpenSSL 3.5 ou une version compatible constitue une base importante pour accéder aux primitives post-quantiques retenues.

4. Fonctionnement d'une communication sécurisée

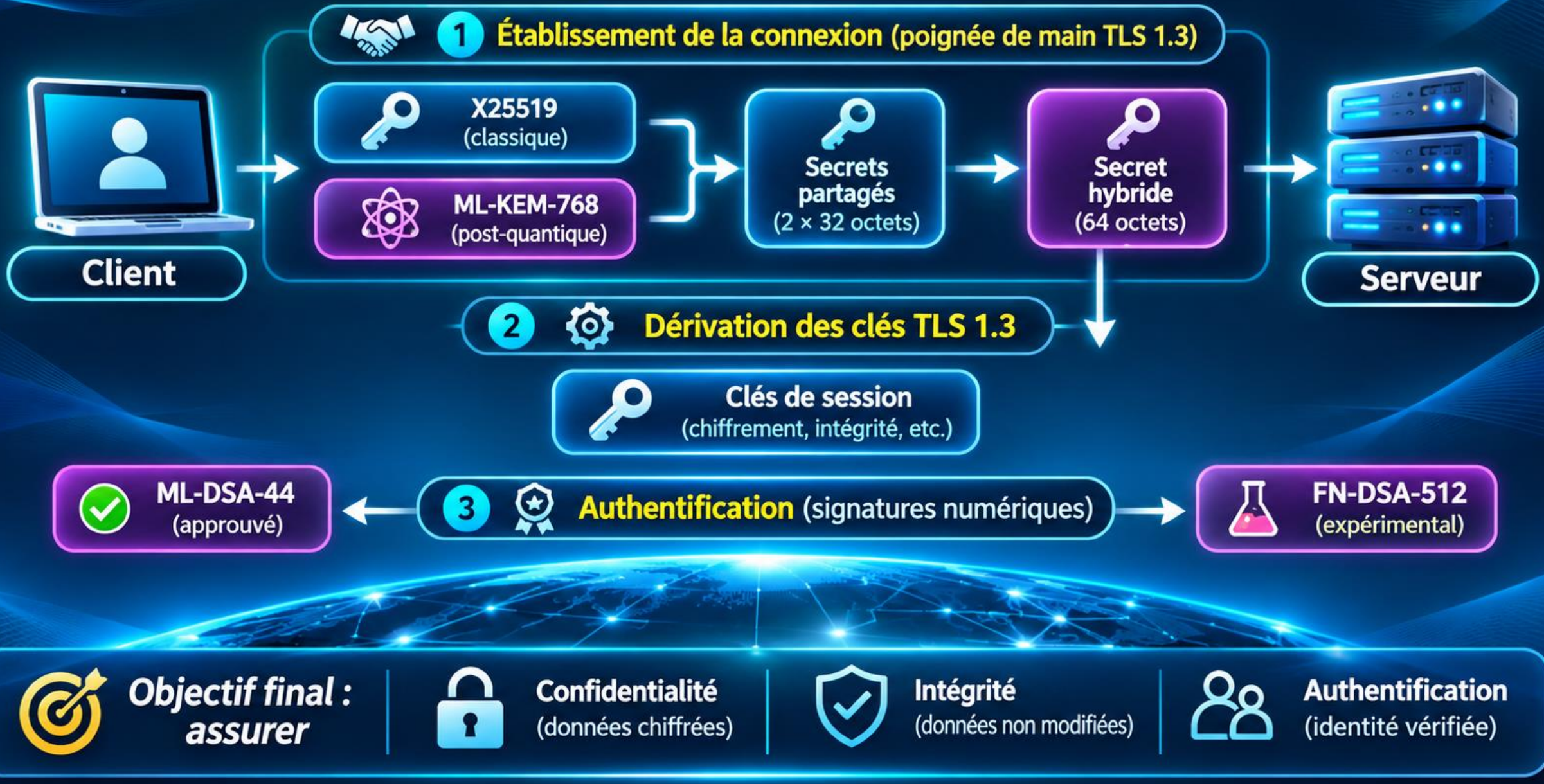
Lors de l'établissement d'une connexion, les mécanismes X25519 et ML-KEM-768 produisent chacun un secret partagé.

Ces deux secrets sont assemblés dans un ordre défini, puis transmis au mécanisme de dérivation des clés de TLS 1.3.

Des signatures numériques permettent ensuite de contribuer à l'authentification des correspondants et à la vérification des messages cryptographiques.

L'ensemble vise à assurer **confidentialité, intégrité et authentification**, sans remplacer le fonctionnement général de TLS 1.3.

QSA / TLSH | Fonctionnement d'une communication sécurisée



Objectif final : assurer

- **Confidentialité** (données chiffrées)
- **Intégrité** (données non modifiées)
- **Authentification** (identité vérifiée)

5. Validation et qualification

Le projet prévoit des tests automatisés couvrant les opérations X25519, l'assemblage des secrets hybrides, les mécanismes de signature et les conditions de disponibilité des fournisseurs cryptographiques.

Le document présente un premier bilan de **7 tests réussis, 0 échec et 3 tests ignorés**, ces derniers correspondant notamment à des fonctionnalités indisponibles dans l'environnement OpenSSL utilisé.

Les prochaines étapes devront confirmer le fonctionnement complet des primitives post-quantiques, leur intégration effective dans une négociation TLS et leur comportement face aux erreurs.

6. Finalité et perspectives

QSA/TLSH constitue une brique technologique destinée à accompagner la transition vers les communications sécurisées post-quantiques.

Son architecture modulaire doit faciliter l'intégration des algorithmes, leur qualification, leur évolution et leur exploitation dans des environnements professionnels.

À terme, cette approche pourra contribuer à protéger les infrastructures web, les échanges interserveurs, les API, les systèmes industriels, les plateformes cloud et les communications sensibles.

L'ambition du projet est de construire un socle cryptographique hybride, portable, vérifiable et évolutif, capable de préparer les infrastructures numériques aux enjeux de sécurité de demain.

6. Finalité et perspectives

QSA/TLSH constitue une brique technologique destinée à accompagner la transition vers les communications sécurisées post-quantiques.

Son architecture modulaire doit faciliter l'intégration des algorithmes, leur qualification, leur évolution et leur exploitation dans des environnements professionnels.

À terme, cette approche pourra contribuer à protéger les infrastructures web, les échanges interserveurs, les API, les systèmes industriels, les plateformes cloud et les communications sensibles.

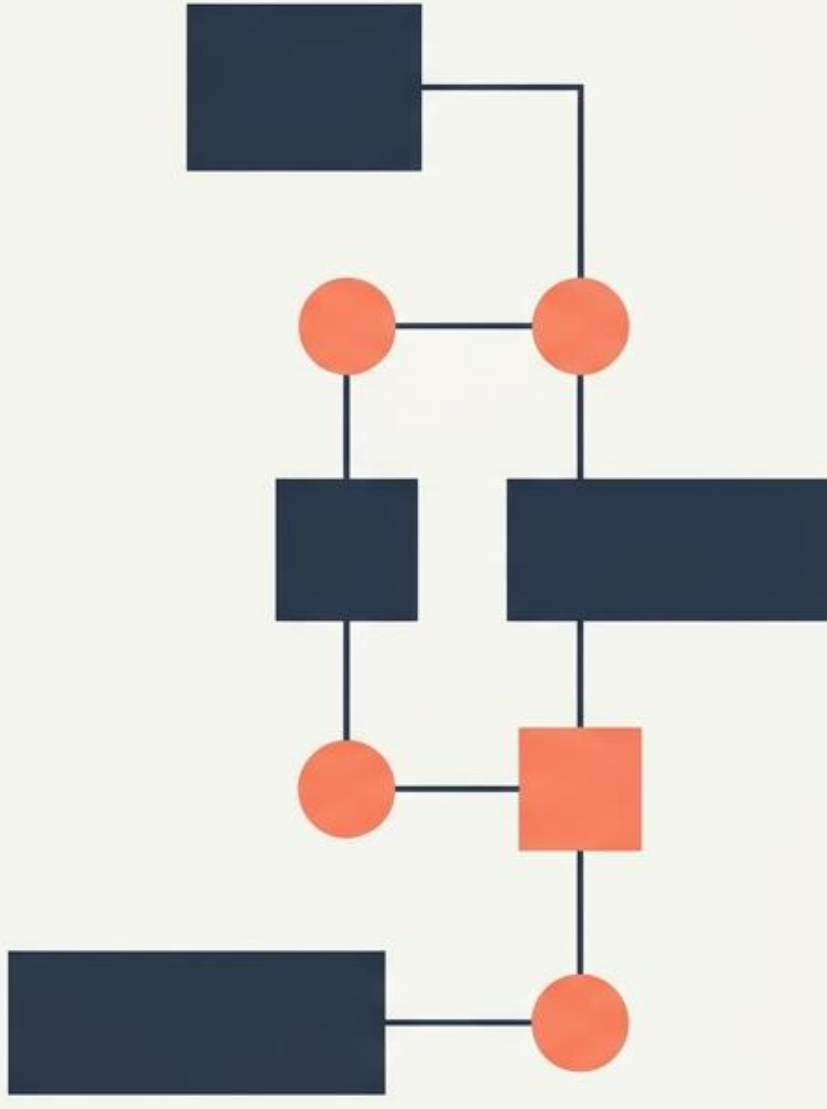
L'ambition du projet est de construire un socle cryptographique hybride, portable, vérifiable et évolutif, capable de préparer les infrastructures numériques aux enjeux de sécurité de demain.


AGENDA
ANALYSE FONCTIONNELLE
QSA

AGENDA ANALYSE FONCTIONELLE QSA

- 1. Finalité du composant et analyse des éléments disponibles
- 2. Comprendre le rôle de QSA dans une connexion TLS hybride
- 3. Périmètre fonctionnel de la version temporaire
- 4. Architecture logicielle et choix des langages
- 5. Déroulement fonctionnel d’une connexion hybride
- 6. Contrat d’interface et comportement opérationnel
- 7. Sécurité et interprétation des exigences NIST
- 8. Stratégie de qualification et critères d’acceptation
- 9. Installation, exploitation et diagnostic
- 10. Plan de réalisation et remplacement par le QSA Original
- 11. Roadmap des phases de développement

Finalité du composant et analyse des éléments disponibles



1.1. Le besoin à satisfaire

Le projet TLSH doit pouvoir progresser pendant que le QSA de Christophe poursuit son développement.

QSA-REF devra donc permettre de :

- développer les interfaces entre TLSH et sa couche cryptographique ;
- établir de véritables connexions TLS hybrides ;
- tester les échanges de clés, l'authentification, les erreurs et la concurrence ;
- mesurer les performances dans des conditions reproductibles ;
- préparer une campagne de comparaison avec le QSA de Christophe ;
- conserver une référence technique après son intégration.

Le caractère temporaire concernera **le moteur sélectionné dans TLSH.** Le contrat d'interface, les tests et les outils de diagnostic auront vocation à rester utiles.

1.2. Les documents décrivent deux périmètres différents

Le document QSA_TLSH - Analyse Fonctionnelle.docx définit un noyau autonome centré sur X25519, ML-KEM-768 et les signatures, explicitement séparé de Cryptonit et des fonctions de chiffrement de fichiers. C'est la référence fonctionnelle principale pour notre travail. Il rapporte également certains essais réussis, mais mentionne trois tests post-quantiques ignorés dans l'environnement OpenSSL 3.0 utilisé.

Le document QSA_NIST - Analyse Fonctionnelle.docx décrit un ensemble plus large : dérivation Cryptonit, conteneurs QSAF/QSAH/QSAS, coffre de secrets, ML-KEM-1024 et signatures supplémentaires.

Décision de périmètre : QSA-REF reproduira les services nécessaires à QSA_TLSH. Les fonctions de QSA_NIST qui ne participent pas à TLSH seront conservées comme contexte d'évolution.

Cela évitera de transformer le composant temporaire en un second projet complet de chiffrement de fichiers.

1.3. Ce que les pièces permettent de définir

Nous disposons d'une description des primitives, de quelques tailles de données, de codes de retour et de principes de sécurité.

En revanche, les pièces ne contiennent pas :

- l'en-tête C réel et complet ;
- les sources du composant ;
- les vecteurs de test annoncés ;
- les journaux détaillés permettant de reproduire les résultats ;
- les scripts de compilation ;
- une démonstration de connexion TLS hybride de bout en bout.

Je retiens donc ton indication actuelle : **le produit de Christophe est une version alpha à qualifier**.

Nous pouvons définir une compatibilité fonctionnelle cible. La compatibilité binaire exacte devra attendre l'examen de son interface réelle.

1.4. Les utilisateurs du composant

Quatre usages guideront la conception :

Utilisateur	Besoin principal
Développeur TLSH	Disposer d'une API stable et d'un moteur fonctionnel
Intégrateur	Installer, configurer et vérifier l'environnement
Responsable des tests	Reproduire les succès, les erreurs et les mesures
Christophe et l'équipe Lockheed LLC	Comparer les deux moteurs et qualifier la substitution

Le résultat attendu est concret : **TLSH doit pouvoir fonctionner avec QSA-REF sans attendre la livraison du QSA officiel.**

SYNTHÈSE DU CHAPITRE 1 : QSA-REF

1.1 Le Besoin à Satisfaire

(fron <IMAGE 0)

Développeur TLSH

stable API

Responsable des Tests

Reproduire succès, erreurs, mesures

Christophe & Lockheed Team

Comparer moteurs
Qualifier substitution

QSA-REF : Découpler le développement TLSH du QSA Officiel

1.2 Les Périmètres Documentaires

(from <IMAGE 1›)

QSA_TLSH (Référence)

A

X25519
ML-KEM-768
Signatures

Noyau Autonome,
Reproduit les services nécessaires

QSA_NIST
(Contexte d'Évolution)

B

ML-KEM-1024
Secret Box
QSA containers

Services

Services contextuels
(Hors périmètre QSA-REF)

Noyau Autonome,
Reproduit les services nécessaires

1.3 Définition Technique & Objectif

(from <IMAGE 2>)

DÉCISION :
QSA-REF reproducira
QSA_TLSH

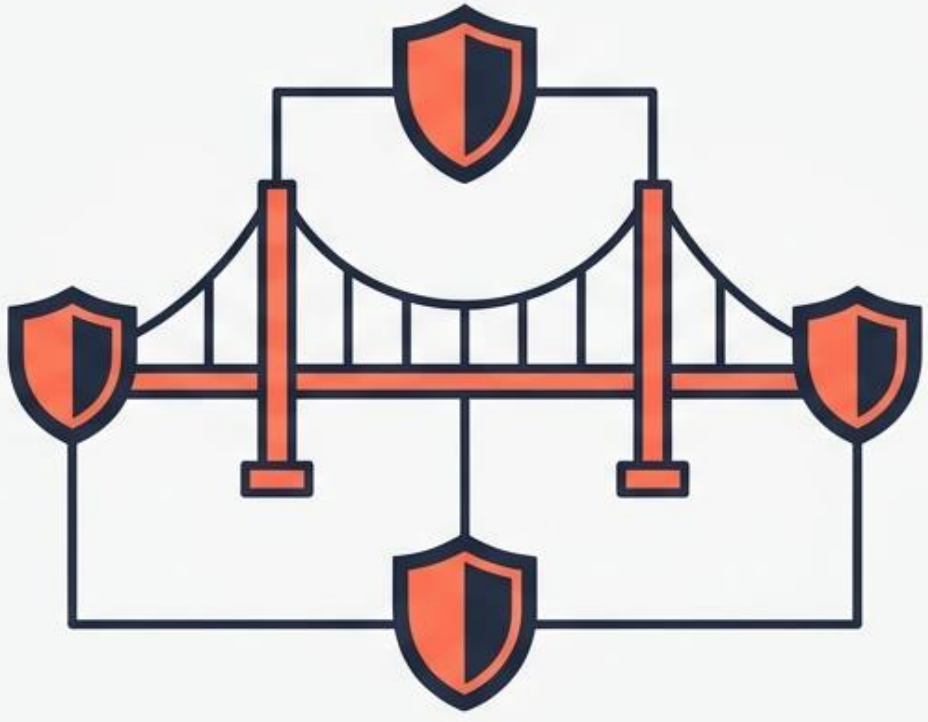
FOURNIT :
- Primitives
- Sécurité
- Tailles de données

QSA-REF

NE FOURNIT PAS :
- En-tête C
- Sources
- Scripts de Compilation
- Démo End-to-End

OBJECTIF : Version Alpha à Qualifier (Compatibilité Fonctionnelle Cible, pas binaire immédiate)

Comprendre le rôle de QSA dans une connexion TLS hybride



2.1. Les fonctions à distinguer

Pour comprendre où intervient QSA, il faut séparer quatre opérations.

Fonction	Question à laquelle elle répond	Mécanisme envisagé
Établissement d'un secret partagé	Comment obtenir un secret commun entre client et serveur ?	X25519 et ML-KEM-768
Authentification	Comment vérifier l'identité du correspondant ?	Certificats et signatures
Dérivation des clés	Comment produire les différentes clés de connexion ?	Mécanisme HKDF de TLS 1.3
Protection des données	Comment protéger les messages échangés ?	Chiffrement authentifié, notamment AES-GCM

Dans TLS 1.3, le groupe d'échange de clés, les algorithmes de signature et la suite de chiffrement sont des choix distincts. L'affichage de `TLS_AES_256_GCM_SHA384` ne suffit donc pas à démontrer qu'un échange post-quantique a eu lieu. RFC Editor

2.2. Ce que signifie « hybride »

Dans le profil retenu, deux mécanismes contribuent à l'établissement des clés :

- **X25519**, pour la composante classique ;
- **ML-KEM-768**, pour la composante post-quantique.

Leurs résultats sont combinés selon le profil TLS défini. L'objectif est de conserver la protection de l'établissement de clés si l'une des composantes reste sûre, dans les conditions prévues par la construction. Cette propriété dépend également d'une intégration correcte. rfc-editor.org

Les données applicatives seront ensuite protégées par le chiffrement symétrique de TLS.

L'hybridation concerne donc ici la création des clés de connexion.

2.3. Ce que fait ML-KEM

ML-KEM est un mécanisme d'encapsulation de clé. Il permet à deux parties d'obtenir un secret partagé à partir d'une clé publique et d'un message d'encapsulation transmis sur le réseau. Il ne sert pas directement à chiffrer chaque page web ou chaque fichier échangé. csrc.nist.gov

Les trois opérations essentielles sont :

1. produire une paire de clés ;
2. encapsuler vers une clé publique ;
3. décapsuler avec la clé privée correspondante.

QSA-REF devra fournir ces trois opérations avec leur véritable sémantique cryptographique.

2.4. Ce que fait ML-DSA

ML-DSA sert à produire et vérifier des signatures numériques. Ces signatures permettent notamment de détecter une modification et d'authentifier le signataire dans un système qui établit correctement la confiance dans sa clé publique. csrc.nist.gov

Cependant, une fonction `sign()` opérationnelle ne suffit pas à démontrer une authentification TLS post-quantique. Il faut aussi intégrer :

- le schéma de signature dans la négociation TLS ;
- les certificats compatibles ;
- la vérification de leur chaîne de confiance ;
- la signature des données exactes prévues par TLS.

2.5. La responsabilité respective de TLSH et QSA-REF

TLSH pilotera les connexions et leur politique. QSA-REF fournira les opérations cryptographiques attendues. Le moteur TLS assurera le déroulement du protocole.

Cette séparation permettra d'identifier précisément l'origine d'un problème : configuration, certificat, protocole, interface ou calcul cryptographique.

2 | QSA – Fonctions et Hybride

Secret partagé
Établir un secret commun

Authentification
Vérifier l'identité

Dérivation des clés
Produire les clés de session

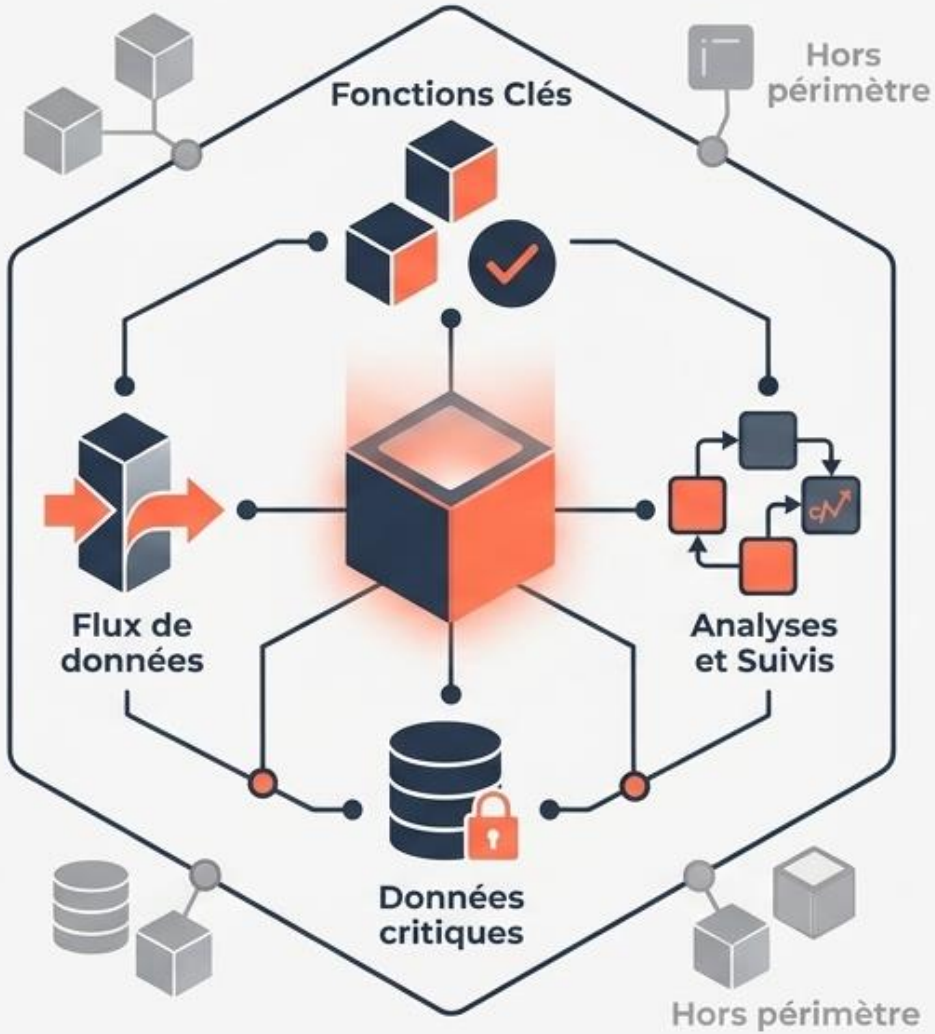
Protection des données
Chiffrer les échanges
(ex. AES-GCM)

Établissement des clés – Hybride



QSA : les briques cryptographiques de TLSH

3. Périmètre fonctionnel de la version temporaire



3.1. Les fonctions obligatoires

Je propose le socle suivant.

Référence	Fonction	Résultat attendu
REF-01	Initialisation et inventaire	Identifier versions, fournisseurs et capacités réellement utilisables
REF-02	X25519	Générer les clés éphémères et calculer le secret partagé
REF-03	ML-KEM-768	Générer, encapsuler et décapsuler
REF-04	Échange hybride	Produire et traiter les données du groupe `X25519MLKEM768`
REF-05	ML-DSA-44	Générer les clés, signer et vérifier
REF-06	Intégration TLS	Participer effectivement à une connexion TLS 1.3
REF-07	Protection des secrets	Maîtriser leur allocation, leur durée de vie et leur effacement
REF-08	Gestion des erreurs	Refuser clairement les opérations impossibles ou interdites
REF-09	Qualification	Exécuter des tests reproductibles avec résultats détaillés

3.2. Le premier profil TLS utilisable

Le premier profil de connexion sera fixé ainsi :

- TLS 1.3 ;
- groupe `X25519MLKEM768` ;
- suite `TLS_AES_256_GCM_SHA384` ;
- authentification serveur par certificat de laboratoire ;
- validation obligatoire de ce certificat ;
- absence de retour automatique à un échange classique.

Pour ce premier jalon, un certificat classique permettra de valider l'échange hybride et le fonctionnement du service.

Le rapport indiquera alors séparément :

Établissement de clés : hybride post-quantique.
Authentification : classique.

Cette distinction devra rester visible dans les résultats.

3.3. Le profil d'authentification post-quantique

La capacité ML-DSA-44 sera obligatoire dans l'API cryptographique de QSA-REF.

Son utilisation dans TLS fera l'objet d'un jalon supplémentaire, avec une infrastructure de certificats de laboratoire et des clients maîtrisés. Au moment de cette vérification, le document IETF consacré à ML-DSA dans TLS 1.3 est en file de publication RFC ; son intégration doit donc être rattachée à une version précise du profil et de la pile utilisée. Use of ML-DSA i...

Nous distinguerons trois résultats :

1. signature ML-DSA opérationnelle ;
2. authentification TLS ML-DSA opérationnelle ;
3. chaîne de confiance qualifiée pour le profil post-quantique annoncé.

3.4. Les extensions prévues

L'architecture réservera la possibilité d'ajouter ML-DSA-65/87, d'autres groupes hybrides et des profils futurs.

FN-DSA/Falcon restera une capacité expérimentale identifiée, désactivée dans le socle initial.

Un futur profil « PQC seul » nécessitera une définition de protocole et des essais d'interopérabilité propres. Il ne sera pas obtenu en retirant simplement X25519 d'un message existant.

3.5. Les fonctions non nécessaires au premier TLSH

Cryptonit, les conteneurs de fichiers, le coffre Supersafe et les optimisations AES spécifiques ne conditionneront pas la disponibilité de QSA-REF.

Leur éventuelle reprise devra correspondre à un besoin fonctionnel explicite du projet.

Rôle de QSA dans le TLS hybride post-quantique

4 fonctions clés

- **Secret partagé** — X25519 + ML-KEM
- **Authentification** — Certificats / signatures
- **Dérivation** — HKDF
- **Protection** — AES-GCM

Hybride

Classique — X25519 + **Post-quantique** — ML-KEM-768 → **Secret de connexion** — Clé hybride

La protection reste assurée si une composante reste sûre.

Primitives QSA

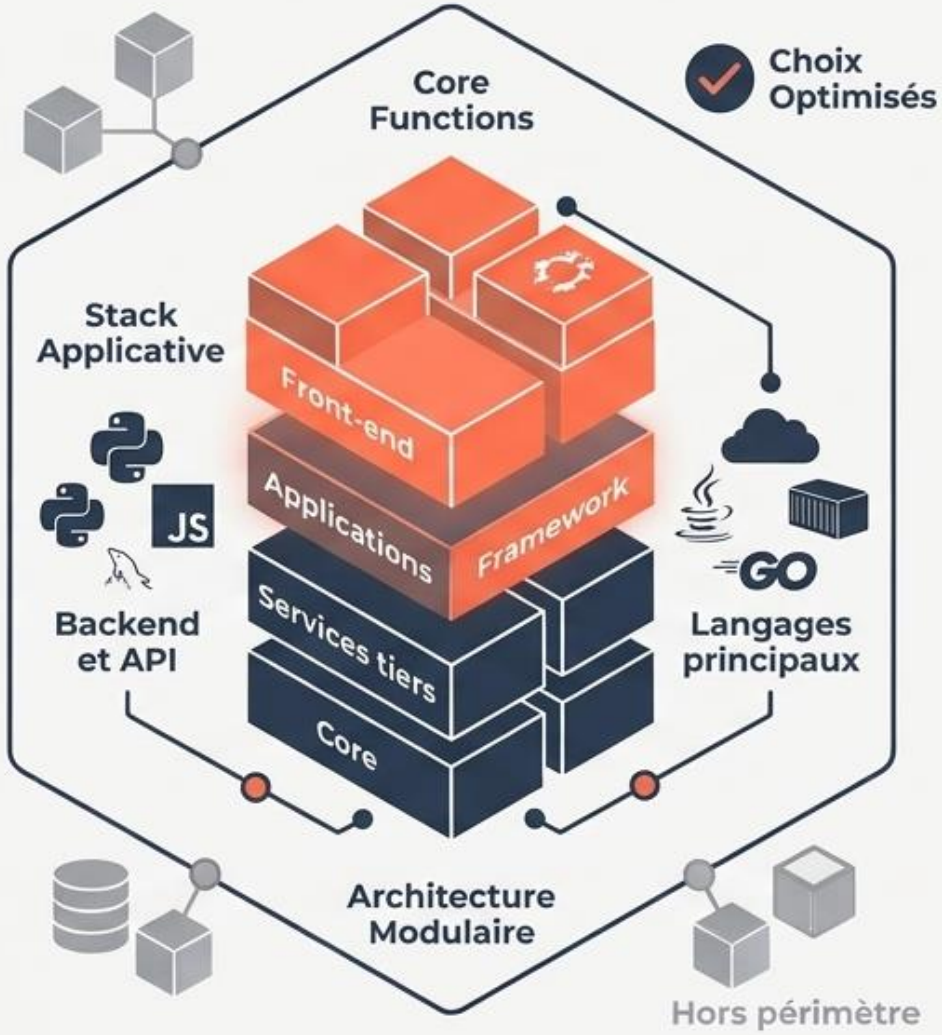
- **ML-KEM** — Établit le secret
- **ML-DSA** — Signe et vérifie

Répartition des responsabilités

TLSH *Pilote* → **QSA-REF** *Crypto* → **Moteur TLS** *Protocole*

QSA *apporte les briques cryptographiques ;* **TLSH** *orchestre le tout.*

4. Architecture logicielle et choix des langages



4.1. L'organisation retenue

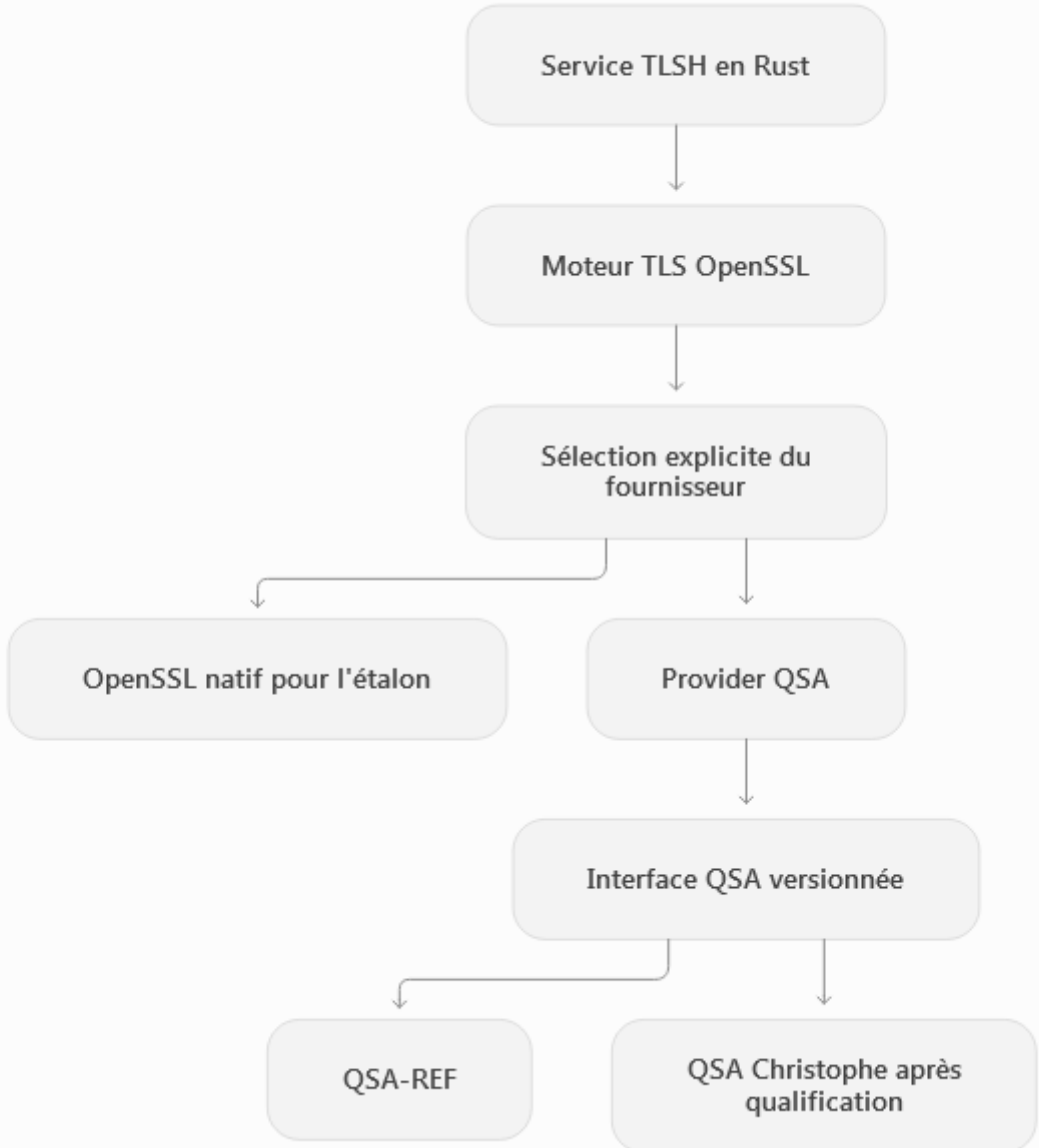
Je recommande une architecture locale, composée d'une bibliothèque native et du service TLSH.

Couche	Technologie	Responsabilité
Service TLSH	Rust	Connexions, configuration, quotas, états et supervision
Moteur TLS	OpenSSL `libssl`	Négociation et traitement du protocole TLS
Interface QSA	API C versionnée	Contrat stable entre les couches
QSA-REF	C utilisant OpenSSL EVP	Opérations cryptographiques de référence
Raccordement QSA à TLS	Provider OpenSSL dédié	Faire utiliser effectivement QSA par `libssl`
Installation et qualification	Python	Vérifications, préparation, tests et rapports

L'API EVP est l'interface générique d'OpenSSL pour accéder aux opérations cryptographiques. Un *provider* est le composant qui expose une implémentation à OpenSSL.

Le socle reposera sur **une révision corrigée et précisément figée d'OpenSSL 3.5 LTS**, qualifiée sur nos plateformes. Cette branche possède le support du groupe hybride retenu. La simple présence d'un numéro de version suffisant ne remplacera pas les essais de disponibilité. OpenSSL Docum...

4.2. Une progression en trois étapes vérifiables



4.2 | Progression en 3 étapes vérifiables



*Un parcours clair, **maîtrisé et vérifiable** de l'intégration QSA dans TLSH*

4.3. Le raccordement réel dans OpenSSL

OpenSSL permet à un provider d'exposer des groupes TLS et les opérations associées. Lorsqu'un même groupe existe dans plusieurs providers, sa sélection doit être maîtrisée ; elle peut autrement être indéterminée. OpenSSL Docum...

Notre raccordement devra donc :

- représenter le groupe hybride complet ;
- gérer les clés et messages combinés ;
- appeler les opérations QSA attendues ;
- appliquer les validations nécessaires ;
- restituer les résultats au moteur TLS.

Exposer uniquement une fonction ML-KEM ne suffira pas.

Le provider de référence utilisera un contexte OpenSSL interne distinct pour ses calculs. Cette séparation devra empêcher qu'il se rappelle lui-même involontairement.

4.4. La preuve du chemin réellement exécuté

Chaque campagne devra identifier le moteur demandé et le moteur effectivement utilisé.

Pour le démontrer, nous prévoirons :

- des compteurs d'appels par opération ;
- une identification du fournisseur chargé ;
- des essais croisés avec OpenSSL natif ;
- une injection de refus réservée au banc de test.

Par exemple, si nous provoquons volontairement un refus d'encapsulation dans QSA-REF, la connexion de test devra échouer à cet endroit. Une connexion qui réussirait malgré ce refus révélerait un contournement du composant.

4.5. Le rôle de Rust

Rust prendra en charge les parties de service : traitement des connexions, délais, concurrence, configuration et journalisation.

La frontière C/Rust restera réduite et documentée :

- propriété de chaque objet ;
- fonctions de libération ;
- règles de concurrence ;
- conversion des erreurs ;
- absence de propagation incontrôlée d'une erreur Rust à travers l'interface C.

Les clés privées resteront autant que possible dans les objets natifs, sans copies intermédiaires dans le service Rust.

4 | Intégration OpenSSL & Preuve d'exécution

Architecture & Rôles

- **Service TLSH** (Rust)
- **Moteur TLS** (OpenSSL / libssl)
- **Interface QSA** (API C versionnée)
- **QSA-REF** (C + OpenSSL EVP)

Raccordement dans OpenSSL

OpenSSL Provider TLSH

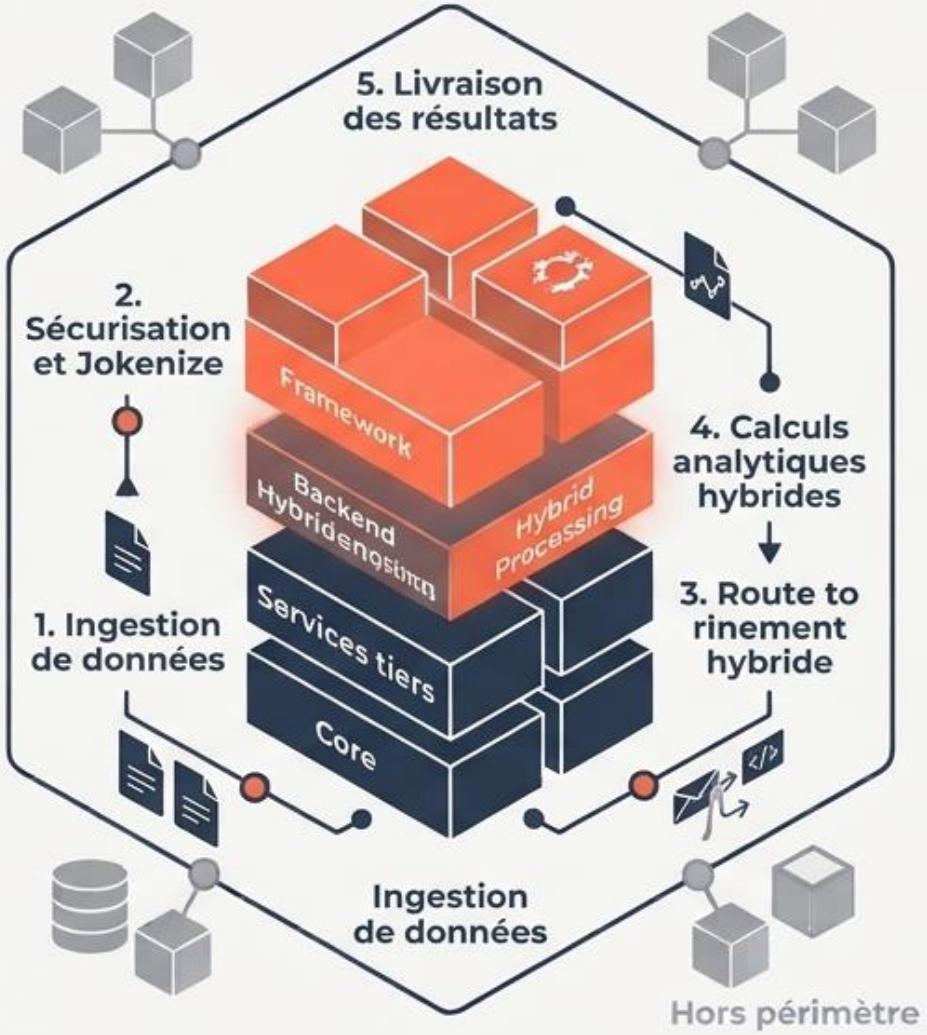
- Groupe hybride complet
- Clés & messages combinés
- Appel des opérations QSA
- Validations (contrôles)
- Résultats vers le moteur TLS

Preuve du chemin exécuté

- Compteurs d'appels par opération ✓
- Identification du provider chargé ✓
- Essais croisés avec OpenSSL natif ✓
- Injection de refus (banc de test) ✓

Une intégration OpenSSL *maîtrisée, vérifiable et sécurisée*

5. Déroulement fonctionnel d'une connexion hybride



5.1. Correction du scénario fourni

Pour le groupe standard `X25519MLKEM768`, **le client génère la paire ML-KEM ; le serveur encapsule ; le client décapsule.**

Le scénario du document, qui place la clé ML-KEM côté serveur et l'encapsulation côté client, ne correspond pas au déroulement de ce groupe TLS. La référence publiée est désormais la RFC 10024. rfc-editor.org

5.2. Les données échangées

Pour ce groupe, les charges utiles sont les suivantes :

Élément	Contenu
Partage client	Clé publique ML-KEM de 1 184 octets, puis partage public X25519 de 32 octets
Partage serveur	Encapsulation ML-KEM de 1 088 octets, puis partage public X25519 de 32 octets
Secret combiné	Secret ML-KEM de 32 octets, puis secret X25519 de 32 octets

Les partages mesurent donc respectivement **1 216** et **1 120 octets**. Ces tailles concernent leur contenu cryptographique, pas les messages TLS complets. L'ordre ML-KEM puis X25519 doit être respecté.

rfc-editor.org

5.3. Le scénario nominal attendu

Préparation. Le service vérifie la configuration, les capacités, le certificat et les résultats des contrôles de démarrage.

Négociation. Le client annonce le profil attendu. Le serveur accepte uniquement une combinaison autorisée.

Calcul cryptographique. Les opérations nécessaires sont exécutées par le fournisseur sélectionné. Chaque connexion possède son propre état.

Authentification et finalisation. Le moteur TLS vérifie les certificats et les messages de contrôle. Le service n'annonce la connexion comme établie qu'après réussite de ces étapes.

Échange applicatif. Le client et le serveur échangent une requête et une réponse vérifiables.

Fermeture. Les objets temporaires sont libérés, les secrets éphémères effacés et le résultat enregistré.

5.4. Le secret combiné ne constitue pas directement la clé de trafic

Le mécanisme de dérivation TLS transforme le matériau de départ en plusieurs clés destinées aux différents sens et phases de la communication. Cette opération doit rester conforme au moteur TLS. RFC Editor

Nous n'insérerons donc ni Cryptonit, ni un pepper, ni une dérivation propriétaire dans ce parcours standard.

De même, assembler deux secrets de 32 octets ne signifie pas obtenir une garantie de sécurité de « 512 bits ».

5.5. Les scénarios d'échec

Le comportement devra être défini pour :

- une composante cryptographique indisponible ;
- un partage public incorrect ;
- un certificat non fiable ou expiré ;
- une signature invalide ;
- un délai de connexion dépassé ;
- une interruption du client ;
- une allocation impossible ;
- un dépassement des limites de charge.

Le profil hybride obligatoire devra échouer si sa composante post-quantique ne peut pas être utilisée.

5 | TLSH – Scénarios, Données et Sécurité

5.1 Scénario corrigé

Groupe hybride standard :

X25519 (classique) + **ML-KEM-768** (post-quantique)

Référence : RFC 10024

5.2 Données échangées

Élément	Contenu (taille)
Partage client	Clé ML-KEM : 1 184 octets + X25519 : 32 octets
Partage serveur	Encapsulation ML-KEM : 1 088 octets + X25519 : 32 octets
Secret combiné	Secret ML-KEM : 32 octets + Secret X25519 : 32 octets

Tailles des partages :
1 216 o / 1 120 o

5.3 Scénario nominal (flux TLSH)

1. **Préparation**
   Vérification de la configuration, capacités, certificat et contrôles de démarrage.
2. **Négociation**
   Annonce du profil, acceptation de la combinaison autorisée.
3. **Calcul cryptographique**
   Opérations via le fournisseur QSA, état propre à chaque connexion.
4. **Authentification & finalisation**
   Vérification des certificats et messages, connexion établie.
5. **Échange applicatif**
   Requête / réponse sécurisées et vérifiables.
6. **Fermeture**
   Libération des objets, effacement des secrets éphémères et journalisation.

5.5 Scénarios d'échec

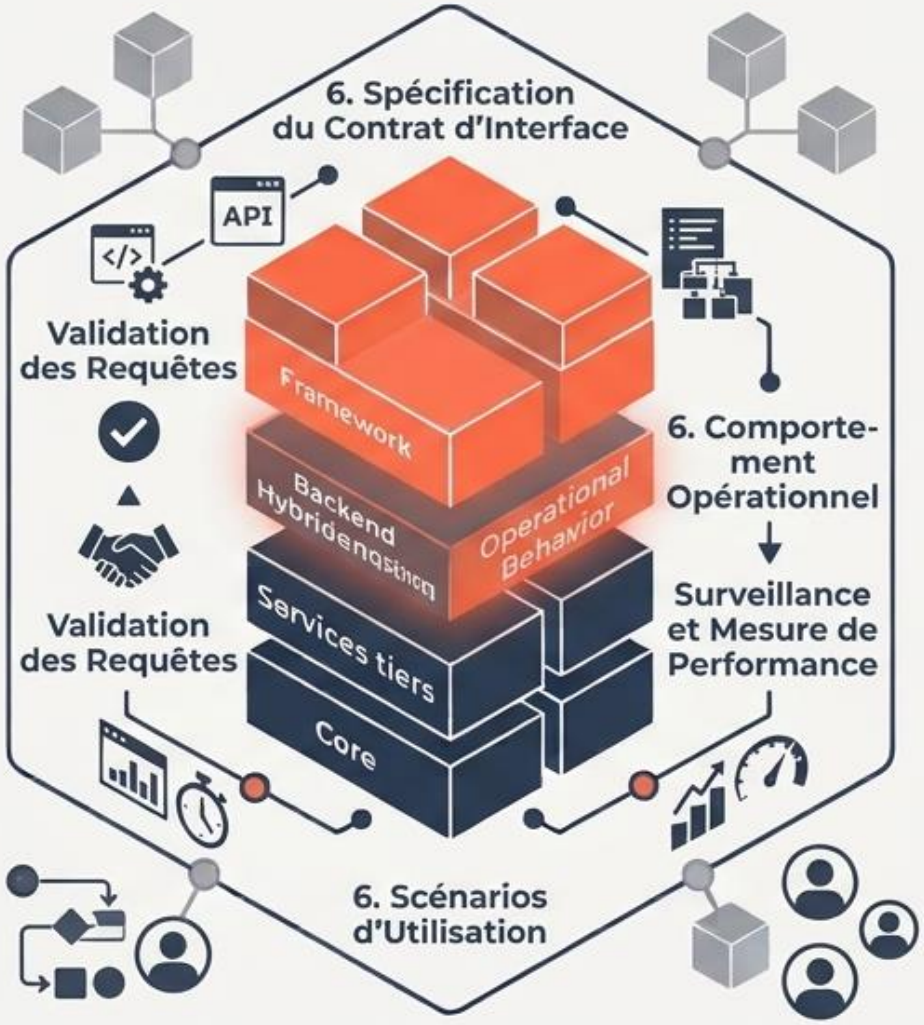
- Composante cryptographique indisponible
- Partage public incorrect
- Certificat non fiable ou expiré
- Signature invalide
- Délai de connexion dépassé
- Interruption du client
- Allocation impossible
- Dépassement des limites de charge

5.4 Point de sécurité clé

Le secret combiné n'est pas directement la clé de trafic.

La dérivation TLS (HKDF) produit des clés dédiées aux différentes phases.

6. Contrat d’interface et comportement opérationnel



6.1. Une interface que nous pouvons stabiliser dès maintenant

Le contrat interne sera versionné indépendamment de l'interface définitive de Christophe.

Il couvrira les opérations suivantes :

Service	Entrées principales	Sorties principales
Initialisation	Configuration et politique	Contexte utilisable ou erreur
Capacités	Contexte	Algorithmes, formats, versions et états
Génération de clés	Algorithme	Objet de clé et données publiques
Dérivation X25519	Clé locale et partage distant	Objet secret
Encapsulation ML-KEM	Clé publique distante	Encapsulation et objet secret
Décapsulation ML-KEM	Clé privée et encapsulation	Objet secret
Assemblage hybride	Résultats des deux composantes	Matériau attendu par le profil
Signature	Clé, message et contexte défini	Signature
Vérification	Clé publique, message et signature	Verdict explicite
Libération	Objet appartenant au composant	Destruction et nettoyage

6.2. Utiliser des références opaques pour les secrets

Je recommande que l'interface principale manipule des *handles* : des références vers des objets cryptographiques internes.

Cela permettra de contrôler :

- qui possède une clé ;
- qui peut l'utiliser ;
- quand elle est détruite ;
- si son export est autorisé ;
- quel fournisseur l'a créée.

Une façade supplémentaire pourra reproduire les tableaux d'octets attendus par Christophe lorsque leurs formats seront confirmés.

6.3. Clarifier les formats de clés

Le tableau privé ML-KEM-768 de 2 400 octets annoncé dans les documents correspond à une représentation développée particulière. OpenSSL distingue cette clé développée, une graine de 64 octets et les enveloppes de sérialisation telles que PKCS#8. Ces objets ne sont pas interchangeables par simple copie. OpenSSL Docum...

Chaque format d'échange devra donc préciser :

- son identifiant ;
- sa longueur ;
- son encodage ;
- les validations à effectuer ;
- les règles d'importation et d'exportation.

6.4. Les erreurs et les tampons

Nous conserverons la correspondance annoncée :

- `OK = 0` ;
- `ERR = 1` ;
- `UNAVAIL = 2`.

Un diagnostic interne précisera la cause : entrée invalide, tampon insuffisant, signature incorrecte, fournisseur absent, échec d'aléa ou politique refusée.

Le contrat devra définir la manière de demander une taille, le propriétaire des allocations et l'état des sorties après erreur. Un appel échoué ne devra jamais laisser croire qu'un secret partiellement produit est exploitable.

6.5. Les états du composant

Le cycle opérationnel sera explicite :

non initialisé → initialisation → contrôles → prêt → arrêt.

Une erreur bloquante conduira à un état **indisponible**, avec diagnostic.

Le service pourra rester consultable pour expliquer l'échec, mais il ne devra plus accepter de nouvelles connexions exigeant une capacité défaillante.

6.6. La concurrence et les changements de configuration

Chaque connexion conservera son contexte et son fournisseur pendant toute sa durée.

Un changement de moteur s'appliquera aux nouvelles connexions après qualification. Une bibliothèque ne pourra pas être déchargée tant que des clés ou des connexions la référencent.

Les limites configurables comprendront le nombre de négociations simultanées, les délais, les files d'attente et les volumes de données temporaires.

6 | QSA – Interface, Gestion et Robustesse

6.1 Interface & Opérations

- Initialisation / Capacités
- Génération de clés
- Encapsulation / Décapsulation
- Assemblage hybride
- Signature / Vérification
- Libération

6.2 Handles (références opaques)

- Contrôle de possession
- Contrôle d'usage
- Traçabilité (fournisseur, cycle)



QSA
Interface C

6.3 Formats de clés

- Identifiant
- Longueur (ex. 1 184 o, 1 088 o)
- Encodage (ex. PKCS#8)
- Validations
- Règles d'import / export

6.4 Erreurs & Tampons

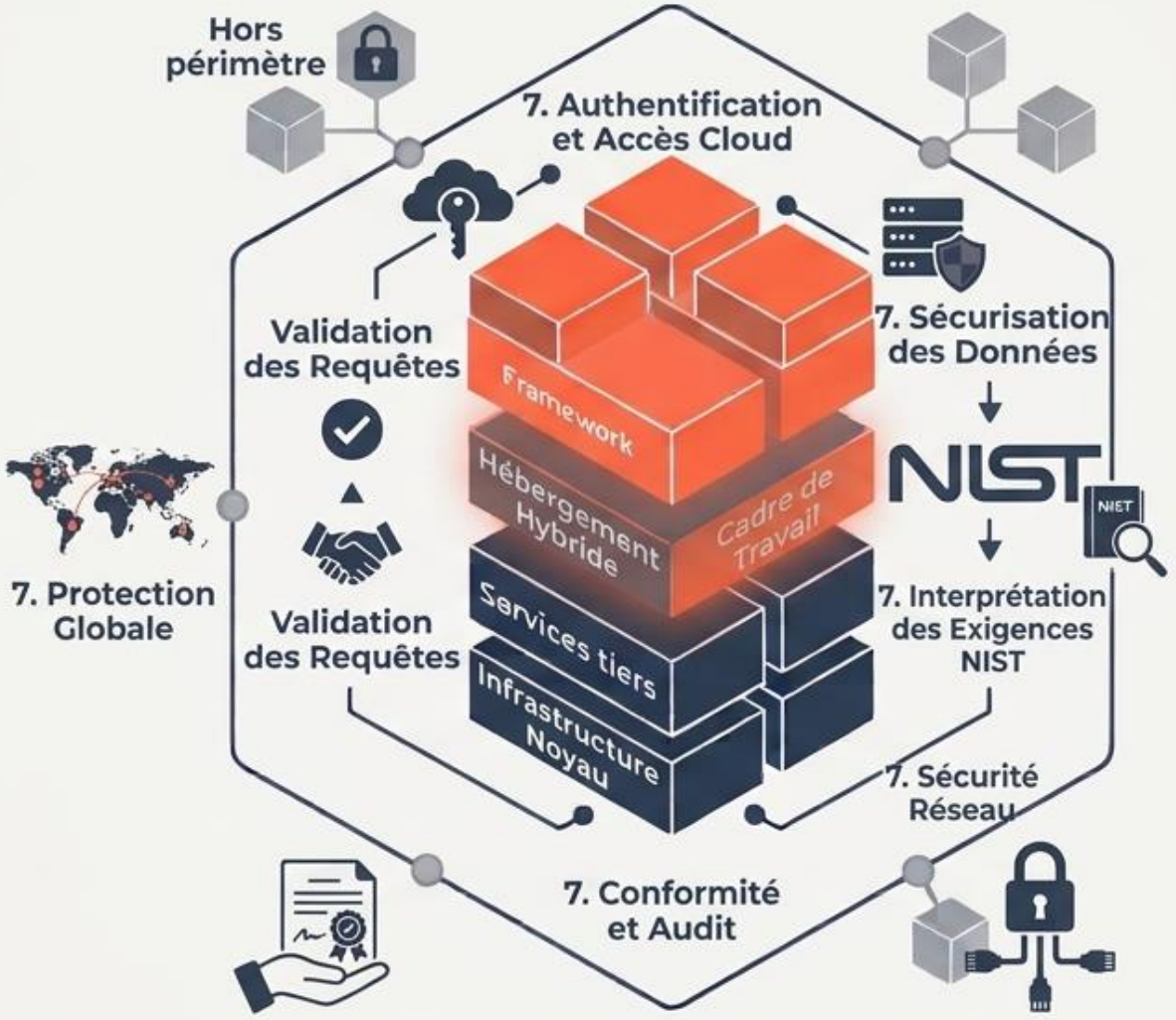
OK = 0	Succès
ERR = 1	Erreur (cause explicite)
UNAVAIL = 2	Indisponible

6.5 États du composant

Non initialisé → Initialisation → Contrôles → Prêt → Arrêt

*Une interface QSA stable, contrôlée et **conforme au cycle TLSH***

7. Sécurité et interprétation des exigences NIST



7.1. Le modèle de menace

QSA-REF devra être conçu pour traiter des clients potentiellement malveillants : messages tronqués, tailles excessives, valeurs invalides, connexions abandonnées et tentatives de saturation.

Les objectifs seront de protéger :

- les clés privées ;
- les secrets de connexion ;
- l'intégrité des échanges ;
- l'identité des correspondants ;
- la disponibilité du service dans ses limites configurées.

Le projet devra également examiner les risques introduits par notre propre raccordement : confusion de fournisseurs, mauvais format, erreurs de mémoire et divulgation par les journaux.

7.2. Le traitement des secrets

Nous imposerons :

- un nombre minimal de copies ;
- des durées de vie courtes pour les secrets éphémères ;
- des fonctions d'effacement adaptées ;
- un nettoyage sur les chemins de succès et d'échec ;
- l'absence de secrets dans les arguments de commande et les journaux ;
- des protections mémoire vérifiées sur chaque plateforme.

Le verrouillage mémoire sera demandé lorsque le profil l'exige. Son échec devra être signalé, et pourra bloquer le démarrage du profil durci.

7.3. L'aléa

La génération des clés reposera sur le générateur cryptographique de la bibliothèque, alimenté par les mécanismes du système.

Le fonctionnement normal n'acceptera pas de graine fixe.

Les données déterministes nécessaires aux vecteurs de test seront limitées au banc de qualification. OpenSSL réserve précisément certains paramètres d'injection d'aléa ML-KEM à cet usage. OpenSSL Docum...

7.4. Préserver le rejet implicite de ML-KEM

Ce point devra figurer explicitement dans les exigences.

Une encapsulation altérée mais de longueur correcte peut déclencher le mécanisme de **rejet implicite** : ML-KEM produit alors un secret de remplacement sans exposer le résultat interne de cette vérification. Le standard interdit de retourner ce drapeau secret. nvlpubs.nist.gov

Nous ne demanderons donc pas systématiquement un retour « encapsulation invalide » à la primitive. Le test de bout en bout devra vérifier que la connexion TLS ne peut pas aboutir avec des secrets incompatibles.

7.5. Distinguer normalisation et validation

Les analyses QSA_NIST - Analyse de conformité.docx et QSA_TLSH - Analyse de conformité.docx apportent un cadrage utile, mais certaines conclusions doivent être affinées.

7.5. Distinguer normalisation et validation

Les analyses QSA_NIST - Analyse de conformité.docx et QSA_TLSH - Analyse de conformité.docx apportent un cadrage utile, mais certaines conclusions doivent être affinées.

Nous enregistrerons séparément :

1. le statut de normalisation de l'algorithme ;
2. sa disponibilité dans l'environnement ;
3. son autorisation par notre politique ;
4. le fournisseur qui exécute l'opération ;
5. l'éventuelle validation applicable au module et à ses conditions d'utilisation.

Un booléen `sig_is_approved()` ne remplacera pas ces informations.

Le certificat OpenSSL **#4985** concerne le provider **3.1.2**. Sa politique classe notamment X25519 parmi les algorithmes non autorisés dans son mode approuvé. Il ne constitue donc pas une validation générale de notre chaîne TLS hybride. csrc.nist.gov +1

La page officielle OpenSSL consultée indique par ailleurs que le provider 3.5.8 est en revue CMVP. La disponibilité des fonctions post-quantiques et la validation du module restent deux états distincts. OpenSSL Library

QSA-REF sera présenté comme un composant de développement et de qualification, sans revendication de validation FIPS propre.

7.6. Corriger la conclusion relative à 2035

Le NIST IR 8547 final a bien été publié le 27 octobre 2026. CSRC

Cependant, son traitement des constructions composites permet, sous conditions, une composante issue d'un mécanisme non approuvé, voire interdit, lorsqu'elle est correctement combinée à une composante approuvée. On ne peut donc pas en déduire automatiquement que tout hybride comprenant X25519 devra disparaître en 2035. nvlpubs.nist.gov

Nous prévoirons l'agilité cryptographique, avec des profils versionnés et qualifiés individuellement.

7 | QSA – Sécurité, Aléa et Conformité

7.1 Modèle de menace



- Clients malveillants
- Messages tronqués / invalides
- Connexions abandonnées
- Tentatives de saturation

7.2 Traitement des secrets



- Peu de copies
- Durées de vie courtes
- Effacement fiable
- Pas de secrets en arguments / journaux
- Mémoire protégée (si requis)

7.3 Aléa



- Génération via le fournisseur cryptographique
- Pas de graine fixe
- Vecteurs de test limités (banc de qualification)

7.4 Rejet implicite de ML-KEM



Une encapsulation altérée peut être rejetée implicitement : un secret de remplacement est produit, sans l'exposer.

7.5 Normalisation et validation

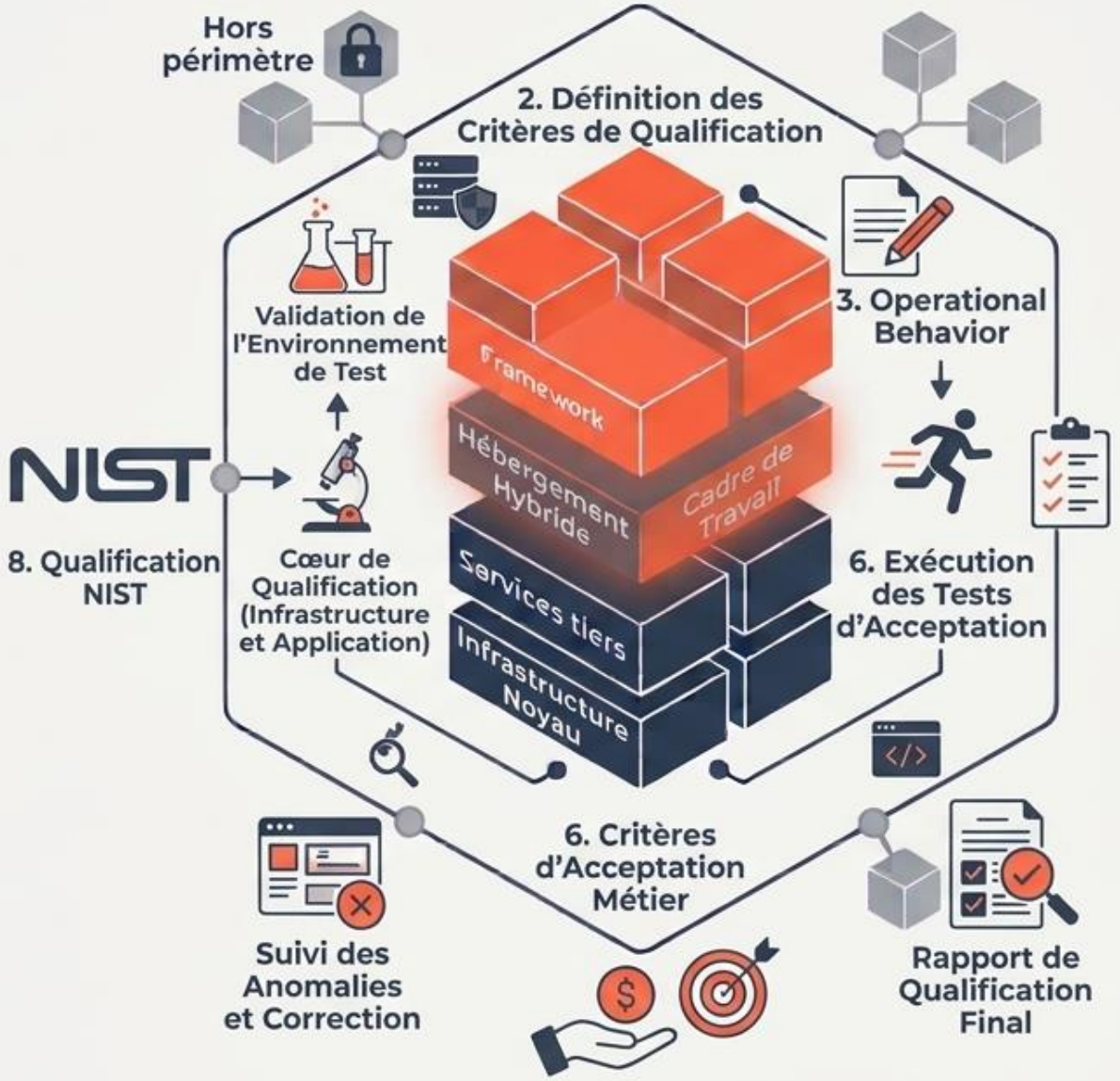


- **Statut de normalisation**
- **Disponibilité environnement**
- **Autorisation** par politique
- **Fournisseur** (exécution)
- Validation du module (et conditions)



Un composant robuste et maîtrisé, orienté développement et qualification

8. Stratégie de qualification et critères d'acceptation



8.1. Qualifier plusieurs niveaux

La validation sera organisée en quatre niveaux :

1. **primitives** : les opérations produisent les résultats attendus ;
2. **interface** : formats, tailles, erreurs et durées de vie sont corrects ;
3. **TLS** : les connexions utilisent réellement le composant ;
4. **service** : le système tient la charge et se comporte correctement en exploitation.

Un succès au premier niveau ne validera pas automatiquement les suivants.

8.2. Les campagnes indispensables

Campagne	Vérification attendue
Vecteurs connus	Résultats conformes aux vecteurs de référence retenus
Échanges normaux	Secrets concordants dans les essais contrôlés
Signatures	Acceptation des signatures correctes et rejet des modifications
Entrées incorrectes	Traitement maîtrisé des tailles et valeurs invalides
Défaillances	Refus propre en cas d'indisponibilité ou d'erreur interne
Interopérabilité	Connexions croisées avec OpenSSL natif
Concurrence	Absence de mélange entre contextes et connexions
Mémoire	Recherche de débordements, usages après libération et fuites
Endurance	Stabilité des ressources et du comportement
Observabilité	Résultats explicables sans divulgation de secrets

8.3. Les tests ignorés

Chaque contrôle affichera un statut explicite :

- réussi ;
- échoué ;
- non disponible ;
- non applicable ;
- non exécuté.

Une capacité obligatoire absente empêchera la qualification du profil correspondant.

Un bilan contenant des tests ML-KEM ignorés ne pourra pas être présenté comme une qualification complète du profil hybride.

8.4. Le temps constant

Le document QSA_TLSH rapport.docx rapporte des travaux de timing et des optimisations AES/CTR. Il mélange toutefois des éléments qui ne correspondent pas tous au noyau TLSH minimal. Ses chiffres ne peuvent donc pas être utilisés directement comme mesures de performance de notre futur TLSH.

Nous organiserons des campagnes reproductibles de recherche de fuites, en conservant la plateforme, le compilateur, les options et les résultats.

Un résultat favorable de `dudect` est un élément d'évaluation. Le projet officiel précise qu'il ne prouve pas, à lui seul, qu'un code est à temps constant. GitHub

8.5. Les performances à mesurer

Le banc distinguera :

- les temps de génération, encapsulation, décapsulation et signature ;
- la durée de négociation TLS ;
- les percentiles de latence, notamment médiane et 95e percentile ;
- le débit applicatif après établissement ;
- la consommation CPU et mémoire ;
- les octets échangés ;
- les taux d'échec et causes de refus.

Les comparaisons porteront d'abord sur **OpenSSL natif et QSA-REF**, dans le même environnement.

Je propose une campagne initiale de **10 000 connexions complètes**, complétée par **une heure d'endurance** à concurrence contrôlée. Ce sont des objectifs de recette proposés, à fixer dans le protocole de test, sans prétention de preuve générale de sécurité.

8.6. La règle d'acceptation

Le profil sera déclaré utilisable lorsque ses capacités obligatoires auront été exécutées, les essais négatifs validés, les échanges croisés réussis et les anomalies bloquantes corrigées.

Les performances serviront à repérer le surcoût de notre intégration. Aucun débit commercial ne sera annoncé avant mesure.

8 | Qualification, Tests et Performance

TLSH / QSA

8.1 Niveaux de qualification



8.2 Campagnes indispensables

- Vecteurs connus
- Échanges normaux
- Signatures
- Entrées incorrectes
- Interopérabilité
- Concurrence
- Mémoire
- Endurance
- Défaillances
- Observabilité

8.3 / 8.4 Contrôle des tests

Statuts des contrôles

- Réussi
- Échoué
- Non disponible
- Non applicable
- Non exécuté

Temps constant

- Campagnes reproductibles
- dudect = indicateur, pas preuve

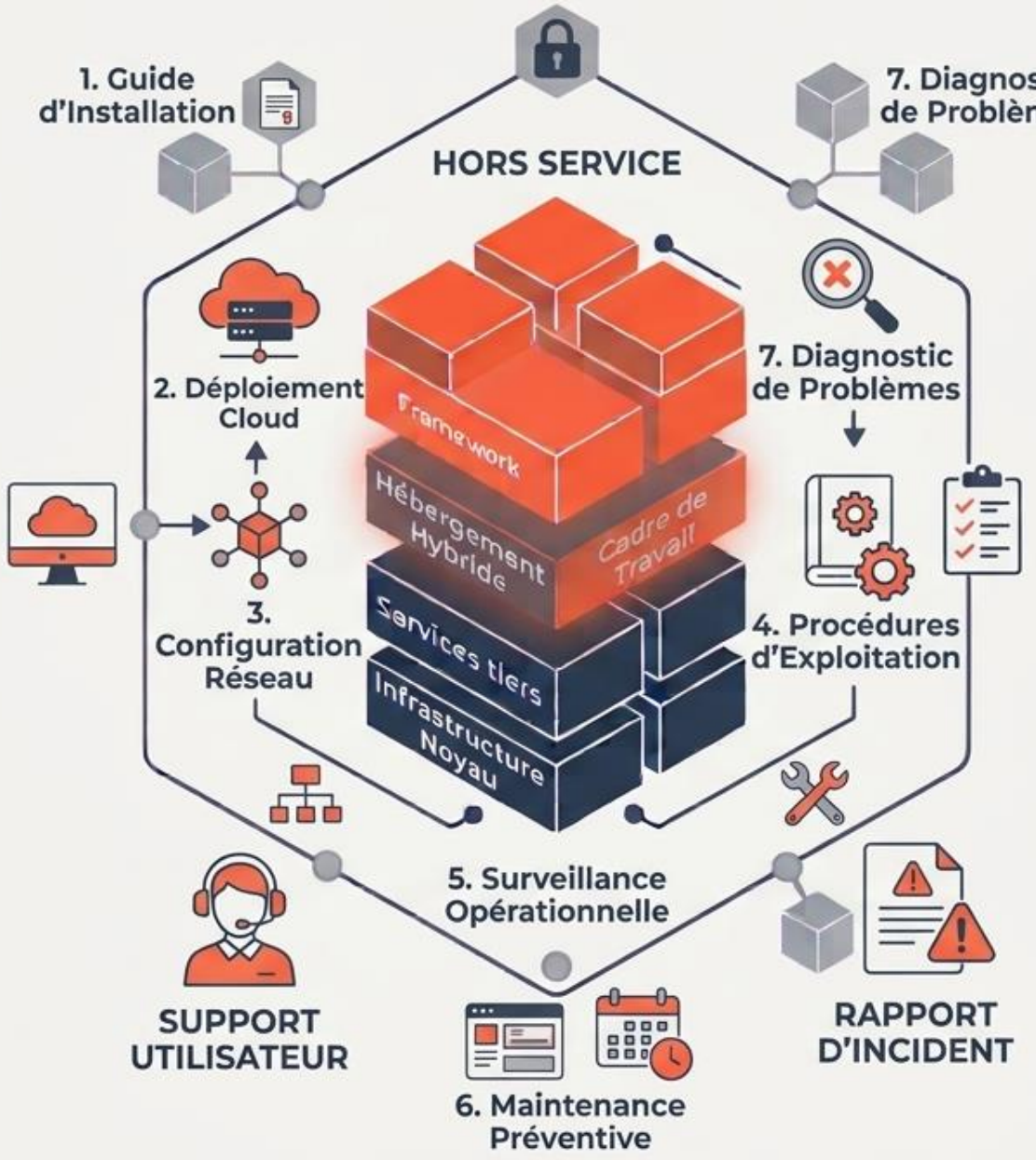
8.5 / 8.6 Mesures & acceptation

- Latence
- Débit
- CPU / Mémoire
- Taux d'échec
- 10 000 connexions + endurance

Profil accepté si capacités obligatoires, tests négatifs et échanges croisés sont validés.

Une qualification multi-niveaux, mesurable et vérifiable avant mise en service

9. Installation, exploitation et diagnostic



9.1. Les environnements cibles

Le premier périmètre couvrira :

- Linux pour le service et les campagnes automatisées ;
- Windows pour le développement et les essais locaux ;
- une exécution sur CPU ;
- des versions de dépendances figées.

Chaque plateforme disposera de son résultat de qualification propre.

9.2. Un installateur Python intégré

L'installation suivra ton principe d'automatisation.

Un script Python prendra en charge :

1. la détection de l'environnement ;
2. la vérification des dépendances ;
3. la préparation des bibliothèques et exécutables ;
4. la génération contrôlée de la configuration ;
5. les contrôles locaux ;
6. la préparation du service ;
7. la vérification du démarrage ;
8. la conservation du rapport.

Son interface utilisateur sera centrée sur `check` et `repair`.

9.3. La maîtrise des dépendances

L'installation utilisera une version dédiée d'OpenSSL pour le projet, avec des chemins de chargement explicites.

Les livrables seront accompagnés d'un manifeste comprenant versions, empreintes, provenance, options de compilation et licences.

Une mise à jour sera préparée dans un emplacement distinct, vérifiée, puis activée. Le retour à la version précédente devra être prévu.

9.4. La configuration

La configuration couvrira notamment :

- le moteur demandé ;
- le profil TLS ;
- les fournisseurs et bibliothèques attendus ;
- les certificats et autorités de confiance ;
- l'adresse et le port d'écoute ;
- les limites de concurrence ;
- les délais ;
- les niveaux et la rotation des journaux.

Les premiers essais utiliseront des connexions complètes. La reprise de session et le `0-RTT` resteront désactivés dans ce profil initial afin de rendre les mesures et le parcours cryptographique faciles à vérifier.

9.5. Les certificats et le réseau

Une autorité de laboratoire dédiée permettra de générer les certificats de test. Les clients utiliseront un magasin de confiance explicite, avec vérification du nom attendu et de la validité.

Le point de terminaison TLS devra être clairement identifié.

Si Nginx est placé devant TLSH pour ces essais, son rôle prévu sera un relais TCP jusqu'au service TLSH. La négociation étudiée devra bien arriver dans notre moteur.

L'installateur vérifiera aussi les conflits de port et la configuration réseau nécessaire.

9.6. Le rapport opérationnel

L'écran principal ou le rapport devra montrer, pour chaque essai :

- le profil demandé ;
- le groupe réellement négocié ;
- le type d'authentification ;
- le moteur effectivement utilisé ;
- les contrôles exécutés ;
- les durées ;
- le résultat et la cause d'échec éventuelle.

La progression sera affichée par étapes terminées, avec un détail consultable. Les exports de diagnostic excluront les clés, secrets et graines.

9 | Déploiement, Installation et Exploitation

9.1 Environnements cibles

- **Linux** : service & campagnes automatisées
- **Windows** : développement & essais locaux
- Exécution CPU
- Dépendances figées

Qualification propre par plateforme

9.2 Installateur Python intégré

1. Détection environnement
2. Vérification dépendances
3. Préparation bibliothèques / exécutables
4. Génération configuration
5. Contrôles locaux
6. Préparation du service
7. Vérification du démarrage
8. Conservation du rapport

Interface : **check** | **repair**

9.3 / 9.4 Dépendances & configuration

- OpenSSL dédié au projet
- Manifeste : versions / empreintes / provenance
- Mise à jour vérifiée + retour arrière

- Profil TLS
- Fournisseurs / bibliothèques
- Certificats / AC
- Adresse / port
- Concurrence / délais / journaux

0-RTT et reprise de session désactivés au profil initial

9.5 Certificats & réseau

- AC de laboratoire pour les tests
- Point de terminaison TLS dentifié

- Vérification du nom et de la validité
- Nginx éventuel = relais TCP vers TLSH

Vérification des ports et de la configuration réseau

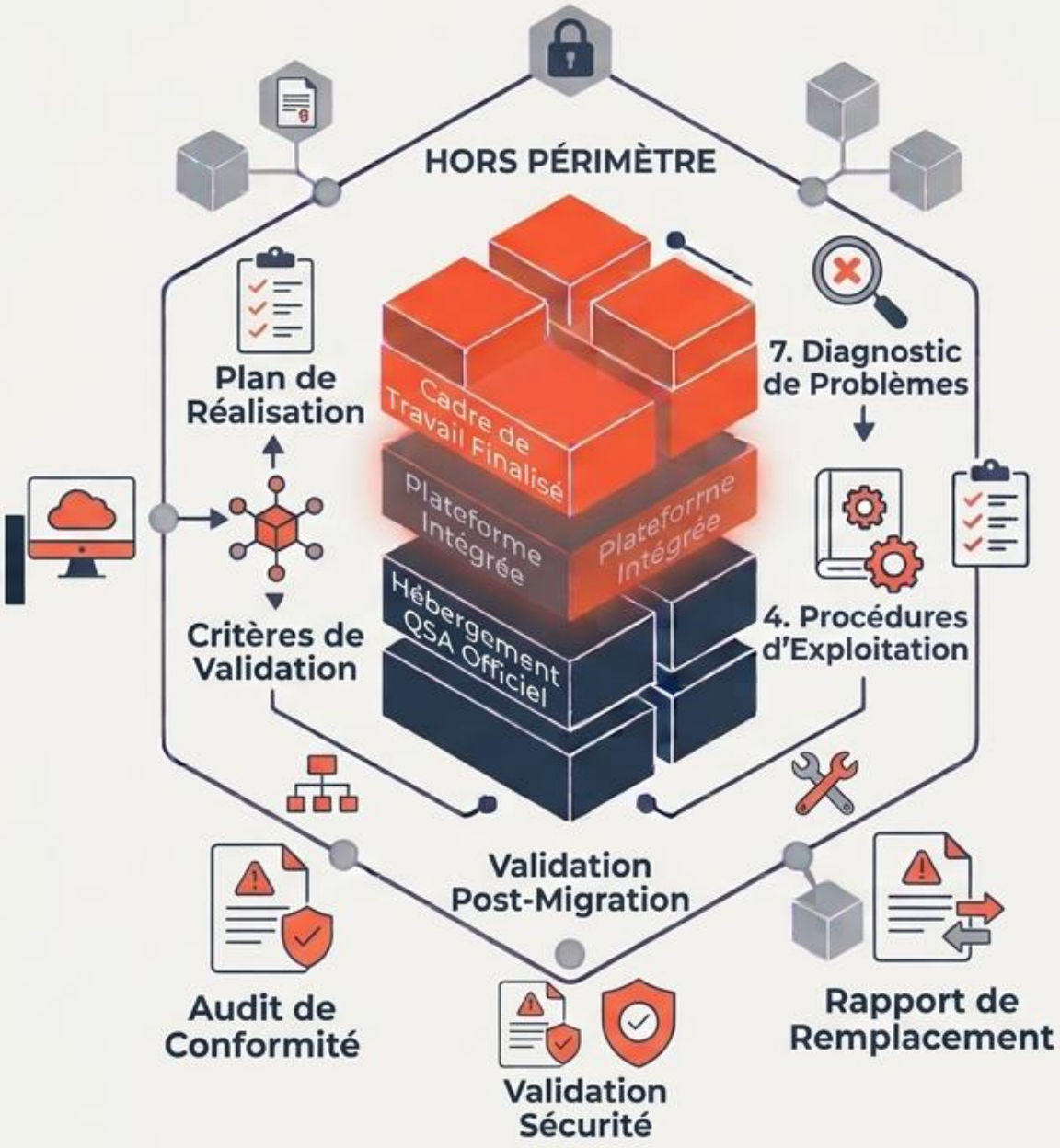
9.6 Rapport opérationnel

- Profil demandé
- Groupe négocié
- Authentification
- Moteur utilisé
- Contrôles exécutés
- Durées
- Résultat / cause d'échec

Progression par étapes terminées, détail consultable, sans clés ni secrets dans les exports.

Un déploiement automatisé, maîtrisé et vérifiable pour TLSH

10. Plan de réalisation et remplacement par le QSA officiel



10.1. Mega Batch QSA-REF-001 — Socle et opérations cryptographiques

Objectif : disposer d'une bibliothèque de référence réellement fonctionnelle.

Ce lot comprendra l'API C versionnée, le contexte de configuration, les capacités, X25519, ML-KEM-768, ML-DSA-44, les erreurs, la gestion des objets et les tests de primitives.

Il intégrera dès le départ le squelette de l'installateur Python et le manifeste des dépendances.

Critère de sortie : toutes les opérations obligatoires sont exécutées avec succès, et leurs principaux cas d'erreur sont couverts.

10.2. Mega Batch QSA-REF-002 — Connexion TLSH et raccordement réel

Objectif : démontrer que TLSH fonctionne avec le composant sélectionné.

Ce lot réunira :

- le service Rust ;
- le profil TLS hybride natif servant d'étalon ;
- le provider QSA ;
- son raccordement à QSA-REF ;
- les certificats de laboratoire ;
- les preuves du chemin exécuté ;
- les tests croisés avec OpenSSL natif.

Critère de sortie : une requête applicative traverse une connexion TLS hybride utilisant effectivement QSA-REF, avec authentification serveur vérifiée.

10.3. Mega Batch QSA-REF-003 — Qualification opérationnelle et authentification avancée

Objectif : obtenir un environnement complet et reproductible.

Ce lot regroupera :

- l'authentification TLS ML-DSA de laboratoire, avec ses conditions de compatibilité ;
- les essais négatifs approfondis ;
- la concurrence et l'endurance ;
- les mesures comparatives ;
- le diagnostic exploitable ;
- l'installation Linux et Windows ;
- la gestion automatisée du service ;
- la procédure de mise à jour et de retour arrière.

Critère de sortie : les profils disponibles sont identifiés séparément, avec leur périmètre exact et leurs preuves de qualification.

Une difficulté sur le profil d'authentification ML-DSA ne devra pas empêcher de continuer les travaux TLSH couverts par le profil hybride déjà qualifié.

10.4. Mega Batch QSA-REF-004 — Adaptation au composant de Christophe

Ce lot sera engagé à réception du paquet nécessaire :

- sources et en-têtes ;
- instructions de compilation ;
- dépendances et versions ;
- vecteurs et tests ;
- formats de clés et signatures ;
- règles mémoire et concurrence ;
- droits d'utilisation.

Il comprendra l'adaptateur, les tests de contrat, les échanges croisés et les comparaisons de comportement.

La substitution transparente du groupe standard ne sera possible que si les algorithmes, formats et comportements sont compatibles. Une construction cryptographique différente devra disposer de son propre profil et de sa qualification.

10.5. Les conditions de bascule

La sélection du QSA de Christophe sera explicite.

Son activation exigera :

1. la compatibilité de l'interface ;
2. la réussite des opérations obligatoires ;
3. la réussite des connexions croisées ;
4. le respect des règles de refus et de protection des secrets ;
5. des résultats de performance documentés ;
6. un retour maîtrisé vers la référence.

Un échec de qualification bloquera l'activation de ce moteur. QSA-REF restera disponible sous son identité propre, conformément au principe de ta capture.

10 | Mega Batches – Plan d'industrialisation et de qualification

10.1 QSA-REF-001

Socle crypto (bibliothèque)



Objectif

Bibliothèque de référence fonctionnelle.

- API C versionnée
- Primitives (X25519, ML-KEM-768, ML-DSA-44)
- Erreurs et gestion des objets
- Tests de primitives
- Installateur Python (squelette)

10.2 QSA-REF-002

Connexion TLSH et raccordement réel



Objectif

Prouver le fonctionnement de TLSH avec QSA-REF.

- Service Rust (TLSH)
- Profil TLS hybride (natif étalon)
- Provider QSA et raccordement
- Certificats de laboratoire
- Preuves du chemin exécuté
- Tests croisés (OpenSSL natif)

10.3 QSA-REF-003

Qualification opérationnelle et authentification avancée



Objectif

Environnement complet et reproductible.

- Authentification TLS ML-DSA
- Essais négatifs approfondis
- Concurrence et endurance
- Mesures comparatives
- Diagnostic exploitable
- Installation Linux / Windows
- Gestion du service et mises à jour

10.4 QSA-REF-004

Adaptation au composant de Christophe



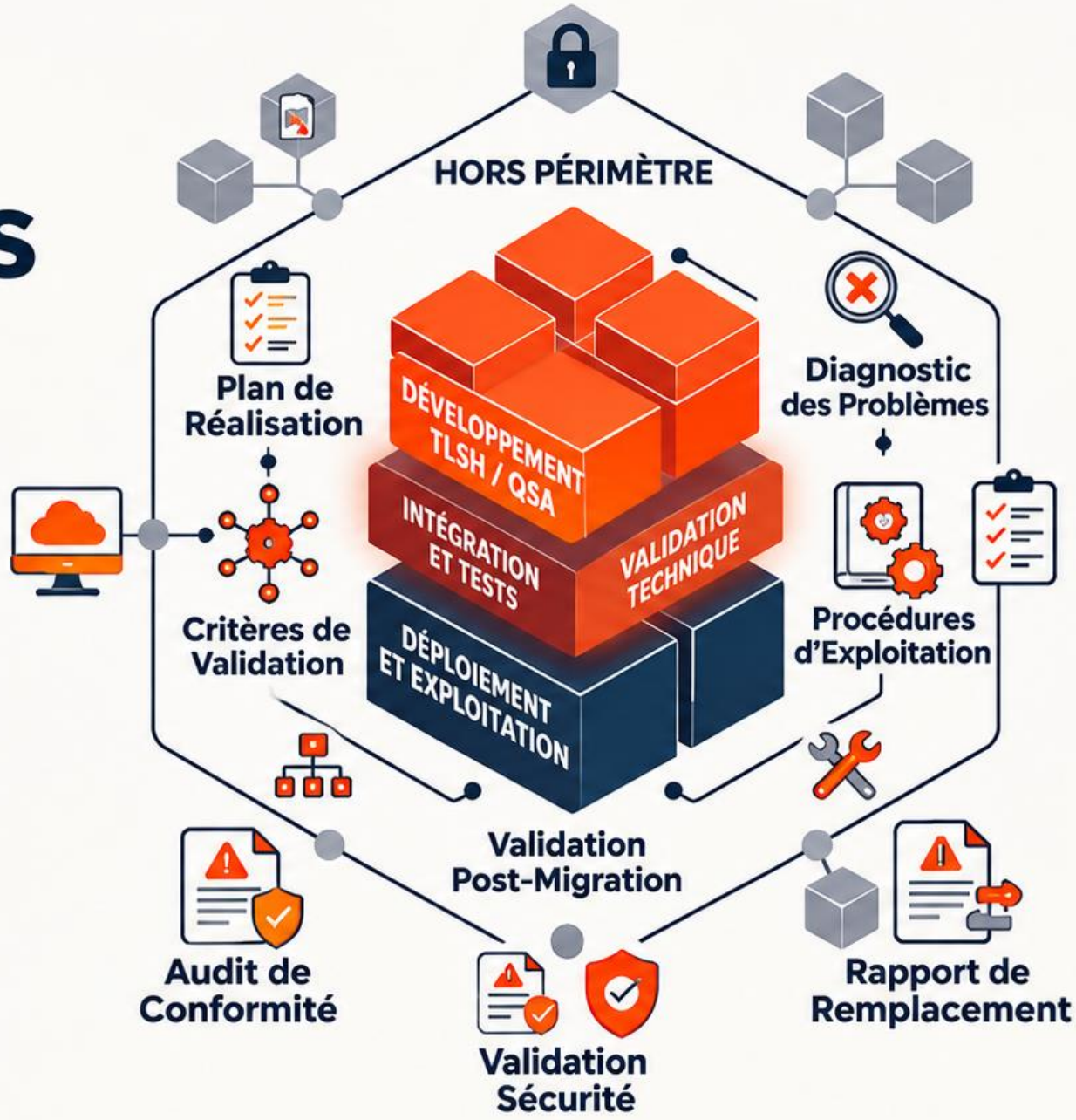
Objectif

Réception et intégration du paquet de Christophe.

- Sources et en-têtes
- Compilation et dépendances
- Vecteurs et tests
- Formats de clés et signatures
- Règles mémoire et concurrence
- Droits d'utilisation

Résultat : un composant qualifié, intégré et prêt pour l'industrialisation

11. Roadmap des phases de développement



AGENDA TECHNIQUE DE LA ROADMAP

- 1. Le choix technique recommandé
- 2. Vue d’ensemble de la roadmap
- 3. Mega Batch QSA-REF-001 — Socle Python et primitives réelles
- 4. Mega Batch QSA-REF-002 — Raccordement natif et connexion TLSH
- 5. Mega Batch QSA-REF-003 — Qualification et exploitation
- 6. Mega Batch QSA-REF-004 — Intégration du QSA original
- 7. Ce que nous conserverons du développement Python

GENERALITES SUR LA MISE EN OEUVRE

Élément	Rôle de Python
Configuration	Charger et vérifier les paramètres, sélectionner les profils
Interface QSA-REF	Présenter des fonctions cohérentes au reste du projet
Gestion des capacités	Identifier les algorithmes réellement disponibles
Contrôle des appels	Vérifier les paramètres et traduire les erreurs
Scénarios de test	Enchaîner les opérations et vérifier leurs résultats
Diagnostic	Expliquer les réussites, indisponibilités et échecs
Rapports	Produire les résultats lisibles et les exports
Installation	Vérifier et préparer automatiquement l'environnement
Qualification	Piloter les essais de concurrence, d'endurance et de performance

1.2. Ce qui serait contre-indiqué

Je déconseille de réécrire les calculs internes de ML-KEM, ML-DSA, X25519 ou AES sous forme d'algorithmes Python propres au projet.

Notre travail portera sur **leur utilisation correcte, leur intégration, les contrôles et les preuves de fonctionnement**.

Une deuxième limite concerne la mémoire. Certaines API Python retournent les secrets sous forme d'objets `bytes`, dont l'effacement fiable ne peut pas être garanti. Supprimer une variable avec `del` ne démontre pas que toutes les copies du secret ont disparu. Cette limitation est explicitement documentée par `cryptography`. Cryptography 50....

Nous distinguerons donc :

- **la référence fonctionnelle Python**, destinée au développement et aux essais ;
- **le parcours natif intégré à TLSH**, dans lequel nous maîtriserons davantage les objets secrets, leurs propriétaires et leur destruction.

La seconde étape devra elle aussi être testée : écrire du C ne constitue pas, à lui seul, une garantie de sécurité.

1.3. Pourquoi Python reste intéressant pour les performances

Les opérations cryptographiques seront exécutées par le code natif de la bibliothèque. Le coût propre à Python concernera surtout la préparation des appels, les contrôles, l'enchaînement des opérations et la collecte des résultats.

Nous mesurerons séparément :

1. le temps des opérations cryptographiques ;
2. le coût de notre interface ;
3. le temps d'une connexion TLS complète.

Cela permettra de déterminer où se situe réellement un éventuel ralentissement.

Nous n'aurons donc pas besoin de traduire immédiatement tout le projet en Rust pour commencer à produire des résultats utiles.

1.4. La distinction entre les bibliothèques installées

Le diagnostic devra identifier séparément :

- la bibliothèque utilisée par `cryptography` ;
- celle utilisée par le module Python `ssl`, s'il intervient ;
- celle utilisée par TLSH ;
- l'éventuel exécutable OpenSSL utilisé pour les essais.

Ces composants peuvent employer des versions différentes. Les distributions binaires de `cryptography` peuvent embarquer leur propre OpenSSL : leur installation ne met pas automatiquement à niveau celui de Python ou du système. Cryptography 50....

Les interfaces TLS disponibles dépendent également de la version de Python. Par exemple, Python 3.15 documente désormais des fonctions permettant de sélectionner les groupes et de connaître le groupe négocié. Notre installateur devra vérifier les capacités réellement présentes, sans les déduire uniquement du nom « Python ». Python 3.15.2 do...

1.5. L’architecture que je retiens

Partie du projet	Technologie retenue
QSA-REF initial, pilotage et qualification	Python
Calculs cryptographiques de référence	Implémentations natives accessibles par `cryptography`
Interface directement utilisable par TLSH	Bibliothèque native avec API C
Raccordement au moteur TLS	Provider OpenSSL dédié
Service TLSH	Rust, conformément à son architecture prévue
Installation et mises à jour	Scripts Python intégrés

2. Vue d’ensemble de la roadmap

Cette roadmap porte sur **QSA-REF, le composant temporaire destiné à débloquer TLSH.**

Mega Batch	Objectif principal	Résultat concret
QSA-REF-001	Socle Python et primitives cryptographiques réelles	Un QSA de référence autonome, installable et testable
QSA-REF-002	Raccordement natif et intégration dans TLSH	Une connexion TLS hybride utilisant effectivement le parcours QSA
QSA-REF-003	Qualification, exploitation et authentification avancée	Un environnement reproductible, mesuré et documenté
QSA-REF-004	Adaptation au QSA de Christophe	Une substitution qualifiée, avec retour possible à la référence

Mega Batch QSA-REF-001 — Socle Python et primitives réelles

Premier Mega Batch de Patch

Mettre en place une référence Python simple, utile et qualifiée pour QSA / TLSH

But

- Piloter QSA-REF en Python
- Tester, diagnostiquer, qualifier
- Préparer l'intégration TLSH

Rôle de Python

- Configuration
- Contrôle des appels
- Scénarios de test
- Rapports & diagnostic
- Installation & qualification

Python orchestre, il ne refait pas la cryptographie.

À ne pas faire

- Ne pas réécrire ML-KEM, ML-DSA, X25519 ou AES en Python
- Ne pas gérer les secrets critiques uniquement via objets Python

Architecture retenue

Python → Implémentations natives via cryptography → Bibliothèque native API C → Provider OpenSSL → Service TLSH en Rust

- Mesures de performance séparées
- Détection des bibliothèques réelles
- Base exploitable pour la suite

Résultat attendu : une base de référence Python simple, fiable et immédiatement exploitable

3.2. Fichiers prévus

Les noms ci-dessous sont **proposés pour le nouveau projet** ; ils ne désignent pas des fichiers déjà existants.

Fichier	Responsabilité
`qsa_ref.py`	Point d'entrée et pilotage des opérations
`qsa_ref_config.py`	Configuration, profils et validation des paramètres
`qsa_ref_contracts.py`	Types de résultats, capacités, erreurs et version du contrat
`qsa_ref_crypto.py`	Accès aux opérations cryptographiques natives
`qsa_ref_checks.py`	Diagnostic et contrôles fonctionnels
`install_qsa_ref.py`	Installation et réparation automatisées
`requirements_qsa_ref.txt`	Dépendances précisément verrouillées

Fonctions livrées

Initialisation et inventaire

Le programme devra identifier :

- son propre numéro de version ;
- la version de Python ;
- la version de la bibliothèque cryptographique ;
- le moteur natif réellement utilisé ;
- les algorithmes disponibles ;
- les fonctions obligatoires absentes.

Une capacité annoncée devra être vérifiée par une opération réelle.

X25519

Le composant devra permettre de :

- générer les clés de deux correspondants ;
- produire leurs données publiques ;
- calculer le secret partagé de chaque côté ;
- vérifier la concordance des résultats ;
- traiter les entrées invalides.

ML-KEM-768

Le composant devra permettre de :

- générer une paire de clés ;
- encapsuler vers la clé publique ;
- décapsuler avec la clé privée ;
- vérifier que les deux opérations produisent le secret attendu ;
- contrôler les formats et les longueurs.

Les résultats seront représentés par des champs nommés. Cela évitera une confusion le secret partagé lors du raccordement des bibliothèques.

ML-DSA-44

Le composant devra permettre de :

- générer les clés de signature ;
- signer un message ;
- vérifier une signature ;
- détecter une modification du message ;
- détecter une modification de la signature ;
- refuser une vérification avec une clé incompatible.

3.5. Installation

L'installateur proposera les deux opérations que tu souhaites privilégier :

- `check` : examiner l'environnement et produire le diagnostic ;
- `repair` : préparer ou corriger l'installation selon les paramètres du projet.

Il vérifiera les dépendances, préparera l'environnement dédié, exécutera les contrôles et conservera les résultats.

Les versions exactes seront verrouillées après vérification de leur compatibilité et des correctifs nécessaires. L'installation utilisera les distributions binaires adaptées lorsqu'elles sont disponibles.

3.6. Résultat visible

À la fin du lot, tu pourras obtenir un rapport présentant notamment :

Contrôle	Résultat attendu
Environnement Python	Compatible
Moteur cryptographique	Identifié
X25519	Opérations exécutées
ML-KEM-768	Opérations exécutées
ML-DSA-44	Opérations exécutées
Entrées incorrectes	Traitement conforme
Capacités manquantes	Refus explicite
Journalisation	Aucun secret exposé

Une progression indiquera les étapes terminées et les éventuels points bloquants.