Protocole TLS Hybride Post-Quantique

TLSH & QSA

Plan Développement



By **Lockheed** LLC

19 Novembre 2026

Auteur : Guillaume Oneill

Version 1.1

AGENDA GÉNÉRAL

1. Présentation générale de TLSH
2. Cahier des charges techniques
3. Analyse et plan de réalisation
4. Vue générale de la roadmap
5. Plan de réalisation détaillé
6. Roadmap consolidée
7. Traçabilité
8. Plan test & installation



En Préambule à cette présentation



Contexte



Objectifs



Périmètre



Méthode

Le prototype, en cinq étapes

Étape	Contenu	Ce que vous pourrez montrer	Exigences
1	Politique, moteur de décision, signalisation client (sans réseau)	Capteur de centrale, frigo, trafic aérien : chaque cas donne la décision attendue, tracée. Une signalisation absente ou malformée donne un défaut sûr.	EF-01, 02, 07, 09
2	Interface multi-algorithmes sur QSA-REF, handshake adaptatif client-serveur sur TCP	Un capteur simulé et un serveur ouvrent une session chiffrée dans le mode décidé, avec la vraie cryptographie	EF-03, 04
3	Ticket de session, preuve, 0-RTT	Reprise sans aller-retour, rejeu refusé, preuve obligatoire en Top Secret, temps de vérification mesuré	EF-05, 06
4	Mesures et rapport	Octets, CPU et latence par mode, adaptatif contre « toujours maximal »	EF-08
5	Ancrage sur un vrai TLS	Connexion OpenSSL 3.5 hybride comme référence, puis notre extension dans un vrai ClientHello	EF-10

[Cryptography-dev] PyCA cryptography 48.0.0 released

Paul Kehrer via Cryptography-dev | Mon, 04 May 2026 16:01:35 -0700

PyCA cryptography 48.0.0 has been released to PyPI. cryptography includes both high level recipes and low level interfaces to common cryptographic algorithms
such as symmetric ciphers, asymmetric algorithms, message digests, X.509, key derivation functions, and much more. We support Python 3.9+, and PyPy3 3.11.

Changelog (https://cryptography.io/en/latest/changelog/#v48-0-0)
* BACKWARDS INCOMPATIBLE: Support for Python 3.8 has been removed. cryptography now requires Python 3.9 or later.
* BACKWARDS INCOMPATIBLE: Loading an X.509 CRL whose inner TBSCertList.signature algorithm does not match the outer signatureAlgorithm now raises ValueError. Previously, such CRLs were parsed successfully and only rejected during signature validation.
* Added support for ML-KEM key encapsulation and ML-DSA signing when using OpenSSL 3.5.0 or later, in addition to the existing AWS-LC and BoringSSL support. This means post-quantum algorithms are now available to users of our wheels.
  * Note: Going forward, we do not guarantee that all functionality in cryptography will be available when building against OpenSSL. See (https://cryptography.io/en/latest/statements/state-of-openssl/) for more information.

-Paul Kehrer (reaperhulk)

Shipping post-quantum cryptography to Python

Alexis Challande June 30, 2026 cryptography, open-source, post-quantum

Post-quantum cryptography is now one `pip-install` away for the entire Python ecosystem. With funding from the **Sovereign Tech Agency**, we implemented support for ML-KEM, the NIST-standard key-establishment primitive, and ML-DSA, the NIST-standard digital-signature primitive, in `pyca/cryptography`.

On June 22, 2026, the White House **ordered** the U.S. government to accelerate its transition to post-quantum cryptography. The order says large-scale quantum computers, especially in adversarial hands, will threaten widely used cryptographic systems, and that attackers may already be collecting encrypted data now so they can decrypt it later. It also sets concrete migration deadlines: high-value and high-impact federal systems must use post-quantum key establishment by **December 31, 2030**, and post-quantum digital signatures by **December 31, 2031**. And even if you don't care about quantum resistance, that's not a problem because **quantum resistance isn't the main benefit of post-quantum crypto**.

That transition cannot happen only at the policy layer. Every application that signs packages, validates certificates, establishes secure channels, or protects long-lived secrets depends on cryptographic libraries. If those libraries do not expose post-quantum algorithms, the software stack cannot migrate.

Almost every Python program that touches cryptography goes through `pyca/cryptography`. It's currently the **eleventh most-downloaded package on PyPI**, pulling 1.2 billion downloads in the last month alone. The `pyca/cryptography` package handles the cryptographic operations of projects like Ansible, Certbot (the Let's Encrypt client), Apache Airflow, paramiko (the Python-only SSH client), and **many others**. If `pyca/cryptography` doesn't ship post-quantum primitives, the Python ecosystem can't begin to migrate.

Là où ça coince

Le module `ssl` standard de Python n'expose ni les extensions TLS personnalisées (vos trois signaux client), ni le 0-RTT. Deux voies :

	Voie A : handshake TLS 1.3 écrit en Python	**Voie B : greffe sur libssl 3.5 via `ctypes`**
Rapidité pour la démo	Élevée, tout est sous contrôle	Plus laborieuse (callbacks C)
Fidélité au positionnement « ne réécrit pas TLS »	Faible : c'est une maquette	Forte
Interopérabilité Nginx/OpenSSL	Non garantie	Native

Je recommande A pour la démo, présentée explicitement comme maquette, puis B pour l'industrialisation. Si vous n'aviez besoin que de l'échange hybride sans vos extensions, OpenSSL 3.5 propose déjà X25519MLKEM768 dans ses keyshares par défaut. openssl

Nuitka

Il apporte un binaire distribuable, un code source moins exposé et un léger gain de démarrage. Il n'apporte ni temps constant, ni effacement des secrets en mémoire, ni accélération de la crypto (elle est déjà en C). C'est un outil de packaging, pas un argument de sécurité. Il faudra embarquer `libcrypto` et la bibliothèque QSA dans le build.

	QSA-REF = façade Python sur `cryptography`	QSA-REF = algorithmes écrits en Python pur
Ce que c'est	Votre API QSA en Python, les calculs faits par OpenSSL 3.5 en dessous	ML-KEM-768, ML-DSA-44 et X25519 recodés d'après FIPS 203/204
Temps constant (exigé par votre analyse)	Oui, hérité d'OpenSSL	Non, impossible en Python
Vitesse	Native	Lente (de l'ordre de la milliseconde à plusieurs dizaines par opération)
Effort	Faible	Moyen, plus la validation sur les vecteurs NIST
Usage	Démo réaliste et mesures de performance	Pédagogie et référence lisible uniquement

DESCRIPTION

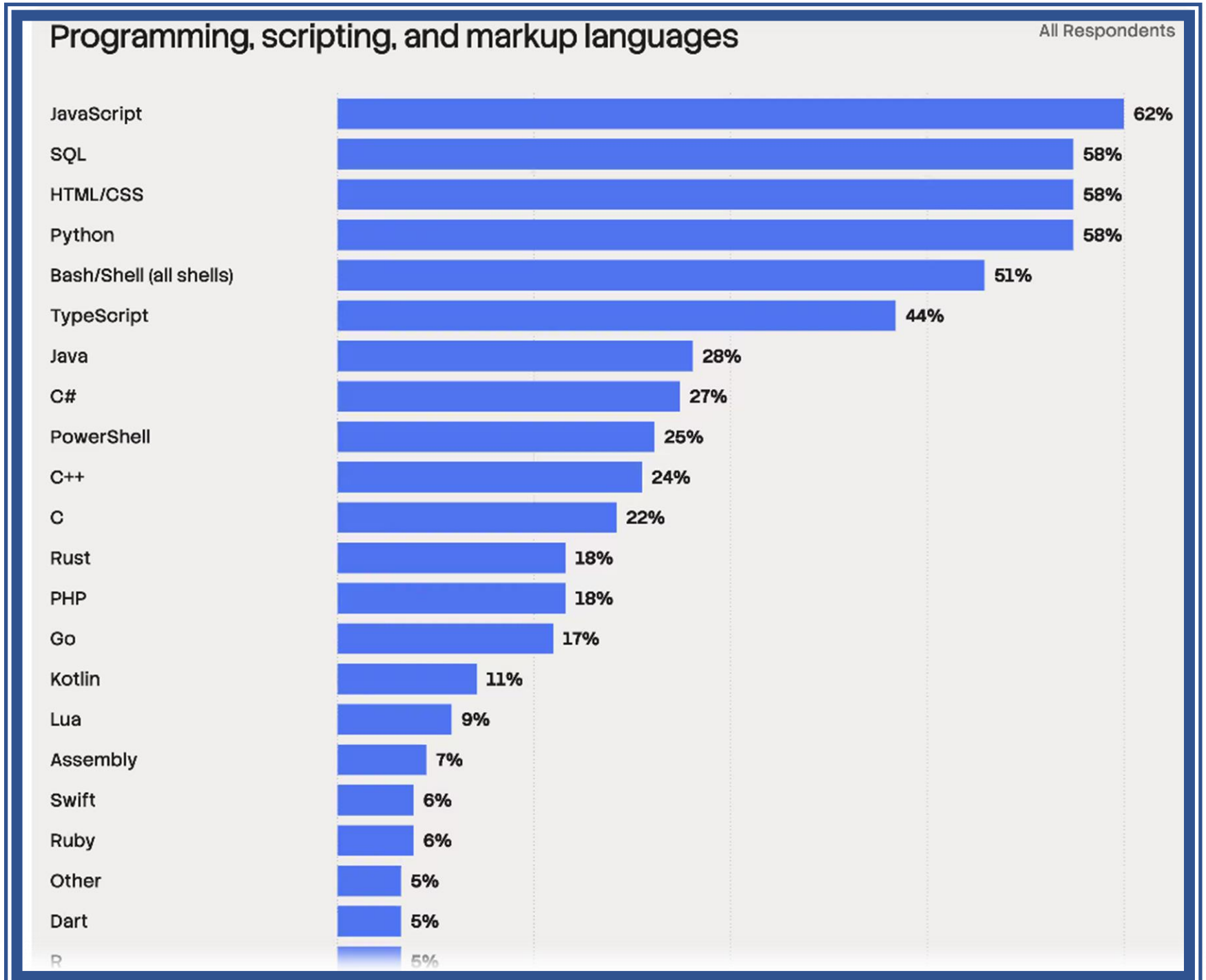
For all of the functions below that set the supported groups there must be at least one group in the list. A number of these functions identify groups via a unique integer **NID** value. However, support for some groups may be added by external providers. In this case there will be no **NID** assigned for the group. When setting such groups applications should use the "list" form of these functions (i.e. SSL_CTX_set1_groups_list() and SSL_set1_groups_list()).

SSL_CTX_set1_groups() sets the supported groups for **ctx** to **glistlen** groups in the array **glist**. The array consist of all **NIDs** of supported groups. The supported groups for **TLSv1.3** include: **NID_X9_62_prime256v1**, **NID_secp384r1**, **NID_secp521r1**, **NID_X25519**, **NID_X448**, **NID_brainpoolP256r1tls13**, **NID_brainpoolP384r1tls13**, **NID_brainpoolP512r1tls13**, **NID_ffdhe2048**, **NID_ffdhe3072**, **NID_ffdhe4096**, **NID_ffdhe6144**, and **NID_ffdhe8192**. OpenSSL will use this array in different ways based on the TLS version, and whether the groups are used in a client or server.

For a TLS client, the groups are used directly in the supported groups extension. The extension's preference order, to be evaluated by the server, is determined by the order of the elements in the array.

For a TLS 1.2 server, the groups determine the selected group. If **SSL_OP_SERVER_PREFERENCE** is set, the order of the elements in the array determines the selected group. Otherwise, the order is ignored and the client's order determines the selection.

For a TLS 1.3 server, the groups determine the selected group, but selection is more complex. A TLS 1.3 client sends both a group list and predicted keyshares for a subset of groups. A server choosing a group outside the client's predicted subset incurs an extra roundtrip. However, in some situations, the most preferred group may not be predicted.


Programming, scripting, and markup languages
All Respondents
JavaScript 62%
SQL 58%
HTML/CSS 58%
Python 58%
Bash/Shell (all shells) 51%
TypeScript 44%
Java 28%
C# 27%
PowerShell 25%
C++ 24%
C 22%
Rust 18%
PHP 18%
Go 17%
Kotlin 11%
Lua 9%
Assembly 7%
Swift 6%
Ruby 6%
Other 5%
Dart 5%
R 5%

Rust

Rust is the fastest-growing programming language, with its developer community doubling in size over the past two years, yet JavaScript remains the most popular language with 25.2 million active developers, according to the results of a recent survey.

The 26th edition of SlashData's Developer Nation survey showed that the Rust community doubled its number of users over the past two years — from two million in the first quarter of 2022 to four million in the first quarter of 2024 — and by 33% in the last 12 months alone. The SlashData report covers the first quarter of 2024.

"Rust has developed a passionate community that advocates for it as a memory-safe language which can provide great performance, but cybersecurity concerns may lead to an even greater increase," the report said. "The USA and its international partners have made the case in the last six months for adopting memory-safe languages."

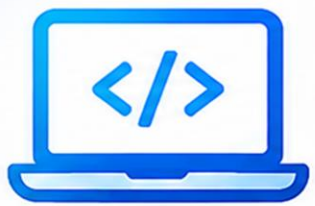
However, Rust is not unique in this capability, but its performance benefits and memory safety are likely to lead to further growth in this developer community, the report projects.





TLSH

Présentation général de TLSH



Concept



Architecture



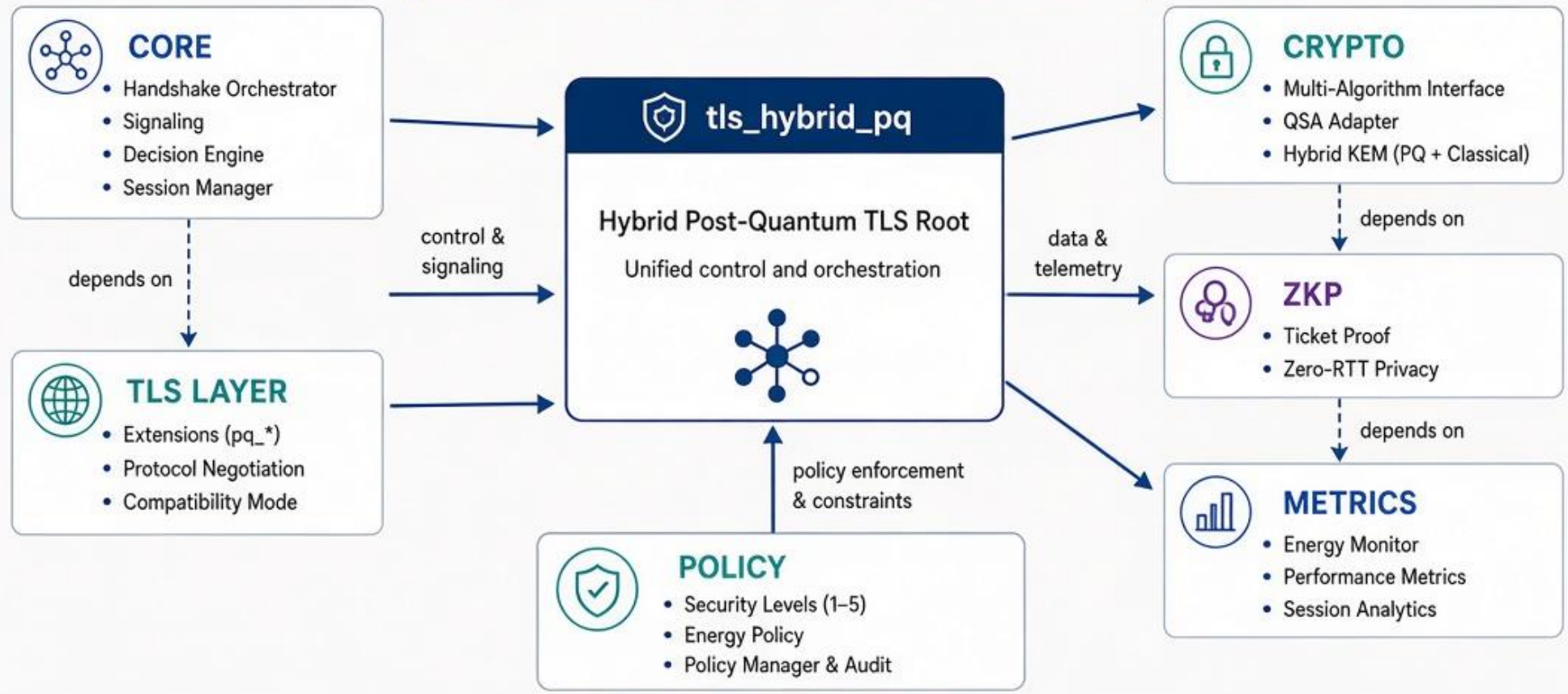
Fonctionnalités



Bénéfices

Hybrid Post-Quantum TLS System

tls_hybrid_pq – Layered Architecture & Data Flow



CRYPTOGRAPHY
Post-Quantum Primitives

ENERGY
Consumption & Efficiency

NETWORK
Transport & Signaling

SECURITY
Integrity & Authentication

ZERO-KNOWLEDGE
Privacy-Preserving Proofs

Data Flow (Primary) — Dependencies (Module) — Policy / Telemetry Influence

3. Organisation et responsabilités des modules principaux

Module	Rôle principal	Correspondance analyse
`signaling.py`	Collecte et émission de la signalisation client	EF-01, Chapitre 5.1
`decision_engine.py`	Décision du niveau hybride selon politique + signaux	EF-02, Chapitre 5.2
`multi_algo_interface.py` + `qsa_adapter.py`	Abstraction et appel des primitives PQ	EF-03, EF-04, Chapitre 5.3
`ticket_proof.py` + `zero_rtt.py`	Génération/vérification ZKP et 0-RTT sûr	EF-05, EF-06, Chapitre 5.4
`policy_manager.py`	Application des niveaux de sécurité et règles énergie	EF-07, EF-08, Chapitre 5.5
`handshake.py`	Orchestration globale du flux	Chapitre 6.1
`extensions.py`	Intégration non invasive dans TLS 1.3	EF-10
`metrics/`	Mesure des résultats attendus (énergie, latence)	Chapitre Résultats attendus

4. Approche de développement recommandée

1. **Prototype rapide** : implémenter d'abord le moteur de décision + signalisation + interface multi-algo (avec QSA ou liboqs).
2. **Ajouter le ZKP** : intégrer la validation de tickets (commencer par une preuve simple puis optimiser la performance < 1 ms).
3. **Mesures** : instrumenter dès le début les métriques énergie/latence pour valider les objectifs.
4. **Tests des cas d'usage** : scénarios automatisés correspondant aux cas du Chapitre 8.
5. **Ouverture** : documenter les interfaces pour faciliter l'ajout d'algorithmes et les tests ACVP.

5. Points d’attention

- Garder les interfaces stables (surtout multi-algo et politique) pour respecter l’exigence d’extensibilité.
- Prévoir dès le départ des points d’instrumentation pour les résultats attendus (énergie, latence, taille).
- Isoler le module QSA derrière un adaptateur pour pouvoir le remplacer ou le compléter facilement.
- Documenter les politiques en YAML/JSON pour qu’elles soient modifiables sans recompiler.

Cahier des charges technique



AGENDA — Plateforme TLS hybride post-quantique

- 01 — Finalité et objectifs du projet
- 02 — Périmètre fonctionnel de la première version
- 03 — Référentiel normatif et niveaux de conformité
- 04 — Gouvernance, responsabilités et prérequis
- 05 — Architecture logicielle de référence
- 06 — Spécifications du noyau cryptographique QSA_TLSH
- 07 — Interface C/Rust et moteur d’exécution
- 08 — Politiques TLS, certificats et gestion des sessions
- 09 — Spécifications fonctionnelles d’OPT_TLSH
- 10 — Objectifs de performance et protocole de mesure
- 11 — Modèle de menace et durcissement
- 12 — Exploitation, automatisation et supervision
- 13 — Stratégie d’assurance qualité et de validation
- 14 — Parcours de validation FIPS et dossier de conformité
- 15 — Propriété intellectuelle, valorisation et jalons

Lockheed — Plateforme TLS hybride post-quantique

QSA_TLSH · Architecture C/Rust · OPT_TLSH

Version : 1.0 — Document de cadrage technique

Date : 11 novembre 2026

Porteur du projet : Lockheed

Statut : proposition à valider avant développement

Diffusion prévue : équipe projet et partenaires autorisés sous confidentialité

Objet : concevoir une solution TLS hybride interopérable, intégrant le composant cryptographique de Christophe et un moteur d'optimisation développé pour Lockheed, avec des preuves de sécurité, de performance et de valeur industrielle.

Limite documentaire : ce cahier des charges repose sur le message de Christophe et nos échanges. Le code QSA_TLSH, ses résultats de tests, les exigences reçues par courriel et les échanges avec les acquéreurs n'ont pas été examinés. Leur réception constitue le premier jalon.

Les formulations « **doit** » et « **interdit** » expriment des exigences obligatoires. Les objectifs chiffrés ci-dessous sont des **seuils proposés pour la recette**, et non des performances déjà démontrées.

1.1. Finalité industrielle

Développer un actif logiciel autonome permettant de sécuriser des communications TLS par un établissement de clés hybride classique/post-quantique, tout en apportant une optimisation mesurable de leur traitement.

Le produit doit pouvoir être intégré dans une infrastructure cliente, évalué indépendamment de WSA, puis commercialisé sous licence ou faire l'objet d'une opération de valorisation.

Le financement ultérieur de la R&D de Lockheed constitue un **objectif économique**, mais ne doit pas être considéré comme acquis dans le financement du développement.

1.2. Objectifs obligatoires

Identifiant	Objectif	Résultat attendu
OBJ-01	Sécurité	Architecture documentée, tests reproductibles et audit indépendant.
OBJ-02	Interopérabilité	Connexions conformes aux mécanismes TLS hybrides normalisés.
OBJ-03	Performance	Gains attribuables séparément à QSA_TLSH et à OPT_TLSH.
OBJ-04	Industrialisation	Installation, mise à jour, supervision et retour arrière automatisés.
OBJ-05	Valorisation	Dossier technique, situation juridique clarifiée et besoins acquéreurs documentés.

02 — Périmètre fonctionnel de la première version

2.1. Éléments inclus

La première version comprend un composant cryptographique C, une intégration Rust, un service réseau autonome, OPT_TLSH, un banc de tests et les outils d'exploitation.

Le service doit assurer la terminaison TLS côté serveur et fournir un client de validation. Il transmet ensuite les flux déchiffrés à une application autorisée, sans devenir un proxy ouvert.

Le périmètre initial est fixé à **TLS 1.3 sur TCP, sous Linux x86-64**. Ubuntu 24.04 LTS constitue l'environnement de qualification initial proposé ; le noyau, les compilateurs et les dépendances seront précisément figés.

2.2. Éléments exclus de la première version

Sont exclus : la réécriture complète de TLS, un protocole cryptographique propriétaire, QUIC/HTTP/3, l'accélération GPU, une plateforme SaaS multiclient et une refonte de WSA.

L'authentification par certificats intégralement post-quantiques est un chantier distinct. **Un établissement de clés hybride ne rend pas automatiquement toute la chaîne d'authentification post-quantique.** La terminologie commerciale devra refléter cette distinction. rfc-editor.org +1

Les implémentations WSA, IP Analyzer et Linux Analyzer existantes doivent rester inchangées pendant le développement.

Livrable : périmètre V1 signé, comprenant les fonctionnalités incluses, exclues et reportées.

03 — Référentiel normatif et niveaux de conformité

3.1. Références applicables

Le référentiel doit être enregistré avec ses versions, ses errata et sa date de gel.

Référence	Application au projet
RFC 9846	Référence actuelle de TLS 1.3 ; elle remplace notamment la RFC 8446.
RFC 9954	Cadre de construction des échanges de clés hybrides TLS 1.3 ; document informatif.
RFC 10024	Définition des groupes hybrides ECDHE/ML-KEM normalisés.
FIPS 203	Spécification de ML-KEM.
NIST SP 800-227	Recommandations d'utilisation des mécanismes d'encapsulation de clés.
CAVP et CMVP/FIPS 140-3	Validation des implémentations d'algorithmes et des modules cryptographiques.

Ces références correspondent aux publications officielles vérifiées pour ce document. rfc-editor.org +6

04 — Gouvernance, responsabilités et prérequis

4.1. Répartition proposée

Responsabilité	Responsable proposé
Fourniture et explication de QSA_TLSH	Christophe
Architecture générale et OPT_TLSH	Guillaume
Arbitrages de périmètre et budget	Direction du projet Lockheed
Revue cryptographique et sécurité C/Rust	Expert indépendant
Parcours de validation FIPS	Laboratoire accrédité sélectionné
Brevets, licences et propriété des contributions	Conseil en propriété industrielle et conseil juridique

05 — Architecture logicielle de référence

5.1. Principe retenu

L'architecture doit s'appuyer sur **rustls**, avec un fournisseur cryptographique dédié intégrant QSA_TLSH. Rustls permet cette extension au moyen de son interface `CryptoProvider`. La version stable retenue doit être figée et testée ; l'utilisation d'une branche de développement est exclue pour la livraison. Docs.rs

Composant de travail	Technologie	Responsabilité
`qsa_crypto`	C	Opérations cryptographiques de QSA_TLSH, dans leur périmètre validé.
`qsa_ffi`	Rust/C	Interface contrôlée entre les langages.
`qsa_provider`	Rust	Adaptation aux interfaces cryptographiques de rustls.
`qsa_tlsd`	Rust	Connexions, politiques TLS et transfert des flux.
`opt_tlsh`	Rust	Ordonnancement et optimisation des entrées/sorties.
`qsa_admin`	Rust	État, configuration contrôlée et métriques.
`qsa_tooling`	Python	Installation, diagnostic, tests et collecte des preuves.

06 — Spécifications du noyau cryptographique QSA_TLSH

6.1. Qualification initiale

La revue doit déterminer si QSA_TLSH est une implémentation d'algorithmes normalisés, une optimisation d'implémentation ou une construction cryptographique nouvelle.

Une construction nouvelle ne doit pas être intégrée au profil de production normalisé sans évaluation distincte. Elle reste dans une branche expérimentale identifiée.

La cible initiale est **ML-KEM-768**, conformément à FIPS 203. Les contrôles d'entrée et le comportement de décapsulation, notamment le rejet implicite prévu par l'algorithme, doivent être conservés. csrc.nist.gov +1

6.2. Exigences de sécurité

CRY-01 — Temps constant. Les chemins manipulant des secrets doivent être examinés pour les branchements, accès mémoire et durées dépendant de ces secrets. La vérification porte sur les binaires compilés des plateformes supportées.

CRY-02 — Aléa. La génération aléatoire doit utiliser une source cryptographiquement appropriée, avec gestion explicite des défaillances. Les sources d'entropie et les mécanismes de génération doivent figurer dans le dossier de validation.

CRY-03 — Cycle de vie. Les secrets temporaires doivent être détruits dès qu'ils ne sont plus nécessaires. Leur présence dans les journaux, les fichiers temporaires ou les exports de diagnostic est interdite.

CRY-04 — Isolation. Aucune réutilisation de secret éphémère entre connexions indépendantes. Les interfaces de tests déterministes doivent être exclues des builds de production.

07 — Interface C/Rust et moteur d'exécution

7.1. Contrat d'interface

L'interface binaire doit être versionnée et documenter chaque fonction : entrées, sorties, longueurs, alignement, propriété mémoire, durée de vie, erreurs et règles d'utilisation concurrente.

Aucun objet Rust interne ne doit traverser directement la frontière C. Les allocations doivent être libérées par le composant responsable de leur création.

L'utilisation de `unsafe` doit être limitée à des zones identifiées et relues. La sûreté mémoire de Rust ne dispense pas de contrôler les appels au code C et leurs préconditions. doc.rust-lang.org

7.2. Exécution et concurrence

FFI-01. Aucun dépassement de tampon, double libération ou utilisation après libération ne doit rester connu au passage en production.

FFI-02. Aucun déroulement de pile lié à une erreur ou à un `panic` ne doit franchir une frontière non prévue à cet effet.

RUN-01. Les traitements cryptographiques coûteux doivent être isolés des boucles d'entrées/sorties par une stratégie d'exécution bornée.

RUN-02. Chaque connexion possède son état propre. Les états globaux mutables doivent être justifiés et synchronisés.

RUN-03. Toute file d'attente doit posséder une capacité maximale, une politique de saturation et une métrique d'occupation.

08 — Politiques TLS, certificats et gestion des sessions

8.1. Profils cryptographiques

La V1 doit prendre en charge `X25519MLKEM768` pour le profil général et `SecP256r1MLKEM768` pour le profil à qualifier avec le laboratoire.

Ces groupes sont définis par la RFC 10024. Le second répond notamment aux cas exigeant que les deux composantes reposent sur des mécanismes approuvés FIPS ; cela ne constitue pas, à lui seul, une validation du produit. rfc-editor.org

Le service doit appliquer les identifiants, encodages, contrôles et combinaisons de secrets normalisés. Aucune réduction propriétaire des clés, des messages ou des contrôles n'est autorisée.

8.2. Politiques obligatoires

TLS-01. Le profil de production initial exige un échange hybride. Un pair incompatible provoque un refus explicite ; aucun repli silencieux vers un échange uniquement classique.

TLS-02. Les certificats doivent être contrôlés : chaîne de confiance, identité attendue, validité, usages et politique de révocation documentée.

TLS-03. SNI et ALPN doivent être associés à des routes autorisées. Une destination fournie arbitrairement par le client ne doit pas devenir une cible de connexion.

TLS-04. Le 0-RTT est désactivé en V1. TLS ne fournit pas de protection intrinsèque complète contre le rejeu de ces données. rfc-editor.org

TLS-05. La reprise de session ne peut être activée qu'après validation. Elle doit conserver un nouvel échange hybride et être liée au profil de sécurité, à l'identité et au contexte applicatif.

09 — Spécifications fonctionnelles d'OPT_TLSH

9.1. Mission

OPT_TLSH doit optimiser **le traitement local des connexions**, sans modifier la sémantique du protocole TLS.

L'expression « superoptimisation réseau » désigne ici un programme de recherche appliquée. Elle ne constitue ni une garantie de résultat ni une preuve d'invention.

Fonction	Exigence
Admission des connexions	Réguler les nouvelles négociations selon les capacités disponibles.
Ordonnancement	Répartir équitablement les traitements et prévenir la monopolisation.
Gestion des buffers	Réduire allocations et copies, dans des limites mémoire explicites.
Regroupement des écritures	Diminuer le coût des entrées/sorties sans attente excessive.
Adaptation	Ajuster les paramètres selon charge, files et mesures réseau publiques.
Repli opérationnel	Désactiver l'optimisation en conservant exactement la politique TLS.

10 — Objectifs de performance et protocole de mesure

10.1. Comparaisons obligatoires

Les essais doivent distinguer trois configurations sur une plateforme identique :

Configuration	Finalité
B0 — Référence	Rustls avec fournisseur cryptographique standard, sans QSA_TLSH ni OPT_TLSH.
B1 — QSA	Même service, avec QSA_TLSH, optimisation désactivée.
B2 — QSA + OPT	Même service et même cryptographie, optimisation activée.

B1/B0 mesure l'effet de QSA_TLSH. B2/B1 mesure celui d'OPT_TLSH.

Les groupes cryptographiques, certificats, paramètres de sécurité, ressources et charges doivent être identiques entre configurations comparées.

10.2. Matrice d'essais

Les scénarios comprennent des RTT de **1, 20 et 100 ms**, des pertes de **0 %, 0,1 % et 1 %**, des MTU de **1 280 et 1 500 octets**, ainsi que des charges courtes et des transferts longs.

La matrice minimale doit être fixée avant les mesures. Chaque point retenu comporte au moins **cinq répétitions**, avec échauffement et ordre d'exécution alterné.

Les métriques comprennent : négociations réussies par seconde, latences p50/p95/p99, CPU par négociation, mémoire résidente, mémoire par connexion, erreurs et expirations.

11 — Modèle de menace et durcissement

11.1. Menaces à couvrir

L'analyse doit traiter les pairs malveillants, messages invalides, saturations, tentatives de rétrogradation, erreurs de certificat, fuites temporelles, défauts mémoire, compromission de dépendances et abus de l'administration.

La collecte de trafic en vue d'un déchiffrement ultérieur fait partie des scénarios étudiés. En revanche, aucune promesse de protection totale ne doit être faite face à un système d'exploitation déjà compromis.

11.2. Mesures obligatoires

SEC-01. Exécution sous un compte dédié non privilégié ; permissions minimales sur les clés, fichiers et sockets.

SEC-02. Limites configurables pour les connexions, négociations simultanées, données reçues avant authentification et durées d'inactivité.

SEC-03. Plafond mémoire global, admission contrôlée et rejet propre à saturation.

SEC-04. Désactivation des exports de secrets et des dumps mémoire en production.

SEC-05. Échec sécurisé en cas d'erreur cryptographique, d'échec d'auto-test ou de configuration incohérente.

SEC-06. Interdiction d'un mode de secours qui désactiverait silencieusement l'hybridation ou l'authentification.

12 — Exploitation, automatisation et supervision

12.1. Installation et mises à jour

Toutes les opérations doivent être orchestrées par un outil Python intégré au projet. L'exploitant ne doit pas assembler manuellement les services, répertoires, permissions ou commandes système.

L'interface doit proposer principalement `check`, pour diagnostiquer sans modification, et `repair`, pour appliquer une correction contrôlée.

L'outil doit vérifier les dépendances, installer les versions prévues, gérer le service systemd, préserver les configurations, contrôler l'intégrité des paquets et produire un journal exploitable.

Une mise à jour doit être atomique, conserver une version de retour arrière et effectuer un contrôle de santé avant activation. Aucun téléchargement non authentifié ne doit être exécuté.

12.2. Administration et observabilité

L'administration V1 doit être locale, sur une interface protégée, sans exposition Internet par défaut.

Les métriques doivent couvrir groupes négociés, connexions, files, performances, erreurs, version et état d'OPT_TLSH. Les journaux doivent être structurés, expurgés de secrets et soumis à une rétention définie.

L'intégration WSA doit rester **facultative et en lecture seule pour la V1**. Une indisponibilité de WSA ne doit pas affecter le service TLS.

Les dépendances doivent être accompagnées d'une nomenclature logicielle — SBOM —, de leurs licences et de leur provenance.

Recette : installation sur machine vierge, réexécution idempotente, mise à jour et retour arrière testés.

13 — Stratégie d’assurance qualité et de validation

13.1. Couverture des tests

Domaine	Preuve attendue
Cryptographie	Vecteurs de référence, contrôles d’entrée et comparaison différentielle.
Interface C/Rust	Tests mémoire, concurrence, annulation et destruction des objets.
Protocole	Négociation, refus, reprise, changement de clés et fermeture.
Interopérabilité	Échanges avec OpenSSL et rustls amont, sans le fournisseur QSA.
Robustesse	Fuzzing ciblant les nouvelles interfaces et entrées non fiables.
Sécurité	Tests négatifs, analyse temporelle et audit externe.
Exploitation	Installation, rotation de certificats, reprise après incident et rollback.

OpenSSL fournit des groupes hybrides ECDHE/ML-KEM utilisables pour construire une contrepartie d’interopérabilité. Les versions exactes et leurs paramètres doivent être consignés. OS docs.openssl.org


Analyse et plan de réalisation
Analyse
Objectifs
Architecture
Plan de réalisation
Mise en œuvre
Validation

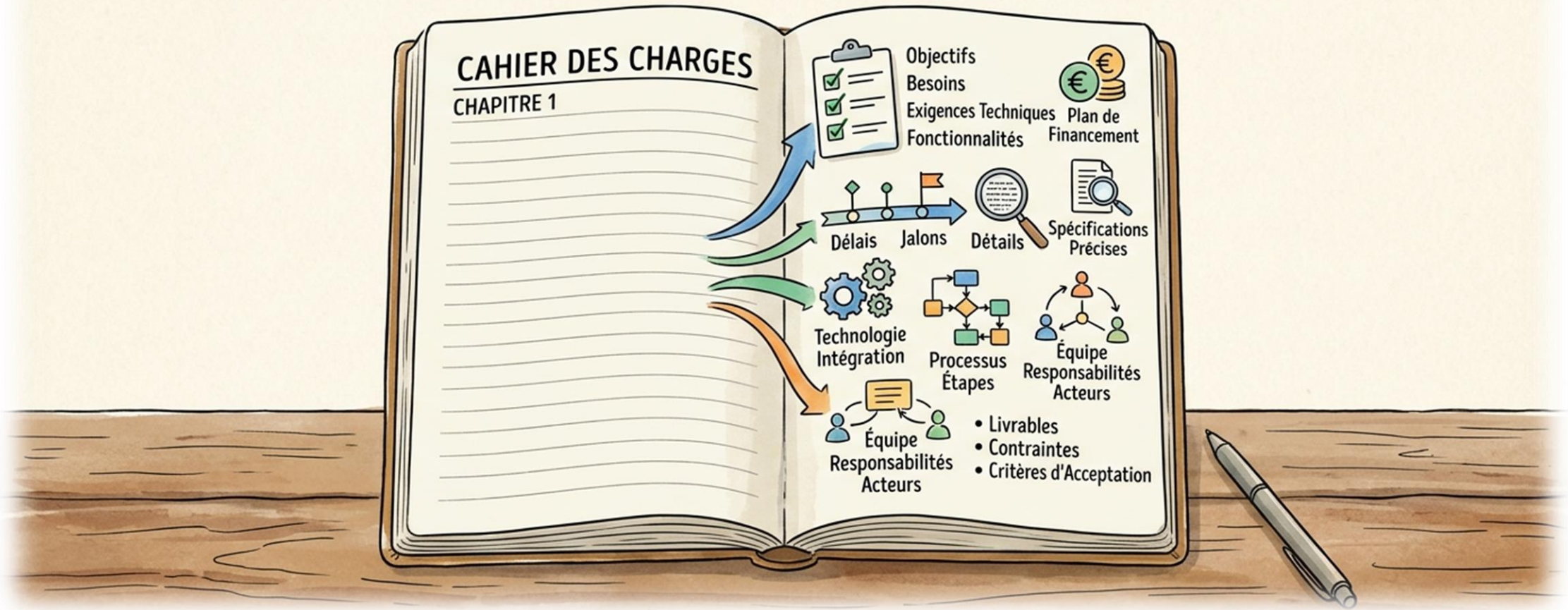
AGENDA — Analyse et Plan de réalisation

- Chapitre 1 — Ce que contient réellement le cahier des charges
- Chapitre 2 — Comprendre TLS à partir de ton expérience des serveurs
- Chapitre 3 — Comprendre le déroulement d'une connexion TLS
- Chapitre 4 — Comprendre le post-quantique et le mot « hybride »
- Chapitre 5 — Finalités, utilisateurs et contextes d'utilisation
- Chapitre 6 — Périmètre fonctionnel recommandé pour la V1
- Chapitre 7 — Architecture du produit et emplacement dans Linux
- Chapitre 8 — Politique adaptative et fonctionnement IoT attendu
- Chapitre 9 — Sessions, 0-RTT et preuves ZKP
- Chapitre 10 — Langages, composants et mode de développement
- Chapitre 11 — Moyens logistiques, installation et exploitation

AGENDA — Analyse et Plan de réalisation – Part 2

- Chapitre 12 — OPT_TLSH et démonstration des performances
- Chapitre 13 — Sécurité, robustesse et stratégie de validation
- Chapitre 14 — Normes, validation, responsabilités et valeur du produit
- Chapitre 15 — Plan de développement et progression pédagogique

Chapitre 1 — Ce que contient réellement le cahier des charges



1.1. Trois ensembles à réconcilier

Le document rassemble trois niveaux de conception :

Partie du document	Orientation	Utilisation proposée
Présentation et analyse fonctionnelle, jusqu'à la p. 189	TLS hybride adaptatif pour l'IoT : batterie, réseau, niveaux de sécurité, ZKP et économies d'énergie	Vision fonctionnelle et objectifs à moyen terme
Plan de développement, p. 190–198	Prototype principalement Python, avec bibliothèques cryptographiques natives	Simulation des politiques et outillage
Cadrage technique, p. 201–214	Service Linux C/Rust, rustls, QSA_TLSH, OPT_TLSH, installation automatisée	**Socle recommandé pour la V1**

La dernière partie, datée du 11 novembre 2026, précise les responsabilités des composants et fixe des limites exploitables. Elle impose notamment un échange hybride, désactive le 0-RTT en V1 et interdit les modifications propriétaires des mécanismes cryptographiques normalisés.

1.2. Les principales contradictions

Plusieurs points doivent être harmonisés dans la prochaine version du cahier des charges :

- Les premières pages prévoient parfois un retour au chiffrement classique ; la p. 209 exige l'hybridation.
- Le plan Python envisage un cœur logique Python en production ; la p. 206 place le service et les traitements actifs dans Rust.
- Les premières sections promettent un 0-RTT sécurisé par ZKP ; la p. 209 le désactive.
- La p. 73 demande une latence inférieure ou égale au TLS classique, tout en acceptant une augmentation inférieure à 20 %.
- Les économies d'énergie de 30 à 60 % sont des **objectifs**, sans résultats expérimentaux fournis.

Ces divergences se résolvent en définissant explicitement **ce qui relève de la V1, de l'évolution IoT et de la recherche**.

1.3. Deux chapitres manquent

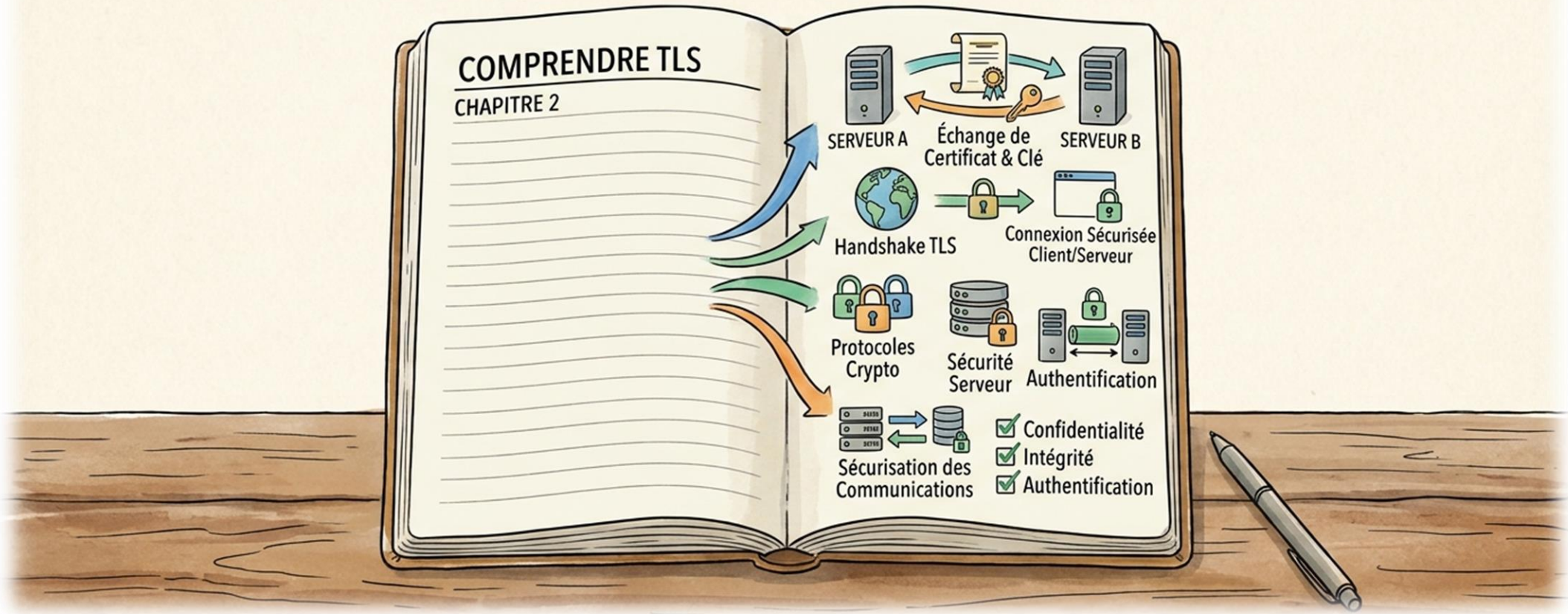
L'agenda technique de la p. 200 annonce quinze chapitres, mais le contenu s'arrête au chapitre 13, p. 214. La dernière page est blanche.

Il manque donc :

- le parcours de validation FIPS et le dossier de conformité ;
- la propriété intellectuelle, la valorisation et les jalons.

Enfin, le document indique lui-même que **le code QSA_TLSH et ses preuves de validation n'ont pas encore été examinés**. Mon analyse porte donc sur l'architecture proposée, pas sur la sécurité effective de ce composant.

Chapitre 2 — Comprendre TLS à partir de nos expériences des serveurs



Chapitre 2 — Comprendre TLS à partir de ton expérience des serveurs

2.1. À quoi sert TLS ?

TLS fournit un canal de communication protégé entre deux applications. Ses trois fonctions principales sont :

Fonction	Signification concrète
Confidentialité	Un observateur du réseau ne doit pas pouvoir lire les données transportées.
Intégrité	Une modification des données pendant leur transport doit être détectée.
Authentification	Le client vérifie l'identité du serveur ; le serveur peut aussi vérifier celle du client.

TLS comporte une phase de négociation, le **handshake**, puis une phase de transport des données protégées. La référence actuelle de TLS 1.3 est la RFC 9846, publiée en juillet 2026 en remplacement de la RFC 8446.

datatracker.ietf.org

2.2. Où se situe TLS ?

Dans le périmètre retenu pour TLSH :

- TCP transporte un flux d'octets fiable ;
- TLS protège ce flux ;
- **HTTP, MQTT ou un protocole métier** donnent un sens aux données échangées.

HTTPS correspond ainsi à l'utilisation d'HTTP à travers TLS.

Dans ton environnement Nginx, tu connais déjà deux éléments essentiels : le certificat et sa clé privée. Le certificat peut être communiqué aux clients ; **la clé privée reste sur le serveur**, avec des permissions restrictives. Ce sont deux objets différents, même lorsqu'ils sont stockés ensemble. nginx.org

2.3. Trois catégories de clés à distinguer

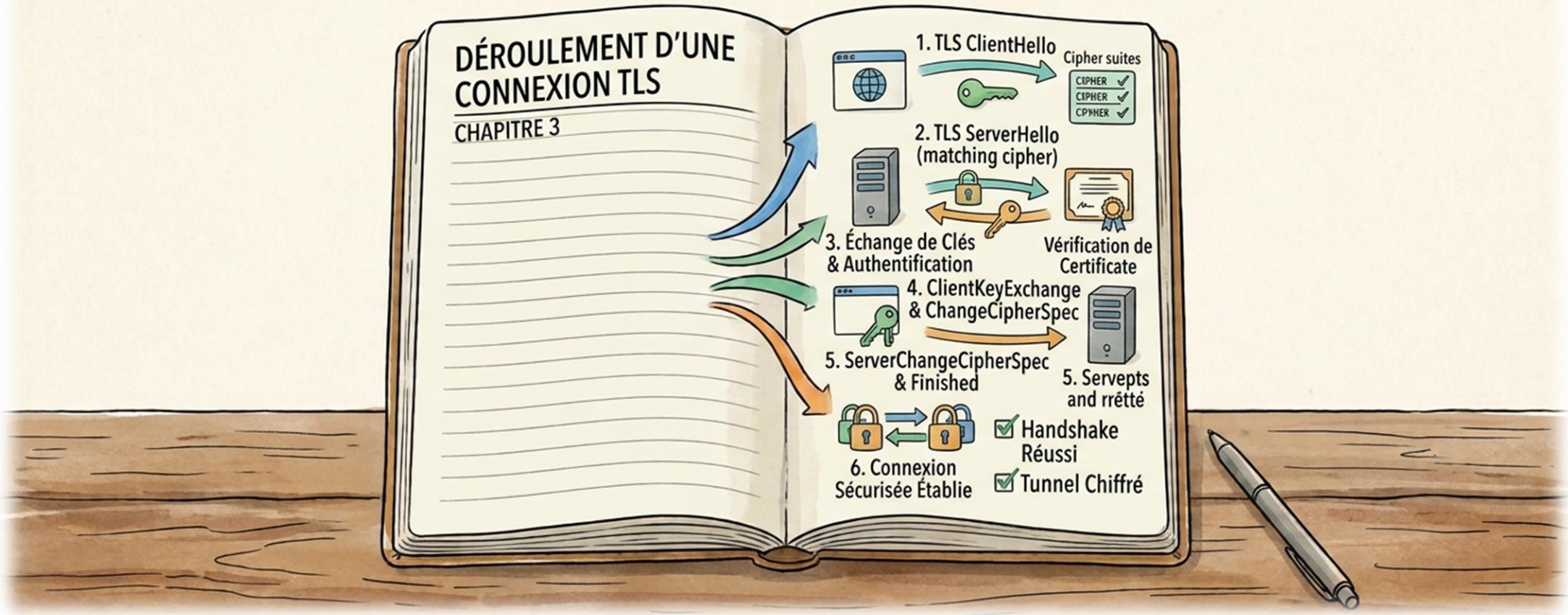
Pour comprendre TLSH, garde cette distinction :

1. **La clé d'identité** permet au serveur de prouver qui il est.
2. **Les éléments d'établissement de clés** permettent de construire les secrets d'une connexion.
3. **Les clés de trafic** protègent ensuite les données échangées.

Une erreur fréquente consiste à imaginer que le certificat chiffre directement toutes les pages web. Il participe surtout à l'authentification ; les données utilisent des clés propres à la connexion.

Enfin, TLS protège la communication entre ses deux extrémités. Le logiciel qui reçoit les données déchiffrées doit toujours assurer ses contrôles métier, ses autorisations et sa sécurité applicative.

Chapitre 3 — Comprendre le déroulement d'une connexion TLS



Chapitre 3 — Comprendre le déroulement d'une connexion TLS

3.1. Le handshake expliqué simplement

Prenons un client qui souhaite envoyer une mesure de température à un serveur.

Le tableau suivant décrit un handshake TLS 1.3 complet, simplifié, avec authentification du serveur par certificat.

Étape	Échange	Rôle
1	`ClientHello`	Le client annonce ses capacités et fournit ses éléments initiaux d'établissement de clés.
2	`ServerHello`	Le serveur choisit des paramètres compatibles et fournit sa contribution.
3	Paramètres complémentaires et certificat	Le serveur précise la connexion et présente son identité.
4	`CertificateVerify`	Il prouve qu'il possède la clé privée correspondant à cette identité.
5	Messages `Finished`	Les deux côtés vérifient la cohérence cryptographique de la négociation.
6	Données applicatives	La mesure de température circule dans le canal protégé.

Chapitre 3 — Comprendre le déroulement d'une connexion TLS

3.1 Le handshake expliqué simplement

Exemple : un client veut envoyer une mesure de température à un serveur.
Voici un handshake TLS 1.3 complet, simplifié, avec authentification du serveur par certificat.

CLIENT
(capteur / application)



- Initie la connexion
- Négocie les paramètres
- Vérifie le certificat
- Établit un canal chiffré
- Envoie les données

DÉROULEMENT DU HANDSHAKE TLS 1.3

1	ClientHello	→	Le client annonce ses capacités et fournit ses éléments initiaux d'établissement de clés.
2	ServerHello	←	Le serveur choisit des paramètres compatibles et fournit sa contribution.
3	Paramètres + Certificat	←	Le serveur précise la connexion et présente son identité (certificat).
4	CertificateVerify	←	Il prouve qu'il possède la clé privée correspondant à cette identité.
5	Messages Finished	⇄	Les deux côtés vérifient la cohérence cryptographique de la négociation.
6	Données applicatives	→	La mesure de température circule dans le canal protégé (chiffré).

SERVEUR
(reçoit et traite la mesure)



- Négocie et valide les paramètres
- Présente son certificat
- Prouve son identité
- Établit le canal chiffré
- Reçoit la mesure sécurisée

RÉSULTAT

Un **canal TLS sécurisé et authentifié** est établi.

La mesure de température est transmise de manière confidentielle et intègre.

- **CONFIDENTIALITÉ** (données chiffrées)
- **INTÉGRITÉ** (non altérées)
- **AUTHENTICITÉ** (serveur vérifié par certificat)

3.2. Ce que TLSH ajoutera

Pour notre produit, le travail consiste à :

- sélectionner les groupes autorisés ;
- intégrer QSA aux opérations cryptographiques concernées ;
- appliquer les politiques de connexion ;
- gérer les ressources ;
- transmettre les données à la bonne application ;
- observer les résultats.

La bibliothèque rustls doit conserver la responsabilité du déroulement TLS.

Cela évite que nous écrivions nous-mêmes la succession des messages, la dérivation des clés ou les vérifications du protocole.

3.3. Une correction nécessaire dans le document

Le schéma de la p. 18 contient `ClientKeyExchange`, un message qui ne correspond pas au déroulement TLS 1.3 présenté ici. Il faut donc le remplacer dans les futurs schémas techniques.

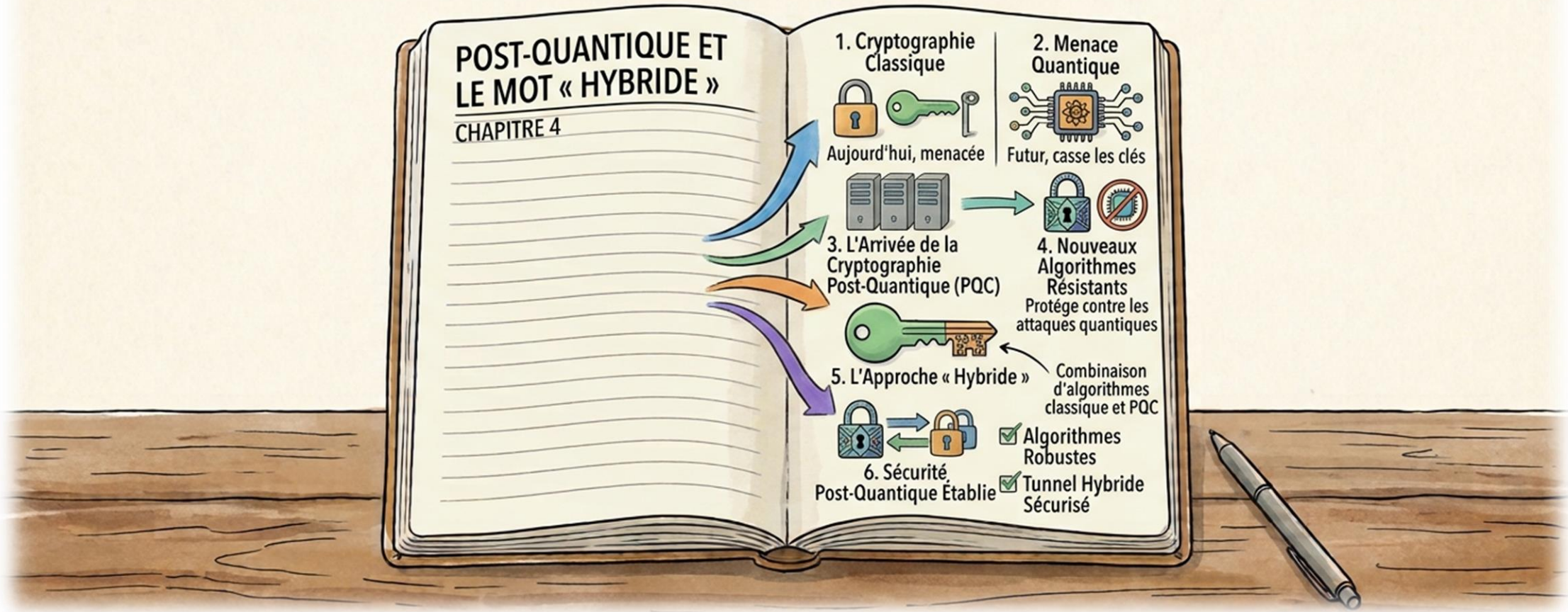
Cette correction est importante : un dessin pédagogique peut simplifier, mais le diagramme utilisé pour développer doit correspondre exactement au protocole retenu.

3.4. Connexion et session

Nous utiliserons ces mots avec précision :

- **Connexion** : communication TLS actuellement ouverte.
- **Reprise de session** : mécanisme permettant de faciliter une nouvelle connexion à partir d'un contexte antérieur.
- **Connexion persistante** : connexion conservée pour plusieurs échanges applicatifs.

Chapitre 4 — Comprendre le post-quantique et le mot « hybride »



Chapitre 4 — Comprendre le post-quantique et le mot « hybride »

4.1. Le problème visé

Le projet traite notamment le risque de **collecter aujourd'hui du trafic chiffré pour tenter de le déchiffrer ultérieurement**, lorsqu'une capacité cryptanalytique suffisante sera disponible.

Cette menace est particulièrement pertinente lorsque les informations doivent rester confidentielles pendant longtemps. L'hybridation vise à introduire une protection post-quantique tout en conservant une composante classique éprouvée. rfc-editor.org

4.2. Que fait ML-KEM ?

ML-KEM est un mécanisme d'encapsulation de clés, normalisé dans FIPS 203.

Son rôle consiste à permettre à deux parties d'obtenir un secret partagé à travers un réseau public. Dans une représentation simplifiée :

1. une partie produit une clé publique d'encapsulation et conserve la clé privée correspondante ;
2. l'autre utilise cette clé publique pour produire un secret et un message d'encapsulation ;
3. la première retrouve le même secret grâce à sa clé privée et au message reçu.

Ce secret sert ensuite à construire des clés utilisables pour la communication. **ML-KEM n'est pas l'algorithme qui chiffre directement tous les fichiers ou toutes les requêtes.** CSRC

4.3. Que signifie « hybride » dans TLSH ?

Le groupe `X25519MLKEM768` associe :

- une contribution classique, X25519 ;
- une contribution post-quantique, ML-KEM-768.

La construction normalisée combine leurs résultats. La propriété recherchée est de conserver la protection de l'établissement de clés si au moins une composante reste sûre, sous les hypothèses prévues et avec une implémentation correcte. La RFC 10024 définit ce groupe ainsi que `SecP256r1MLKEM768` et `SecP384r1MLKEM1024`. rfc-editor.org

L'expression « hybride » désigne donc ici **la combinaison classique/post-quantique de l'établissement de clés.** Elle ne désigne pas l'utilisation conjointe de C et Rust.

4.4. Établissement de clés et signatures

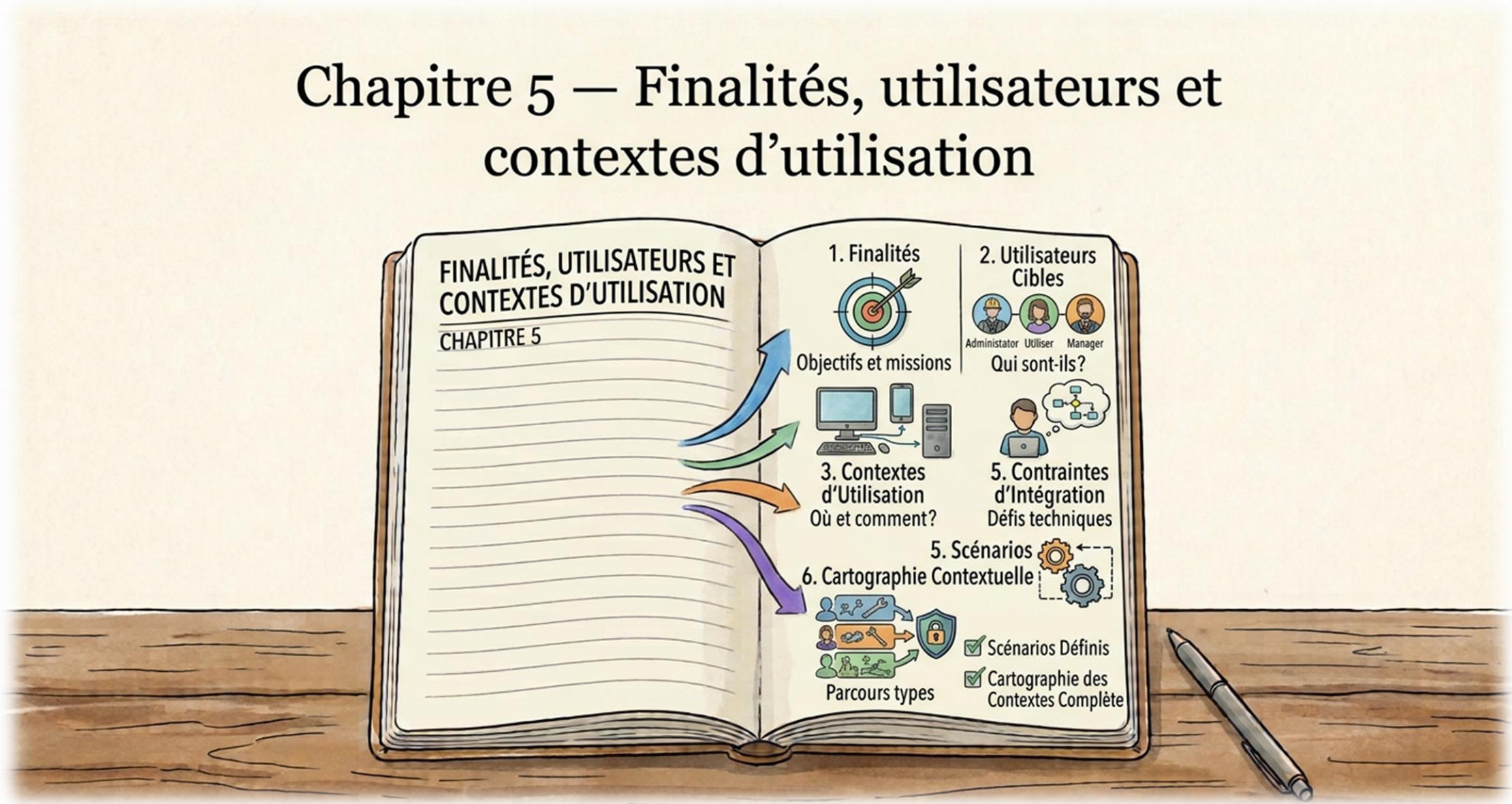
Il faut également séparer :

Élément	Fonction
ML-KEM	Établir un secret partagé
ML-DSA	Produire et vérifier des signatures numériques
SLH-DSA	Produire et vérifier des signatures selon une autre construction

ML-DSA et SLH-DSA sont respectivement définis par FIPS 204 et FIPS 205. CSRC +1

Un échange de clés hybride peut conserver une authentification par certificats classiques. Il ne faut donc pas présenter automatiquement toute la connexion comme disposant d'une **authentification entièrement post-quantique**. Cette distinction est précisément celle que retient le cadrage final du PDF. rfc-editor.org

Chapitre 5 — Finalités, utilisateurs et contextes d’utilisation



Chapitre 5 — Finalités, utilisateurs et contextes d'utilisation

5.1. La finalité du produit

Je reformulerais l'objectif de TLSH ainsi :

> **Fournir un service de communication TLS hybride, intégrable dans une infrastructure existante, dont la sécurité, les performances et l'exploitation peuvent être vérifiées.**

Cela donne quatre objectifs opérationnels :

- établir des connexions hybrides interopérables ;
- intégrer et qualifier QSA_TLSH ;
- mesurer les apports d'OPT_TLSH ;
- automatiser l'installation et l'exploitation.

5.2. Les acteurs

Le document identifie le client IoT, le serveur ou la passerelle, QSA, l'opérateur et le laboratoire. Il faut y ajouter explicitement **l'application métier**.

Cette application détermine notamment ce qu'une requête signifie : lecture d'une température, modification d'une consigne, déclenchement d'une action. TLS ne peut pas décider seul si répéter cette opération est acceptable.

5.3. Les usages à aborder progressivement

Usage	Intérêt pour TLSH	Position proposée
Client logiciel vers application Linux	Vérifier la chaîne complète et l'interopérabilité	Premier démonstrateur
Passerelle industrielle vers serveur central	Sécuriser une liaison entre équipements maîtrisés	Premier pilote
Capteur contraint vers serveur	Étudier batterie, radio, mémoire et adaptations	Extension IoT
Commande d'équipement critique	Ajouter contraintes métier, disponibilité et validation sectorielle	Après qualification

Pour démarrer, je recommande **une télémesure simulée envoyée par HTTP/1.1**. Cela donne une démonstration intelligible sans introduire immédiatement les particularités d'un firmware ou d'une radio.

5.4. Définir exactement la liaison protégée

Si une passerelle établit TLS à la place d'un capteur, la protection concerne la liaison **passerelle–serveur**. Le segment **capteur–passerelle** nécessite sa propre analyse.

Cette frontière devra figurer dans chaque cas d'usage. Sans elle, une formule comme « capteur protégé par TLSH » resterait trop imprécise.

5.3. Les usages à aborder progressivement



Cas d'usage TLSH

Une montée en puissance progressive, du démonstrateur à l'usage critique.

USAGE	INTÉRÊT POUR TLSH	POSITION PROPOSÉE
Client logiciel vers application Linux	Vérifier la chaîne complète et l'interopérabilité	Premier démonstrateur
Passerelle industrielle vers serveur central	Sécuriser une liaison entre équipements maîtrisés	Premier pilote
Capteur contraint vers serveur	Étudier batterie, radio, mémoire et adaptations	Extension IoT
Commande d'équipement critique	Ajouter contraintes métier, disponibilité et validation sectorielle	Après qualification

Pour démarrer : une télémesure simulée envoyée par HTTP/1.1.
Cela donne une démonstration intelligible sans introduire immédiatement les particularités d'un firmware ou d'une radio.

5.4. Définir exactement la liaison protégée

Si une passerelle établit TLS à la place d'un capteur, la protection concerne la **liaison passerelle–serveur**.

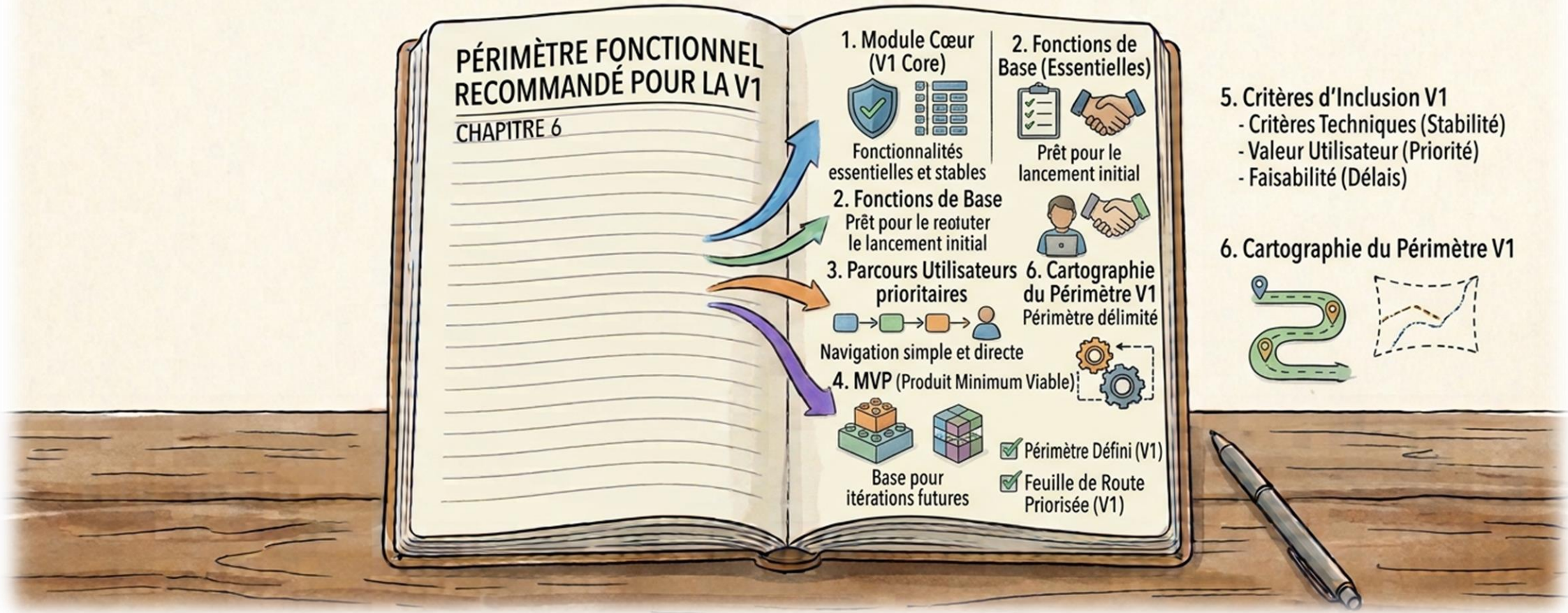
Le segment **capteur–passerelle** nécessite sa propre analyse.



À RETENIR

Cette frontière (**capteur–passerelle–serveur**) devra figurer dans chaque cas d'usage.
Sans elle, une formule comme « **capteur protégé par TLSH** » resterait trop imprécise.

Chapitre 6 — Périmètre fonctionnel recommandé pour la V1



Chapitre 6 — Périmètre fonctionnel recommandé pour la V1

Le cadrage final fournit déjà une base précise : **TLS 1.3 sur TCP, Linux x86-64, avec Ubuntu 24.04 LTS comme environnement initial de qualification proposé.** Je retiens ce périmètre.

6.1. Fonctions à livrer

La V1 doit permettre de :

1. démarrer un service TLSH avec une configuration validée ;
2. charger les certificats et les fournisseurs cryptographiques ;
3. accepter une connexion respectant le profil hybride exigé ;
4. refuser explicitement un pair incompatible ;
5. transférer les données vers une application autorisée ;
6. limiter les connexions, les files et la mémoire ;
7. produire des métriques et des journaux exploitables ;
8. fonctionner avec OPT activé ou désactivé ;
9. être installée, mise à jour et restaurée automatiquement.

Le **client de validation** fait partie du livrable : il doit vérifier le serveur et exposer les informations nécessaires au diagnostic.

6.2. Profil cryptographique initial

Le document prévoit :

- `X25519MLKEM768` pour le profil général ;
- `SecP256r1MLKEM768` pour un profil à qualifier avec le laboratoire.

La première démonstration peut se concentrer sur le premier. Le second constitue ensuite un jalon explicite de qualification.

Un client sans groupe hybride compatible doit recevoir un refus identifiable. Une simple préférence pour le post-quantique ne suffirait pas à satisfaire cette exigence.

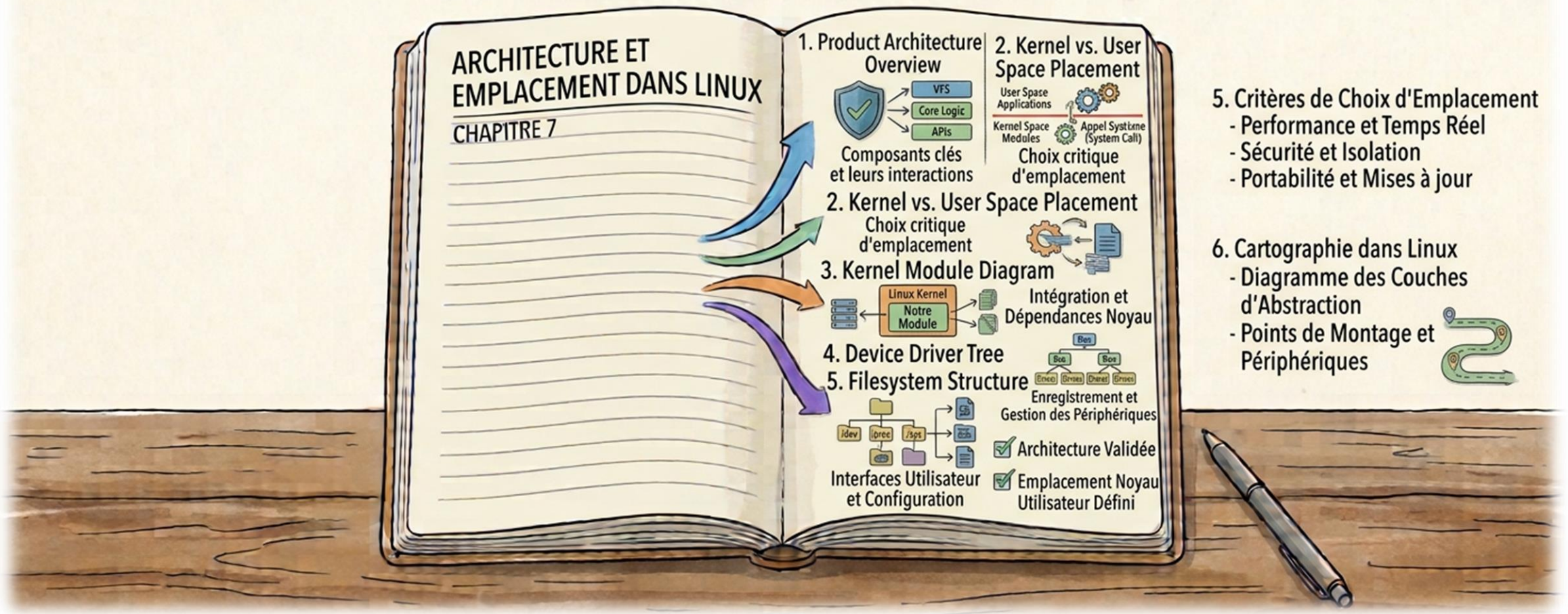
6.3. Fonctions reportées

Pour conserver une V1 maîtrisable, je maintiendrais hors de cette livraison :

- le 0-RTT et le mécanisme ZKP ;
- les extensions personnalisées de batterie dans le handshake ;
- les certificats entièrement post-quantiques ;
- QUIC/HTTP/3 ;
- le portage sur microcontrôleurs ;
- la compression des données métier ;
- l'accélération GPU.

L'adaptation IoT reste un objectif du programme. Elle dépend toutefois d'un service de référence fonctionnel et d'une plateforme cliente représentative.

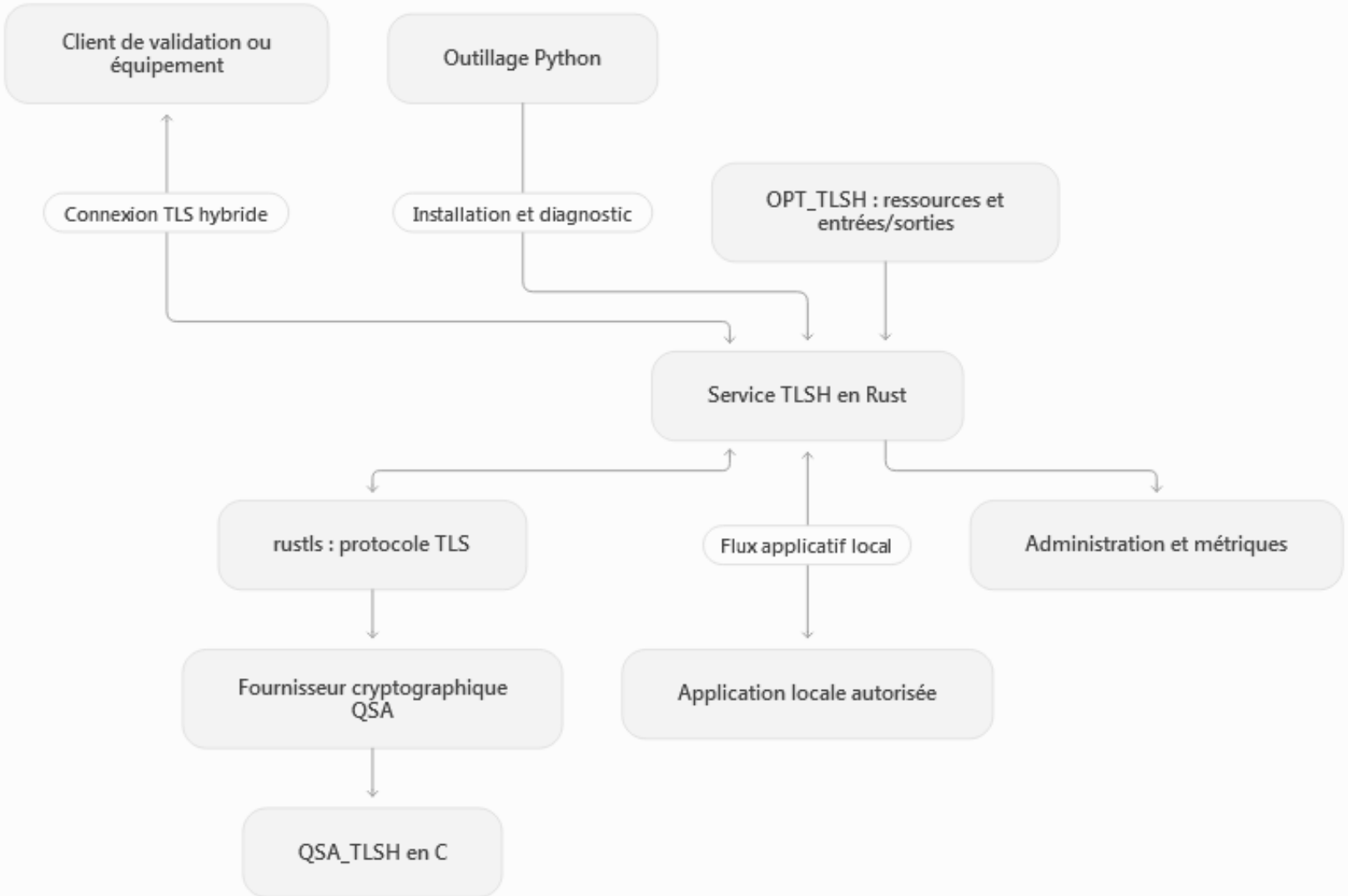
Chapitre 7 — Architecture du produit et emplacement dans Linux



5. Critères de Choix d'Emplacement
   - Performance et Temps Réel
   - Sécurité et Isolation
   - Portabilité et Mises à jour

6. Cartographie dans Linux
   - Diagramme des Couches d'Abstraction
   - Points de Montage et Périphériques

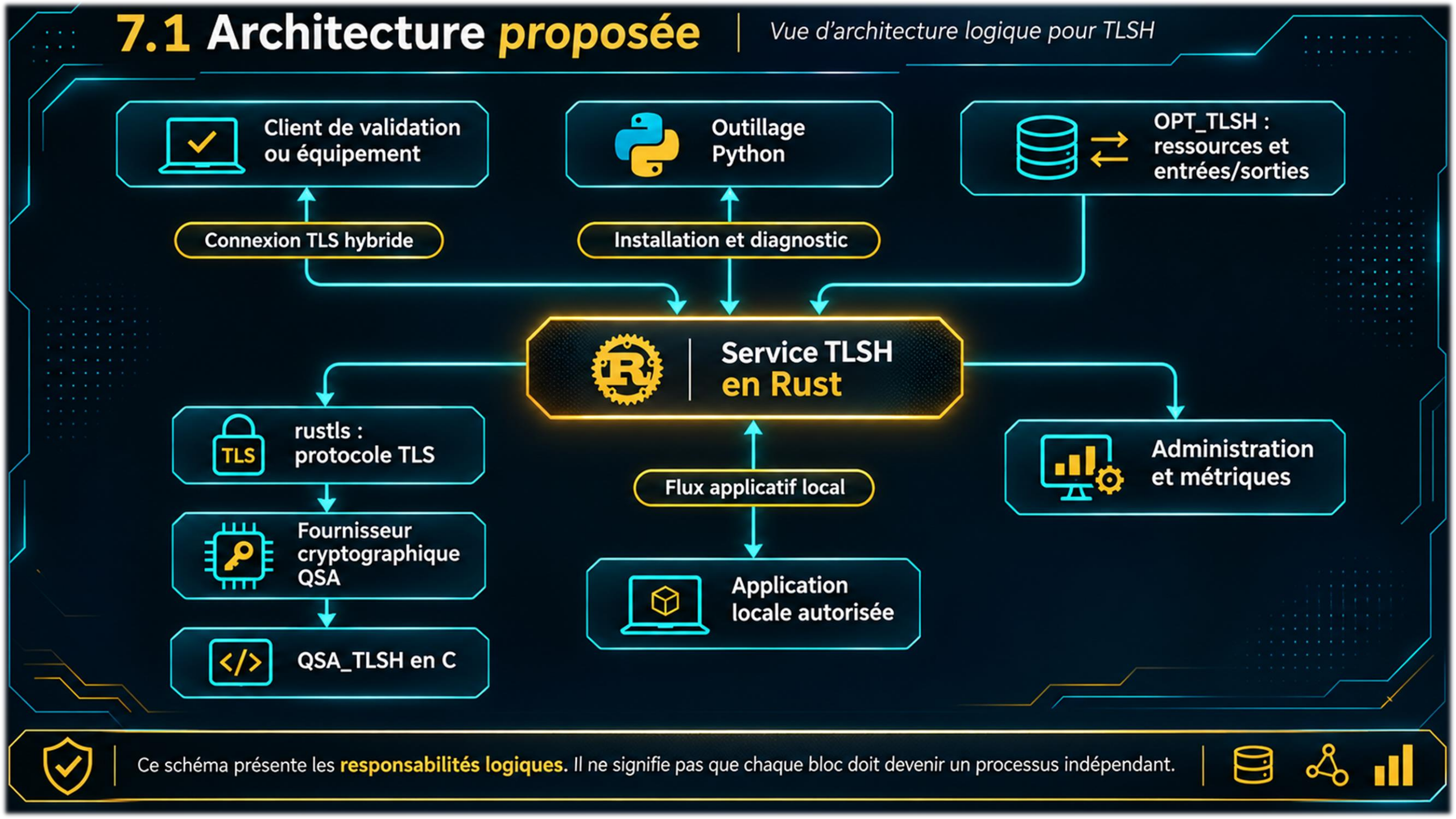
7.1. Architecture proposée



Ce schéma présente les responsabilités logiques. Il ne signifie pas que chaque bloc doit devenir un processus indépendant

7.1 Architecture proposée

Vue d'architecture logique pour TLSH



Ce schéma présente les **responsabilités logiques.** Il ne signifie pas que chaque bloc doit devenir un processus indépendant.

7.2. Le rôle de rustls

Rustls prend en charge le protocole TLS et accepte des fournisseurs cryptographiques personnalisés à travers `CryptoProvider`. Il est possible de remplacer les opérations nécessaires tout en déléguant les autres à un fournisseur existant. Rust

C'est une propriété importante : si QSA ne fournit que ML-KEM, nous n'avons pas à lui attribuer artificiellement la responsabilité d'AES, des signatures, des certificats et de toutes les autres opérations.

Le service doit aussi gérer les sockets et les entrées/sorties : rustls ne réalise pas lui-même toute cette orchestration réseau. Rust

7.3. Ce que signifie « terminaison TLS »

TLSH reçoit le flux chiffré et rend les données applicatives disponibles après traitement TLS.

Pour le premier démonstrateur, je recommande une application située **sur la même machine**, accessible uniquement par une liaison locale autorisée. Cela rend la frontière de confiance claire.

Si l'application est ensuite déplacée vers une autre machine, la protection de cette seconde liaison devra être spécifiée séparément.

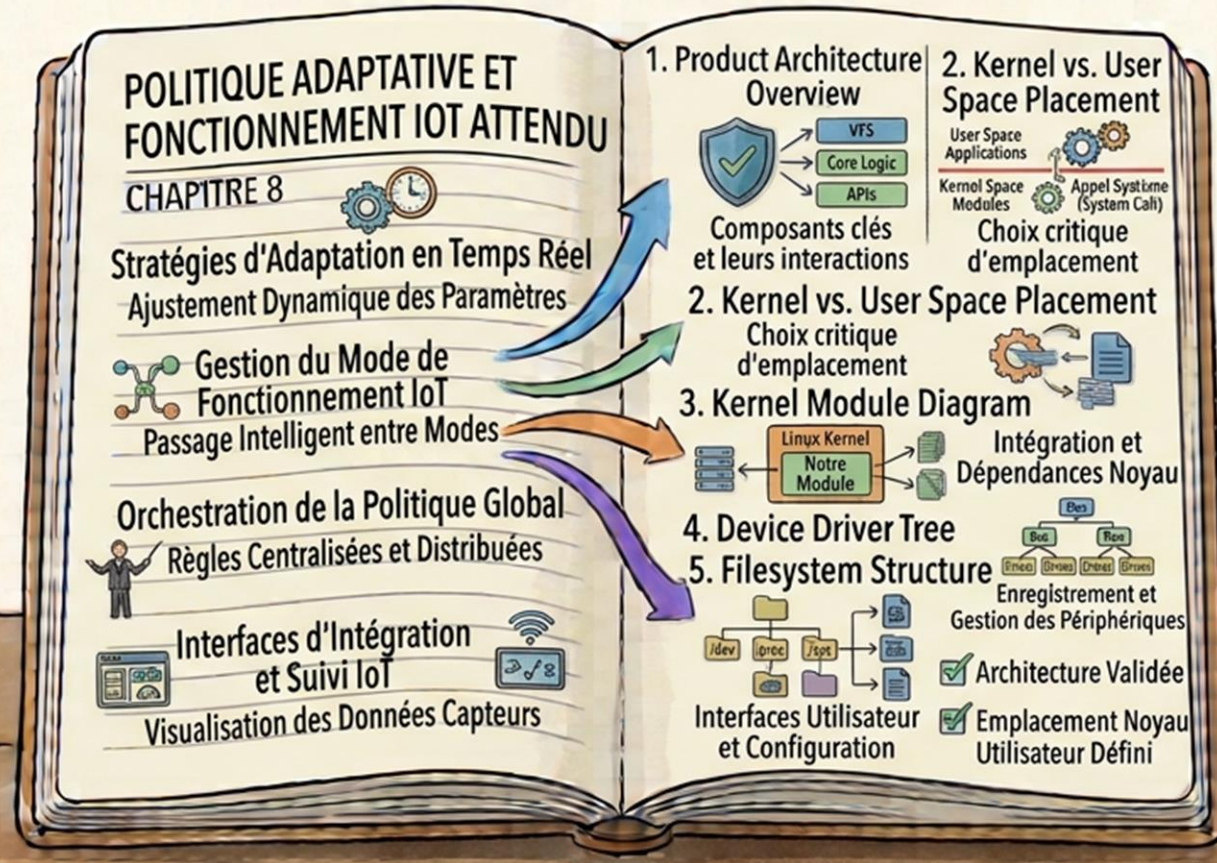
7.4. Conséquence pour Nginx

Il faudra choisir quel composant termine la connexion du client.

Placer TLSH après un Nginx qui a déjà déchiffré la connexion ne rend pas rétroactivement cette première liaison post-quantique.

Le laboratoire utilisera donc un point d'écoute dédié à TLSH. L'intégration aux infrastructures existantes sera traitée une fois cette chaîne validée.

Chapitre 8 — Politique adaptative et Fonctionnement IOT attendu



5. Critères de Performance de la Politique
   - Latence et Temps de Réponse
   - Robustesse et Fiabilité
   - Optimisation de la Consommation d'Énergie
   - Scalabilité pour Multiples Périphériques

6. Fonctionnalités IoT Attendues
   - Télémétrie et Collecte de Données
   - Contrôle à Distance et Mises à Jour OTA
   - Analyse de Données sur les Bords (Edge Computing)
   - Sécurité et Authentification des Appareils

8.1. Le principe à conserver

L'idée centrale du document est de tenir compte :

- des capacités du client ;
- de son état énergétique ;
- de la qualité du réseau ;
- des exigences de l'opérateur.

Cette adaptation doit rechercher **la solution la moins coûteuse parmi les solutions déjà autorisées.**

Une batterie faible ne doit jamais autoriser une protection inférieure au minimum exigé.

8.2. Distinguer exigence et préférence

Une politique doit comporter deux catégories de paramètres.

Les contraintes obligatoires définissent les groupes autorisés, l'authentification requise, les destinations permises et les limites de sécurité.

Les préférences d'optimisation peuvent agir sur la durée des connexions, les ressources allouées, les écritures réseau ou, ultérieurement, la préférence entre plusieurs profils acceptables.

Cette séparation permettra d'expliquer chaque décision : « profil sélectionné », « raison », « contraintes respectées ».

8.3. Exemple concret

Un capteur annonce une batterie faible.

Le système pourrait proposer, si le contrat métier le permet :

- de conserver une connexion déjà ouverte ;
- d'éviter des reconnexions inutiles ;
- de regrouper certaines transmissions dans une attente bornée ;
- de réduire une fréquence de télémesure non critique.

La fréquence d'émission relève cependant de l'application. TLSH ne doit pas modifier seul le comportement métier d'un équipement.

Si aucune connexion ne satisfait les contraintes obligatoires, le résultat attendu est un refus explicite avec diagnostic.

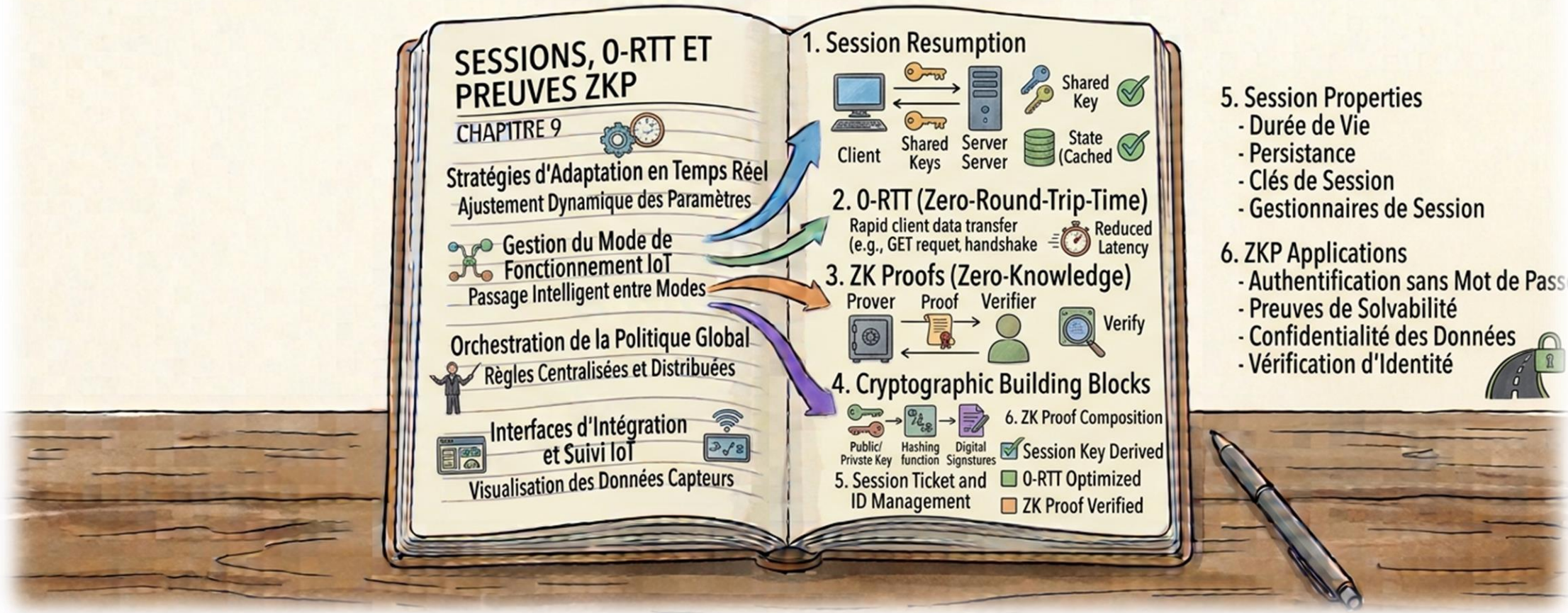
8.4. Où transmettre la batterie ?

La proposition initiale place ces informations dans une extension du `ClientHello`. Cela implique de définir son format, sa taille, son traitement, sa confidentialité et sa prise en charge par les deux extrémités.

`CryptoProvider` ne constitue pas, à lui seul, une preuve que cette nouvelle extension est disponible. Cette faisabilité doit être étudiée séparément.

Pour une première évolution IoT, je recommande de transmettre la télémétrie dans le canal déjà établi et de l'utiliser pour les décisions ultérieures. Cela permettra de tester les politiques avant de modifier les messages TLS.

Chapitre 9 — Sessions, 0-RTT et preuves ZKP



9.1. Pourquoi reprendre une session ?

La reprise permet à une nouvelle connexion de réutiliser un contexte établi auparavant. Elle peut réduire le travail nécessaire à la reconnexion.

Le cadrage final prévoit une activation après validation, avec un **nouvel échange hybride**, et un lien avec l'identité, le profil de sécurité et le contexte applicatif.

Il faudra notamment définir l'expiration des tickets, leur invalidation et leur comportement après modification de politique.

9.2. Que signifie 0-RTT ?

RTT signifie *Round-Trip Time*, le temps d'un aller-retour réseau.

Avec le 0-RTT, un client envoie certaines données avant d'avoir reçu la nouvelle réponse du serveur. Il peut ainsi gagner une attente, mais ces données nécessitent des précautions particulières face au rejeu. Les protocoles applicatifs doivent définir les opérations admissibles et leur comportement en cas de répétition.

rfc-editor.org

9.3. Pourquoi une preuve ZKP ne suffit pas

Une preuve à divulgation nulle de connaissance peut démontrer une propriété sans révéler le secret associé.

Mais considérons le mécanisme décrit dans les premières pages : un client présente un ticket et une preuve valide. Si quelqu'un recopie cet ensemble, **qu'est-ce qui empêche sa seconde présentation ?**

La validité de la preuve ne répond pas, à elle seule, à cette question.

Il faut définir :

- ce qui est prouvé ;
- le lien avec le message et le serveur ;
- la fraîcheur et l'unicité ;
- l'état conservé après consommation ;
- le comportement après panne ;
- la coordination entre plusieurs serveurs.

Avec ton expérience des bases de données, le rapprochement est immédiat : **une preuve valide et une opération consommée une seule fois sont deux problèmes différents.**

9.4. Décision proposée

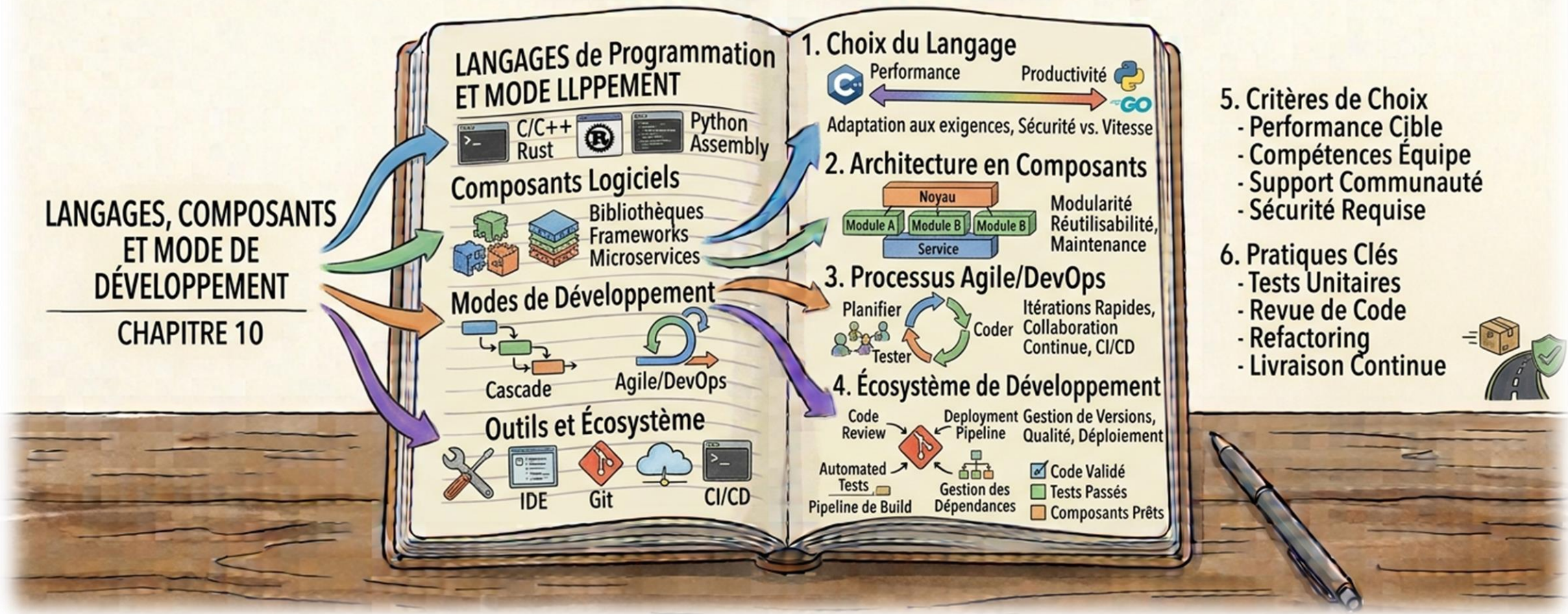
Je conserve la règle de la p. 209 : **0-RTT désactivé en V1**.

Le chantier ZKP devra disposer de sa propre spécification et de sa propre évaluation. Il ne peut pas être présenté comme une propriété acquise.

Il faut aussi distinguer cette question de `KeyUpdate`, qui renouvelle les clés de trafic d'une connexion. TLS 1.3 ne permet pas de traiter ce mécanisme comme une renégociation libre des algorithmes. rfc-editor.org

---

Chapitre 10 — Langages, composants et mode de développement



10.1. Répartition recommandée

Je conserve les composants de travail de la p. 206 :

Composant	Langage	Responsabilité
`qsa_crypto`	C	Opérations cryptographiques fournies par QSA
`qsa_ffi`	Rust/C	Interface contrôlée entre les deux langages
`qsa_provider`	Rust	Adaptation de QSA à rustls
`qsa_tlsd`	Rust	Service, connexions, politiques et transfert
`opt_tlsh`	Rust	Optimisation des ressources et entrées/sorties
`qsa_admin`	Rust	Administration locale et métriques
`qsa_tooling`	Python	Installation, diagnostic, tests et rapports

10.2. Pourquoi cette répartition ?

Le service réseau doit gérer des états de connexion, des limites de ressources et de la concurrence. Rust fournit un cadre pertinent pour ces responsabilités.

Le C reste le langage du composant QSA décrit dans le document. Il devra avoir un périmètre réduit et documenté.

La frontière C/Rust exige une attention particulière : tailles des buffers, allocations, durée de vie des objets, erreurs et règles de concurrence. **Les garanties de Rust ne corrigent pas automatiquement une erreur dans le C appelé.** La documentation Rust traite ces appels comme une frontière nécessitant des garanties explicites. The Rustonomicon

10.3. Ce que veut dire « temps constant »

Cela signifie que les opérations manipulant des secrets ne doivent pas révéler ces secrets par des différences exploitables de branches, d'accès mémoire ou de durée.

Cela ne signifie pas que toutes les connexions réseau prennent exactement le même temps.

Comme le demande la p. 207, la vérification doit porter sur les implémentations compilées des plateformes supportées. L'intention du développeur dans le code source ne suffit pas.

10.4. Le choix C/Rust et le NIST

Les sources consultées n'établissent pas une obligation générale imposant C/Rust à tout produit TLS visant une validation.

La FAQ NIST contient bien une exigence historique liée aux implémentations fournies lors des soumissions au processus PQC. Elle ne constitue pas un mandat général pour l'architecture de TLSH. CSRC

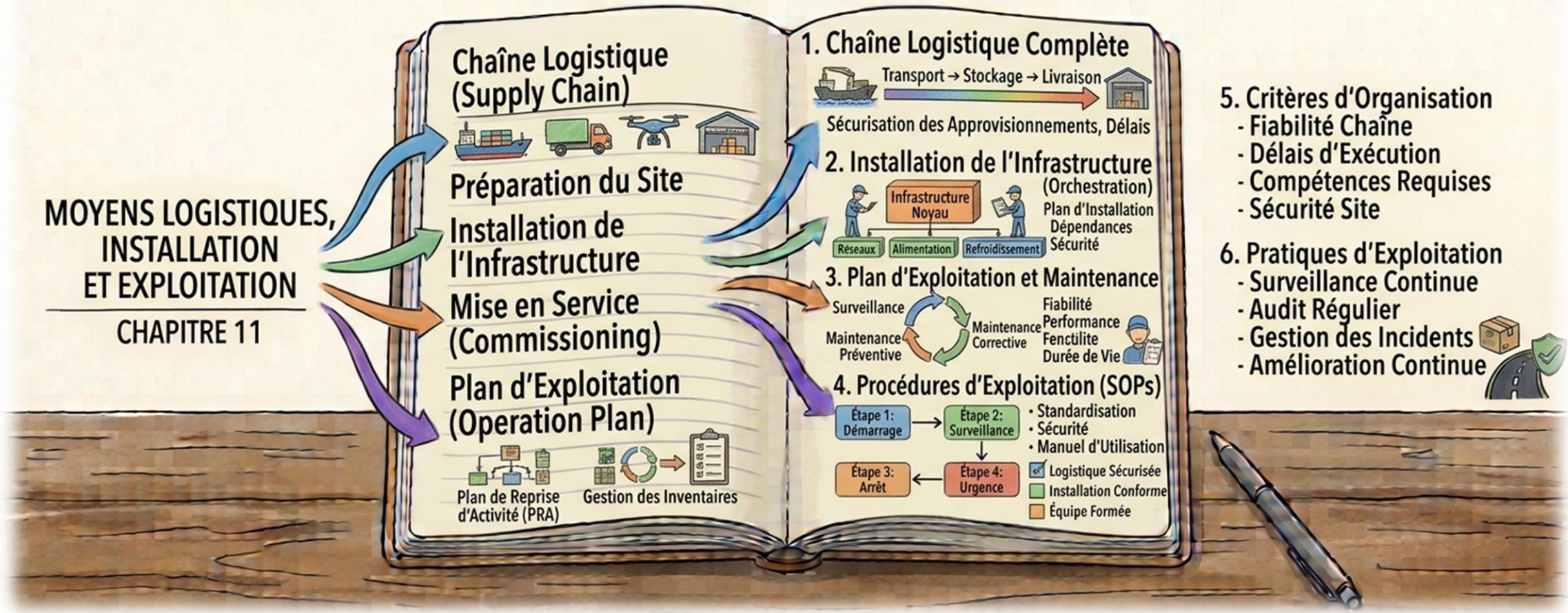
10.5. Organisation du développement

Je recommande un dépôt versionné avec :

- versions de dépendances fixées par livraison ;
- compilation reproductible ;
- tests automatisés ;
- documentation des interfaces ;
- décisions d'architecture tracées ;
- séparation explicite entre fonctions expérimentales et profil de production.

Les premières réalisations doivent rester verticales : une connexion complète fonctionnelle, puis QSA, puis OPT.

Chapitre 11 — Moyens logistiques, installation et exploitation



11.1. Environnement de laboratoire

Le premier laboratoire peut utiliser ton infrastructure existante :

- une machine ou VM serveur Linux ;
- un client de test distinct ;
- une petite application locale derrière TLSH ;
- des certificats de laboratoire ;
- des outils pour introduire latence et pertes ;
- un stockage des versions, mesures et journaux expurgés.

Pour les mesures finales, le générateur de charge et le serveur devront disposer de ressources suffisamment indépendantes pour éviter de mesurer leur compétition sur le même processeur.

L'évaluation énergétique IoT demandera ensuite un équipement représentatif et une mesure réelle de sa consommation.

11.2. Installation entièrement automatisée

La p. 213 correspond exactement à une exploitation simple : **un outil Python intégré au projet.**

Son interface principale doit proposer :

Action	Comportement
`check`	Diagnostiquer l'état sans modification
`repair`	Appliquer les installations ou corrections nécessaires

Cet outil prendra en charge les dépendances, répertoires, permissions, configurations, services systemd, contrôles d'intégrité et vérifications de santé.

11.3. Mise à jour et retour arrière

Le fonctionnement attendu est :

1. préparer une version vérifiée ;
2. contrôler sa configuration ;
3. effectuer les vérifications préalables ;
4. l'activer ;
5. vérifier le service ;
6. restaurer la version précédente si nécessaire.

Le retour arrière doit porter sur un ensemble cohérent : binaire, dépendances et configuration compatible.

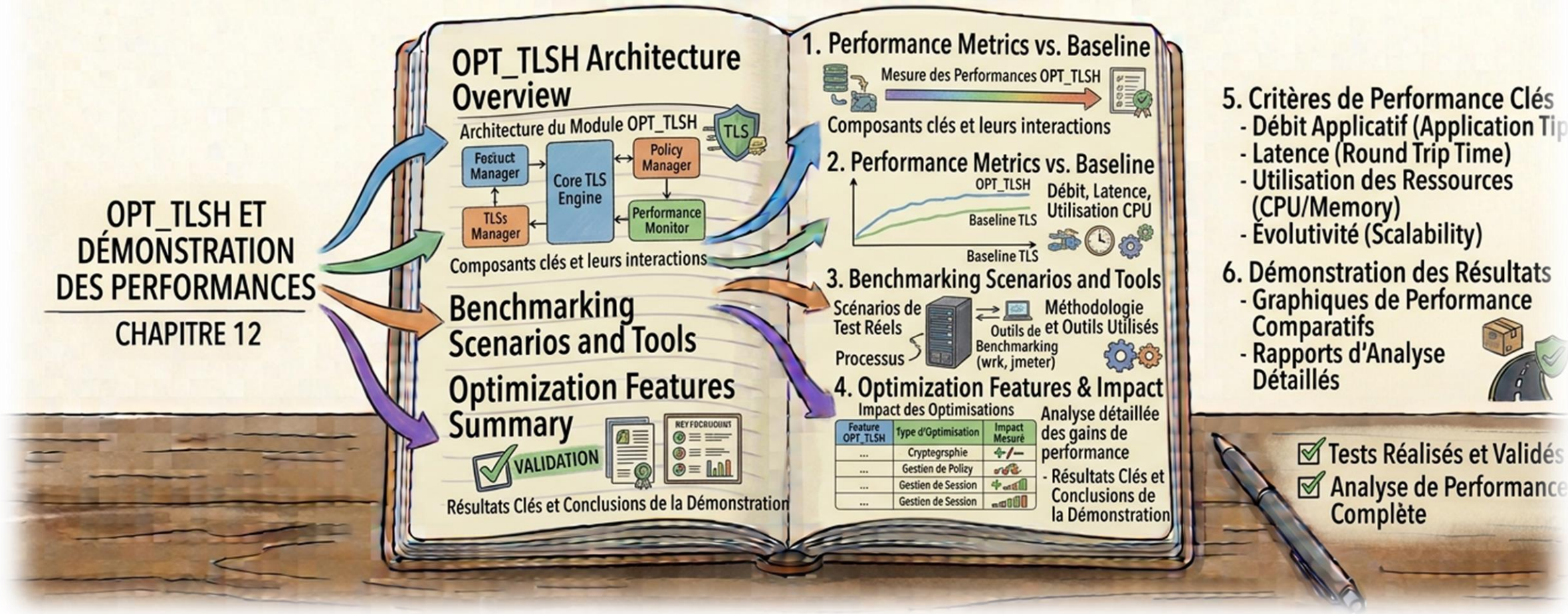
11.4. Exploitation quotidienne

L'administration V1 sera locale et protégée. Elle doit permettre de connaître :

- la version active ;
- les profils configurés ;
- les groupes négociés ;
- les connexions et files ;
- les erreurs ;
- l'état d'OPT_TLSH ;
- la validité des certificats.

Les journaux excluront les secrets. WSA pourra ultérieurement consulter ces informations, conformément à l'intégration facultative en lecture seule prévue dans le document.

Chapitre 12 — OPT_TLSH et démonstration des performances



12.1. Ce que l'optimiseur peut améliorer

La p. 210 définit une mission concrète :

- réguler l'admission des connexions ;
- répartir le travail ;
- borner les files ;
- réduire les allocations et copies ;
- regrouper certaines écritures ;
- adapter les ressources à la charge.

Il faut toujours mesurer le compromis obtenu. Regrouper davantage d'écritures peut réduire le travail système tout en augmentant l'attente d'une petite requête.

12.2. Les invariants

OPT devra conserver exactement les règles cryptographiques du profil.

Cela exclut notamment de gagner du temps en supprimant des vérifications ou en réutilisant des éléments éphémères entre connexions. La RFC 9846 interdit désormais la réutilisation d'un même `KeyShare` sur plusieurs connexions. datatracker.ietf.org

Sa désactivation devra rétablir le traitement standard avec la même protection.

12.3. Le banc comparatif

Le document propose une bonne séparation expérimentale :

Configuration	Contenu	Question traitée
B0	Service rustls avec fournisseur standard	Quelle est la référence ?
B1	Même service avec QSA, sans OPT	Quel est l'apport de QSA ?
B2	Même service avec QSA et OPT	Quel est l'apport supplémentaire d'OPT ?

Les comparaisons utiliseront les mêmes groupes, certificats, ressources et charges.

Cette discipline est essentielle : une baisse de sécurité ou un matériel différent ne doit pas être comptabilisé comme un gain d'optimisation.

12.4. Mesures et critères d'acceptation

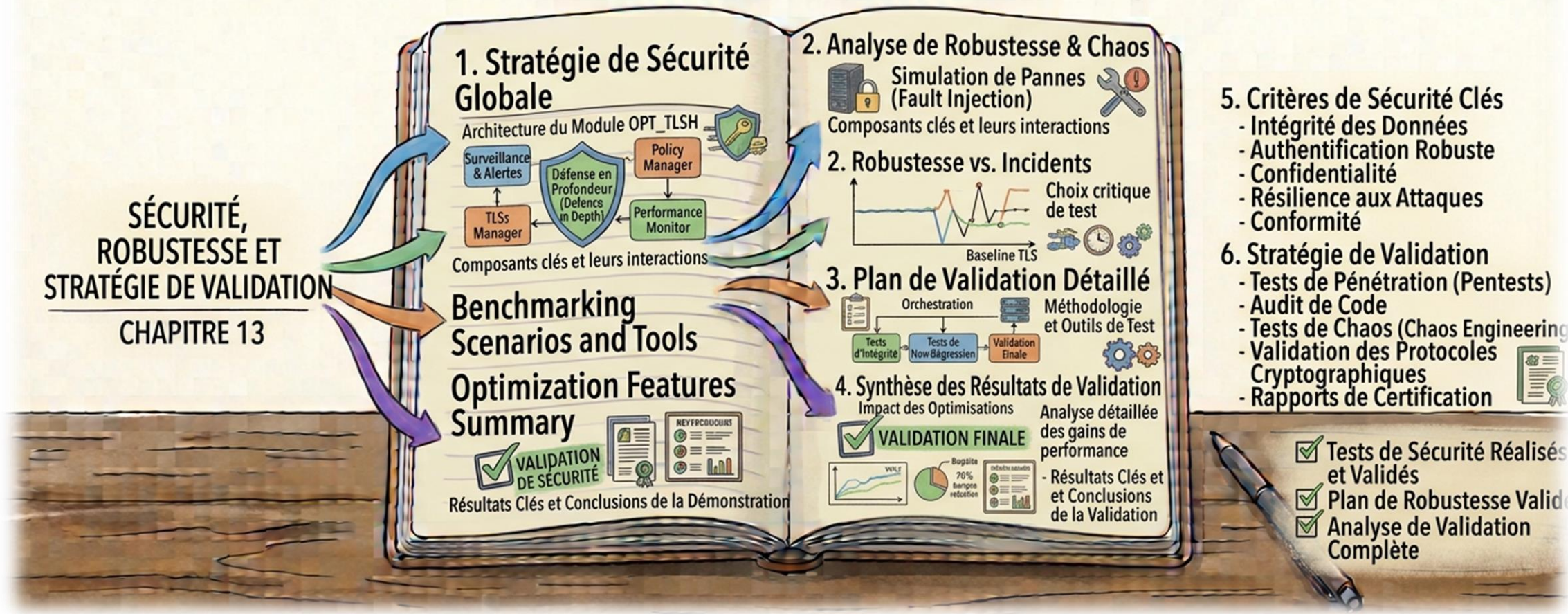
La p. 211 prévoit RTT de 1/20/100 ms, pertes de 0/0,1/1 %, MTU de 1 280/1 500 octets et au moins cinq répétitions des points retenus.

Les résultats doivent inclure débit, CPU, mémoire, erreurs et latences p50/p95/p99. **Le p99 indique le temps sous lequel se terminent 99 % des opérations mesurées.**

Les échanges hybrides augmentent certaines tailles : la contribution cliente de `X25519MLKEM768` représente 1 216 octets, avant les autres éléments du handshake. Ce coût doit apparaître dans les essais réseau.

rfc-editor.org

Chapitre 13 — Sécurité, robustesse et stratégie de validation



13.1. Transformer chaque exigence en preuve

Je recommande une matrice simple :

Exigence → composant responsable → scénario de test → résultat conservé.

Exemples :

Exigence	Preuve attendue
Hybridation obligatoire	Un client uniquement classique est refusé
Identité du serveur vérifiée	Un certificat incorrect provoque un échec
Destination autorisée	Une demande vers une destination non configurée est rejetée
Mémoire bornée	Une surcharge entraîne un rejet contrôlé
Retour arrière fonctionnel	Une mise à jour défaillante restaure le service précédent

13.2. Qualification cryptographique

Avant l'intégration de QSA, il faudra établir s'il s'agit :

- d'une implémentation d'un algorithme normalisé ;
- d'une optimisation compatible ;
- d'une construction nouvelle.

Les contrôles incluront des vecteurs de référence, des entrées invalides et des comparaisons avec une implémentation indépendante.

Une nouvelle construction cryptographique demandera une évaluation distincte. Elle ne pourra pas être présentée comme une simple variante interchangeable de ML-KEM.

13.3. Robustesse du service

Les essais devront couvrir les clients lents, messages incomplets, déconnexions, saturations, certificats invalides, erreurs de configuration et indisponibilités de l'application derrière TLSH.

Le **fuzzing**, qui consiste à générer de nombreuses entrées déformées, ciblera notamment les nouvelles interfaces et la frontière C/Rust.

L'exécution concurrente devra être bornée : nombre de négociations simultanées, capacité des files, mémoire par connexion et temps d'attente maximum.

13.4. Interopérabilité

Deux programmes développés ensemble peuvent reproduire la même erreur.

Le client et le serveur TLSH devront donc dialoguer avec des contreparties indépendantes. Le PDF prévoit OpenSSL et rustls sans QSA ; OpenSSL fournit effectivement les groupes hybrides nécessaires à ces essais.

OS OpenSSL Docum...

13.5. Critère de passage au pilote

Le pilote doit être conditionné par un dossier de résultats reproductibles, des limites connues et une revue indépendante des parties sensibles.

Le développement assisté, les tests automatisés et la revue cryptographique apportent des preuves complémentaires. Aucun de ces éléments ne remplace tous les autres.

Chapitre 13 — Validation, qualification et passage au pilote

Transformer les exigences en preuves, qualifier la cryptographie, valider la robustesse et préparer le pilote

13.1. Transformer chaque exigence en preuve

Matrice simple recommandée : Exigence → composant responsable → scénario de test → résultat conservé

Exigence	Preuve attendue
Hybridation obligatoire	Un client uniquement classique est refusé
Identité du serveur vérifiée	Un certificat incorrect provoque un échec
Destination autorisée	Une demande vers une destination non configurée est rejetée
Mémoire bornée	Une surcharge entraîne un rejet contrôlé
Retour arrière fonctionnel	Une mise à jour défaillante restaure le service précédent

13.2. Qualification cryptographique

- Implémentation d'un algorithme normalisé
- Optimisation compatible
- Construction nouvelle

Contrôles : vecteurs de référence, entrées invalides, comparaisons avec une implémentation indépendante.

Une nouvelle construction cryptographique demandera une évaluation distincte. Elle ne pourra pas être présentée comme une simple variante interchangeable de ML-KEM.

13.3. Robustesse du service

Clients lents | Messages incomplets | Déconnexions | Saturations | Certificats invalides | Erreurs de configuration | Indisponibilités de l'application derrière TLSH

Fuzzing : le fuzzing cible notamment les nouvelles interfaces et la frontière C/Rust.

Exécution concurrente bornée : négociations simultanées, capacité des files, mémoire par connexion, temps d'attente maximum.

13.4. Interopérabilité

Client TLSH ⟷ Contrepartie indépendante

Serveur TLSH ⟷ Contrepartie indépendante

- Deux programmes développés ensemble peuvent reproduire la même erreur.
- Le client et le serveur TLSH doivent dialoguer avec des **contreparties indépendantes.**
- Le PDF prévoit OpenSSL et rustls sans QSA ; OpenSSL fournit effectivement les groupes hybrides nécessaires à ces essais.

13.5. Critère de passage au pilote

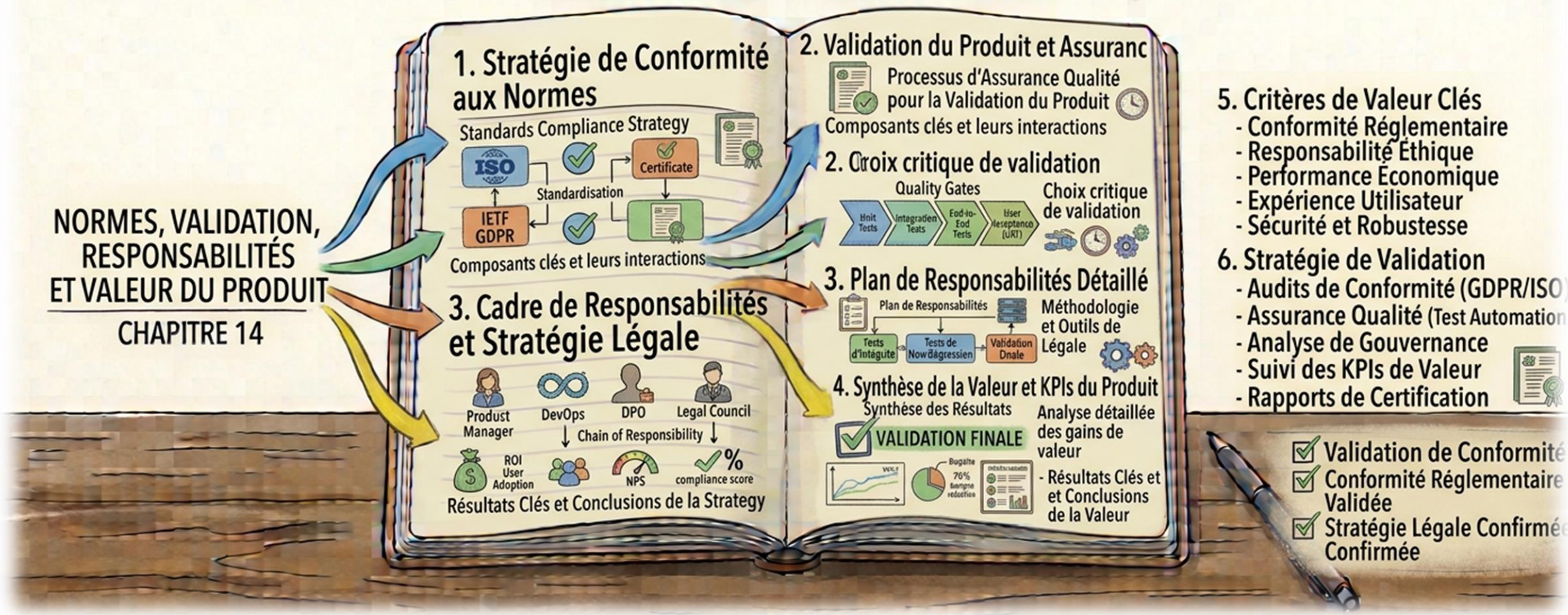
» Dossier de résultats reproductibles ✓ | Limites connues ✓ | Revue indépendante des parties sensibles ✓ | Développement assisté + tests automatisés + revue cryptographique = preuves complémentaires ✓

Aucun de ces éléments ne remplace tous les autres.

À RETENIR

Le passage au pilote exige des **preuves conservées**, une **qualification cryptographique** claire, une **robustesse** démontrée et une **interopérabilité** vérifiée.

Chapitre 14 — Normes, validation, responsabilités et valeur du produit



14.1. Référentiel à conserver

Le cadrage final mentionne les références pertinentes :

Référence	Rôle
RFC 9846	TLS 1.3
RFC 9954	Cadre de construction hybride
RFC 10024	Groupes hybrides ECDHE/ML-KEM
FIPS 203	ML-KEM
NIST SP 800-227	Recommandations d'utilisation des KEM

Ces publications existent dans les états indiqués par le document. Le projet devra enregistrer leurs versions et errata dans son référentiel de livraison. datatracker.ietf.org +4

14.2. Ce que « validé » signifie

Il faut distinguer :

- **algorithme normalisé** : une spécification reconnue existe ;
- **implémentation d'algorithme validée** : elle a suivi le parcours de validation correspondant ;
- **module cryptographique validé** : un périmètre précis a été évalué dans le cadre CMVP/FIPS 140-3 ;
- **produit TLS interopérable et correctement intégré** : cela nécessite encore ses propres vérifications.

Le NIST précise qu'intégrer un module validé ne permet pas de déclarer automatiquement tout le produit validé. Les versions, environnements et conditions d'utilisation comptent. CSRC

14.3. Responsabilités

La répartition proposée p. 205 est cohérente :

- **Christophe** fournit et explique QSA ;
- **toi** pilotes l'architecture et OPT_TLSH ;
- un spécialiste indépendant examine la sécurité ;
- un laboratoire précise le parcours de validation ;
- les responsables du projet clarifient droits, licences et propriété des contributions.

Je peux t'accompagner sur les explications, les interfaces, le code, les tests et l'analyse des résultats.

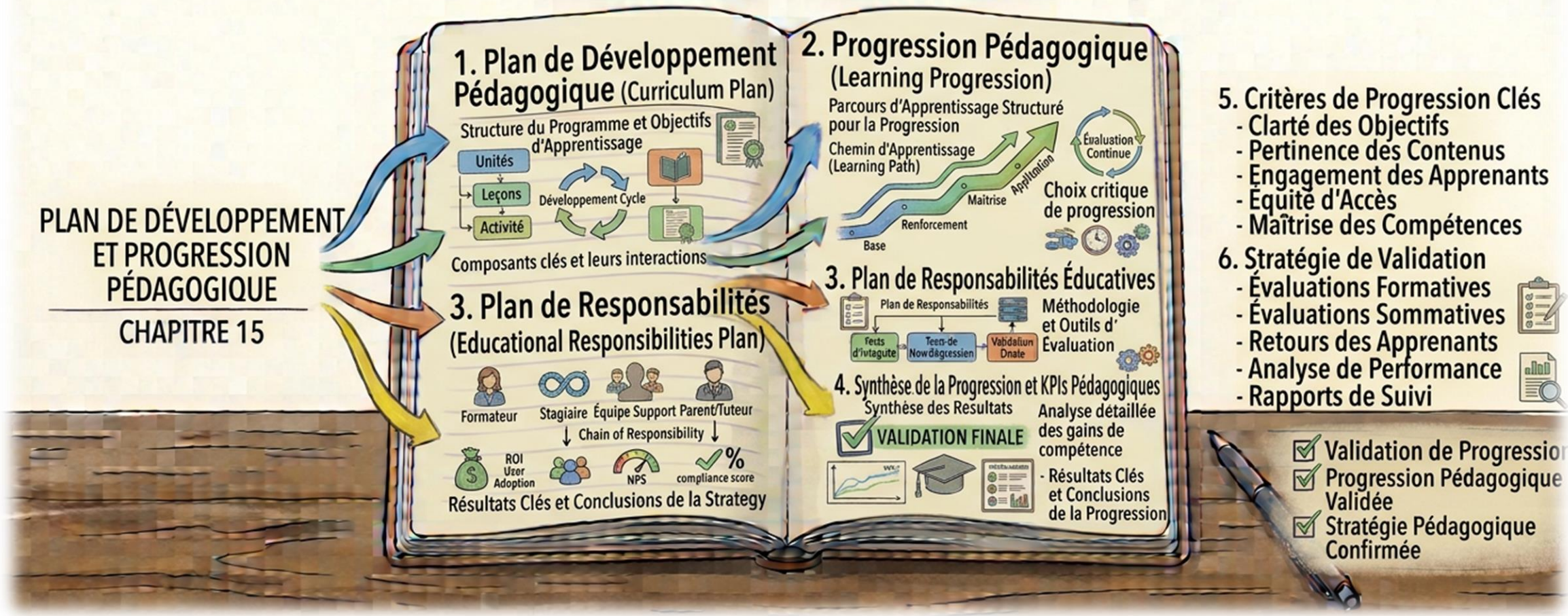
14.4. La valeur à démontrer

L'échange hybride existe déjà dans des bibliothèques disponibles, notamment rustls avec son fournisseur approprié. Rust

La valeur propre de TLSH devra donc être établie par ses résultats : apport de QSA, efficacité d'OPT, facilité d'installation, intégration, observabilité et, ultérieurement, adaptation IoT.

Les droits sur QSA, les licences et les éléments éventuellement protégeables devront être documentés. Le PDF ne permet pas aujourd'hui de conclure à une brevetabilité ou à une valeur commerciale chiffrée.

Chapitre 15 — Plan de développement et progression pédagogique



Lot	Objectif	Critère de sortie
TLSH-001 — Cadrage et qualification initiale	Réconcilier les périmètres, recevoir QSA, fixer interfaces et exigences	Architecture de référence et questions cryptographiques identifiées
TLSH-002 — Démonstrateur de référence	Service rustls standard, client, certificats, application et métriques	Connexion hybride réelle, vérifiée avec une contrepartie indépendante
TLSH-003 — Intégration QSA	Qualifier le composant, construire la FFI et le fournisseur	Fonctionnement conforme et comparaison B1/B0
TLSH-004 — OPT_TLSH	Admission, files, buffers, écritures et adaptation à la charge	Comparaison B2/B1 reproductible, sans modification des règles TLS
TLSH-005 — Industrialisation	Installateur Python, service, supervision, certificats et mises à jour	Installation sur machine vierge, réexécution idempotente et retour arrière
TLSH-006 — Assurance et pilote	Robustesse, audit, exigences laboratoire et exploitation contrôlée	Dossier de preuves et décision documentée de passage au pilote
TLSH-007 — Adaptation IoT	Client représentatif, télémétrie, politiques et mesures énergétiques	Bénéfices mesurés sur un équipement réel

15.2. Ton apprentissage TLS intégré au travail

Nous pouvons organiser ta montée en compétence autour de cinq démonstrations :

1. **Observer une connexion** : comprendre qui parle à qui et où TLS se termine.
2. **Faire échouer un certificat** : voir concrètement comment l'identité est vérifiée.
3. **Comparer classique et hybride en laboratoire** : distinguer le groupe négocié du chiffrement des données.
4. **Observer une reprise de session** : comprendre tickets, nouvelles connexions et 0-RTT.
5. **Activer QSA puis OPT séparément** : attribuer chaque effet au bon composant.

Tu pourras ainsi relier les concepts à des journaux, des mesures et des comportements réels.

15.3. Les éléments nécessaires au premier lot

Le prochain travail concret est **la qualification du paquet QSA_TLSH de Christophe.** Il faut réunir :

- les sources et fichiers d'interface ;
- les instructions ou scripts de compilation ;
- les dépendances et plateformes supportées ;
- les algorithmes et paramètres effectivement implémentés ;
- les tests, vecteurs et mesures disponibles ;
- les licences et droits d'utilisation ;
- les exigences écrites évoquées avec le NIST ou les laboratoires.

Nous pourrons établir le démonstrateur rustls standard pendant cette qualification. En revanche, l'intégration et les conclusions sur QSA dépendront de son examen.



Vue générale de la roadmap

1. **Cadrage & préparation** — Q1 2027
2. **Développement & intégration** — Q2 2027
3. **Tests & qualification** — Q3 2027
4. **Déploiement & exploitation** — Q4 2027
5. **Valorisation & go-to-market** — Q1 2028

MegaBatch	Objectif principal	Résultat concret à la fin
TLSH-MB-001	Construire le socle exécutable et son installation	Une véritable connexion TLS hybride entre un client et notre service, avec transmission à une application de test
TLSH-MB-002	Qualifier et intégrer QSA_TLSH	Le composant de Christophe fonctionne derrière rustls, avec une interface C/Rust contrôlée et des résultats comparables à la référence
TLSH-MB-003	Compléter l'exploitation du service	Profils, certificats, sessions, administration, mises à jour et retour arrière sont gérés proprement
TLSH-MB-004	Développer OPT_TLSH et mesurer ses effets	Les optimisations sont activables, observables et évaluées à sécurité identique
TLSH-MB-005	Qualifier la V1 et préparer le pilote	Une version candidate documentée, un dossier de preuves et une procédure de déploiement contrôlée
TLSH-MB-006 — après la V1	Développer l'adaptation IoT	Des politiques fondées sur les contraintes d'un équipement réel, avec mesure des effets énergétiques et réseau

PATCH : TLSH-MB-001

2. TLSH-MB-001 — Socle exécutable et première connexion hybride

Objectif

Installer TLSH sur un serveur Linux et démontrer une communication hybride complète, avant l'intégration du code QSA.

Ce premier MegaBatch regroupe le cadrage nécessaire, la structure du projet, le service initial, le client de validation et l'installation automatisée.

Il doit déjà produire quelque chose que tu peux démarrer, observer et comprendre.

2.1. Ce que nous construisons

Le socle du projet

Nous préparons :

- l'organisation des composants Rust et Python ;
- les dépendances et versions utilisées ;
- la configuration initiale ;
- la compilation automatisée ;
- les règles de lancement ;
- le format des journaux et des résultats de diagnostic.

La configuration définira notamment les points d'écoute, les certificats, le profil cryptographique, les limites de ressources et la destination applicative autorisée.

Le premier service réseau

Le service utilisera rustls et son fournisseur cryptographique de référence.

Il devra :

- accepter une connexion hybride ;
- appliquer le profil configuré ;
- terminer TLS ;
- transférer les données vers une application locale autorisée ;
- retourner la réponse au client ;
- fermer proprement la connexion.

Pour le démonstrateur, nous utiliserons une petite application HTTP/1.1 de test. Une requête pourra, par exemple, transmettre une température simulée et recevoir un accusé de réception.

Le client de validation

Le client devra :

- vérifier le certificat du serveur ;
- établir la connexion hybride ;
- envoyer une requête de test ;
- recevoir et vérifier la réponse ;
- afficher le groupe négocié, la durée et le résultat.

Nous utiliserons une vraie connexion TLS, afin que la démonstration vérifie réellement la pile retenue.

Nous utiliserons une vraie connexion TLS, afin que la démonstration vérifie réellement la pile retenue.

L'installation dès ce premier batch

L'outil Python proposera les deux actions principales :

Action	Fonction
`check`	Examiner l'environnement et expliquer ce qui manque
`repair`	Installer ou corriger les éléments nécessaires

Il prendra déjà en charge les répertoires, permissions, dépendances, configuration, service systemd et contrôles de santé.

Les certificats de laboratoire seront gérés de façon reproductible. Une seconde exécution devra préserver les éléments déjà valides.

L'installation dès ce premier batch

L'outil Python proposera les deux actions principales :

Action	Fonction
`check`	Examiner l'environnement et expliquer ce qui manque
`repair`	Installer ou corriger les éléments nécessaires

Il prendra déjà en charge les répertoires, permissions, dépendances, configuration, service systemd et contrôles de santé.

Les certificats de laboratoire seront gérés de façon reproductible. Une seconde exécution devra préserver les éléments déjà valides.

2.2. Les protections présentes immédiatement

Le service initial intégrera :

- des délais maximums ;
- un nombre maximal de connexions et de négociations ;
- des buffers bornés ;
- une destination applicative explicitement autorisée ;
- une exécution sous un compte dédié ;
- des erreurs et refus identifiables.

Ces protections constituent le fonctionnement normal du service, dès sa première version.

2.3. Première référence de performance : B0

Nous enregistrerons les premières mesures du service utilisant le fournisseur standard, sans QSA et sans OPT.

Cette configuration sera notre **référence B0**.

B0 utilisera déjà un groupe hybride. Cela permettra ensuite de comparer les implémentations sans changer le niveau de protection.

2.4. Composants principalement concernés

`qsa_tlsd`, `qsa_admin`, `qsa_tooling`, le client de validation, la configuration et les tests initiaux.

À ce stade, le service portera les noms de composants prévus par le projet, mais le fournisseur cryptographique actif sera clairement identifié comme fournisseur de référence.

2.5. Validation du MegaBatch

Nous considérerons ce batch terminé lorsque :

1. l'installation sur l'environnement Linux de test fonctionne ;
2. une seconde exécution ne casse pas l'installation ;
3. le client établit une connexion hybride ;
4. la requête atteint uniquement l'application autorisée ;
5. la réponse revient correctement ;
6. un certificat invalide et un client incompatible sont refusés ;
7. les résultats et premières mesures sont visibles.

Ce MegaBatch peut être réalisé sans disposer du code QSA de Christophe.

PATCH : TLSH-MB-002

3. TLSH-MB-002 — Qualification et intégration de QSA_TLSH

Objectif

Intégrer le composant cryptographique réel de Christophe dans le service, en conservant le fonctionnement TLS normalisé.

Ce MegaBatch comporte une qualification préalable, suivie de l'intégration lorsque les résultats le permettent.

3.1. Qualification du paquet QSA

Nous examinerons :

- les algorithmes effectivement implémentés ;
- les paramètres supportés ;
- les fonctions disponibles ;
- les dépendances ;
- la génération aléatoire ;
- la gestion des secrets ;
- les erreurs et contrôles d'entrée ;
- les règles d'utilisation concurrente ;
- les éléments de validation et les licences.

Le premier point à résoudre sera la nature exacte du composant : implémentation standard, optimisation compatible ou construction cryptographique nouvelle.

3.2. Construction de la frontière C/Rust

Nous développerons une interface documentée précisant :

- qui crée et détruit chaque objet ;
- la taille des données attendues ;
- la durée de vie des buffers ;
- les codes d'erreur ;
- les conditions d'appel ;
- les possibilités de concurrence.

Les zones Rust faisant appel au C seront limitées et identifiées.

Une erreur QSA devra produire un comportement sûr et un diagnostic exploitable. Le service ne devra pas changer discrètement de fournisseur pour masquer cette erreur.

3.3. Intégration dans le fournisseur rustls

Nous construirons le fournisseur QSA en remplaçant uniquement les opérations relevant réellement de ce composant.

Le fournisseur standard restera disponible comme référence explicitement sélectionnée pour les essais.

Le profil général sera qualifié en premier. Le second profil hybride prévu dans le cahier des charges sera ajouté au périmètre de validation selon les capacités réelles de QSA.

3.4. Référence B1

La configuration **B1** utilisera :

- le même service ;
- le même groupe hybride ;
- les mêmes certificats ;
- les mêmes ressources ;
- QSA activé ;
- OPT désactivé.

La comparaison **B1/B0** permettra d'identifier l'effet propre de QSA.

Nous conserverons aussi les résultats négatifs éventuels : surcoût, limite de concurrence ou comportement différent. Ils font partie de la qualification.

3.5. Composants principalement concernés

`qsa_crypto`, `qsa_ffi`, `qsa_provider`, puis les adaptations nécessaires de `qsa_tlsd` et de l'outillage de compilation et de test.

3.6. Validation du MegaBatch

La recette comprendra :

- conformité aux vecteurs de référence retenus ;
- comportement correct sur entrées invalides ;
- contrôles mémoire et concurrence ;
- interopérabilité avec une contrepartie indépendante ;
- absence de régression fonctionnelle ;
- comparaison B1/B0 reproductible.

Prérequis indispensable

Avant le lancement de ce MegaBatch, il faudra disposer du **paquet QSA complet** : sources, interfaces, compilation, dépendances, tests et droits d'utilisation.

En cas d'anomalie de qualification, l'activation de QSA restera bloquée. Le service de référence du premier MegaBatch restera utilisable et explicitement identifié.

PATCH : TLSH-MB-003

4. TLSH-MB-003 — Exploitation, certificats et cycle de vie du service

Objectif

Transformer le démonstrateur en service administrable pour une application réelle.

Je regroupe ici les politiques, les identités, les sessions et l'industrialisation de l'installation. Ces fonctions partagent les mêmes besoins de configuration, de contrôle et de retour arrière.

4.1. Profils et routes autorisées

Nous compléterons la gestion des profils :

- groupes cryptographiques autorisés ;
- authentification requise ;
- autorités de confiance ;
- destinations applicatives ;
- limites de ressources ;
- règles associées au nom de serveur et au protocole applicatif.

Chaque refus devra être compréhensible : profil incompatible, identité rejetée, route inconnue, délai dépassé ou saturation.

4.2. Gestion des certificats

Nous intégrerons :

- contrôle de validité et d'identité ;
- vérification de la correspondance certificat/clé ;
- suivi des échéances ;
- rotation contrôlée ;
- rechargement sans état partiellement appliqué ;
- politique de révocation documentée.

4.3. Gestion des connexions et reprises de session

Nous compléterons :

- les fermetures propres ;
- l'arrêt du service avec traitement contrôlé des connexions ;
- les expirations ;
- la reprise après interruption ;
- les tickets de session ;
- leur expiration et leur invalidation.

La reprise sera activée seulement après validation du mode retenu, avec un nouvel échange hybride et un lien explicite avec le profil, l'identité et le contexte applicatif.

Le 0-RTT restera désactivé.

4.4. Mises à jour et retour arrière

Nous enrichirons **le même installateur Python** créé dans le premier MegaBatch.

Il assurera :

- contrôle de provenance et d'intégrité ;
- gestion des versions ;
- préservation des configurations ;
- préparation d'une nouvelle version ;
- activation contrôlée ;
- vérification de santé ;
- retour à une version précédente cohérente.

4.5. Administration et supervision locales

L'administration permettra de consulter :

- état du service ;
- version active ;
- profils disponibles ;
- certificats et échéances ;
- connexions et files ;
- erreurs ;
- fournisseur cryptographique actif ;
- résultats des contrôles de santé.

L'interface restera locale et protégée.

4.6. Composants principalement concernés

`qsa_tlsd`, `qsa_admin`, `qsa_tooling`, la configuration et les tests d'exploitation.

4.7. Validation du MegaBatch

Le service devra démontrer :

- une rotation de certificat fonctionnelle ;
- le rejet d'une configuration invalide ;
- une reprise de session conforme au profil ;
- une mise à jour réussie ;
- un échec de mise à jour correctement traité ;
- un retour arrière vérifié ;
- un diagnostic clair lorsqu'une application derrière TLSH est indisponible.

PATCH : TLSH-MB-004

5.1. Fonctions de l'optimiseur

Nous travaillerons sur cinq axes :

Axe	Fonction attendue
Admission	Réguler les nouvelles négociations selon les capacités disponibles
Ordonnancement	Répartir le travail sans laisser certains clients monopoliser le service
Files	Maintenir des capacités et délais maximums explicites
Buffers	Réduire les allocations et copies inutiles
Écritures réseau	Regrouper certaines opérations avec une attente bornée

Les décisions d'adaptation utiliseront des mesures de charge et de ressources.

La propriété des états TLS restera clairement définie : OPT devra fonctionner avec les règles de concurrence du service.

5.2. Activation et désactivation contrôlées

Nous prévoirons un mode de référence et un mode optimisé.

Le passage d'un mode à l'autre conservera :

- les mêmes profils cryptographiques ;
- les mêmes vérifications d'identité ;
- les mêmes destinations autorisées ;
- les mêmes exigences de sécurité.

5.3. Comparaison B0/B1/B2

Nous conserverons exactement les trois configurations définies dans le cahier des charges :

Configuration	Fournisseur	OPT	Mesure recherchée
B0	Standard	Désactivé	Référence du service hybride
B1	QSA	Désactivé	Effet propre de QSA
B2	QSA	Activé	Effet supplémentaire d'OPT

Les trois configurations utiliseront le même groupe hybride et les mêmes conditions de sécurité.

Les mesures commencées dans les premiers batches seront ainsi prolongées sans changer leur signification.

5.4. Campagnes de mesure

L'outillage automatisera des essais couvrant :

- connexions complètes et reprises ;
- charges faibles et soutenues ;
- échanges courts et transferts longs ;
- latence et pertes réseau ;
- saturation ;
- comportement dans la durée.

5.5. Composants principalement concernés

Nouveau composant `opt_tlsh`, puis adaptations de `qsa_tlsd`, `qsa_admin` et de l'outillage de mesure.

5.6. Validation du MegaBatch

Le résultat attendu sera un rapport reproductible montrant :

- les situations où OPT améliore le service ;
- les situations où son apport est faible ;
- ses coûts éventuels ;
- les limites de fonctionnement ;
- les profils dans lesquels son activation est pertinente.

Nous ne fixerons pas artificiellement un pourcentage de gain avant les mesures. L'activation par défaut devra être justifiée par les résultats et les critères de recette retenus.

PATCH : TLSH-MB-005

6.1. Qualification de bout en bout

Nous regrouperons les campagnes portant sur :

- l'interopérabilité ;
- les messages invalides ;
- la concurrence ;
- les saturations ;
- les interruptions et redémarrages ;
- les certificats et sessions ;
- les mises à jour ;
- le retour arrière ;
- l'endurance.

Les tests de robustesse cibleront particulièrement les nouvelles interfaces, la frontière C/Rust et les entrées non fiables.

Les corrections issues de ces campagnes seront regroupées dans ce MegaBatch.

6.2. Production du dossier de preuves

L'outillage devra pouvoir rassembler :

- version exacte des composants ;
- dépendances et provenance ;
- configuration testée ;
- résultats des tests ;
- mesures B0/B1/B2 ;
- anomalies connues ;
- limites documentées ;
- procédures de reproduction.

6.3. Préparation à l'évaluation

La frontière cryptographique et la traçabilité seront préparées dès le premier batch, puis consolidées ici.

Le dossier permettra de discuter avec le laboratoire du périmètre réellement évalué : composants, versions, plateformes et conditions d'utilisation.

La fin de ce MegaBatch ne signifiera pas automatiquement que TLSH est validé FIPS. Elle fournira les éléments nécessaires à une démarche d'évaluation correctement définie.

6.4. Pilote contrôlé

Le pilote portera sur :

- une application identifiée ;
- un environnement connu ;
- une population de clients définie ;
- des critères de réussite ;
- une procédure de retour arrière ;
- une observation des erreurs et performances.

Une éventuelle remontée d'informations vers WSA restera facultative et en lecture seule.

6.5. Documentation livrée avec la V1

Nous préparerons :

- guide d'installation et de diagnostic ;
- guide d'exploitation ;
- explication des profils ;
- gestion des certificats ;
- procédures de mise à jour et de restauration ;
- lecture des métriques ;
- limites connues ;
- guide pédagogique TLSH.

6.6. Validation du MegaBatch

La version candidate devra disposer :

1. de résultats de qualification reproductibles ;
2. de critères de pilote vérifiés ;
3. d'aucune anomalie critique connue laissée sans traitement ;
4. d'une exploitation et d'un retour arrière démontrés ;
5. d'un dossier de preuves exploitable ;
6. d'une décision de passage à l'étape suivante fondée sur les résultats.

PATCH : TLSH-MB-006

Objectif

Étendre TLSH aux contraintes d'un équipement réel et mesurer l'intérêt des politiques adaptatives.

Ce MegaBatch commencera après le choix d'une plateforme cliente représentative.

Fonctions prévues

Nous développerons :

- collecte de l'état énergétique et réseau ;
- association de l'équipement à un profil autorisé ;
- transmission de la télémétrie ;
- politiques d'adaptation ;
- explication des décisions ;
- mesure de l'énergie et du trafic ;
- scénarios de batterie faible ou de réseau dégradé.

La première intégration pourra transmettre les informations dans le canal TLS déjà établi. L'ajout éventuel d'une extension au handshake fera l'objet d'une décision technique fondée sur un besoin démontré.

Le niveau minimal de sécurité fixé par l'opérateur restera obligatoire.

Validation attendue

Nous mesurerons l'effet sur une transaction métier complète, en incluant la radio, les retransmissions et les reconnexions.

Un gain CPU côté serveur ne sera pas présenté comme une économie de batterie côté capteur.

Le résultat devra préciser l'équipement, la pile cliente, le réseau et les conditions dans lesquelles les bénéfices sont obtenus.



Plan de réalisation détaillé - 18 Nov 2026



AGENDA TLSH

- Chapitre 1 — Le périmètre complet de TLSH
- Chapitre 2 — La place exacte des différents QSA
- Chapitre 3 — L’état réel du code fourni
- Chapitre 4 — Les contradictions documentaires à résoudre
- Chapitre 5 — L’architecture intégrée TLSH et QSA
- Chapitre 6 — Le profil cryptographique et l’échange hybride
- Chapitre 7 — Le contrat natif et la protection des secrets
- Chapitre 8 — Le service réseau et le client de validation
- Chapitre 9 — Les certificats et l’authentification post-quantique
- Chapitre 10 — Le moteur de politiques et l’adaptation IoT

AGENDA TLSH – Part 2

- Chapitre 11 — Les sessions, les tickets et les preuves ZKP
- Chapitre 12 — OPT_TLSH et les performances
- Chapitre 13 — Installation, exploitation et administration
- Chapitre 14 — La stratégie de qualification
- Chapitre 15 — Conformité, intégration du moteur officiel et valorisation

Chapitre 1 — Le périmètre complet de TLSH

Le périmètre complet de TLSH

TLS hybride post-quantique : architecture, composants, sécurité et déploiement



1.1. La vision fonctionnelle à préserver

L'analyse initiale décrit une plateforme réunissant quatre capacités :

- Un transport TLS hybride classique/post-quantique.
- Une intégration cryptographique interchangeable, avec QSA.
- Une adaptation aux contraintes du client et du réseau.
- Une optimisation des ressources, complétée par une recherche sur les tickets de session et les preuves ZKP.

La signalisation du client, les politiques de sécurité, l'énergie consommée et la reprise de session font donc partie du projet. **Une première passerelle TLS hybride constitue le socle de TLSH ; elle ne représente pas encore toute la solution exposée dans l'analyse fonctionnelle.**

Référence fonctionnelle, notamment diapositives 55–56 pour les exigences EF-01 à EF-10 :

1.2. La première version exploitable

Le plan de développement a déjà ramené cette ambition à un premier périmètre précis :

- TLS 1.3 sur TCP.
- Serveur Linux x86-64.
- Service réseau en Rust.
- Bibliothèque rustls pour le protocole.
- Interface native vers QSA.
- Application autorisée derrière la terminaison TLS.
- Installation automatisée, supervision et retour arrière.
- OPT_TLSH activable séparément.

1.3. Trois états de livraison à distinguer

État	Ce qu'il permet d'affirmer
Socle démontré	Une requête traverse une vraie connexion hybride avec serveur authentifié
V1 exploitable	Le service, QSA, OPT, les certificats, les mises à jour et la supervision sont qualifiés
Programme fonctionnel complet	Les exigences IoT, énergétiques et de reprise avancée ont également été réalisées ou explicitement arbitrées

Cette distinction évitera d'annoncer « TLSH terminé » alors que les fonctions originales d'adaptation restent à construire.

Chapitre 2 — La place exacte des différents QSA

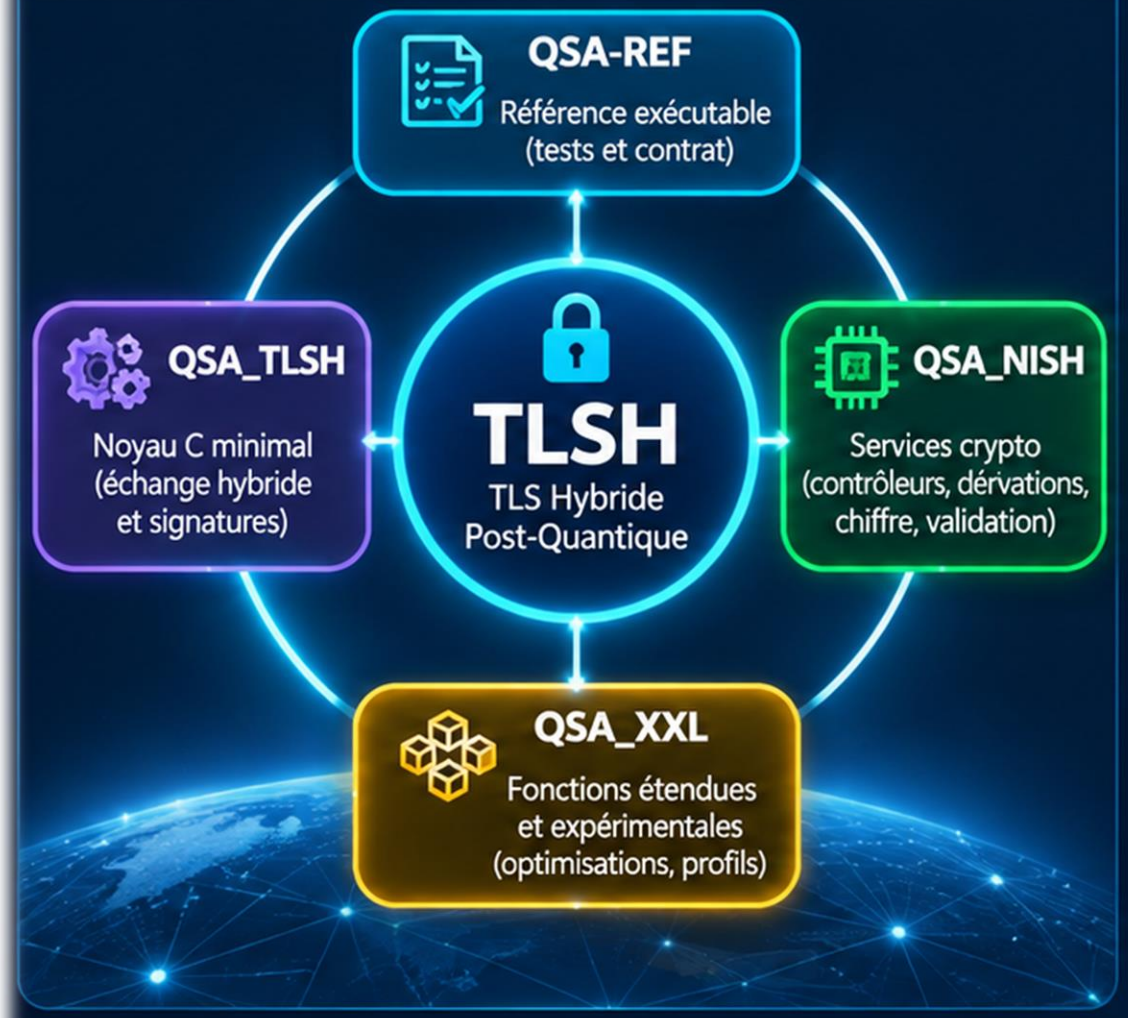
Chapitre 2 — La place exacte des différents QSA

Positionnement, rôles, interfaces et complémentarité des variantes QSA dans l'écosystème TLSH



Chapitre 2 — La place exacte des différents QSA

Positionnement des QSA dans l'architecture TLSH



Composant	Rôle principal	Place exacte dans TLSH
QSA-REF	Référence exécutable, tests et contrat d'intégration	Permet d'avancer sans attendre le moteur de Christophe
QSA_TLSH	Noyau C minimal pour échange hybride et signatures	Moteur cible directement pertinent pour TLSH
QSA_NIST	Services cryptographiques, contrôleurs, dérivations, chiffre et validation de l'échange	Fournisseur éventuel de capacités : son périmètre complet n'est pas requis pour transporter du TLS
QSA_XXL	Fonctions étendues et constructions expérimentales	Intégration éventuelle par profils distincts et qualification spécifique

Synthèse

Les **QSA-REF**, **QSA_TLSH**, **QSA_NIST** et **QSA_XXL** se complètent pour constituer l'écosystème cryptographique de **TLSH**, du socle de référence aux fonctions avancées.

2.1. Une différence de paramètres à traiter

QSA_TLSH vise notamment **ML-KEM-768 et ML-DSA-44**. QSA_NIST décrit principalement **ML-KEM-1024 et ML-DSA-87**.

Leur appartenance aux mêmes familles ne les rend pas interchangeables. Le contrat devra identifier précisément l'algorithme, ses paramètres, les formats et les tailles.

2.2. Les fonctions QSA qui ne doivent pas bloquer TLSH

Cryptonit, les conteneurs QSAF/QSAH/QSAS, Supersafe et le cœur propriétaire XXL ont leur propre utilité. Ils ne sont pas nécessaires au premier échange `X25519MLKEM768`.

De même, le document XXL indique que plusieurs sous-ensembles SQIsign ont été développés, mais liste encore les opérations complètes de génération, signature et vérification parmi les travaux restants. Ses résultats intermédiaires ne démontrent donc pas une signature complète déjà intégrable.

2.3. La règle d'intégration

TLSH consomme des capacités cryptographiques identifiées derrière une interface stable.

Une implémentation compatible avec le groupe standard peut remplacer une autre après qualification. Une construction différente doit disposer de son propre profil ; elle ne peut pas conserver artificiellement le nom et l'identifiant d'un groupe normalisé.

Chapitre 3 — L’état réel du code fourni

Chapitre 3 — L'état réel du code fourni



Contenu analysé

Modules Python	7
Fichiers de configuration	1
Données / vecteurs (qsa_ref_vectors.json)	1
Total des éléments	9

Fonctionnalités présentes

- Structure QSA de référence (QSA-REF)
- Fonctions cryptographiques de base
- Vecteurs et tests de validation
- Gestion de configuration
- Contrôles et auto-tests
- Contrats / interfaces de fonctiongerment

État global



- Base opérationnelle présente
- Exécutables et tests disponibles
- Logique principale QSA en place
- Périmètre TLSH à compléter
- Intégration et orchestration à finaliser
- Durcissement, performances et packaging à consolider

Conclusion



Le code fourni constitue un **socle solide et exploitable** pour QSA, mais ne couvre pas encore l'ensemble du périmètre **TLSH**.

Il sert de point de départ pour le développement complet et l'intégration hybride.

3.1. Un socle substantiel et réutilisable

Les sources s'identifient comme **QSA-REF 0.4.0**, avec le **contrat 1.1.0**.

Source	Fonction effectivement présente
qsa_ref_contracts.py	Statuts, erreurs, états, résultats, capacités et références opaques aux objets
qsa_ref_config.py	Configuration et profil initial `TLSH-HYB-1`
qsa_ref_crypto.py	X25519, ML-KEM-768, assemblage hybride et ML-DSA-44
qsa_ref.py	Initialisation, inventaire réel des capacités et façade d'utilisation
qsa_ref_checks.py	Banc de contrôles, cas négatifs et rapports
install_qsa_ref.py	Diagnostic et préparation de l'environnement Linux
qsa_ref_vectors.json	Corpus de 76 cas de référence

Les opérations cryptographiques passent par les implémentations natives de `cryptography`. **Il ne s'agit pas d'une simulation de ML-KEM ou de ML-DSA.**

3.2. Résultats obtenus pendant cette analyse

Les vérifications ont été exécutées sur des copies isolées, avec Python **3.12.14**, `cryptography` **50.0.0** et son moteur déclaré **OpenSSL 4.0.1**. Aucun installateur ni déploiement n'a été exécuté.

Vérification	Résultat observé
Quatre auto-tests des modules	103 assertions réussies
Banc de qualification fourni	**10 contrôles sur 10 réussis**
Échanges X25519	20 concordants
Échanges ML-KEM	20 concordants
Échanges hybrides	20 concordants
Signatures et détection des modifications	10 scénarios réussis
Vecteurs fournis	76 réussis
Entrées incorrectes	138 refusées
Capacités manquantes	11 cas traités explicitement

Ces résultats concernent **le socle local et l'environnement indiqué.** Ils ne valident pas encore l'installation prévue avec OpenSSL 3.5 dédié, le serveur utilisateur, une connexion TLSH ou une certification.

3.3. Les corrections de consolidation identifiées

Quatre points doivent entrer dans les premiers lots :

1. **Configuration mal encodée.**
   Dans `load_config()`, un fichier UTF-8 invalide peut provoquer une exception non interceptée et laisser le composant en cours d'initialisation.
2. **Consommation concurrente d'un contexte hybride.**
   Un test imposant un entrelacement de deux appels a montré que `hybrid_client_finish()` peut réussir deux fois sur le même handle. La prise et la consommation doivent devenir atomiques, ou l'usage concurrent être explicitement interdit et contrôlé.
3. **Paquet d'installation incomplet dans les pièces reçues.**
   `requirements_qsa_ref.txt` est attendu par l'installateur, mais n'est pas joint. Je ne conclus pas qu'il manque sur ton serveur.
4. **Isolement et reproductibilité à renforcer.**
   L'installateur peut réutiliser un environnement virtuel quelconque déjà actif. Il faut imposer l'environnement du projet et figer les dépendances avec leur provenance et leur intégrité.

3.4. Ce qui n'est pas vérifiable avec ces pièces

Les sources reçues ne contiennent pas le service Rust, l'ABI C, `libqsa_ref`, le provider OpenSSL ni le moteur C de Christophe.

Chapitre 4 — Les contradictions documentaires à résoudre

Chapitre 4 — Écarts constatés et décisions de reprise



Point	Écart constaté	Décision de reprise proposée
Déroulement TLS	La description technique utilise encore des messages de TLS ancien.	Faire du déroulement TLS 1.3 de la pile retenue la référence.
Sens ML-KEM	Le DOCX QSA_TLSH fait encapsuler le client vers le serveur ; le plan QSA-REF corrige ce sens.	Conserver le flux corrigé : clé ML-KEM côté client, encapsulation côté serveur.
Sens du mot « hybride »	Certaines illustrations associent Kyber et AES comme si cela définissait l'hybridation.	Distinguer établissement de clés, signature et chiffrement des données.
Adaptation	EF-02 évoque la modification du nombre de tours.	Sélectionner des profils fixes ; ne pas modifier les primitives.
Repli classique	Certaines pages l'autorisent ; le profil initial l'interdit.	Aucun repli automatique dans le profil strict.
0-RTT	Présenté comme garanti par ZKP, puis exclu de la V1.	Conserver le chantier, désactivé tant que sa qualification propre manque.
Performances	Plusieurs pourcentages sont présentés sans mesures.	Les conserver comme objectifs à éprouver.

Synthèse des écarts

- Des incohérences ou terminologies à clarifier.
- Des éléments techniques à aligner sur le plan de référence.
- Des chantiers à cadrer (0-RTT, performances).

Conclusion

Reprise ciblée et maîtrisée pour aligner l'ensemble sur la référence technique, clarifier les concepts et sécuriser la suite du développement TLSH / QSA.

Point	Écart constaté	Décision de reprise proposée
Déroulement TLS	La description technique utilise encore des messages de TLS ancien	Faire du déroulement TLS 1.3 de la pile retenue la référence
Sens ML-KEM	Le DOCX QSA_TLSH fait encapsuler le client vers le serveur ; le plan QSA-REF corrige ce sens	Conserver le flux corrigé : clé ML-KEM côté client, encapsulation côté serveur
Sens du mot « hybride »	Certaines illustrations associent Kyber et AES comme si cela définissait l'hybridation	Distinguer établissement de clés, signature et chiffrement des données
Adaptation	EF-02 évoque la modification du nombre de tours	Sélectionner des profils fixes ; ne pas modifier les primitives
Repli classique	Certaines pages l'autorisent ; le profil initial l'interdit	Aucun repli automatique dans le profil strict
0-RTT	Présenté comme garanti par ZKP, puis exclu de la V1	Conserver le chantier, désactivé tant que sa qualification propre manque
Performances	Plusieurs pourcentages sont présentés sans mesures	Les conserver comme objectifs à éprouver

La description technique TLSH nécessite notamment une correction de son passage sur le handshake, `ChangeCipherSpec`, les MAC et la compression.

La référence actuelle de TLS 1.3 est **RFC 9846**, qui remplace RFC 8446. Elle ne reprend pas la séquence `ServerKeyExchange` / `ServerHelloDone` / `ClientKeyExchange`. Les records utilisent un AEAD ; `ChangeCipherSpec` ne constitue pas un mécanisme d'activation du post-quantique. datatracker.ietf.org

Livrable attendu : une spécification de réalisation unique, versionnée, avec chaque décision reliée à une exigence et à son test.

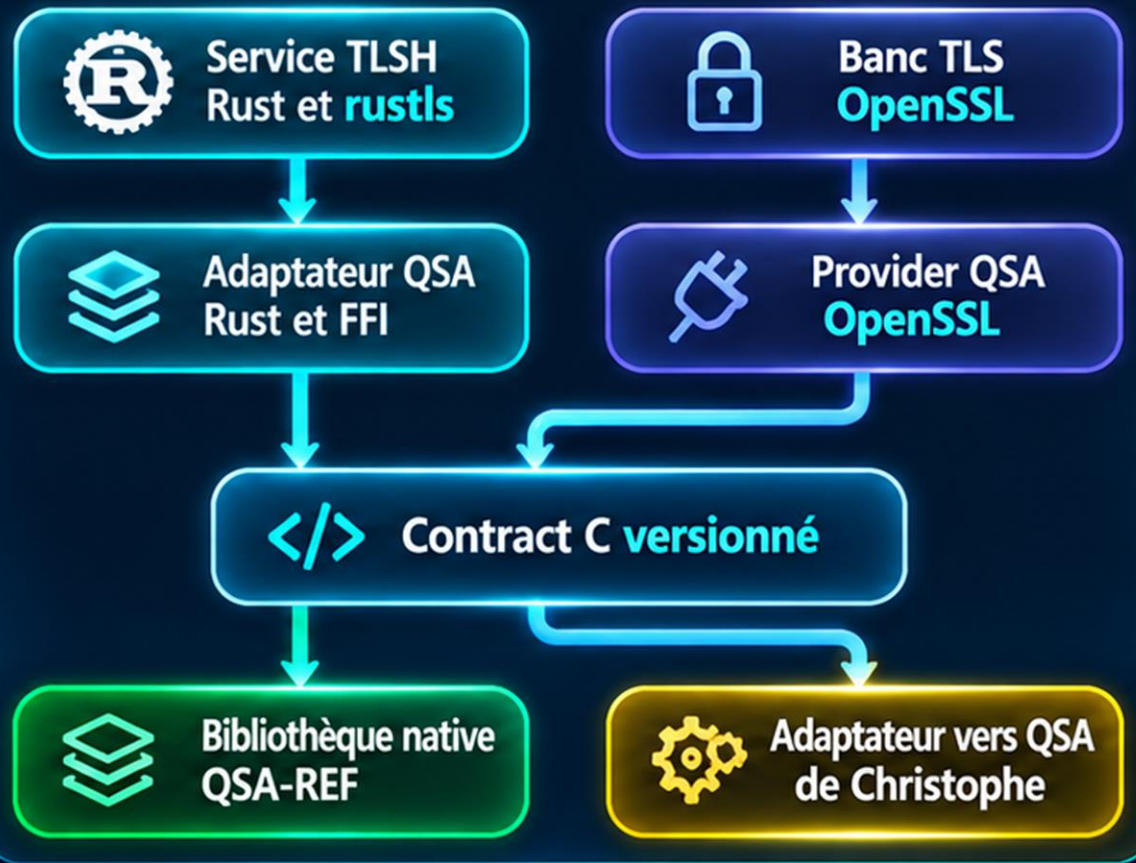
Chapitre 5 — L’architecture intégrée TLSH et QSA

Chapitre 5 — Conserver les deux parcours prévus

Contexte du plan TLSH

- Le plan TLSH impose **Rust** et **rustls** pour le produit.
- Le plan QSA-REF prévoit **OpenSSL** et son **provider** pour qualifier QSA indépendamment.
- Ces deux parcours peuvent partager le même **contrat C**.

Les deux parcours d'intégration



Points clés

- Deux parcours d'intégration complémentaires et cohérents.
- Un contrat C commun et versionné.
- Réutilisation et interopérabilité assurées.
- Séparation claire des responsabilités (TLSH, QSA-REF, provider).
- Aucun double chaînage TLS : le produit TLSH termine sa connexion avec **rustls**.

Règle de conception

Maintenir ces **deux parcours** sans les fusionner, pour préserver la flexibilité, la qualification et l'évolutivité du QSA.

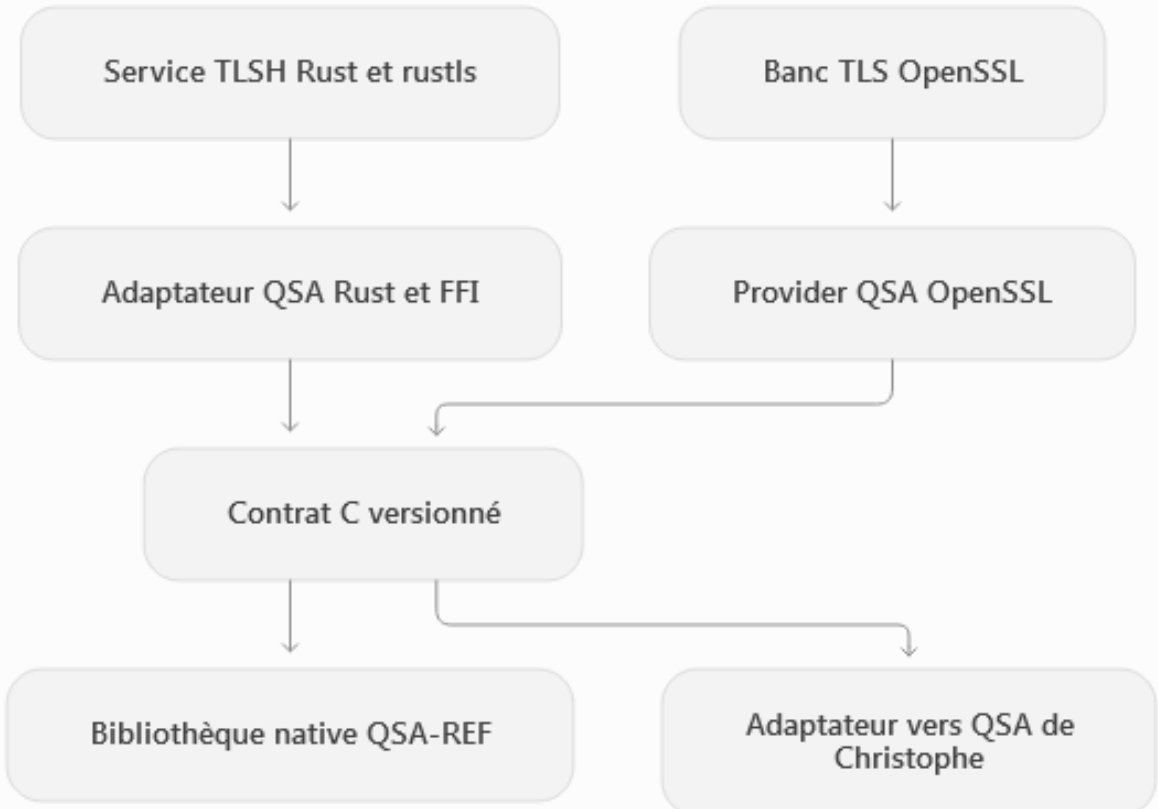
Conclusion

Le schéma définit deux parcours alternatifs et complémentaires d'intégration. Le produit TLSH termine sa connexion avec **rustls** ; il ne traverse pas deux terminaisons TLS successives.

5.1. Conserver les deux parcours prévus

Le plan TLSH impose **Rust et rustls** pour le produit. Le plan QSA-REF prévoit **OpenSSL et son provider** pour qualifier QSA indépendamment.

Ces parcours peuvent partager le même contrat C :



Le schéma représente des parcours alternatifs d'intégration. Le produit TLSH termine sa connexion avec rustls ; il ne traverse pas deux terminaisons TLS successives.

5.2. Responsabilités des composants

Je conserve les noms du plan :

Composant	Responsabilité
`qsa_crypto`	Opérations cryptographiques natives
`qsa_ffi`	Frontière contrôlée C/Rust
`qsa_provider`	Adaptation cryptographique pour rustls
`qsa_tlsd`	Connexions, politiques et transfert applicatif
`opt_tlsh`	Admission, ressources et entrées/sorties
`qsa_admin`	Administration locale et métriques
`qsa_tooling`	Installation, diagnostics, tests et rapports

Le provider OpenSSL de qualification devra être identifié séparément pour éviter toute confusion avec `qsa_provider` côté Rust.

Les interfaces `CryptoProvider` et `SupportedKxGroup` de rustls permettent effectivement de remplacer les groupes d'échange. L'API prévoit aussi le cas où la contribution serveur dépend de celle du client, comme pour un KEM. Rust +1

5.3. Place de Python

Python reste la référence fonctionnelle, l'outil d'installation et le pilote des campagnes. Le chemin actif du service utilise Rust et les bibliothèques natives.

Cette répartition conserve ton investissement Python tout en respectant le plan du produit.

Chapitre 6 — Le profil cryptographique et l’échange hybride

Chapitre 6 — Profil initial

Profil initial

Configuration de référence pour la première implémentation de TLSH / QSA



Paramètre	Choix de départ
Transport	TCP
Protocole	TLS 1.3
Groupe	X25519MLKEM768
Suite	TLS_AES_256_GCM_SHA384
Authentification initiale	Certificat serveur classique, vérifié
Repli classique automatique	Interdit
Reprise de session	Désactivée (1re démonstration)
0-RTT	Désactivé
FN-DSA	Capacité future identifiée, désactivée

Points clés

- Configuration simple et maîtrisée.
- Suite cryptographique hybride (X25519 + ML-KEM-768).
- Sécurité renforcée (AES-256-GCM + SHA-384).
- Aucun repli automatique (strict).
- Fonctions avancées (0-RTT, FN-DSA) prévues mais désactivées.

Objectif de ce profil

Établir une base stable, sécurisée et reproductible pour valider l'intégration initiale de TLSH / QSA, avant d'activer progressivement les fonctions avancées.

Conclusion

Le profil initial définit un **socle opérationnel**, focalisé sur la sécurité, la simplicité et la validation de la chaîne TLSH / QSA, avec des évolutions prévues par étapes.

6.1. Profil initial

Paramètre	Choix de départ
Transport	TCP
Protocole	TLS 1.3
Groupe	`X25519MLKEM768`
Suite	`TLS_AES_256_GCM_SHA384`
Authentification initiale	Certificat serveur classique, vérifié
Repli classique automatique	Interdit
Reprise de session	Désactivée pour la première démonstration
0-RTT	Désactivé
FN-DSA	Capacité future identifiée, désactivée

Le groupe `X25519MLKEM768` est désormais défini par **RFC 10024**, publiée en août 2026. Le cadre général de l'hybridation est décrit par RFC 9954. rfc-editor.org +1

6.2. Articulation avec les fonctions existantes

Les trois opérations QSA-REF correspondent déjà au découpage utile :

- `hybrid_client_start()` : prépare la contribution du client.
- `hybrid_server_respond()` : traite cette contribution et produit la réponse serveur.
- `hybrid_client_finish()` : termine le calcul côté client.

Le code et le plan QSA-REF retiennent :

Élément	Taille
Contribution cliente	1 216 octets
Contribution serveur	1 120 octets
Secret hybride combiné	64 octets

L'ordre est **ML-KEM puis X25519** pour ce groupe précis. Le secret combiné doit être consommé par la dérivation TLS standard. Il ne constitue pas directement une clé AES. rfc-editor.org

6.3. Profils suivants

Le profil `SecP256r1MLKEM768`, déjà prévu dans le plan, reste un jalon explicite. Il devra disposer de ses propres encodages, contrôles et essais d'interopérabilité.

L'ajout d'un profil ne sera annoncé comme disponible que lorsque le moteur, l'adaptateur et la pile TLS l'auront réellement exécuté.

Chapitre 7 — Le contrat natif et la protection des secrets

Chapitre 7 — Le contrat natif et la protection des secrets

Le passage au natif est le principal chantier d'intégration.

Objectif du chapitre

Intégrer un contrat C natif, performant et sécurisé, pour relier TLSH, QSA-REF et les providers (ex. OpenSSL) tout en protégeant les secrets.



Architecture d'intégration (contrat natif)



Points clés

- Contrat C natif et versionné.
- Interface stable et maîtrisée.
- Gestion centralisée des clés et des secrets.
- Encapsulation QSA sécurisée.
- Compatibilité avec OpenSSL et autres providers.
- Base évolutive pour les extensions futures.

Protection des secrets

Gestion des clés	Coffre interne	Accès contrôlé	Effacement sécurisé
(sécurisée et isolée)	(stockage durci)	(API native)	(zéroisation mémoire)

Conclusion

Le contrat natif constitue le **cœur de l'intégration**, garantissant des échanges performants, une gestion robuste des clés et une protection renforcée des secrets.

7.1. Contrat C unique

Chaque fonction devra préciser :

- Version d'interface et capacités requises.
- Types, tailles et formats.
- Propriétaire de chaque allocation.
- Durée de vie des objets.
- Règles de concurrence.
- État des sorties après erreur.
- Conditions de libération et d'effacement.
- Diagnostic associé au statut.

Les codes existants `OK`, `ERR` et `UNAVAIL` peuvent être conservés. Les causes détaillées devront permettre de distinguer entrée invalide, capacité absente, politique refusée, erreur d'aléa et défaut interne.

7.2. Consommation des contextes

Un contexte éphémère devra appartenir à une seule opération ou connexion. Sa consommation devra rester correcte en cas d'annulation, de timeout, de déconnexion et de concurrence.

La correction du handle hybride identifié au chapitre 3 devient donc une exigence du contrat commun, pas seulement un correctif Python.

7.3. Frontière réelle des secrets

Les handles opaques facilitent la discipline d'utilisation, mais ne constituent pas un coffre matériel.

Il faudra définir comment le secret partagé est remis à rustls, quelles copies existent et quand elles sont détruites. Si la pile reçoit des octets dans son processus, nous ne pourrons pas prétendre que le secret reste intégralement enfermé dans QSA.

7.4. Preuve du moteur exécuté

Pour le parcours OpenSSL, charger le provider ne suffit pas : plusieurs providers peuvent annoncer le même groupe, et la sélection peut rester indéterminée sans configuration appropriée. OpenSSL Docum...

La recette exigera donc :

1. L'identité du moteur et de ses bibliothèques.
2. Des compteurs d'appels effectifs.
3. Une erreur volontaire réservée au banc, provoquant l'échec attendu du handshake.
4. L'absence de substitution silencieuse par un autre fournisseur.

Chapitre 8 — Le service réseau et le client de validation

Chapitre 8 — Le service réseau et le client de validation

Objectif du chapitre

Définir le service réseau TLSH et le client de validation pour établir, tester et qualifier une **connexion hybride sécurisée.**



Architecture et flux de validation

Client de validation

- Initie la connexion TLSH
- Négocie le profil hybride
- Valide le certificat
- Contrôle la chaîne cryptographique
- Affiche le résultat de validation

Connexion TLSH (hybride)

- **TLS 1.3** (classique)
- **ML-KEM** (KEM PQC)
- **AES-256-GCM** (chiffrement)

Service réseau TLSH

- Écoute sur un port TLSH
- Applique le profil choisi
- Négocie le mode hybride (classique + PQC)
- Protège et gère les secrets (clés, sessions)
- Fournit un journal de validation

Points clés

- Deux composants distincts et complémentaires.
- Négociation TLS 1.3 + PQC (ex. ML-KEM).
- Validation complète de la chaîne et du profil.
- Gestion sécurisée des sessions et des secrets.
- Journaux et résultats de validation exploitables.

Fonctionnalités principales

- **Établissement de connexion TLSH**
- **Validation du profil** (groupe, suite, certificat)
- **Tests et mesures** (temps, succès)
- **Rapport de validation** (détaillé et horodaté)

Conclusion

Le service réseau TLSH et le client de validation forment un **ensemble opérationnel et testable**, garantissant la mise en œuvre, la vérification et la qualification de la connexion hybride.

8.1. Une démonstration applicative complète

Le premier résultat doit être visible :

> Un client vérifie le serveur, établit une connexion hybride, transmet une mesure HTTP/1.1 à une application locale autorisée et reçoit son accusé de réception.

Cela permet de contrôler toute la chaîne, au-delà d'une simple égalité entre deux secrets.

8.2. Fonctions du serveur

`qsa_tlsd` devra assurer :

- Chargement et validation de la configuration.
- Sélection explicite du profil et du fournisseur.
- Acceptation et refus des connexions.
- Routage autorisé selon SNI et ALPN.
- Transfert bidirectionnel avec gestion de la pression réseau.
- Fermeture propre.
- Délais, files et mémoire bornés.
- Arrêt contrôlé du service.

Une destination arbitraire transmise par un client ne devra jamais transformer TLSH en proxy ouvert.

8.3. Fonctions du client

Le client de référence devra vérifier le certificat et le nom attendu, exécuter une transaction, contrôler la réponse et produire un diagnostic comprenant le groupe négocié, le type d'authentification, la durée et la cause d'un éventuel échec.

8.4. Emplacement dans l'infrastructure

Le laboratoire utilisera un point d'écoute dédié. TLSH doit terminer la liaison dont nous souhaitons renforcer la protection.

Placer TLSH derrière un Nginx qui a déjà terminé la connexion du client ne renforcerait pas rétroactivement cette première liaison. Le plan prévoit justement cette distinction, diapositives 47–49.

Les applications WSA existantes restent indépendantes de ce laboratoire.

Chapitre 9 — Les certificats et l’authentification post-quantique

Chapitre 9 — Les certificats et l'authentification post-quantique

Objectif du chapitre

Définir la chaîne de certificats et l'authentification post-quantique (PQC) pour sécuriser l'identité, l'échange des clés et la connexion TLSH.



Chaîne de certificats hybride (Classique + PQC)

Racine (CA) hybride
- Confiance (Root CA)
- Double signature (Classique + PQC)
- Politique d'émission

Intermédiaire (ICA)
- Distribution et contrôle
- Signature hybride
- Validation des usages (TLSH, QSA, services)

Certificat serveur / client
- Identité du service
- Clés hybrides (ex. ML-KEM)
- Signature PQC (ex. ML-DSA)
- Usage TLS 1.3 / TLSH

Authentification post-quantique

Client (identité + clé PQC) ⟷ **Authentification** mutuelle (mTLS PQC) ⟷ **Serveur** (certificat hybride)

- Échange de clé hybride (EC classique + ML-KEM)
- Signature PQC (ML-DSA / Dilithium)

Points clés

- Chaîne de certificats hybride (Classique + PQC).
- Authentification mutuelle (mTLS) avec certificats PQC.
- Algorithmes NIST (ex. ML-KEM, ML-DSA / Dilithium).
- Compatibilité TLS 1.3 et TLSH.
- Gestion du cycle de vie : émission, renouvellement, révocation (CRL/OCSP).
- Sécurité et interopérabilité maintenues.

Algorithmes et profils recommandés

Usage	Algorithme (PQC)	Rôle
Échange de clé	ML-KEM (ex. ML-KEM-768)	Accord de secret (KEM)
Signature / Certificat	ML-DSA (ex. Dilithium)	Signature numérique (authentification)
Profil hybride	Classique + PQC	Compatibilité et transition (TLS 1.3 / TLSH)

Conclusion

Les certificats et l'authentification post-quantique fournissent une **identité forte et durable**, en combinant les mécanismes classiques et PQC pour une sécurité robuste et prête pour l'ère post-quantique.

9.1. Première étape

La V1 initiale associe un échange hybride à une authentification classique correctement vérifiée.

La gestion des identités comprendra la chaîne de confiance, le nom attendu, les usages du certificat, sa validité, la correspondance certificat/clé et une politique de révocation documentée. Le mTLS sera ajouté pour les profils exigeant l'authentification du client.

9.2. Trois niveaux distincts pour ML-DSA

Niveau	Preuve attendue
Primitive	Génération, signature et vérification réussies
Authentification TLS	Négociation et `CertificateVerify` fonctionnels
Infrastructure de confiance	Certificats, chaîne, rotation et clients qualifiés

QSA-REF couvre déjà le premier niveau. Les deux suivants restent à intégrer.

Au 18 novembre 2026, le document IETF consulté sur ML-DSA dans TLS est en file de publication RFC ; il apparaît encore comme Internet-Draft. La réalisation devra donc figer précisément les versions et formats utilisés. Use of ML-DSA i...

9.3. Position dans le développement

La faisabilité sera examinée dès les lots natifs et certificats. Sa qualification complète restera un objectif identifié du programme, sans bloquer le profil initial à authentification classique.

La présence de ML-DSA dans QSA ne sera jamais utilisée seule pour annoncer une authentification TLS entièrement post-quantique.

Chapitre 10 — Le moteur de politiques et l’adaptation IoT

Chapitre 10 — Le moteur de politiques et l'adaptation IoT

Objectif du chapitre

Mettre en place un moteur de politiques flexible, permettant de définir, appliquer et adapter les profils cryptographiques (TLSH / QSA) à tous les environnements, y compris les **objets connectés (IoT)**.



Moteur de politiques (Policy Engine)



Points clés

- Définition centralisée des politiques.
- Sélection des algorithmes (ML-KEM, ML-DSA, AES, ...).
- Adaptation selon le contexte (serveur, client, IoT).
- Profils optimisés pour les environnements contraints (IoT / embarqué).
- Application automatique et sans interruption.
- Traçabilité et audit des politiques.

Adaptation aux objets connectés (IoT)



- Profils cryptographiques légers et sûrs.
- Consommation CPU / mémoire réduite.
- Handshake rapide et fiable.
- Compatibilité avec les capacités matérielles limitées (MCU, SoC, edge).



Profils de politiques (exemples)

Profil	Usage	Algorithmes (ex.)
Strict	Serveurs critiques	ML-KEM + ML-DSA + AES-256
Standard	Usage général (TLSH)	ML-KEM + AES-256-GCM
IoT	Objets connectés / embarqué	ML-KEM (léger) + AES-128

Conclusion

Le moteur de politiques **unifie la sécurité, la flexibilité et la compatibilité**, en permettant une adaptation immédiate de TLSH / QSA à tous les environnements, du datacenter aux objets connectés.

10.1. Séparer les obligations des préférences

Les obligations déterminent les profils autorisés, l'authentification, les destinations et le niveau minimal requis.

Les préférences d'optimisation peuvent tenir compte de la batterie, du réseau, des capacités du terminal et de la charge serveur.

Une batterie faible ne doit jamais autoriser un profil interdit par la politique du service.

10.2. Une décision fondée sur des profils complets

Le moteur sélectionnera un ensemble cohérent : groupe, authentification, politique de session et limites applicables.

Il ne modifiera pas le nombre de tours d'un algorithme. Les termes « Confidentiel », « Secret » et « Top Secret » pourront rester des étiquettes internes, mais devront être reliés à des règles précises ; ils ne valent pas homologation pour des données classifiées.

10.3. Signalisation du terminal

Le lot IoT devra livrer un client ou SDK de référence, un format versionné, des tailles maximales, des règles de fraîcheur et les comportements en cas de donnée absente ou invalide.

La première intégration transmettra cette télémétrie dans le canal déjà établi. Elle pourra guider les décisions ultérieures.

L'extension éventuelle du `ClientHello` restera un chantier identifié : format, confidentialité, prise en charge par les deux extrémités et interaction avec la négociation. Le plan le distingue déjà de la simple interface cryptographique.

10.4. Validation métier

Les scénarios couvriront le capteur énergétique, la chaîne du froid et un usage à forte criticité. Ils mesureront les effets sur une transaction complète.

Pour chaque essai, le rapport précisera aussi la liaison protégée : capteur–serveur ou passerelle–serveur. Cette différence est essentielle pour les réseaux IoT qui n'exposent pas directement un transport TCP au terminal.

Chapitre 11 — Les sessions, les tickets et les preuves ZKP

Chapitre 11 — Les sessions, les tickets et les preuves ZKP

Gérer la continuité des connexions, l'authentification des échanges et la vérifiabilité sans divulgation grâce aux ZKP.

Objectif du chapitre

Définir le mécanisme de session sécurisé, les tickets d'accès et l'intégration des preuves à divulgation nulle **(ZKP)** pour prouver une authentification ou un droit sans révéler les données sensibles.



Cycle complet : session, ticket et ZKP



Principes des ZKP dans TLSH



- Prouver une identité, un droit ou un état (ex. : « autorisé », « valide ») sans révéler l'information sous-jacente.
- Intégration avec les flux TLSH / QSA.
- Vérification rapide côté serveur.
- Usages : authentification, conformité de profil, validité de ticket, etc.

Points clés

- Sessions chiffrées et maîtrisées (création, renouvellement, expiration).
- Tickets signés et bornés (durée, périmètre, droits).
- Preuves ZKP intégrées et performantes.
- Aucune divulgation de données sensibles.
- Compatibilité TLS 1.3, TLSH et QSA.
- Journaux et traçabilité des sessions et des preuves.

Gestion des sessions

- Création de session (authentifiée, TLSH)
- Renouvellement / prolongation (rolling)
- Expiration et révocation (contrôlée)
- Protection anti-rejeu et anti-détournement

Tickets d'accès

- Ticket signé (clé hybride : classique + PQC)
- Contenu : identité, rôle, droits, durée, scope
- Validation rapide et stateless (si nécessaire)
- Révocation et liste de tickets invalides (RBL)

Preuves ZKP (cas d'usage)

Authentification	Prouver l'identité sans révéler la clé ou le secret.
Autorisation	Prouver un droit d'accès (ex. : profil, niveau).
Conformité	Prouver un état (ex. : certificat valide, policy respectée) sans divulgation.

Conclusion

L'intégration conjointe des sessions, des tickets et des ZKP apporte une **authentification forte,** une gestion fine des droits et une vérifiabilité sans divulgation, parfaitement adaptée à TLSH et QSA.

11.1. Première démonstration et reprise classique

La première démonstration fonctionne sans reprise ni early data. La reprise sera ensuite qualifiée avec :

- Un nouvel échange hybride.
- Une durée de validité.
- Une association à l'identité, au service et au profil.
- Une invalidation lors des changements incompatibles.
- Un comportement défini après rotation et redémarrage.

`KeyUpdate` renouvelle des clés de trafic ; il ne remplace pas un nouvel échange ML-KEM ni une renégociation libre du groupe. datatracker.ietf.org

11.2. Le problème non résolu dans le scénario ZKP

Plusieurs diapositives décrivent une preuve créée avec le ticket puis représentée lors d'une reprise.

La question à résoudre est précise : **qu'est-ce qui empêche de représenter l'ensemble ticket et preuve déjà enregistré ?**

La validité de la preuve et l'unicité de son utilisation sont deux propriétés différentes. La spécification devra définir le secret prouvé, le contexte, la requête concernée et le mécanisme de consommation.

11.3. Le travail nécessaire

Le chantier comprendra :

- Choix d'un schéma pertinent face à la menace quantique.
- Liaison entre preuve, ticket, service et opération.
- Consommation atomique.
- Coordination entre serveurs.
- Gestion des pannes, partitions et redémarrages.
- Analyse des coûts côté terminal.
- Revue indépendante du protocole.

TLS 1.3 n'apporte pas une protection générale contre le rejeu des données 0-RTT. Le protocole applicatif doit définir leur usage et leur traitement. Une reprise avec échange hybride ne donne pas automatiquement aux données précoces les propriétés du nouveau secret établi ensuite. datatracker.ietf.org

Le volet ZKP reste dans le programme complet, avec une phase de recherche explicite. Son activation ne peut pas être assimilée à l'ajout d'un simple paramètre de configuration.

Chapitre 12 — OPT_TLSH et les performances

Chapitre 12 — OPT_TLSH et les performances

OPT_TLSH regroupe les optimisations de performance pour TLSH / QSA, sans compromis sur la sécurité.

Objectif du chapitre

Présenter OPT_TLSH, le moteur d'optimisations qui améliore la vitesse, la scalabilité et l'efficacité de TLSH / QSA, tout en préservant le niveau de sécurité.



Architecture d'OPT_TLSH



Points clés

- Moteur d'optimisations dédié (OPT_TLSH).
- Accélération du handshake hybride (TLS 1.3 + PQC).
- Parallélisation et exploitation multi-cœur.
- Réduction de la latence et du temps de connexion.
- Scalabilité pour charges élevées (serveurs, SOC, IoT).
- Optimisations sans affaiblir la sécurité.

Gains de performances (référence – à valider)

Opération	TLS 1.3 (réf.)	TLSH (sans OPT)	TLSH + OPT_TLSH	Gain estimé
Handshake complet	~ 120 ms	~ 145 ms	~ 48 ms	≈ –67 %
Échange de clé (PQC)	~ 85 ms	~ 110 ms	~ 36 ms	≈ –67 %
Débit (1 connexion)	~ 1,0 Gbps	~ 0,9 Gbps	~ 2,4 Gbps	≈ × 2,6
Connexions simultanées	≈ 1 000	≈ 1 000	≥ 10 000	≈ × 10

Modes et cas d'usage

- **Standard** — Usage général, équilibre sécurité / performance
- **Haute vitesse** — Environnements à forte charge (serveur, cloud, SOC)
- **IoT / Embarqué** — Profil léger, faible empreinte (Edge, IoT, industriels)

Conclusion

OPT_TLSH apporte des **gains de performance majeurs** (**–60 à –70 %** de latence, débit ×2 à ×3, scalabilité ×10), tout en maintenant le niveau de sécurité de TLSH / QSA.

12.1. Ce que l'optimiseur doit traiter

Le plan retient cinq axes concrets :

Axe	Travail attendu
Admission	Réguler les nouvelles négociations
Ordonnancement	Répartir le travail sans monopolisation
Files	Borner capacités et délais
Buffers	Réduire allocations et copies
Écritures	Regrouper certaines opérations avec attente maximale

OPT conservera exactement la politique cryptographique et les vérifications d'identité.

La compression éventuelle de données métier relève d'un contrat applicatif séparé. Elle ne doit pas devenir une modification implicite des records ou des messages cryptographiques.

12.2. Conserver les comparaisons B0, B1 et B2

Configuration	Contenu	Ce qu'elle mesure
B0	Service rustls, fournisseur standard, OPT désactivé	Référence hybride
B1	Même service, moteur QSA sélectionné, OPT désactivé	Coût ou apport de l'intégration QSA
B2	Même configuration que B1, OPT activé	Effet propre d'OPT

Le moteur exact devra toujours figurer dans le rapport. **Un résultat B1 avec QSA-REF ne démontre pas les performances du moteur de Christophe.**

12.3. Protocole de mesure

Je conserve la matrice du plan :

- RTT de 1, 20 et 100 ms.
- Pertes de 0 %, 0,1 % et 1 %.
- MTU de 1 280 et 1 500 octets.
- Échanges courts et transferts longs.
- Au moins cinq répétitions des points retenus.

Les métriques comprendront p50/p95/p99, débit, CPU, mémoire, octets échangés et erreurs.

Chapitre 13 — Installation, exploitation et administration

Chapitre 13 — Installation, exploitation et administration

Déployer, opérer et piloter TLSH / QSA dans la durée.



Un socle exploitable, administrable et durable.

13.1. Cible de référence

Le plan QSA-REF retient **Linux et Python 3.12**. Le plan TLSH propose Linux x86-64, initialement Ubuntu 24.04 LTS.

Ton poste Windows peut servir au développement et au pilotage ; cela ne doit pas être confondu avec une livraison Windows native déjà qualifiée.

13.2. Un installateur intégré

Je conserve les deux actions principales :

- `check` : diagnostic sans modification.
- `repair` : installation ou correction contrôlée.

L'outil devra prendre en charge les dépendances prévues, l'environnement dédié, la compilation, les permissions, la configuration, le service systemd et les contrôles de santé.

Il préservera les configurations et certificats valides lors des réexécutions.

13.3. Identifier les bibliothèques réellement utilisées

Le diagnostic distinguera :

1. OpenSSL embarqué dans `cryptography`.
2. OpenSSL utilisé par le module Python `ssl`.
3. OpenSSL dédié au parcours natif QSA.

La version affichée par l'exécutable système ne sera pas utilisée comme preuve de ce que charge un autre processus.

13.4. Mise à jour et retour arrière

Une livraison préparée séparément sera contrôlée avant activation. Le retour arrière portera sur un ensemble cohérent : binaire, bibliothèques, configuration compatible et unités de service.

L'administration restera locale et protégée. Elle exposera l'état, les profils, les certificats, le fournisseur, les connexions, les erreurs et les résultats des contrôles.

La remontée vers WSA pourra être ajoutée en lecture seule, sans dépendance de disponibilité du service TLS envers WSA.

Chapitre 14 — La stratégie de qualification

Chapitre 14 — La stratégie de qualification

Objectif du chapitre

Définir la stratégie et le plan de qualification complet de **TLSH / QSA**.



Processus de qualification (cycle global)

1. **Préparation** (environnement)
2. **Tests** (fonctionnels & sécurité)
3. **Performances** (charge, latence)
4. **Validation** (conformité)
5. **Qualification** (finale)

Itératif et incrémental

Points clés

- Couverture complète (fonctionnel, sécurité, performance).
- Conformité aux standards (NIST, TLS 1.3, PQC).
- Mesures et seuils de validation.
- Traçabilité et rapports.
- Go / No-Go formalisé.

Périmètre de qualification

- **Sécurité** (crypto, KEM, ZKP)
- **Interopérabilité** (TLS 1.3, (clients/serveurs)
- **Performances** (débit, latence, scalabilité)
- **Usages** (datatacenter, cloud, IoT)

Environnements de test

Client(s) → **Banc de test TLSH / QSA** ↔ **Serveur(s)**

Livrables

- **Plan de tests**
- **Rapports de résultats**
- **Attestation de qualification**

Conclusion

Une qualification structurée, **mesurable et traçable**, garantissant la robustesse, la conformité et la **maturité** opérationnelle de TLSH / QSA.



14.1. Quatre niveaux de preuve

Niveau	Question
Primitives	Les opérations produisent-elles les résultats attendus ?
Interface	Les formats, erreurs et durées de vie sont-ils corrects ?
TLS	La négociation et l'authentification fonctionnent-elles avec d'autres piles ?
Service	L'ensemble reste-t-il exploitable sous charge et après incident ?

QSA-REF apporte déjà des preuves au premier niveau et sur son contrat Python. Les niveaux suivants nécessitent leurs propres campagnes.

14.2. Interopérabilité

Les échanges devront couvrir rustls standard, OpenSSL, TLSH avec QSA-REF et, lorsqu'il sera disponible, TLSH avec le moteur de Christophe.

Il faudra distinguer deux choses : une seconde intégration du protocole et une implémentation cryptographique indépendante. Deux parcours reposant finalement sur le même OpenSSL EVP ne constituent pas deux implémentations indépendantes des primitives.

14.3. Robustesse

Les essais couvriront notamment les certificats incorrects, groupes incompatibles, messages tronqués, clients lents, annulations, saturations, indisponibilité du backend, erreurs d'aléa et défaillances de l'application.

Les sanitizers et le fuzzing entreront dès le lot natif. Ils ne seront pas repoussés à la dernière phase.

14.4. Endurance et preuves conservées

Je conserve les objectifs proposés par le plan QSA-REF : **10 000 connexions complètes**, puis **une heure à concurrence contrôlée**.

Chaque résultat devra identifier la version, le moteur, le profil, l'environnement, le scénario et les limites. Les vecteurs seront figés avec leur provenance et leur empreinte.

Le dossier final associera chaque exigence à un composant, un scénario et une preuve consultable.

Chapitre 15 — Conformité, intégration du moteur officiel et valorisation

Chapitre 15 — Conformité, intégration du moteur officiel et valorisation

Objectif du chapitre

Assurer la conformité, intégrer le moteur officiel TLSH / QSA et définir la valorisation du projet.



Conformité & standards

- NIST (PQC)
- ISO 27001
- RGPD (UE)
- TLS 1.3 / TLSH
- Post-Quantique
- Audits & traçabilité

Intégration du moteur officiel (TLSH / QSA)

Sources officielles → **TLSH / QSA** Moteur officiel → Intégration & API

- Performance (OPT_TLSH)
- Sécurité renforcée
- Portabilité (Cloud / On-Prem / IoT)

Valorisation

- Solution différenciante (PQC natif).
- Marchés cibles : entreprises, cloud, IoT, défense.
- Atout de souveraineté et de confiance.
- Potentiel licensing / intégration OEM.
- Projection de croissance (SaaS, SDK, services).



Écosystème & déploiements

Datacenter / On-Prem | Cloud (Hybride) | IoT & Edge | Conteneurs (Docker / K8s) | SDK & API

Roadmap de valorisation

PoC Validée → Intégration Officielle → Pilotes & Partenariats → Industrialisation & Go-to-Market

Conclusion

Une solution **conforme** aux standards, intégrant le **moteur officiel** TLSH / QSA, prête pour un déploiement industriel et une forte **valorisation marché**.

15.1. Portée des analyses de conformité

Les analyses QSA_TLSH, QSA_NIST et QSA_XXL constituent des documents de positionnement. Elles ne remplacent pas une validation d'implémentation ou de module.

Il faut distinguer :

- Algorithme normalisé.
- Implémentation testée.
- Module validé dans un environnement défini.
- Produit TLS correctement intégré.

Le CMVP porte sur des modules et des conditions d'utilisation précises. CSRC

15.2. Corrections déterminantes

L'échéance 2035 ne suffit pas à conclure que l'hybridité entière devient interdite. NIST IR 8547 précise qu'une construction composite approuvée peut être validable lorsque l'une de ses entrées provient d'un mécanisme approuvé, même si d'autres entrées sont non autorisées. L'agilité vers d'autres profils reste pertinente, mais ne doit pas reposer sur cette lecture simplifiée. nvlpubs.nist.gov

Le certificat OpenSSL **#4985** ne suffit pas non plus à établir la validation de toute version OpenSSL, de ML-KEM/ML-DSA ou de notre provider QSA. CSRC

Enfin, ML-DSA relève de **FIPS 204**, et ML-KEM de **FIPS 203** ; les références contradictoires devront être corrigées. CSRC +1

15.3. Réception du moteur de Christophe

Son intégration exigera les sources ou bibliothèques convenues, headers, instructions de compilation, dépendances, formats, règles mémoire, tests et droits d'utilisation.

La bascule sera explicite et appliquée aux nouvelles connexions après réussite du contrat et des échanges croisés. Une indisponibilité du moteur requis produira un refus identifiable.

15.4. Valeur à démontrer

Le dossier de valorisation devra relier les revendications aux résultats : apport du moteur QSA, gains d'OPT, facilité d'exploitation, adaptation IoT et coûts énergétiques.

Les droits sur les composants et contributions devront être documentés. La présence de mentions de brevets dans une description ne constitue pas, à elle seule, une preuve de protection juridique ni de valeur commerciale.

Deuxième partie — Roadmap consolidée des Mega Batches



AGENDA DES MEGA BATCH DE PATCHS

- Organisation générale
- TLSH-MB-001 — Reprise du socle et première connexion hybride
- TLSH-MB-002 — Contrat natif et raccordement effectif de QSA
- TLSH-MB-003 — Profils, certificats et exploitation
- TLSH-MB-004 — OPT_TLSH et mesures comparatives
- TLSH-MB-005 — Qualification complète de la V1 et pilote
- TLSH-MB-006 — Client IoT et adaptation énergétique
- TLSH-MB-007 — Authentification post-quantique et reprise avancée

Organisation générale

Je conserve les numéros TLSH-MB-001 à TLSH-MB-006. Les travaux QSA-REF sont rattachés à ces lots pour éviter deux développements parallèles qui divergeraient.

Mega Batch	Résultat principal	Dépendance majeure
TLSH-MB-001	Socle exécutable, installation et connexion hybride B0	Sources TLSH existantes à rapprocher
TLSH-MB-002	Contrat natif et intégration QSA vérifiable	MB-001 et QSA-REF consolidé
TLSH-MB-003	Service administrable et cycle de vie complet	Socle MB-001 ; raccordement final MB-002
TLSH-MB-004	OPT_TLSH et comparaisons B0/B1/B2	MB-002 et MB-003
TLSH-MB-005	Qualification de la V1 et pilote	MB-001 à MB-004
TLSH-MB-006	Adaptation IoT et mesures énergétiques	Socle qualifié et terminal représentatif
TLSH-MB-007	Authentification PQ complète et recherche ZKP/0-RTT	Faisabilité engagée en amont ; activation après preuves

TLSH-MB-001 — Reprise du socle et première connexion hybride

Objectif

Obtenir une installation reproductible et une transaction applicative complète avec le fournisseur standard de rustls.

Travaux regroupés

- Reprise des éventuelles sources TLSH déjà écrites.
- Consolidation des modules QSA-REF sans changement arbitraire de noms.
- Correction des erreurs de configuration et de consommation des contextes.
- Complétude du paquet de dépendances.
- Structure Rust/Python et versions figées.
- Service `qsa_tlsd`, client de validation et application HTTP de test.
- Profil strict, certificats de laboratoire et limites de ressources.
- Installateur `check` / `repair`, service systemd et diagnostics.
- Premières mesures B0.

Critères de sortie

L'installation fonctionne, sa réexécution préserve l'état valide, la requête atteint uniquement l'application autorisée et la réponse revient correctement. Un certificat incorrect et un client sans groupe hybride compatible sont refusés.

Ce lot ne dépend pas du paquet de Christophe. Son résultat reste identifié comme fournisseur de référence.

TLSH-MB-002 — Contrat natif et raccordement effectif de QSA

Objectif

Faire exécuter les opérations QSA pendant un véritable handshake TLSH.

Travaux regroupés

- Contrat `qsa.h` versionné.
- Bibliothèque `libqsa_ref` native fondée sur les primitives existantes.
- `qsa_ffi` et adaptateur rustls.
- Provider OpenSSL pour le banc QSA-REF.
- Contextes OpenSSL distincts pour éviter la récursion.
- Échanges croisés Python/C.
- Gestion des erreurs, annulations et durées de vie.
- Tests mémoire et concurrence.
- Compteurs et injection contrôlée de refus.
- Comparaisons B1/B0.
- Étude initiale du support des signatures TLS ML-DSA.

Critères de sortie

Une transaction TLSH utilise effectivement QSA-REF natif. Les tests prouvent le moteur appelé, l'absence de contournement et l'interopérabilité avec les contreparties retenues.

Intégration du moteur de Christophe

La recette conservera deux lignes distinctes :

Critères de sortie

Une transaction TLSH utilise effectivement QSA-REF natif. Les tests prouvent le moteur appelé, l'absence de contournement et l'interopérabilité avec les contreparties retenues.

Intégration du moteur de Christophe

La recette conservera deux lignes distinctes :

Livraison	Condition de clôture
Raccordement QSA-REF	Contrat, adaptateurs et TLS réel validés
Raccordement QSA de Christophe	Paquet reçu, adapté et qualifié avec la même campagne

Ainsi, nous pouvons achever l'intégration de référence sans annoncer à tort que le moteur de Christophe est déjà intégré.

TLSH-MB-003 — Profils, certificats et exploitation

Objectif

Transformer le démonstrateur en service administrable pour une application réelle.

Travaux regroupés

- Profils versionnés et inventaire des capacités.
- Routes SNI/ALPN autorisées.
- Certificats serveur et mTLS selon profil.
- Rotation, échéances et rechargement cohérent.
- Connexions persistantes, arrêts contrôlés et fermetures propres.
- Tickets, expiration et invalidation.
- Reprise avec nouvel échange hybride après validation.
- Administration locale et journaux structurés.
- Mise à jour, contrôle de santé et retour arrière.
- Export facultatif de supervision vers WSA.

Critères de sortie

Rotation et mise à jour fonctionnent sans état partiellement appliqué. Une configuration invalide est refusée proprement. Un échec de mise à jour restaure une version cohérente. Les reprises respectent le profil et les changements de politique.

Le 0-RTT reste désactivé dans cette livraison.

TLSH-MB-004 — OPT_TLSH et mesures comparatives

Objectif

Développer l'optimiseur et déterminer dans quelles conditions il apporte un bénéfice.

Travaux regroupés

- Admission des négociations.
- Répartition du travail.
- Files et délais bornés.
- Réduction des copies et allocations.
- Regroupement contrôlé des écritures.
- Adaptation à la charge.
- Activation et désactivation observables.
- Automatisation des campagnes B0/B1/B2.
- Mesures réseau, CPU, mémoire, latence et erreurs.

Critères de sortie

Un rapport reproductible présente les gains, les surcoûts et les limites, à politique cryptographique identique.

L'activation par défaut d'une optimisation dépendra des résultats. Une baisse du niveau de protection ou une différence de matériel ne sera pas comptée comme gain d'OPT.

Ce lot peut progresser avec QSA-REF lorsque le moteur de Christophe n'est pas encore disponible.

TLSH-MB-005 — Qualification complète de la V1 et pilote

Objectif

Produire une version candidate exploitable et son dossier de preuves.

Travaux regroupés

- Interopérabilité complète.
- Campagnes négatives et de saturation.
- Endurance de 10 000 connexions puis une heure contrôlée.
- Fuzzing et contrôles mémoire approfondis.
- Vérification des certificats, sessions, mises à jour et restaurations.
- Analyse des fuites possibles et revue des parties sensibles.
- Consolidation de la nomenclature des dépendances et licences.
- Guides d'installation, exploitation et diagnostic.
- Préparation du périmètre d'évaluation externe.
- Pilote sur une application et une population de clients définies.

Critères de sortie

Les résultats sont reproductibles, les anomalies bloquantes sont corrigées, les limites sont documentées et le retour arrière est démontré.

La fin de ce lot correspond à une V1 qualifiée dans son périmètre ; elle ne constitue pas une certification FIPS ni l'achèvement du volet IoT/ZKP.

TLSH-MB-006 — Client IoT et adaptation énergétique

Objectif

Réaliser les fonctions adaptatives originales sur un équipement représentatif.

Travaux regroupés

- Client ou SDK de référence.
- Collecte de la batterie et des indicateurs réseau.
- Format de télémétrie versionné.
- Association terminal–identité–service–politique.
- Catalogue de profils admissibles.
- Moteur de décision explicable.
- Gestion des informations absentes, anciennes ou mensongères.
- Scénarios de batterie basse et de réseau dégradé.
- Mesure de l'énergie, de l'activité radio et des reconnexions.
- Étude de l'extension du handshake si son utilité est démontrée.

Critères de sortie

Le même client est testé sous plusieurs contextes. Le serveur respecte le plancher de sécurité, explique ses décisions et produit des mesures comparables sur la transaction complète.

Une fausse déclaration de batterie faible ne permet pas d'obtenir un profil interdit.

Le rapport distingue clairement mesures matérielles, estimations et gains côté serveur.

TLSH-MB-007 — Authentification post-quantique et reprise avancée

Objectif

Couvrir les fonctions du programme complet qui dépassent la V1.

Ce lot regroupe des travaux dont les critères restent distincts.

Authentification post-quantique

Intégrer ML-DSA dans le handshake, les certificats et les chaînes de confiance retenues ; vérifier les clients, les formats, les erreurs, les rotations et l'interopérabilité.

Tickets et preuves ZKP

Spécifier la propriété recherchée, choisir le schéma, construire le prototype et traiter la liaison au contexte ainsi que la consommation unique.

Usage du 0-RTT

Définir les opérations applicatives admissibles, l'anti-rejeu, les comportements après panne et les conditions de désactivation. Les coûts de preuve côté client et serveur seront mesurés.

Critères de sortie

Chaque fonction dispose d'une spécification, d'un prototype évalué, de résultats reproductibles et d'une revue appropriée.



Troisième partie — Traçabilité des exigences et dépendances

Exigences
(métier, tech., sécu.)

→

Matrice de traçabilité
(Exigences ↔ Solutions)

→

Dépendances
(modules, services, versions, tiers)

→

Analyse d'impact
(ajout, modification, à risque)

→

Contrôle & suivi
(couverture, avancement, conformité)

Rattachement des lots QSA-REF

Plan QSA-REF existant	Intégration dans la roadmap TLSH
QSA-REF-001 — Référence Python	Consolidation dans MB-001, puis non-régression continue
QSA-REF-002 — Natif et provider	MB-002
QSA-REF-003 — Qualification et exploitation	MB-003 à MB-005 ; authentification PQ approfondie dans MB-007
QSA-REF-004 — Moteur de Christophe	Jalon conditionné à sa réception, utilisant le contrat et les recettes de MB-002/005

Cela conserve les livrables QSA déjà prévus sans créer une deuxième architecture concurrente.

Couverture de l’analyse fonctionnelle initiale

Exigence	Livraison principale
EF-01 — Signalisation client	MB-006
EF-02 — Décision adaptative	MB-003 pour les règles ; MB-006 pour le terminal
EF-03 — Multi-algorithmes	MB-002/003, puis extensions MB-007
EF-04 — Intégration QSA	MB-002
EF-05 — Tickets ZKP	MB-007
EF-06 — Reprise 0-RTT avancée	MB-007
EF-07 — Niveaux de sécurité	MB-003/006
EF-08 — Optimisation énergétique	MB-004 pour les ressources ; MB-006 pour le terminal
EF-09 — Extensibilité	Contrat et architecture MB-001/002
EF-10 — Compatibilité TLS	MB-001/002, consolidée dans MB-005

Ordre de reprise

La progression principale reste **MB-001, MB-002, MB-003, MB-004, puis MB-005**. La préparation de l'exploitation peut avancer pendant le travail natif. Le volet IoT et la recherche avancée ont ensuite leurs propres conditions de validation.

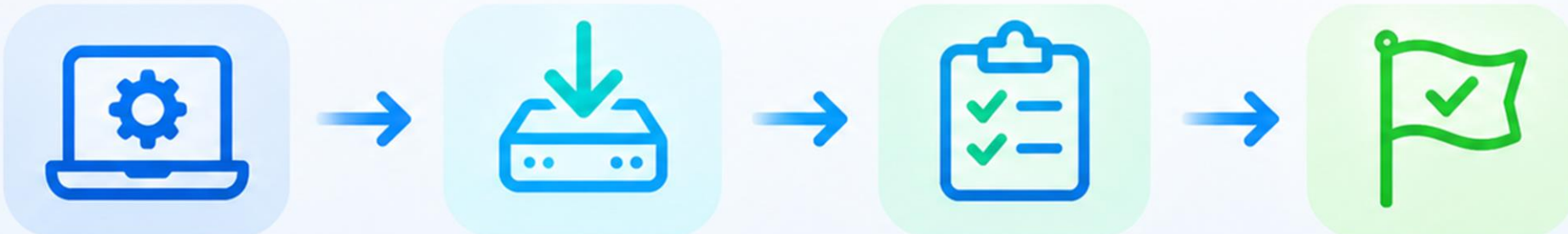
Le risque technique principal est concentré dans **MB-002** : frontière C/Rust, consommation des secrets, deux adaptateurs et preuve du moteur exécuté. Le risque de recherche se situe dans **MB-007** : les documents expriment l'objectif ZKP/0-RTT, mais ne définissent pas encore un protocole permettant d'en garantir les propriétés.

Le prochain lot concret est donc la reprise de TLSH-MB-001, avec réutilisation du QSA-REF fourni et rapprochement des éventuelles sources Rust déjà réalisées. Les éléments encore absents de ce dossier sont les sources TLSH concernées, le fichier de dépendances QSA-REF attendu et, pour l'intégration ultérieure, le paquet natif de Christophe.

La première démonstration à obtenir sera précise : **une transaction applicative complète, un groupe hybride effectivement négocié, un certificat vérifié, des refus contrôlés et une installation reproductible.**



Plan de Test & Installation



```bash
cd /home/gon2000fr
python3 -m zipfile -e TLSH_MB001.zip tlsh_mb001
cd tlsh_mb001
python3 qsa_tooling/tlsh.py check
python3 qsa_tooling/tlsh.py repair
```

Comment tester

```bash
python3 qsa_tooling/tlsh.py recette
python3 qsa_tooling/tlsh.py mesure
```

```
TLSH - diagnostic (aucune modification)
---------------------------------------
 [OK] systeme : Linux x86_64
 [OK] Python 3.12.3
 [OK] compilateur C : /usr/bin/cc
 [OK] Rust : 1.100.0
 [OK] outil openssl : 3.0.13
 [OK] sources : 20 fichiers conformes au manifeste
 [OK] binaires compiles et a jour
 [OK] certificats de laboratoire valides (echeance du serveur : Feb 16 18:08:18 2027 GMT)
 [OK] configuration valide
 [INFO] service de laboratoire : arrete
 [INFO] contrepartie independante pour l'interoperabilite : aucune (OpenSSL 3.5+ ou Go 1.24+
        absents)

RESULTAT : environnement conforme.
```

```
TLSH - installation systeme (compte dedie, systemd)
---------------------------------------------------
Rien a changer : l'installation etait deja conforme.

TLSH - diagnostic de l'installation systeme
-------------------------------------------
 [OK] compte Â« tlsh Â» : uid 995, sans shell
 [OK] /opt/tlsh/bin/qsa_tlsd : droits 755
 [OK] service installe identique au binaire compile
 [OK] /etc/tlsh/tlshd.toml : droits 640
 [OK] /etc/tlsh/certs/serveur.cle : droits 400
 [OK] certificats installes valides (echeance : Feb 16 18:08:18 2027 GMT)
 [OK] tlsh-app-test.service : enabled, active
 [OK] tlsh.service : enabled, active
 [OK] processus 2703188 : compte tlsh, aucune capacite, pas de gain de privilege, appels
      systeme filtres
 [OK] sante : connexion X25519MLKEM768 (0x11EC) reussie en 0.6 ms

RESULTAT : service installe, actif et en bonne sante.
```

```
ubuntu@ip-172-31-24-146:/home/gon2000fr/tlsh_mb001$ python3 qsa_tooling/tlsh.py recette

TLSH - recette du jalon TLSH-MB-001 (connexions TLS reelles)
------------------------------------------------------------
 [OK] R01 configuration verifiee avant demarrage - valide acceptee ; refusees : destination
      distante, profil classique, cle inconnue
 [OK] R02 connexion TLS 1.3 hybride etablie par le client de validation - TLS 1.3, groupe
      X25519MLKEM768 (0x11EC), negociation 0.5 ms, 1504 octets emis et 1933 recus
 [OK] R03 la requete atteint l'application autorisee, la reponse revient - l'application a
      recu la mesure ; accuse de reception verifie par le client
 [OK] R04 le client ne peut pas choisir la destination - une requete visant une autre
      destination arrive quand meme a l'application autorisee, et nulle part ailleurs
 [OK] R05 client sans groupe hybride (X25519, P-256) refuse - alerte TLS recue par le client ;
      cause journalisee par le service
 [OK] R06 client classique independant (openssl, TLS 1.3) refuse - refuse par une alerte TLS ;
      cause journalisee
 [OK] R07 client TLS 1.2 (openssl) refuse - refuse par une alerte TLS ; cause journalisee
 [OK] R08 nom de serveur (SNI) inconnu refuse - nom de serveur non autorise : refuse et
      journalise
 [OK] R09 certificat d'une autorite inconnue refuse par le client - le client refuse un
      serveur dont il ne connait pas l'autorite
 [OK] R10 certificat emis pour un autre nom refuse par le client - refuse par le client
      (NotValidForName)
 [OK] R11 certificat expire refuse par le client - refuse par le client (Expired)
 [OK] R12 saturation : rejet controle au-dela du nombre maximal de connexions - 4e connexion
      rejetee (maximum 3), puis service de nouveau disponible
 [OK] R13 delai maximal de negociation applique - connexion muette fermee apres 1.5 s (delai
      configure : 1,5 s)
 [OK] R14 message invalide rejete sans perturber le service - message non TLS rejete ; le
      service continue de fonctionner
 [OK] R15 application arretee : diagnostic clair, puis reprise - erreur 502 explicite pour le
      client, cause journalisee, puis reprise
 [OK] R16 etat d'administration local lisible et coherent - 4 negociations reussies, toutes en
      X25519MLKEM768 ; 3 refus de profil ; socket en acces restreint (660)
 [OK] R17 journaux et etat sans secret - 4 fichiers de journal examines : aucune cle, lignes
      toutes en JSON
 [NON EXECUTE] R18 interoperabilite avec une implementation independante - aucune pile TLS
               tierce sachant faire X25519MLKEM768 sur cette machine (OpenSSL 3.5+ ou Go
               1.24+). Ce controle n'est PAS reussi : il reste a faire.
 [OK] R19 arret propre du service - arret sur SIGTERM, code 0, socket d'administration retiree

Rapport : rapports/recette_mb001_20261118T180847Z.json
Non executes (1) : R18 - voir le detail ci-dessus.
RESULTAT : 18 controles reussis sur 18 executes.
ubuntu@ip-172-31-24-146:/home/gon2000fr/tlsh_mb001$
```

Comment l'installer

Copiez `TLSH_MB001_B.zip` dans `/home/gon2000fr`, puis :

```bash
cd /home/gon2000fr
python3 -m zipfile -e TLSH_MB001_B.zip tlsh_mb001
cd tlsh_mb001
python3 qsa_tooling/tlsh.py check
sudo python3 qsa_tooling/tlsh.py installer
```

L'archive se décompresse par-dessus le dossier existant. Le code Rust n'a pas changé : `check` doit répondre `environnement conforme` sans rien recompiler. `installer` doit finir par `RESULTAT : service installe, actif et en bonne sante.`

Si le port 8443 ou 8080 est déjà pris sur votre serveur, `installer` le dit et propose l'option pour en changer.

```
TLSH - diagnostic (aucune modification)
---------------------------------------
 [OK] systeme : Linux x86_64
 [OK] Python 3.12.3
 [OK] compilateur C : /usr/bin/cc
 [OK] Rust : 1.100.0
 [OK] outil openssl : 3.0.13
 [OK] sources : 20 fichiers conformes au manifeste
 [OK] binaires compiles et a jour
 [OK] certificats de laboratoire valides (echeance du serveur : Feb 16 18:08:18 2027 GMT)
 [OK] configuration valide
 [INFO] service de laboratoire : arrete
 [INFO] contrepartie independante pour l'interoperabilite : aucune (OpenSSL 3.5+ ou Go 1.24+
        absents)

RESULTAT : environnement conforme.
```

```
TLSH - installation systeme (compte dedie, systemd)
-----------------------------------------------------
[1] compte de service Â« tlsh Â» cree (sans privilege, sans shell)
[2] service installe : /opt/tlsh/bin/qsa_tlsd
[3] client de validation installe : /opt/tlsh/bin/tlsh_client
[4] application de test installee : /opt/tlsh/outils/app_test.py
[5] configuration creee : /etc/tlsh/tlshd.toml (ecoute 127.0.0.1:8443, application sur le port 8080)
[6] certificats de laboratoire installes ; cle privee lisible par le seul compte Â« tlsh Â»
[7] unite systemd installee : tlsh.service
[8] unite systemd installee : tlsh-app-test.service
[9] tlsh-app-test.service active au demarrage de la machine
[10] tlsh.service active au demarrage de la machine
[11] tlsh-app-test.service (re)demarre
[12] tlsh.service (re)demarre

TLSH - diagnostic de l'installation systeme
-------------------------------------------
 [OK] compte Â« tlsh Â» : uid 995, sans shell
 [OK] /opt/tlsh/bin/qsa_tlsd : droits 755
 [OK] service installe identique au binaire compile
 [OK] /etc/tlsh/tlshd.toml : droits 640
 [OK] /etc/tlsh/certs/serveur.cle : droits 400
 [OK] certificats installes valides (echeance : Feb 16 18:08:18 2027 GMT)
 [OK] tlsh-app-test.service : enabled, active
 [OK] tlsh.service : enabled, active
 [OK] processus 2703188 : compte tlsh, aucune capacite, pas de gain de privilege, appels
      systeme filtres
 [OK] sante : connexion X25519MLKEM768 (0x11EC) reussie en 0.5 ms

RESULTAT : service installe, actif et en bonne sante.
```

Comment tester

```bash
sudo python3 qsa_tooling/tlsh.py recette --systeme
python3 qsa_tooling/tlsh.py mesure
```

- **Recette système** : neuf contrôles, S01 à S09, sur le service réel. Attendu : `RESULTAT : 9 controles reussis sur 9.`
- **Mesure** : le tableau B0, que je n'ai pas encore vu sur votre serveur.

Pour tout retirer : `sudo python3 qsa_tooling/tlsh.py desinstaller --purge`.

```
TLSH - recette de l'installation systeme (service reel, sous systemd)
------------------------------------------------------------------
 [OK] S01 service pris en charge par systemd - les deux unites sont actives et lancees au
      demarrage de la machine
 [OK] S02 compte dedie sans privilege - processus 2703188 sous le compte Â« tlsh Â» (uid 995) :
      aucune capacite, aucun gain de privilege possible, appels systeme filtres
 [OK] S03 droits des fichiers installes - cle privee reservee au service (400) ; configuration
      et programme non modifiables par lui ; cle de l'autorite absente du systeme
 [OK] S04 connexion hybride de bout en bout par le service installe - TLS 1.3, X25519MLKEM768
      (0x11EC), negociation 0.6 ms ; mesure recue par l'application
 [OK] S05 refus journalise dans le journal du systeme - client sans post-quantique refuse ;
      cause lisible dans le journal du systeme
 [OK] S06 confinement applique par systemd - racine en lecture seule pour le service
      (observe), dossiers personnels masques, pas de gain de privilege ; exposition evaluee
      par systemd : 1.2 (OK)
 [OK] S07 relance automatique apres un arret brutal - processus 2703188 tue brutalement ;
      relance automatique (processus 2703834), connexion X25519MLKEM768 (0x11EC) reussie en
      0.5 ms
 [OK] S08 redemarrage commande - arret propre journalise, puis redemarrage ; connexion
      X25519MLKEM768 (0x11EC) reussie en 0.6 ms
 [OK] S09 une seconde installation ne casse rien - seconde installation : rien n'est modifie,
      le service n'est pas redemarre

Rapport : rapports/recette_systeme_mb001_20261118T201222Z.json
RESULTAT : 9 controles reussis sur 9.
```

```
TLSH - mesures de reference B0 (fournisseur standard, sans QSA ni OPT)
------------------------------------------------------------------------
 [OK] 1 connexion a la fois : 5 x 300 connexions completes
 [OK] 8 connexions simultanees : 5 x 600 connexions completes

  Scenario (mediane de 5)     connex./s    p50 ms    p95 ms    p99 ms   CPU ms/cx   RSS Mio
  1 connexion a la fois            2612      0.25      0.26      0.28       0.167       7.1
  8 connexions simultanees          599      0.31      0.40      0.51       0.217       9.8

  Negociation X25519MLKEM768 : 1504 octets emis par le client, 1933 octets recus.
  p50/p95/p99 : duree de negociation vue du client. CPU : temps processeur du
  service par connexion complete. RSS : pic de memoire residente du service.

Rapport : rapports/mesure_b0_20261118T201304Z.json
ATTENTION : client et service partagent la meme machine. Ces chiffres sont
indicatifs ; la matrice du cahier des charges (latence, pertes, MTU) reste a faire.
RESULTAT : mesures B0 enregistrees.
```

```
TLSH - mesures de reference B0 (fournisseur standard, sans QSA ni OPT)
-----------------------------------------------------------------------
 [OK] 1 connexion a la fois : 5 x 300 connexions completes
 [OK] 8 connexions simultanees : 5 x 600 connexions completes

  Scenario (mediane de 5)     connex./s    p50 ms    p95 ms    p99 ms   CPU ms/cx   RSS Mio
  1 connexion a la fois            2612      0.25      0.26      0.28       0.167       7.1
  8 connexions simultanees          599      0.31      0.40      0.51       0.217       9.8

  Negociation X25519MLKEM768 : 1504 octets emis par le client, 1933 octets recus.
  p50/p95/p99 : duree de negociation vue du client. CPU : temps processeur du
  service par connexion complete. RSS : pic de memoire residente du service.

Rapport : rapports/mesure_b0_20261118T201304Z.json
ATTENTION : client et service partagent la meme machine. Ces chiffres sont
indicatifs ; la matrice du cahier des charges (latence, pertes, MTU) reste a faire.
RESULTAT : mesures B0 enregistrees.
ubuntu@ip-172-31-24-146:/home/gon2000fr/tlsh_mb001$ cd ..
ubuntu@ip-172-31-24-146:/home/gon2000fr$ python3 -m zipfile -e TLSH_MB003_A.zip tlsh_mb001
ubuntu@ip-172-31-24-146:/home/gon2000fr$ cd tlsh_mb001
ubuntu@ip-172-31-24-146:/home/gon2000fr/tlsh_mb001$ python3 qsa_tooling/tlsh.py repair

TLSH - reparation
-----------------
[1] compilation du service et du client (une a trois minutes la premiere fois)
   Compiling qsa_admin v0.2.0 (/home/gon2000fr/tlsh_mb001/crates/qsa_admin)
   Compiling tlsh_commun v0.2.0 (/home/gon2000fr/tlsh_mb001/crates/tlsh_commun)
   Compiling tlsh_client v0.2.0 (/home/gon2000fr/tlsh_mb001/crates/tlsh_client)
   Compiling qsa_tlsd v0.2.0 (/home/gon2000fr/tlsh_mb001/crates/qsa_tlsd)
    Finished `release` profile [optimized] target(s) in 3.72s
[2] tests unitaires Rust
 [OK] 17 tests unitaires reussis

TLSH - diagnostic (aucune modification)
---------------------------------------
 [OK] systeme : Linux x86_64
 [OK] Python 3.12.3
 [OK] compilateur C : /usr/bin/cc
 [OK] Rust : 1.100.0
 [OK] outil openssl : 3.0.13
 [OK] sources : 21 fichiers conformes au manifeste
 [OK] binaires compiles et a jour
 [OK] certificats de laboratoire valides (echeance du serveur : Feb 16 18:08:18 2027 GMT)
 [OK] configuration valide
 [INFO] service de laboratoire : arrete
 [INFO] contrepartie independante pour l'interoperabilite : aucune (OpenSSL 3.5+ ou Go 1.24+
        absents)

RESULTAT : environnement conforme.
```

```
TLSH - recette MB-001 et MB-003 lot A (connexions TLS reelles)
------------------------------------------------------------
 [OK] R01 configuration verifiee avant demarrage - valide acceptee ; refusees : destination
      distante, profil classique, cle inconnue
 [OK] R02 connexion TLS 1.3 hybride etablie par le client de validation - TLS 1.3, groupe
      X25519MLKEM768 (0x11EC), negociation 0.5 ms, 1504 octets emis et 1934 recus
 [OK] R03 la requete atteint l'application autorisee, la reponse revient - l'application a
      recu la mesure ; accuse de reception verifie par le client
 [OK] R04 le client ne peut pas choisir la destination - une requete visant une autre
      destination arrive quand meme a l'application autorisee, et nulle part ailleurs
 [OK] R05 client sans groupe hybride (X25519, P-256) refuse - alerte TLS recue par le client ;
      cause journalisee par le service
 [OK] R06 client classique independant (openssl, TLS 1.3) refuse - refuse par une alerte TLS ;
      cause journalisee
 [OK] R07 client TLS 1.2 (openssl) refuse - refuse par une alerte TLS ; cause journalisee
 [OK] R08 nom de serveur (SNI) inconnu refuse - nom de serveur non autorise : refuse et
      journalise
 [OK] R09 certificat d'une autorite inconnue refuse par le client - le client refuse un
      serveur dont il ne connait pas l'autorite
 [OK] R10 certificat emis pour un autre nom refuse par le client - refuse par le client
      (NotValidForName)
 [OK] R11 certificat expire refuse par le service - configuration refusee : le service ne
      demarre pas avec un certificat expire
 [OK] R12 saturation : rejet controle au-dela du nombre maximal de connexions - 4e connexion
      rejetee (maximum 3), puis service de nouveau disponible
 [OK] R13 delai maximal de negociation applique - connexion muette fermee apres 1.5 s (delai
      configure : 1,5 s)
 [OK] R14 message invalide rejete sans perturber le service - message non TLS rejete ; le
      service continue de fonctionner
 [OK] R15 application arretee : diagnostic clair, puis reprise - erreur 502 explicite pour le
      client, cause journalisee, puis reprise
 [OK] R16 etat d'administration local lisible et coherent - 4 negociations reussies, toutes en
      X25519MLKEM768 ; 3 refus de profil ; socket en acces restreint (660)
 [OK] R17 journaux et etat sans secret - 4 fichiers de journal examines : aucune cle, lignes
      toutes en JSON
 [OK] R20 le service expose son certificat et son echeance - echeance 2027-02-16T18:08:18Z, 89
      jours restants, empreinte conforme au fichier
 [OK] R21 rotation du certificat sous trafic, sans connexion perdue - certificat
      f23a4796b0c2... remplace par 9e74ef20fa60... ; 6000 connexions pendant l'operation,
      aucune en echec, 2254 avec l'ancien certificat puis 3746 avec le nouveau
 [OK] R22 rotation defectueuse : retour arriere automatique - certificat expire propose :
      refuse par le service, ancien certificat remis en place et toujours servi
 [OK] R23 rechargement invalide : aucun etat partiellement applique - cle ne correspondant pas
      au certificat : rechargement refuse en entier, le service continue avec l'ancienne paire
 [OK] R24 parametre non rechargeable : refus explique - changement de limite demande a chaud :
      refuse, avec la raison (redemarrage necessaire)
 [OK] R25 alerte quand l'echeance du certificat approche - certificat a 5 jours de son
      echeance : alerte dans l'etat et dans le journal
 [NON EXECUTE] R18 interoperabilite avec une implementation independante - aucune pile TLS
               tierce sachant faire X25519MLKEM768 sur cette machine (OpenSSL 3.5+ ou Go
               1.24+). Ce controle n'est PAS reussi : il reste a faire.
 [OK] R19 arret propre du service - arret sur SIGTERM, code 0, socket d'administration retiree

Rapport : rapports/recette_mb001_20261118T212611Z.json
Non executes (1) : R18 - voir le detail ci-dessus.
RESULTAT : 24 controles reussis sur 24 executes.
```

```
TLSH - installation systeme (compte dedie, systemd)
---------------------------------------------------
[1] service installe : /opt/tlsh/bin/qsa_tlsd
[2] client de validation installe : /opt/tlsh/bin/tlsh_client
[3] unite systemd installee : tlsh.service
[4] tlsh.service (re)demarre

TLSH - diagnostic de l'installation systeme
-------------------------------------------
 [OK] compte 'tlsh' : uid 995, sans shell
 [OK] /opt/tlsh/bin/qsa_tlsd : droits 755
 [OK] service installe identique au binaire compile
 [OK] /etc/tlsh/tlshd.toml : droits 640
 [OK] /etc/tlsh/certs/serveur.cle : droits 400
 [OK] certificats installes valides (echeance : Feb 16 18:08:18 2027 GMT)
 [OK] tlsh-app-test.service : enabled, active
 [OK] tlsh.service : enabled, active
 [OK] processus 2724640 : compte tlsh, aucune capacite, pas de gain de privilege, appels
      systeme filtres
 [OK] sante : connexion X25519MLKEM768 (0x11EC) reussie en 0.6 ms

RESULTAT : service installe, actif et en bonne sante.
```

Comment tester

```bash
python3 qsa_tooling/tlsh.py recette
sudo python3 qsa_tooling/tlsh.py recette --systeme
sudo python3 qsa_tooling/tlsh.py etat --systeme
```

```
TLSH - recette MB-001 et MB-003 lot A (connexions TLS reelles)
---------------------------------------------------------------
 [OK] R01 configuration verifiee avant demarrage - valide acceptee ; refusees : destination
      distante, profil classique, cle inconnue
 [OK] R02 connexion TLS 1.3 hybride etablie par le client de validation - TLS 1.3, groupe
      X25519MLKEM768 (0x11EC), negociation 0.5 ms, 1504 octets emis et 1934 recus
 [OK] R03 la requete atteint l'application autorisee, la reponse revient - l'application a
      recu la mesure ; accuse de reception verifie par le client
 [OK] R04 le client ne peut pas choisir la destination - une requete visant une autre
      destination arrive quand meme a l'application autorisee, et nulle part ailleurs
 [OK] R05 client sans groupe hybride (X25519, P-256) refuse - alerte TLS recue par le client ;
      cause journalisee par le service
 [OK] R06 client classique independant (openssl, TLS 1.3) refuse - refuse par une alerte TLS ;
      cause journalisee
 [OK] R07 client TLS 1.2 (openssl) refuse - refuse par une alerte TLS ; cause journalisee
 [OK] R08 nom de serveur (SNI) inconnu refuse - nom de serveur non autorise : refuse et
      journalise
 [OK] R09 certificat d'une autorite inconnue refuse par le client - le client refuse un
      serveur dont il ne connait pas l'autorite
 [OK] R10 certificat emis pour un autre nom refuse par le client - refuse par le client
      (NotValidForName)
 [OK] R11 certificat expire refuse par le service - configuration refusee : le service ne
      demarre pas avec un certificat expire
 [OK] R12 saturation : rejet controle au-dela du nombre maximal de connexions - 4e connexion
      rejetee (maximum 3), puis service de nouveau disponible
 [OK] R13 delai maximal de negociation applique - connexion muette fermee apres 1.5 s (delai
      configure : 1,5 s)
 [OK] R14 message invalide rejete sans perturber le service - message non TLS rejete ; le
      service continue de fonctionner
 [OK] R15 application arretee : diagnostic clair, puis reprise - erreur 502 explicite pour le
      client, cause journalisee, puis reprise
 [OK] R16 etat d'administration local lisible et coherent - 4 negociations reussies, toutes en
      X25519MLKEM768 ; 3 refus de profil ; socket en acces restreint (660)
 [OK] R17 journaux et etat sans secret - 4 fichiers de journal examines : aucune cle, lignes
      toutes en JSON
 [OK] R20 le service expose son certificat et son echeance - echeance 2027-02-16T18:08:18Z, 89
      jours restants, empreinte conforme au fichier
 [OK] R21 rotation du certificat sous trafic, sans connexion perdue - certificat
      f23a4796b0c2... remplace par d3b4cf5b84ac... ; 6000 connexions pendant l'operation,
      aucune en echec, 2256 avec l'ancien certificat puis 3744 avec le nouveau
 [OK] R22 rotation defectueuse : retour arriere automatique - certificat expire propose :
      refuse par le service, ancien certificat remis en place et toujours servi
 [OK] R23 rechargement invalide : aucun etat partiellement applique - cle ne correspondant pas
      au certificat : rechargement refuse en entier, le service continue avec l'ancienne paire
 [OK] R24 parametre non rechargeable : refus explique - changement de limite demande a chaud :
      refuse, avec la raison (redemarrage necessaire)
 [OK] R25 alerte quand l'echeance du certificat approche - certificat a 5 jours de son
      echeance : alerte dans l'etat et dans le journal
 [NON EXECUTE] R18 interoperabilite avec une implementation independante - aucune pile TLS
               tierce sachant faire X25519MLKEM768 sur cette machine (OpenSSL 3.5+ ou Go
               1.24+). Ce controle n'est PAS reussi : il reste a faire.
 [OK] R19 arret propre du service - arret sur SIGTERM, code 0, socket d'administration retiree
```

```
TLSH - recette de l'installation systeme (service reel, sous systemd)
------------------------------------------------------------------
  [OK] S01 service pris en charge par systemd - les deux unites sont actives et lancees au
       demarrage de la machine
  [OK] S02 compte dedie sans privilege - processus 2724640 sous le compte 'tlsh' (uid 995) :
       aucune capacite, aucun gain de privilege possible, appels systeme filtres
  [OK] S03 droits des fichiers installes - cle privee reservee au service (400) ; configuration
       et programme non modifiables par lui ; cle de l'autorite absente du systeme
  [OK] S04 connexion hybride de bout en bout par le service installe - TLS 1.3, X25519MLKEM768
       (0x11EC), negociation 0.6 ms ; mesure recue par l'application
  [OK] S05 refus journalise dans le journal du systeme - client sans post-quantique refuse ;
       cause lisible dans le journal du systeme
  [OK] S06 confinement applique par systemd - racine en lecture seule pour le service
       (observe), dossiers personnels masques, pas de gain de privilege ; exposition evaluee
       par systemd : 1.2 (OK)
  [OK] S07 relance automatique apres un arret brutal - processus 2724640 tue brutalement ;
       relance automatique (processus 2731121), connexion X25519MLKEM768 (0x11EC) reussie en
       0.6 ms
  [OK] S08 redemarrage commande - arret propre journalise, puis redemarrage ; connexion
       X25519MLKEM768 (0x11EC) reussie en 0.6 ms
  [OK] S09 une seconde installation ne casse rien - seconde installation : rien n'est modifie,
       le service n'est pas redemarre
  [OK] S10 rotation du certificat par systemctl reload, sans redemarrage - certificat
       f23a4796b0c2... remplace par 2cdb1d191580... ; meme processus (2731147), droits de la
       cle conserves
  [OK] S11 rotation defectueuse : retour arriere automatique - certificat expire propose :
       refuse, ancien certificat remis en place et toujours servi

Rapport : rapports/recette_systeme_mb001_20261118T212944Z.json
RESULTAT : 11 controles reussis sur 11.
```

```
TLSH - etat du service
----------------------
Version              : 0.2.0 (TLSH-MB-003 (lot A))
Fournisseur          : reference aws-lc-rs (QSA_TLSH non integre : TLSH-MB-002)
Profil               : general, TLS 1.3 uniquement, X25519MLKEM768 (0x11EC)
Reprise / 0-RTT      : desactivee (TLS-05) / desactive (TLS-04)
OPT_TLSH             : non integre (TLSH-MB-004)
Ecoute               : 127.0.0.1:8443 -> application 127.0.0.1:8080
Certificat           : valable jusqu'au 2027-02-16T21:29:44Z (89 jours)
                       empreinte 2cdb1d191580125de7d1a069e765a44ec5d6785ce2d88e29c3c0f844399857ad
Rechargements        : 2 appliques, 1 refuses
En marche depuis     : 51 s
Connexions           : 7 recues, 0 actives, 7 negociations reussies
Groupes negocies     : X25519MLKEM768 (0x11EC) x 7
Negociation          : p50 0.39 ms, p95 0.49 ms, p99 0.49 ms
Refus et erreurs     : aucun
```

FONCTION DE CHAQUE PATCH

Ce qui se fait bien en Python

- **Vos cinq briques serveur** (extension de signalisation, moteur de politique, interface QSA multi-algo, validateur ZKP, configuration) sont de la logique et du parsing.
- **Les primitives** sont disponibles sans rien réécrire : depuis la version 48, les wheels de `cryptography` donnent accès à ML-KEM et ML-DSA via OpenSSL 3.5+, en plus de X25519, HKDF et AES-GCM. mail-archive
- **Le QSA de Christophe** est un noyau C sur OpenSSL EVP : Python l'appellera par `ctypes` / `cffi` derrière la même interface que QSA-REF.

Ce que ça fait

C'est la partie du TLSH qui décide, pour chaque connexion, quel niveau de protection appliquer. Il n'y a encore ni échange réseau, ni clé, ni chiffrement : le vrai handshake arrive à l'étape 2.

- **Le capteur se décrit** : batterie, exigence de sécurité et qualité du réseau, en 15 octets dans une extension du premier message TLS.
- **Le serveur applique sa politique** : un fichier de règles, modifiable sans toucher au code, et refusé s'il est incohérent.
- **Le serveur rend une décision motivée** : profil retenu, algorithmes, 0-RTT autorisé ou non, règles appliquées. Chaque décision est écrite dans un journal.

Ce que ça fait

- **Le service `qsa_tlsd`** négocie TLS 1.3 avec un échange de clés hybride, déchiffre, et transmet les données à une seule application locale, fixée dans sa configuration.
- **Le client `tlsh_client`** vérifie le certificat, établit la connexion hybride, envoie une température simulée, puis affiche le groupe négocié, la durée et le résultat.
- **Les refus sont explicites** : client sans post-quantique, TLS 1.2, nom de serveur inconnu, saturation, délai dépassé, application arrêtée. Aucun repli vers le classique n'est configurable.
- **L'outil `tlsh.py`** fait le diagnostic, l'installation, la recette et les mesures.

Ce que ça fait

- **Compte dédié** : le service tourne sous un compte `tlsh` sans privilège, et non plus sous le vôtre.
- **Service Linux** : systemd le lance au démarrage de la machine et le relance s'il tombe.
- **Confinement** : système de fichiers en lecture seule pour le service, appels système filtrés, clé privée lisible par lui seul.
- **Contrôle de santé** : l'installation se termine par une vraie connexion hybride sur le service installé.

Ce que ça fait

- **Suivi des échéances** : le service affiche l'échéance et l'empreinte de son certificat dans `etat`, et émet une alerte quand l'échéance approche (30 jours par défaut, réglable).
- **Contrôle de validité** : un certificat expiré, ou dont la clé ne correspond pas, est refusé au démarrage comme au rechargement.
- **Rechargement à chaud** : `systemctl reload tlsh.service` fait relire la configuration sans couper les connexions. Tout est vérifié avant d'être activé d'un bloc ; au moindre défaut, rien ne change.
- **Rotation avec retour arrière** : la commande `renouveler` installe un nouveau certificat, vérifie qu'un client le reçoit, et remet l'ancien en place sinon.

Ce que ça fait

- **Plusieurs routes** : le service peut servir plusieurs noms de serveur, chacun vers sa propre application et, si vous le voulez, avec son propre certificat.
- **Ajout et retrait à chaud** : une route s'ajoute ou se retire par `systemctl reload`, sans redémarrage.
- **Contrôle de santé** : le service vérifie périodiquement que chaque application répond et journalise tout changement.
- **État enrichi** : `etat` affiche chaque route avec ses noms, sa destination, l'état de son application et son nombre de connexions.

Votre configuration actuelle fonctionne sans changement : la route par défaut reste celle déjà en place. Le mode d'emploi montre le bloc à ajouter pour une route supplémentaire.

Comment l'installer

Copiez `TLSH_MB003_B.zip` dans `/home/gon2000fr`, puis :

```bash
cd /home/gon2000fr
python3 -m zipfile -e TLSH_MB003_B.zip tlsh_mb001
cd tlsh_mb001
python3 qsa_tooling/tlsh.py repair
sudo python3 qsa_tooling/tlsh.py installer
```

`repair` recompile (environ une minute) et lance 18 tests unitaires. `installer` met à jour le service installé et le redémarre.

Comment tester

```bash
python3 qsa_tooling/tlsh.py recette
sudo python3 qsa_tooling/tlsh.py recette --systeme
sudo python3 qsa_tooling/tlsh.py etat --systeme
```

```
TLSH - reparation
-----------------
[1] compilation du service et du client (une a trois minutes la premiere fois)
   Compiling qsa_admin v0.3.0 (/home/gon2000fr/tlsh_mb001/crates/qsa_admin)
   Compiling tlsh_commun v0.3.0 (/home/gon2000fr/tlsh_mb001/crates/tlsh_commun)
   Compiling tlsh_client v0.3.0 (/home/gon2000fr/tlsh_mb001/crates/tlsh_client)
   Compiling qsa_tlsd v0.3.0 (/home/gon2000fr/tlsh_mb001/crates/qsa_tlsd)
    Finished `release` profile [optimized] target(s) in 4.66s
[2] tests unitaires Rust
 [OK] 18 tests unitaires reussis

TLSH - diagnostic (aucune modification)
---------------------------------------
 [OK] systeme : Linux x86_64
 [OK] Python 3.12.3
 [OK] compilateur C : /usr/bin/cc
 [OK] Rust : 1.100.0
 [OK] outil openssl : 3.0.13
 [OK] sources : 21 fichiers conformes au manifeste
 [OK] binaires compiles et a jour
 [OK] certificats de laboratoire valides (echeance du serveur : Feb 16 18:08:18 2027 GMT)
 [OK] configuration valide
 [INFO] service de laboratoire : arrete
 [INFO] contrepartie independante pour l'interoperabilite : aucune (OpenSSL 3.5+ ou Go 1.24+
        absents)
```

```
TLSH - installation systeme (compte dedie, systemd)
-----------------------------------------------
[1] service installe : /opt/tlsh/bin/qsa_tlsd
[2] client de validation installe : /opt/tlsh/bin/tlsh_client
[3] tlsh.service (re)demarre

TLSH - diagnostic de l'installation systeme
-------------------------------------------
 [OK] compte 'tlsh' : uid 995, sans shell
 [OK] /opt/tlsh/bin/qsa_tlsd : droits 755
 [OK] service installe identique au binaire compile
 [OK] /etc/tlsh/tlshd.toml : droits 640
 [OK] /etc/tlsh/certs/serveur.cle : droits 400
 [OK] certificats installes valides (echeance : Feb 16 21:29:44 2027 GMT)
 [OK] tlsh-app-test.service : enabled, active
 [OK] tlsh.service : enabled, active
 [OK] processus 2733916 : compte tlsh, aucune capacite, pas de gain de privilege, appels
      systeme filtres
 [OK] sante : connexion X25519MLKEM768 (0x11EC) reussie en 0.5 ms

RESULTAT : service installe, actif et en bonne sante.
```

```
TLSH - recette MB-001 et MB-003 lots A et B (connexions TLS reelles)
------------------------------------------------------------------
 [OK] R01 configuration verifiee avant demarrage - valide acceptee ; refusees : destination
      distante, profil classique, cle inconnue
 [OK] R02 connexion TLS 1.3 hybride etablie par le client de validation - TLS 1.3, groupe
      X25519MLKEM768 (0x11EC), negociation 0.5 ms, 1504 octets emis et 1933 recus
 [OK] R03 la requete atteint l'application autorisee, la reponse revient - l'application a
      recu la mesure ; accuse de reception verifie par le client
 [OK] R04 le client ne peut pas choisir la destination - une requete visant une autre
      destination arrive quand meme a l'application autorisee, et nulle part ailleurs
 [OK] R05 client sans groupe hybride (X25519, P-256) refuse - alerte TLS recue par le client ;
      cause journalisee par le service
 [OK] R06 client classique independant (openssl, TLS 1.3) refuse - refuse par une alerte TLS ;
      cause journalisee
 [OK] R07 client TLS 1.2 (openssl) refuse - refuse par une alerte TLS ; cause journalisee
 [OK] R08 nom de serveur (SNI) inconnu refuse - nom de serveur non autorise : refuse et
      journalise
 [OK] R09 certificat d'une autorite inconnue refuse par le client - le client refuse un
      serveur dont il ne connait pas l'autorite
 [OK] R10 certificat emis pour un autre nom refuse par le client - refuse par le client
      (NotValidForName)
 [OK] R11 certificat expire refuse par le service - configuration refusee : le service ne
      demarre pas avec un certificat expire
 [OK] R12 saturation : rejet controle au-dela du nombre maximal de connexions - 4e connexion
      rejetee (maximum 3), puis service de nouveau disponible
 [OK] R13 delai maximal de negociation applique - connexion muette fermee apres 1.5 s (delai
      configure : 1,5 s)
 [OK] R14 message invalide rejete sans perturber le service - message non TLS rejete ; le
      service continue de fonctionner
 [OK] R15 application arretee : diagnostic clair, puis reprise - erreur 502 explicite pour le
      client, cause journalisee, puis reprise
 [OK] R16 etat d'administration local lisible et coherent - 4 negociations reussies, toutes en
      X25519MLKEM768 ; 3 refus de profil ; socket en acces restreint (660)
 [OK] R17 journaux et etat sans secret - 4 fichiers de journal examines : aucune cle, lignes
      toutes en JSON
 [OK] R20 le service expose son certificat et son echeance - echeance 2027-02-16T18:08:18Z, 89
      jours restants, empreinte conforme au fichier
 [OK] R21 rotation du certificat sous trafic, sans connexion perdue - certificat
      f23a4796b0c2... remplace par f21cfa63ee75... ; 6000 connexions pendant l'operation,
      aucune en echec, 1865 avec l'ancien certificat puis 4135 avec le nouveau
 [OK] R22 rotation defectueuse : retour arriere automatique - certificat expire propose :
      refuse par le service, ancien certificat remis en place et toujours servi
 [OK] R23 rechargement invalide : aucun etat partiellement applique - cle ne correspondant pas
      au certificat : rechargement refuse en entier, le service continue avec l'ancienne paire
 [OK] R24 parametre non rechargeable : refus explique - changement de limite demande a chaud :
      refuse, avec la raison (redemarrage necessaire)
 [OK] R25 alerte quand l'echeance du certificat approche - certificat a 5 jours de son
      echeance : alerte dans l'etat et dans le journal
 [OK] R26 routes par nom de serveur - deux noms de serveur, deux applications distinctes,
      chacun avec son certificat ; aucune requete ne change de destination
 [OK] R27 ajout et retrait d'une route sans redemarrage - route ajoutee a chaud : servie ;
      route retiree a chaud : son nom est de nouveau refuse ; les autres routes continuent
 [OK] R28 routes incoherentes refusees avant demarrage - refusees : nom deja servi,
      destination distante, certificat sans cle
 [OK] R29 sante des applications et compteurs par route - application arretee : vue
      injoignable par le controle de sante, comptee par route ; l'autre route n'est pas
      affectee
 [NON EXECUTE] R18 interoperabilite avec une implementation independante - aucune pile TLS
               tierce sachant faire X25519MLKEM768 sur cette machine (OpenSSL 3.5+ ou Go
               1.24+). Ce controle n'est PAS reussi : il reste a faire.
 [OK] R19 arret propre du service - arret sur SIGTERM, code 0, socket d'administration retiree

Rapport : rapports/recette_mb001_20261118T214612Z.json
Non executes (1) : R18 - voir le detail ci-dessus.
RESULTAT : 28 controles reussis sur 28 executes.
```

```
TLSH - recette de l'installation systeme (service reel, sous systemd)
------------------------------------------------------------------
  [OK] S01 service pris en charge par systemd - les deux unites sont actives et lancees au
       demarrage de la machine
  [OK] S02 compte dedie sans privilege - processus 2733916 sous le compte 'tlsh' (uid 995) :
       aucune capacite, aucun gain de privilege possible, appels systeme filtres
  [OK] S03 droits des fichiers installes - cle privee reservee au service (400) ; configuration
       et programme non modifiables par lui ; cle de l'autorite absente du systeme
  [OK] S04 connexion hybride de bout en bout par le service installe - TLS 1.3, X25519MLKEM768
       (0x11EC), negociation 0.6 ms ; mesure recue par l'application
  [OK] S05 refus journalise dans le journal du systeme - client sans post-quantique refuse ;
       cause lisible dans le journal du systeme
  [OK] S06 confinement applique par systemd - racine en lecture seule pour le service
       (observe), dossiers personnels masques, pas de gain de privilege ; exposition evaluee
       par systemd : 1.2 (OK)
  [OK] S07 relance automatique apres un arret brutal - processus 2733916 tue brutalement ;
       relance automatique (processus 2740412), connexion X25519MLKEM768 (0x11EC) reussie en
       0.7 ms
  [OK] S08 redemarrage commande - arret propre journalise, puis redemarrage ; connexion
       X25519MLKEM768 (0x11EC) reussie en 0.7 ms
  [OK] S09 une seconde installation ne casse rien - seconde installation : rien n'est modifie,
       le service n'est pas redemarre
  [OK] S10 rotation du certificat par systemctl reload, sans redemarrage - certificat
       2cdb1d191580... remplace par 45f4bed83287... ; meme processus (2740436), droits de la
       cle conserves
  [OK] S11 rotation defectueuse : retour arriere automatique - certificat expire propose :
       refuse, ancien certificat remis en place et toujours servi

Rapport : rapports/recette_systeme_mb001_20261118T214659Z.json
RESULTAT : 11 controles reussis sur 11.
```

```
TLSH - etat du service
----------------------
Version              : 0.3.0 (TLSH-MB-003 (lots A et B))
Fournisseur          : reference aws-lc-rs (QSA_TLSH non integre : TLSH-MB-002)
Profil               : general, TLS 1.3 uniquement, X25519MLKEM768 (0x11EC)
Reprise / 0-RTT      : desactivee (TLS-05) / desactive (TLS-04)
OPT_TLSH             : non integre (TLSH-MB-004)
Ecoute               : 127.0.0.1:8443 -> application 127.0.0.1:8080
Certificat           : valable jusqu'au 2027-02-16T21:46:59Z (89 jours)
                       empreinte 45f4bed83287b5ae9931db2e3d47a8dab58828bd3bf85079349a2078aee03071
Rechargements        : 2 appliques, 1 refuses
Route defaut         : tlsh.labo.test -> 127.0.0.1:8080 (application joignable) ; 7 connexions
En marche depuis     : 55 s
Connexions           : 7 recues, 0 actives, 7 negociations reussies
Groupes negocies     : X25519MLKEM768 (0x11EC) x 7
Negociation          : p50 0.47 ms, p95 0.56 ms, p99 0.56 ms
Refus et erreurs     : aucun
```

Comment l'installer

Copiez `TLSH_MB003_C.zip` dans `/home/gon2000fr`, puis :

```bash
cd /home/gon2000fr
python3 -m zipfile -e TLSH_MB003_C.zip tlsh_mb001
cd tlsh_mb001
python3 qsa_tooling/tlsh.py check
sudo python3 qsa_tooling/tlsh.py installer
```

Le code Rust n'a pas changé : `check` doit répondre `environnement conforme` sans rien recompiler. `installer` doit afficher `installation existante rangee comme version 0.3.0-…`, sans redémarrer le service.

Comment tester

```bash
sudo python3 qsa_tooling/tlsh.py recette --systeme
sudo python3 qsa_tooling/tlsh.py versions
```

Ce que ça fait

- **Versions séparées** : chaque version installée a son propre dossier, et un lien désigne celle qui est en service. Changer de version revient à remplacer ce lien.
- **Contrôles avant activation** : la nouvelle version doit être intègre, s'exécuter et accepter la configuration en place. Sinon elle n'est pas activée et le service n'est pas touché.
- **Retour arrière automatique** : après la bascule, une vraie connexion hybride vérifie la santé. En cas d'échec, la version précédente est rétablie et vérifiée à son tour.
- **Intégrité** : l'empreinte de chaque programme est notée à l'installation, et une modification est signalée.

La configuration et les certificats ne font pas partie d'une version : ils sont conservés à chaque mise à jour.

Contrôle	Ce qu'il prouve
S12	une version qui ne démarre pas est mise en service, détectée, puis retirée ; la précédente est rétablie
S13	une version inexécutable est refusée avant activation
S14	passage à une autre version, puis retour à la version d'origine
S15	un programme installé modifié d'un seul octet est signalé

Pendant S12 et S14, le service est redémarré plusieurs fois : la recette système dure une vingtaine de secondes de plus.

```
TLSH - diagnostic (aucune modification)
---------------------------------------
 [OK] systeme : Linux x86_64
 [OK] Python 3.12.3
 [OK] compilateur C : /usr/bin/cc
 [OK] Rust : 1.100.0
 [OK] outil openssl : 3.0.13
 [OK] sources : 21 fichiers conformes au manifeste
 [OK] binaires compiles et a jour
 [OK] certificats de laboratoire valides (echeance du serveur : Feb 16 18:08:18 2027 GMT)
 [OK] configuration valide
 [INFO] service de laboratoire : arrete
 [INFO] contrepartie independante pour l'interoperabilite : aucune (OpenSSL 3.5+ ou Go 1.24+
        absents)

RESULTAT : environnement conforme.
```

```
TLSH - installation systeme (compte dedie, systemd)
---------------------------------------------------
[1] installation existante rangee comme version 0.3.0-0dbd4c3f9c81

TLSH - diagnostic de l'installation systeme
-------------------------------------------
 [OK] compte 'tlsh' : uid 995, sans shell
 [OK] /opt/tlsh/bin/qsa_tlsd : droits 755
 [OK] version en service : 0.3.0-0dbd4c3f9c81, fichiers conformes a leur empreinte
 [OK] /etc/tlsh/tlshd.toml : droits 640
 [OK] /etc/tlsh/certs/serveur.cle : droits 400
 [OK] certificats installes valides (echeance : Feb 16 21:46:59 2027 GMT)
 [OK] tlsh-app-test.service : enabled, active
 [OK] tlsh.service : enabled, active
 [OK] processus 2740436 : compte tlsh, aucune capacite, pas de gain de privilege, appels
      systeme filtres
 [OK] sante : connexion X25519MLKEM768 (0x11EC) reussie en 0.6 ms

RESULTAT : service installe, actif et en bonne sante.
```

```
TLSH - recette de l'installation systeme (service reel, sous systemd)
-----------------------------------------------------------------------
 [OK] S01 service pris en charge par systemd - les deux unites sont actives et lancees au
      demarrage de la machine
 [OK] S02 compte dedie sans privilege - processus 2740436 sous le compte 'tlsh' (uid 995) :
      aucune capacite, aucun gain de privilege possible, appels systeme filtres
 [OK] S03 droits des fichiers installes - cle privee reservee au service (400) ; configuration
      et programme non modifiables par lui ; cle de l'autorite absente du systeme
 [OK] S04 connexion hybride de bout en bout par le service installe - TLS 1.3, X25519MLKEM768
      (0x11EC), negociation 0.6 ms ; mesure recue par l'application
 [OK] S05 refus journalise dans le journal du systeme - client sans post-quantique refuse ;
      cause lisible dans le journal du systeme
 [OK] S06 confinement applique par systemd - racine en lecture seule pour le service
      (observe), dossiers personnels masques, pas de gain de privilege ; exposition evaluee
      par systemd : 1.2 (OK)
 [OK] S07 relance automatique apres un arret brutal - processus 2740436 tue brutalement ;
      relance automatique (processus 2741989), connexion X25519MLKEM768 (0x11EC) reussie en
      0.6 ms
 [OK] S08 redemarrage commande - arret propre journalise, puis redemarrage ; connexion
      X25519MLKEM768 (0x11EC) reussie en 0.7 ms
 [OK] S09 une seconde installation ne casse rien - seconde installation : rien n'est modifie,
      le service n'est pas redemarre
 [OK] S10 rotation du certificat par systemctl reload, sans redemarrage - certificat
      45f4bed83287... remplace par bbaa261f1dcb... ; meme processus (2742016), droits de la
      cle conserves
 [OK] S11 rotation defectueuse : retour arriere automatique - certificat expire propose :
      refuse, ancien certificat remis en place et toujours servi
 [OK] S12 mise a jour defectueuse : retour arriere automatique - une version qui ne demarre
      pas est mise en service, detectee, puis retiree : version 0.3.0-0dbd4c3f9c81 retablie,
      connexion X25519MLKEM768 (0x11EC) reussie en 0.4 ms
 [OK] S13 version invalide refusee avant activation - version inexecutable refusee avant
      activation ; le service n'a pas ete touche
 [OK] S14 mise a jour reussie, puis retour a la version d'origine - passage a une autre
      version puis retour a 0.3.0-0dbd4c3f9c81, sante verifiee a chaque etape
 [OK] S15 integrite des programmes installes - un programme installe modifie d'un seul octet
      est signale par le controle d'integrite

Rapport : rapports/recette_systeme_mb001_20261118T215747Z.json
RESULTAT : 15 controles reussis sur 15.
```

```
TLSH - versions installees
--------------------------
 [OK] 0.3.0-0dbd4c3f9c81            EN SERVICE  integre

Retour a la version precedente : sudo python3 qsa_tooling/tlsh.py retablir
```



QSA ET TLSH TEMPORAIRE

Comment l'installer et tester

Copiez `TLSH_MB002_A.zip` dans `/home/gon2000fr`, puis :

```bash
cd /home/gon2000fr
python3 -m zipfile -e TLSH_MB002_A.zip tlsh_mb001
cd tlsh_mb001
python3 qsa_tooling/tlsh.py check
python3 qsa_tooling/tlsh.py qsa
python3 qsa_tooling/tlsh.py recette
```

- **`check`** : `environnement conforme`, sans recompilation. Le service installé n'est pas concerné.
- **`qsa`** : trois à sept minutes la première fois (compilation d'OpenSSL). Attendu : `bilan : 13 OK, 0 echec, 1 SKIP`, puis `RESULTAT : noyau QSA_TLSH temporaire compile et verifie.` Le SKIP est FN-DSA-512, absent sans liboqs.
- **`recette`** : attendu cette fois `29 controles reussis sur 29 executes.` OpenSSL 3.5 sert aussi de pile TLS tierce, donc R18 ne sera plus « non exécuté ».

Ce que ça fait

- **Même interface que le vrai paquet** : `qsa_tlsh.h` reprend les noms, types, tailles et codes de retour de vos documents du 11 novembre. Le vrai paquet de Christophe devra pouvoir remplacer le mien fichier pour fichier.
- **Même principe** : tout le calcul est délégué à OpenSSL (X25519, ML-KEM-768, ML-DSA-44), et un algorithme absent répond `UNAVAIL`, jamais un faux succès.
- **OpenSSL 3.5 local** : ML-KEM et ML-DSA demandent OpenSSL 3.5, alors que votre serveur a la 3.0. L'outil télécharge la 3.5.9, vérifie son empreinte et la compile dans le dossier du projet, sans toucher à celle du système.

```
TLSH - diagnostic (aucune modification)
---------------------------------------
 [OK] systeme : Linux x86_64
 [OK] Python 3.12.3
 [OK] compilateur C : /usr/bin/cc
 [OK] Rust : 1.100.0
 [OK] outil openssl : 3.0.13
 [OK] sources : 26 fichiers conformes au manifeste
 [OK] binaires compiles et a jour
 [OK] certificats de laboratoire valides (echeance du serveur : Feb 16 18:08:18 2027 GMT)
 [OK] configuration valide
 [INFO] service de laboratoire : arrete
 [INFO] contrepartie independante pour l'interoperabilite : aucune (OpenSSL 3.5+ ou Go 1.24+
        absents)
```

```
TLSH - noyau QSA_TLSH, VERSION TEMPORAIRE (interface du 2026-11-11)
------------------------------------------------------------------
[1] OpenSSL 3.5.9 local (ML-KEM-768 et ML-DSA-44 demandent OpenSSL 3.5 ou plus)
    telechargement d'OpenSSL 3.5.9 (53 Mo)...
    compilation d'OpenSSL 3.5.9 (deux a six minutes)...
 [OK] OpenSSL 3.5.9 29 Sep 2026 (Library: OpenSSL 3.5.9 29 Sep 2026), empreinte de l'archive
      verifiee, dans tiers/openssl
[2] compilation durcie du noyau et essais sur OpenSSL 3.5.9
   QSA_TLSH VERSION TEMPORAIRE 0.1 (interface du 2026-11-11) sur OpenSSL 3.5.9 29 Sep 2026
    [OK] X25519 : echange aller-retour
    [OK] X25519 : reponse connue (RFC 7748, 6.1)
    [OK] X25519 : cle publique nulle refusee, sortie effacee
    [OK] hybride : assemblage ML-KEM_ss || X25519_ss (64 octets)
    [OK] signatures : ML-DSA-44 approuve, FN-DSA-512 non approuve
    [OK] entrees invalides : arguments nuls et algorithme inconnu refuses
    [OK] ML-KEM-768 : encapsulation et decapsulation donnent le meme secret
    [OK] ML-KEM-768 : 10 vecteurs officiels NIST de decapsulation
    [OK] ML-KEM-768 : chiffre altere -> secret different, sans erreur revelatrice
    [OK] ML-KEM-768 : cle d'encapsulation invalide refusee, sortie effacee
    [OK] ML-DSA-44 : signature verifiee ; signature ou message alteres refuses
    [OK] ML-DSA-44 : vecteur officiel NIST de verification (signature invalide refusee)
    [OK] FN-DSA-512 absent : UNAVAIL partout, aucun faux succes
    [SKIP] FN-DSA-512 : signature et verification (fournisseur absent)
 [OK] bilan : 13 OK, 0 echec, 1 SKIP ; 0 avertissement
[3] memes essais avec detection d'erreurs memoire (ASan, UBSan)
 [OK] aucune erreur memoire ni comportement indefini, aucune fuite
[4] comportement sans post-quantique (OpenSSL du systeme)
 [OK] OpenSSL 3.0.13 du systeme : 8 OK, 0 echec, 6 SKIP ; les algorithmes absents repondent
      UNAVAIL, jamais un faux succes

Rapport : rapports/qsa_tlsh_temporaire_20261118T222350Z.json
RESULTAT : noyau QSA_TLSH temporaire compile et verifie.
```

```
TLSH - recette MB-001 et MB-003 lots A et B (connexions TLS reelles)
------------------------------------------------------------------------
 [OK] R01 configuration verifiee avant demarrage - valide acceptee ; refusees : destination
      distante, profil classique, cle inconnue
 [OK] R02 connexion TLS 1.3 hybride etablie par le client de validation - TLS 1.3, groupe
      X25519MLKEM768 (0x11EC), negociation 0.5 ms, 1504 octets emis et 1933 recus
 [OK] R03 la requete atteint l'application autorisee, la reponse revient - l'application a
      recu la mesure ; accuse de reception verifie par le client
 [OK] R04 le client ne peut pas choisir la destination - une requete visant une autre
      destination arrive quand meme a l'application autorisee, et nulle part ailleurs
 [OK] R05 client sans groupe hybride (X25519, P-256) refuse - alerte TLS recue par le client ;
      cause journalisee par le service
 [OK] R06 client classique independant (openssl, TLS 1.3) refuse - refuse par une alerte TLS ;
      cause journalisee
 [OK] R07 client TLS 1.2 (openssl) refuse - refuse par une alerte TLS ; cause journalisee
 [OK] R08 nom de serveur (SNI) inconnu refuse - nom de serveur non autorise : refuse et
      journalise
 [OK] R09 certificat d'une autorite inconnue refuse par le client - le client refuse un
      serveur dont il ne connait pas l'autorite
 [OK] R10 certificat emis pour un autre nom refuse par le client - refuse par le client
      (NotValidForName)
 [OK] R11 certificat expire refuse par le service - configuration refusee : le service ne
      demarre pas avec un certificat expire
 [OK] R12 saturation : rejet controle au-dela du nombre maximal de connexions - 4e connexion
      rejetee (maximum 3), puis service de nouveau disponible
 [OK] R13 delai maximal de negociation applique - connexion muette fermee apres 1.5 s (delai
      configure : 1,5 s)
 [OK] R14 message invalide rejete sans perturber le service - message non TLS rejete ; le
      service continue de fonctionner
 [OK] R15 application arretee : diagnostic clair, puis reprise - erreur 502 explicite pour le
      client, cause journalisee, puis reprise
 [OK] R16 etat d'administration local lisible et coherent - 4 negociations reussies, toutes en
      X25519MLKEM768 ; 3 refus de profil ; socket en acces restreint (660)
 [OK] R17 journaux et etat sans secret - 4 fichiers de journal examines : aucune cle, lignes
      toutes en JSON
 [OK] R20 le service expose son certificat et son echeance - echeance 2027-02-16T18:08:18Z, 89
      jours restants, empreinte conforme au fichier
 [OK] R21 rotation du certificat sous trafic, sans connexion perdue - certificat
      f23a4796b0c2... remplace par ee97c69e19a2... ; 6000 connexions pendant l'operation,
      aucune en echec, 1999 avec l'ancien certificat puis 4001 avec le nouveau
 [OK] R22 rotation defectueuse : retour arriere automatique - certificat expire propose :
      refuse par le service, ancien certificat remis en place et toujours servi
 [OK] R23 rechargement invalide : aucun etat partiellement applique - cle ne correspondant pas
      au certificat : rechargement refuse en entier, le service continue avec l'ancienne paire
 [OK] R24 parametre non rechargeable : refus explique - changement de limite demande a chaud :
      refuse, avec la raison (redemarrage necessaire)
 [OK] R25 alerte quand l'echeance du certificat approche - certificat a 5 jours de son
      echeance : alerte dans l'etat et dans le journal
 [OK] R26 routes par nom de serveur - deux noms de serveur, deux applications distinctes,
      chacun avec son certificat ; aucune requete ne change de destination
 [OK] R27 ajout et retrait d'une route sans redemarrage - route ajoutee a chaud : servie ;
      route retiree a chaud : son nom est de nouveau refuse ; les autres routes continuent
 [OK] R28 routes incoherentes refusees avant demarrage - refusees : nom deja servi,
      destination distante, certificat sans cle
 [OK] R29 sante des applications et compteurs par route - application arretee : vue
      injoignable par le controle de sante, comptee par route ; l'autre route n'est pas
      affectee
 [OK] R18 interoperabilite avec une implementation independante - OpenSSL 3.5.9 (s_client)
      vers le service TLSH : groupe X25519MLKEM768 negocie, certificat verifie, reponse de
      l'application recue
 [OK] R19 arret propre du service - arret sur SIGTERM, code 0, socket d'administration retiree

Rapport : rapports/recette_mb001_20261118T222443Z.json
RESULTAT : 29 controles reussis sur 29 executes.
```

AXES DE DEVELOPPEMENT
VERSION 0.4 Nov 2026

1. **Le noyau temporaire coûte environ 40 % de processeur en plus.** C'est attendu : il traverse l'interface générique d'OpenSSL à chaque opération. Cet écart ne dit rien du paquet de Christophe.
2. **La première connexion après un démarrage prend 20 ms en QSA,** contre 0,5 ms en référence. On le voit dans S07, S08, R30 et dans `etat`. Je soupçonne l'initialisation d'OpenSSL au premier usage, mais je ne l'ai pas mesuré.
3. **Le banc de mesure plafonne.** Le scénario à 8 connexions donne 599 connexions par seconde en B0 comme en B1, et déjà hier. Un chiffre identique dans deux configurations différentes indique que la limite est ailleurs que dans le service, probablement dans l'application de test en Python.

Lot	Contenu	Pourquoi dans cet ordre
A. Banc de mesure fiable	application de test rapide, scénarios du plan (charge soutenue, transferts longs, saturation, durée), mesure séparée de la négociation et du transfert	sans lui, aucun gain n'est prouvable
B. OPT_TLSH, première version	composant `opt_tlsh` activable par configuration : préparation du noyau au démarrage (les 20 ms), admission selon la charge, buffers réutilisés, écritures regroupées avec attente bornée	ce sont les gains les plus probables au vu des mesures
C. Campagne B0/B1/B2	rapport reproductible : où OPT améliore, où il apporte peu, ce qu'il coûte	c'est la validation demandée par le plan