

Cryptographie Industrielle Avancée // TP 4

Sécurisation d'une Plateforme MedTech de Bout-en-Bout

Loïc Rouquette

28 octobre 2025

HealthChain Dynamics fait face à un défi d'architecture complexe. Le parc de pacemakers de son hôpital partenaire est hétérogène : certains dispositifs sont trop anciens pour supporter des algorithmes de cryptographie post-quantique. Cependant, la nouvelle politique de sécurité de l'hôpital, inspirée du modèle "**Zéro Confiance**" (**Zero Trust**), stipule qu'**aucune donnée ne doit transiter en clair**, même sur le réseau local interne, qui est désormais considéré comme potentiellement hostile.

Votre mission est de prototyper l'architecture de sécurité complète qui adresse cette contrainte :

1. **Chiffrement léger IoT** : Les pacemakers chiffreront leurs données avec un algorithme symétrique rapide et économe en énergie (ChaCha20Poly1305).
 2. **Passerelle de Sécurité (Proxy)** : Un proxy puissant agira comme une passerelle, déchiffrant le flux IoT puis le re-chiffrant via un canal post-quantique robuste pour la communication avec les serveurs centraux.
- **Outils** : Python 3, `oqs-python`, `cryptography`, `microsoft-seal`.
 - **Prérequis** : Assurez-vous que toutes les bibliothèques sont installées : `pip install oqs cryptography microsoft-seal`.

Scénario Global du Pipeline de Données

1. **Émission Chiffrée (Pacemaker → Proxy)** : Le pacemaker chiffre ses données (âge, rythme cardiaque) avec **ChaCha20Poly1305** et une clé pré-partagée (PSK), puis les envoie.
2. **Rupture Protocolaire (Proxy)** : Le proxy déchiffre les données avec la PSK. Il agit comme un point de terminaison pour le protocole IoT.
3. **Sécurisation en Transit (Proxy → Serveur)** : Le proxy établit un canal sécurisé avec le serveur central en utilisant le **KEM Kyber**. Il utilise la clé de session ainsi négociée pour re-chiffrer les données (avec AES-GCM) avant de les envoyer.
4. **Sécurisation au Repos (Serveur Hôpital)** : Le serveur déchiffre les données reçues, puis les re-chiffre avec sa propre clé **AES** interne avant de les stocker en base de données.
5. **Analyse Confidentielle (Serveur Hôpital → Cloud)** : Pour une analyse, le serveur déchiffre les données de sa base, puis les re-chiffre avec le **chiffrement homomorphe (FHE)** avant de les envoyer au cloud.

Mise en Œuvre du Pipeline

Partie 1 : Chiffrement Léger sur le Segment IoT (ChaCha20)

Objectif : Simuler le chiffrement des données par le pacemaker ressource-limité.

Étapes :

1. Configuration Partagée (Pacemaker & Proxy) :

- Un pacemaker serait provisionné avec une clé secrète partagée lors de sa fabrication ou de son installation. Simulons cette clé.

```
import os
import json
from cryptography.hazmat.primitives.ciphers.aead import ChaCha20Poly1305

# Clé pré-partagée (PSK) de 32 bytes connue uniquement du pacemaker et du proxy
iot_psk = os.urandom(32)
chacha = ChaCha20Poly1305(iot_psk)
```

2. Émission par le Pacemaker :

- Le pacemaker prépare ses données, génère un nonce (numéro unique pour chaque message) et chiffre le tout.

```
pacemaker_data = {
    "age": 40,
    "heart_rate_bpm": 82,
    "sensor_id": 42,
    "timestamp": "2025-10-13T12:50:39.430262"
}
data_bytes = json.dumps(pacemaker_data).encode('utf-8')

# Le nonce doit être unique pour chaque chiffrement avec la même clé
nonce = os.urandom(12)

# Chiffrement
encrypted_payload_iot = chacha.encrypt(nonce, data_bytes, None) # None for additional data
print(f"[Pacemaker -> Proxy] Envoi du payload chiffré (nonce + données) sur le réseau")
```

Partie 2 : La Passerelle de Sécurité : Proxy (Kyber)

Objectif : Le proxy déchiffre le trafic IoT et établit un tunnel post-quantique vers le serveur central.

Étapes :

1. Réception et Déchiffrement par le Proxy :

- Le proxy reçoit le nonce et le payload chiffré, puis les déchiffre en utilisant la PSK.

Le proxy déchiffre les données

```
decrypted_bytes_proxy = chacha.decrypt(nonce, encrypted_payload_iot, None)
decrypted_data_proxy = json.loads(decrypted_bytes_proxy.decode('utf-8'))
print(f"[Proxy] Données du pacemaker déchiffrées : {decrypted_data_proxy}")
```

2. Établissement du canal sécurisé (Proxy ↔ Serveur) :

- Maintenant que les données sont en clair sur le proxy, celui-ci doit les envoyer au serveur central. Il utilise **Kyber** pour négocier une clé de session symétrique.
- Le serveur central génère une paire de clés Kyber.
- Le proxy utilise la clé publique du serveur pour encapsuler un secret. Le **shared_secret** retourné par Kyber **est** la clé de session.

3. Re-chiffrement et Transmission :

- Le proxy utilise cette nouvelle clé de session (par exemple, avec AES-GCM, un autre chiffrement authentifié) pour chiffrer les données avant de les envoyer au serveur. Le serveur utilisera la même clé de session pour déchiffrer.

Partie 3 & 4 : Stockage au Repos et Analyse Confidentielle (AES & FHE)

1. **Arrivée au Serveur Central** : Le serveur déchiffre les données reçues du proxy en utilisant la clé de session Kyber.
2. **Sécurité au Repos** : Le serveur re-chiffre immédiatement les données avec sa propre clé **AES** interne (**Fernet**) avant de les stocker dans sa base de données SQLite.
3. **Préparation pour l'Analyse** : Pour une analyse, le serveur :
 - (a) Récupère les données chiffrées de la base.
 - (b) Les déchiffre avec sa clé AES.
 - (c) Les re-chiffre avec le **chiffrement homomorphe (FHE)**.
 - (d) Les envoie au cloud pour le calcul "aveugle".
 - (e) Récupère le résultat, le déchiffre avec la clé FHE et valide le calcul.