CryptoNext security

Transition vers la cryptographie post-quantique dans les systèmes embarqués.

Les algorithmes PQC sont en phase de standardisation et de déploiement. Reste un risque de sécurité majeur : les **timing attacks**.

01 CONTEXTE & ENJEUX

Depuis la publication par le NIST des premiers standards **FIPS 203 / 204 / 205** en août 2024, la PQC entre dans une phase de déploiement en production. Trois premiers algorithmes ont été standardisés :

- **ML-KEM** — mécanisme d'encapsulation de clé.
- **ML-DSA** et **SLH-DSA** — signature numérique.

Certains acteurs comme **Cloudflare**, Google (TLS), Apple, Signal (messagerie) ou encore Amazon (cloud) sont déjà avancés dans le déploiement de la PQC, mais la migration sera incontournable sur la prochaine décennie.

Le programme **CAVP** du NIST valide la correction mathématique des implémentations — *pas leur sécurité*.

02 LES TIMING ATTACKS

Décrites par **Kocher dès 1996**, ces attaques exploitent de légères différences de temps d'exécution pour reconstruire une clé secrète. Elles sont :

- **Montées à distance**, sans accès physique.
- **Invisibles** — aucune trace sur le matériel.
- **Très efficaces** — peuvent compromettre le système entier.

Les **systèmes embarqués** y sont particulièrement exposés : peu de bruit, pas d'OS, interfaces scriptables. Le signal est net.

03 QUATRE SOURCES DE FUITE TEMPORELLE

01 — Flot de contrôle algorithmique
Branchements conditionnels dépendant d'un secret. Deux branches ont rarement le même nombre de cycles → la différence révèle le secret. **Ex :** exponentiation modulaire (Kocher, 1996).

02 — Accès mémoire secrets
Accès position dans la mémoire dépendant d'un secret → **cache miss** mesurable. **Ex :** S-Box des chiffres par blocs.

03 — Instructions à latence variable (ISA)
Division : durée dépendante des opérandes. **Ex : Kyberslash** (2023) — Java, Rust, Python, JavaScript, Zig.

04 — PQC et timing attacks
Avec la cryptographie post-quantique on redécouvre les faiblesses introduites par l'exécution en temps variable. La connaissance acquise en trente ans d'attaques sur la cryptographie classique n'est **pas immédiatement transférable** : mathématiques différentes, objets nouveaux. Une nouvelle prudence est nécessaire.

04 CONTRE-MESURES

BITMASKS	Remplacer les `if/else` par des opérations bit à bit en **temps constant**.
BLINDING	Injecter un **aléa** pour rendre les fuites inexploitables.
RÉDUCTIONS	**Barrett** / **Montgomery** pour éliminer les divisions.
FIXSLICING	S-Box réécrites en **circuits booléens** (AND / OR / XOR) — pas d'accès mémoire secret.

05 CAS ILLUSTRATIF — CLANG VS GCC

UNE INSTRUCTION · ~10 MINUTES

Dans l'implémentation de référence de Kyber/ML-KEM, une ligne de code est écrite suivant l'état de l'art pour obtenir un code en temps constant. Une passe d'optimisation de **Clang** le transforme en un `bt`/jump — comme un simple `if`. **GCC → temps constant. Clang → une instruction de différence.** ~10 minutes suffisent à retrouver la clé **ML-KEM** complète.

> Le temps constant n'est pas une propriété du code source. C'est une propriété du triplet **code — compilateur — architecture**. Changer l'un impose de tout re-vérifier.

06 OUTILS D'AUDIT — ET LEURS LIMITES

dudect / TIMECOP	Analyse dynamique sur processeur cible. *TIMECOP incompatible bare-metal.*
Microsurf	Émulation binaire (**Unicorn Engine**). Trace l'état CPU sur binaires compilés.
Jasmin	Compilateur formel — garantit le temps constant. Portabilité et support ISA limités.

Aucun outil ne couvre l'ensemble des plateformes / architectures / OS. L'expertise humaine reste indispensable — choisir les bons outils, combiner les résultats, analyser manuellement.

07 MESSAGES CLÉS

01 Le **temps constant** est le niveau de sécurité **minimum** de toute bibliothèque cryptographique.

02 C'est une **propriété continue** du triplet *(code, compilateur, architecture)* — pas du code seul.

03 Après 30 ans, des **vulnérabilités émergent encore**. La vigilance et l'expertise humaine restent essentielles.

04 Les **systèmes embarqués** sont exposés : mesures de temps précises et sans bruit, outils de vérification limités, multiplicité de plateformes et cross-compilateurs.

RÉFÉRENCES

NIST FIPS 203 / 204 / 205 · Kocher (1996) · Kyberslash (2023) · crocs-muni.github.io/ct-tools