Université IBN TOFAIL – Kénitra
ECOLE NATIONALE DES SCIENCES APPLIQUEES

Cours Sécurité des Systèmes Embarqués et IoT

CH3: Sécurisation des équipements IoT (Hardware & Firmware)

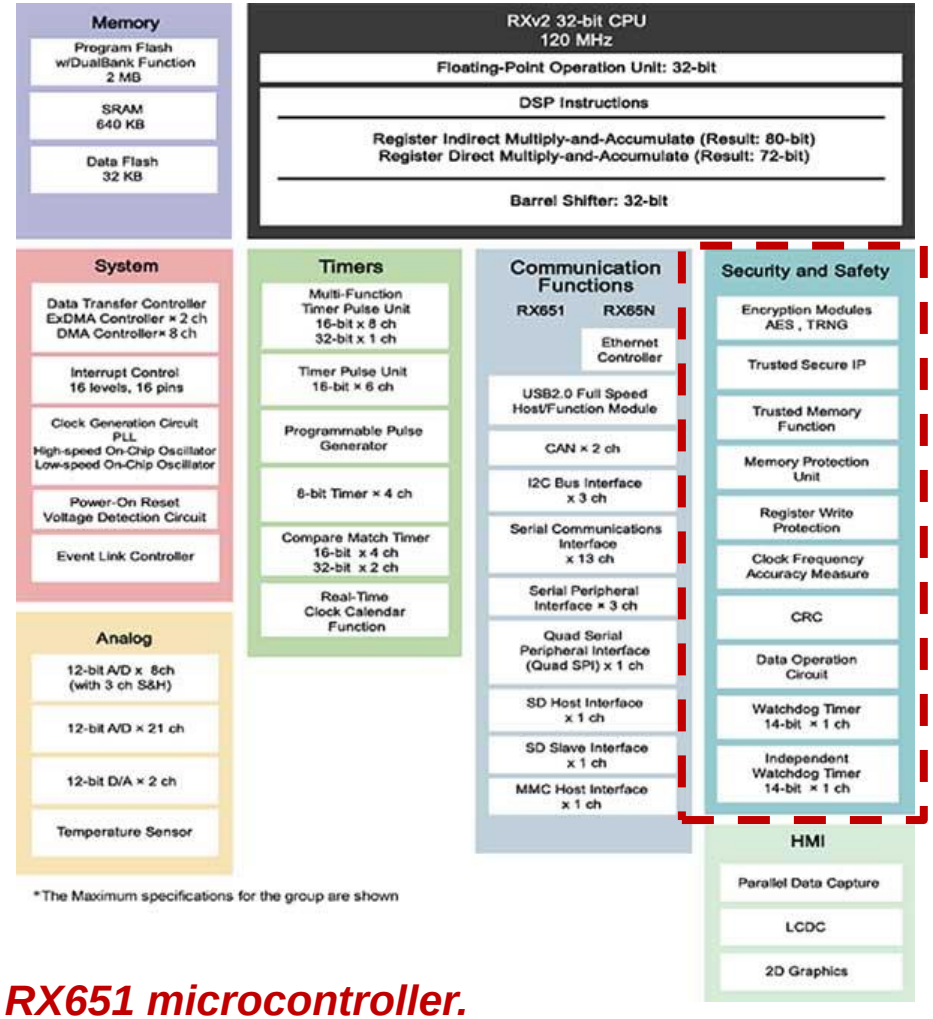
Par:

Hassan EL FADIL

elfadil.h@gmail.com

1. Sécurité des capteurs et microcontrôleurs

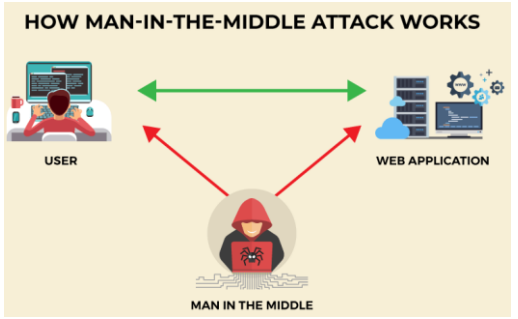
- Dans un environnement IoT, la sécurité des équipements est une priorité.
- En effet, une vulnérabilité au niveau du **hardware (composants physiques)** ou du **firmware (logiciel bas niveau)** peut compromettre l'ensemble du système.

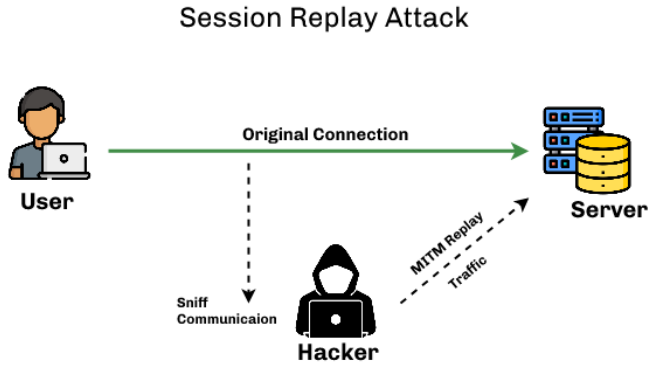


A view of the security building blocks in the RX651 microcontroller.

1.1. Sécurisation des interfaces de communication :

- Les interfaces de communication des objets connectés sont une cible privilégiée des attaques. Les menaces incluent :
  - Attaques par interception (Man-in-the-Middle, sniffing)
  - Usurpation d'identité et injection de paquets malveillants (Identity Spoofing)
  - Attaques par rejeu (Replay Attack) et falsification des messages
- Pour éviter ces menaces, il est essentiel d'implémenter des **mécanismes de chiffrement et d'authentification robustes**.





1. **Chiffrement des communications sans fil** (Wi-Fi, BLE, LoRa, ZigBee) avec TLS 1.2+, DTLS, AES-128/256.

Technologie	Chiffrement recommandé	Protocole de transport sécurisé
Wi-Fi (802.11)	WPA3, WPA2-Enterprise (AES-256)	TLS 1.2+, DTLS
Bluetooth (BLE 4.2+)	AES-128 CCM (LE Secure Connections)	DTLS, TLS 1.3 (si passerelle)
LoRaWAN	AES-128 pour l'authentification et le chiffrement des paquets	End-to-End Encryption (E2EE)
ZigBee	AES-128 CCM (Application Layer Security)	Pas de TLS natif, mais chiffrement des messages

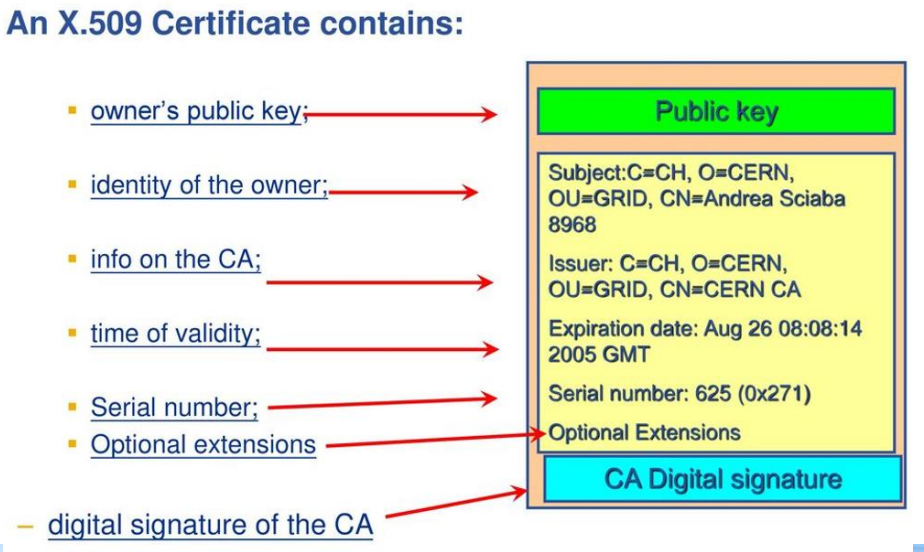
2. Authentification mutuelle des périphériques via certificats X.509 et mTLS.

a) **Certificats X.509** : Identité et Sécurité en IoT

Un certificat **X.509** est un fichier numérique qui sert à authentifier un appareil IoT. Il contient :

- Un identifiant unique (**Common Name - CN**).
- Une clé publique et la signature d'une **autorité de certification (CA)**.
- Une date d'expiration et des informations sur l'émetteur.

-----BEGIN CERTIFICATE-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8A
MIIBCgKCAQEA...
-----END CERTIFICATE-----



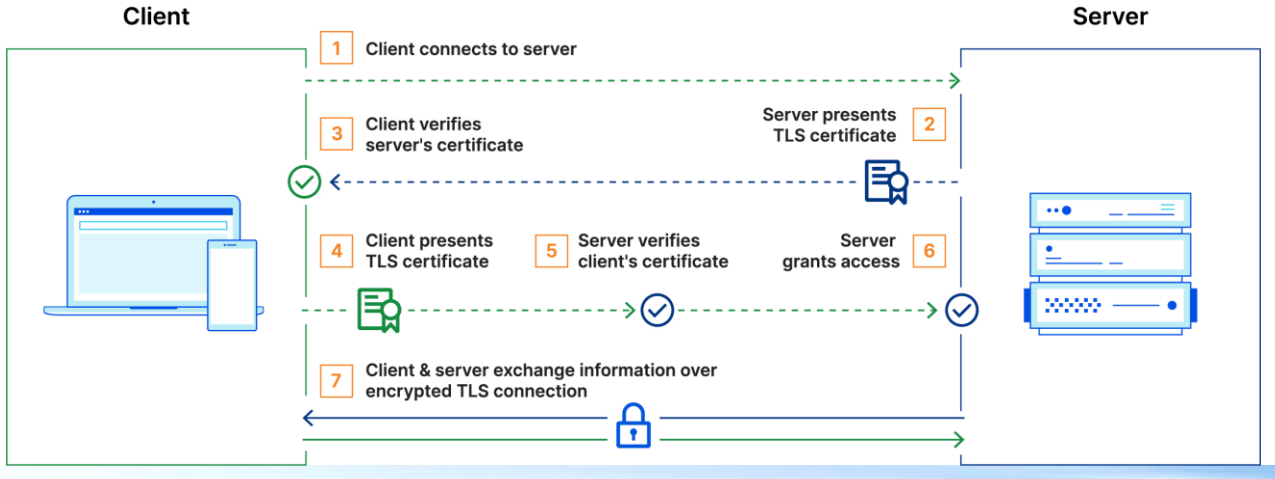
b) TLS Mutuel (mTLS) : Authentification Sécurisée

Le **TLS mutuel (mTLS)** est une version renforcée de **TLS** où le client et le serveur, possèdent chacun un certificat :

1. ***Le serveur*** *vérifie l'identité du périphérique via son certificat X.509.*
2. ***Le périphérique*** *vérifie que le serveur est authentique*

Pourquoi utiliser mTLS ?

- *Prévient les attaques **MITM (Man-In-The-Middle)**.*
- *Empêche les appareils non autorisés de se connecter.*
- *Sécurise **MQTT, HTTPS, CoAP, WebSockets**.*



1.2. Étapes pour sécuriser un périphérique IoT avec mTLS :

1. Générer une paire de clés RSA/ECC pour l'appareil

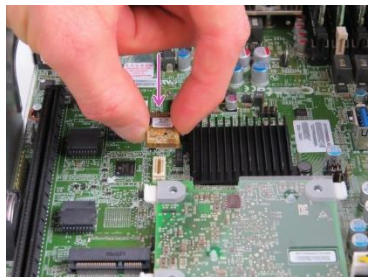
```
openssl genpkey -algorithm RSA -out device.key
openssl req -new -key device.key -out device.csr -subj "/CN=device01"
openssl x509 -req -in device.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out device.crt -days 365
```

2. Stocker le certificat et la clé privée de manière sécurisée

a) Utilisation des puces:

- **Trusted Platform Module** (**TPM**)**: puce cryptographique** intégrée dans les ordinateurs, serveurs et certains appareils IoT.
- **Secure Element (SE)**: Microcontrôleur sécurisé dédié **(**TPM, cartes bancaires, passeports électroniques)

TPM



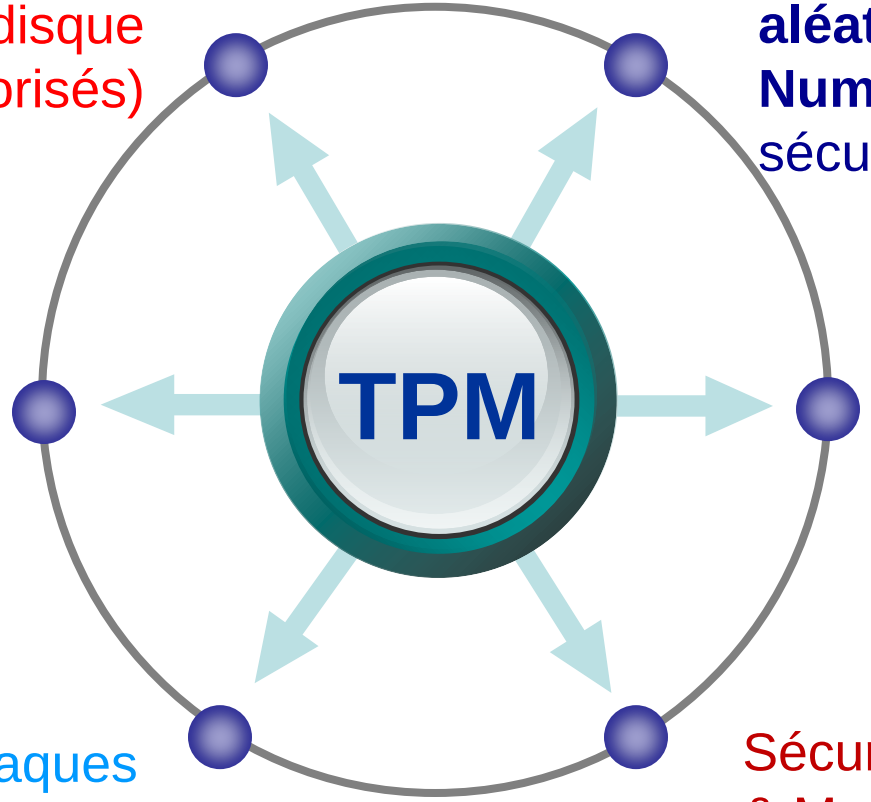
- Appuyez sur **Win + R**
- Tapez **tpm.msc** et appuyez sur **Entrée**
- Vérifiez si **TPM 2.0 est activé**

Secure Element

… Étapes pour sécuriser un périphérique IoT avec mTLS

Sécurisation du chiffrement BitLocker (pour protéger le disque dur contre les accès non autorisés)

Possède un **générateur de nombres aléatoires (TRNG - True Random Number Generator):** Renforcer la sécurité



Authentification et gestion des identités (reconnaissance faciale, empreinte digitale, code PIN)

Gestion des certificats et signatures numériques (pour stocker des **certificats SSL, des clés de chiffrement**)

Protection contre les attaques matérielles et logicielles

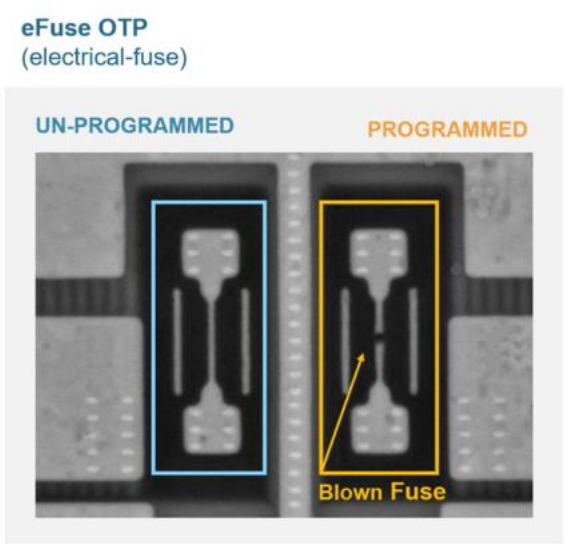
Sécurisation du démarrage (Secure Boot & Measured Boot)

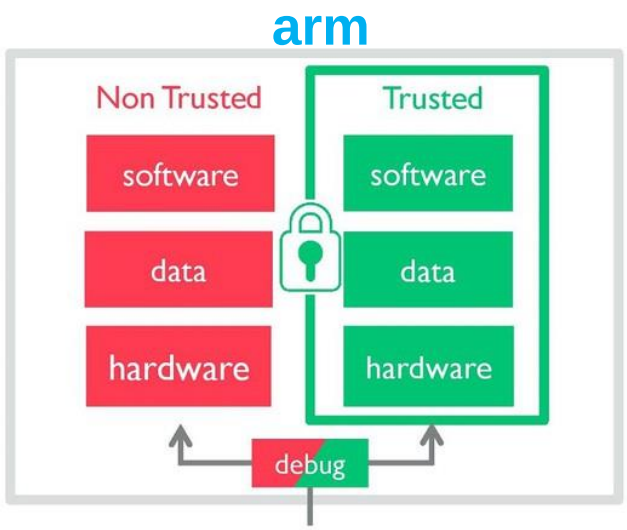
… *Étapes pour sécuriser un périphérique IoT avec mTLS*

b) Protection contre l'extraction avec:

- **eFuse (eFuses programmables: OTP – One-Time Programmable):** Circuit intégré programmable: mémoire programmable de façon irréversible: Une fois programmées, ces clés ne peuvent plus être modifiées, garantissant ainsi l'authenticité du dispositif.

- **TrustZone**: c'est une technologie de sécurité matérielle développée par **Arm** pour créer un environnement d'exécution sécurisé (**TEE - Trusted Execution Environment**) dans les processeurs embarqués (IoT, mobiles, systèmes critiques)





… *Étapes pour sécuriser un périphérique IoT avec mTLS*

- **TPM**: Sécurisation logicielle avancée, mais externe.
- **SE**: Stockage ultra-sécurisé avec isolation physique.
- **eFuse**: Une fois gravé, **non modifiable** (idéal pour empêcher modifications).
- **TrustZone**: Isolation logicielle **dans le processeur** (plus flexible, mais moins protégé qu'un SE)

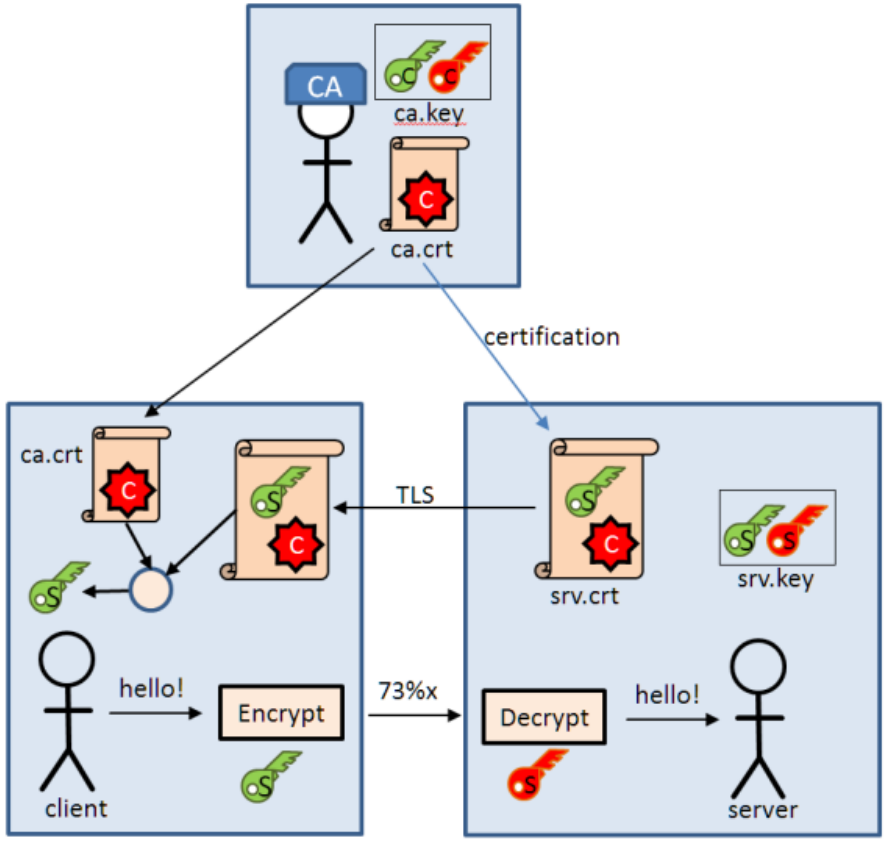
Technologie	Objectif principal	Protection contre les attaques	Intégration	Cas d'usage
TPM (Trusted Platform Module)	Stockage sécurisé des clés, attestation	Fort contre attaques logicielles	Puce externe ou firmware	PC, serveurs, Windows 11, IoT sécurisé
Secure Element (SE)	Stockage sécurisé et isolation physique	Très fort contre attaques physiques	Circuit dédié sécurisé	Paiements, cartes SIM, IoT sécurisé
eFuse	Stockage immuable d'informations (clés, configurations)	Définitif et irréversible	Intégré dans la puce (SoC)	Verrouillage de boot, protection firmware
TrustZone	Isolation logicielle des applications	Fort contre attaques logicielles	Intégré dans le processeur	Smartphones, IoT, gestion de clés, DRM

3. Configurer le serveur IoT (ex. Mosquitto, AWS IoT, Azure IoT) pour exiger mTLS

Exemple: configurer Mosquitto pour exiger mTLS:
(**voir chapitre 4 et chapitre 5**)

On doit générer des certificats avec OpenSSL

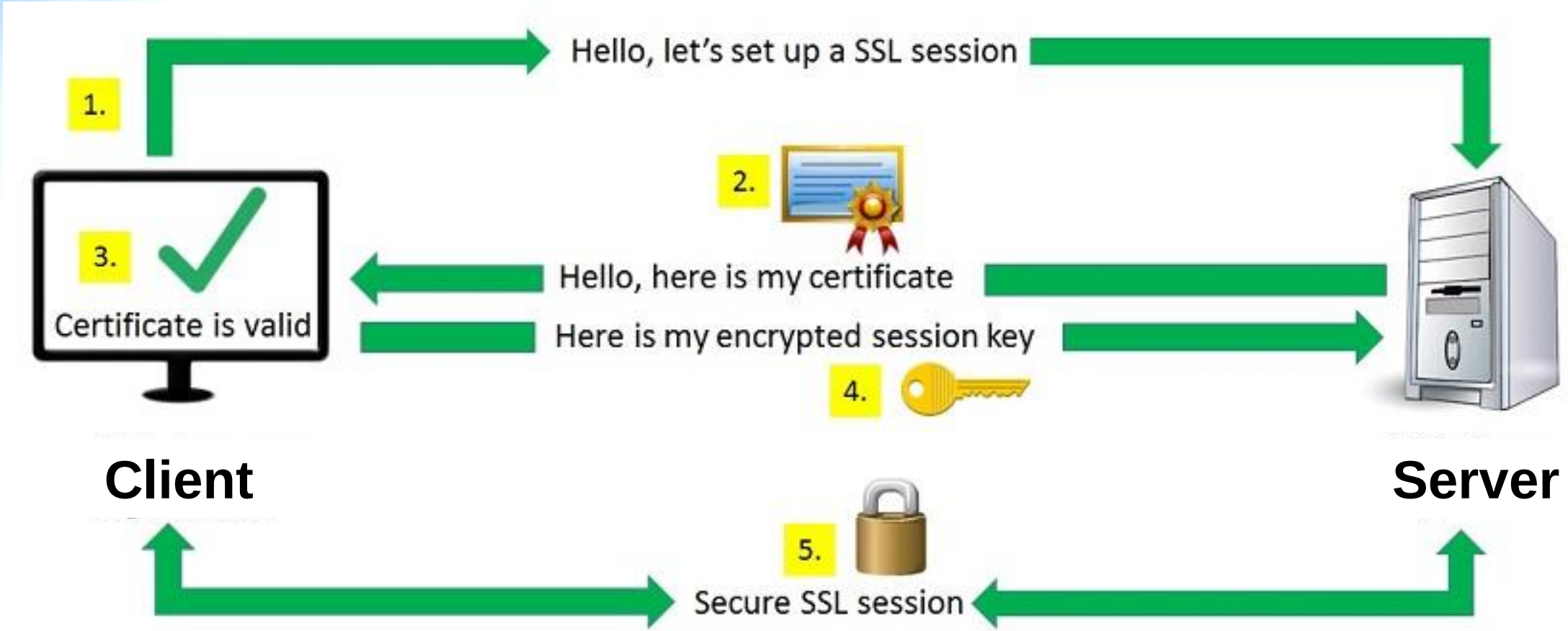
- Un **certificat d'autorité (CA)** 📜 pour signer les certificats.
- Un **certificat serveur** 🔑 pour Mosquitto.
- Un **certificat client** 🔐 pour authentifier les clients IoT.



TLS Handshaking with certificates and keys

… *Étapes pour sécuriser un périphérique IoT avec mTLS*

Processus d'établissement d'une connexion sécurisée TLS entre le client et le serveur



1.3. Protection des ports de débogage :

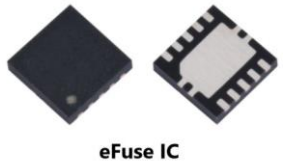
1. Désactivation des interfaces JTAG, SWD, UART en production

- **Pourquoi ?** JTAG (Joint Test Action Group), SWD (Serial Wire Debug), et UART (Universal Asynchronous Receiver-Transmitter) sont des interfaces couramment utilisées pour :
  - Déboguer du firmware.
  - Lire et écrire en mémoire.
  - Charger un nouveau firmware.

- **Problème :** En production, un attaquant peut exploiter ces interfaces pour :
  - Extraire le firmware.
  - Modifier le code et insérer une porte dérobée (backdoor).
  - Lire des informations sensibles (clés de chiffrement, mots de passe).

- **Solution : Désactivation** une fois le produit finalisé et prêt à être utilisé par les clients.

a) **Désactivation par eFuse (Permanent – Méthode la plus courante):** Brûler des **eFuses** pour désactiver définitivement JTAG/SWD et empêcher tout reflashage non autorisé. Méthode utilisée sur les **microcontrôleurs modernes (**ESP32, STM32)



Exemples sur ESP32 (Code en C):

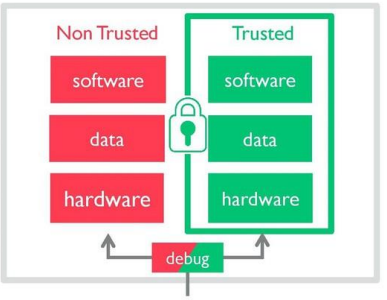
```
esp_efuse_write_field_bit(ESP_EFUSE_DISABLE_JTAG);
```

b) **Désactivation logicielle (Temporaire – Utilisé en test):** Configurer les registres pour **désactiver JTAG/SWD au démarrage** du firmware.

Exemple sur STM32 (Code en C):

```
__HAL_DBGMCU_FREEZE_IWDG(); // Désactive JTAG en mode débogage
```

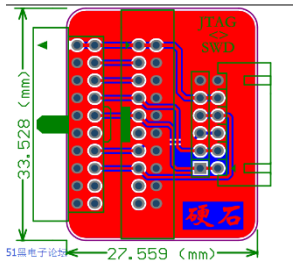
c) Activation de TrustZone ou Secure Boot (Sécurité avancée)



- Activer **TrustZone** (ARM Cortex-M33, A) ou **Secure Boot** pour empêcher l'exécution de firmwares modifiés.
- Empêche un attaquant d'activer JTAG/SWD même si eFuse n'est pas grillé.
- Exemple sur STM32 avec TrustZone activé

HAL_SAU_ConfigRegion(...); // Configure une région sécurisée

d) Désactivation physique des broches (Définitif)



- Couper les pistes JTAG/SWD sur la carte PCB.
- Ne pas connecter les broches à un connecteur.
- Très courant dans les produits IoT sécurisés (montres connectées, objets industriels).

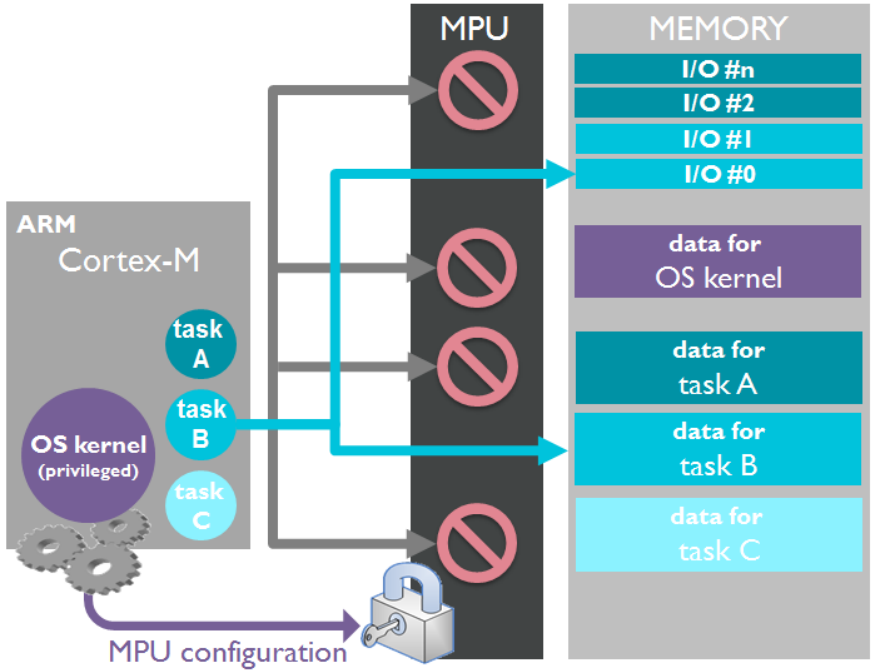
1.4. Contrôle des accès mémoire

La protection des accès mémoire est essentielle pour sécuriser un système embarqué contre les attaques (buffer overflow, exécution de code arbitraire, etc.). Voici comment on met en place ces protections :

1. Utilisation de la Memory Protection Unit (MPU):

La **Memory Protection Unit (MPU)** est un composant matériel présent dans certains microcontrôleurs (ex: ARM Cortex-M) permettant de :

- **Restreindre l'accès à certaines régions mémoire** (lecture/écriture/exécution).
- **Empêcher un processus d'accéder à la mémoire d'un autre**.
- **Éviter l'exécution de code dans des zones mémoire non prévues** (ex: stack).



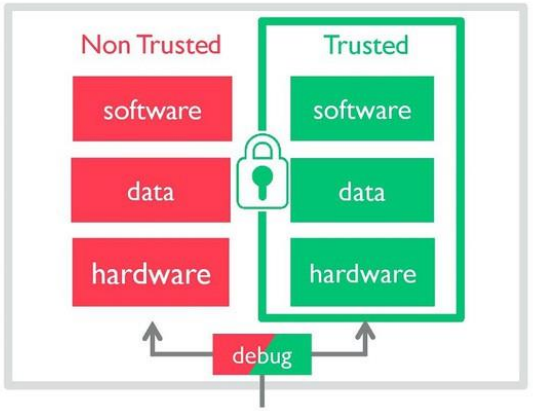
Exemple de configuration MPU sur un Cortex-M4 (ex: STM32F4):

Ce code protège une région mémoire contre l'écriture, empêchant ainsi certaines attaques mémoire.

```
void MPU_Config(void) {
   MPU_Region_InitTypeDef MPU_InitStruct;

   HAL_MPU_Disable();  // Désactiver temporairement la MPU

   // Définir une région protégée (ex: empêcher l'écriture sur une zone critique)
   MPU_InitStruct.Enable          = MPU_REGION_ENABLE;
   MPU_InitStruct.BaseAddress     = 0x20000000; // Début de la RAM
   MPU_InitStruct.Size            = MPU_REGION_SIZE_64KB; // Taille de la région
   MPU_InitStruct.AccessPermission = MPU_REGION_PRIV_RO; // Lecture seule pour mode privilège
   MPU_InitStruct.IsBufferable    = MPU_ACCESS_NOT_BUFFERABLE;
   MPU_InitStruct.IsCacheable     = MPU_ACCESS_NOT_CACHEABLE;
   MPU_InitStruct.IsShareable     = MPU_ACCESS_NOT_SHAREABLE;
   MPU_InitStruct.Number          = MPU_REGION_NUMBER0;
   HAL_MPU_ConfigRegion(&MPU_InitStruct);

   HAL_MPU_Enable(MPU_PRIVILEGED_DEFAULT);  // Réactiver la MPU
}
```

2. TrustZone : Séparation mémoire en "Secure" et "Non-Secure"

- Présent sur **Cortex-M23, M33, M55** et certains **Cortex-A**.
- Permet de créer **deux espaces mémoire distincts** :
  - ***Secure World*** *: exécution du code sensible (crypto, clés, mises à jour).*
  - ***Non-Secure World*** *: exécution du reste de l'application (interfaces, réseau...).*



Exemple d'utilisation de TrustZone:

a) Séparation de la Flash et RAM :
- *Une partie de la mémoire Flash est réservée au code sécurisé.*
- *Une partie de la RAM est protégée contre les accès non sécurisés.*

b) Contrôle d'accès avec SAU (Secure Attribution Unit) :

```
SAU->RNR  = 0;          // Sélectionner la région 0
SAU->RBAR = 0x08000000; // Début de la région sécurisée
SAU->RLAR = 0x0800FFFF; // Fin de la région sécurisée
SAU->CTRL = 1;          // Activer TrustZone
```

2. Mise à jour sécurisée du firmware (OTA) et rollback protection

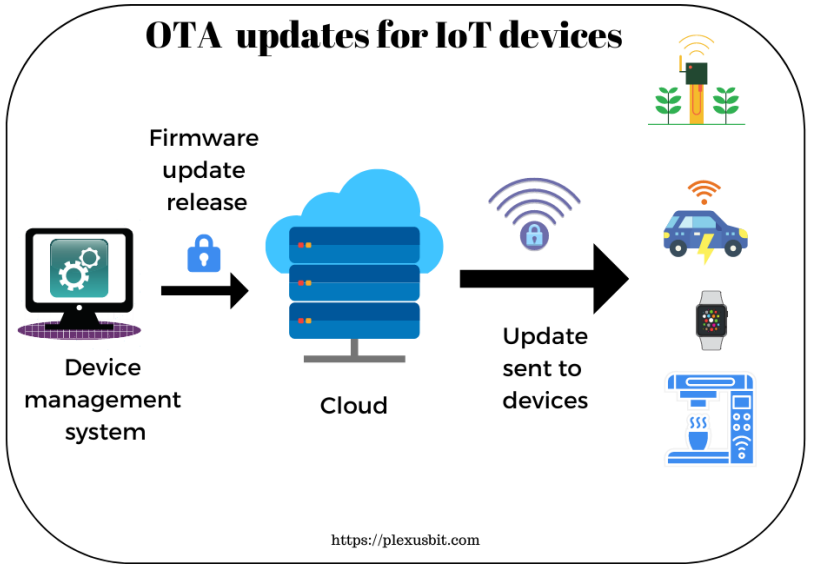
2.1. Mises à jour Over-the-Air (OTA):

- Les mises à jour **OTA** permettent d'installer un nouveau firmware à distance sur un appareil IoT.
- Cependant, elles représentent une cible privilégiée pour les attaques.
- Une mise à jour mal sécurisée peut permettre à un attaquant de **remplacer le firmware par un code malveillant** ou **d'exploiter une ancienne faille de sécurité**

Pourquoi sécuriser les mises à jour OTA ?

Un attaquant peut tenter de :

- **Intercepter et modifier** une mise à jour en transit.
- **Installer un firmware non autorisé** en exploitant une faille.
- **Forcer un retour (rollback) vers une version vulnérable** pour exploiter une faille connue.



2.2. Sécurisation des mises à jour OTA

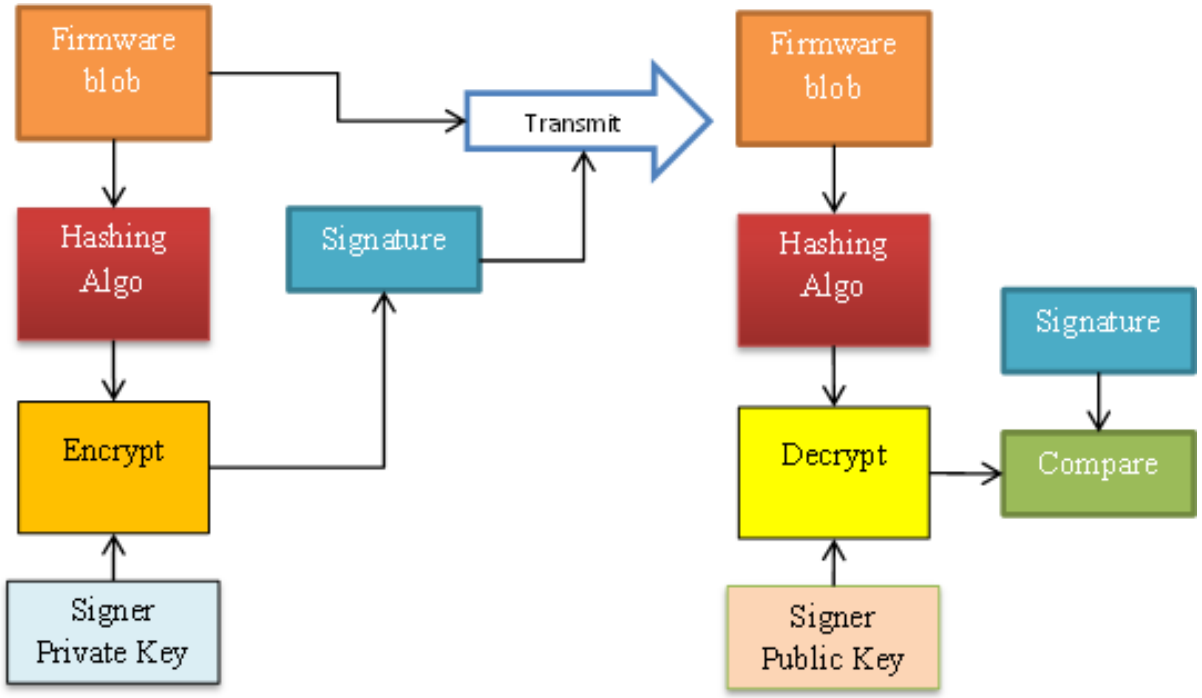
1. Authentification et chiffrement des mises à jour :

Pour empêcher un attaquant d'injecter un firmware malveillant, il faut : **TLS 1.2 ou TLS 1.3** pour chiffrer les mises à jour.

- **Authentification du serveur** pour s'assurer que le firmware provient d'une source de confiance.
- **Validation côté client** du certificat du serveur (PKI, certificats X.509).
- **Exemple :**
  - Un serveur OTA envoie un firmware signé.
  - L'appareil IoT télécharge le firmware via **HTTPS (TLS 1.2+)**.
  - Il vérifie la signature numérique avant d'installer le firmware.
- **AWS IoT, Azure IoT Hub, Google IoT Core** proposent des services OTA sécurisés

2. Signature numérique du firmware (garantir l'intégrité)

- Chaque mise à jour doit être signée numériquement par le fabricant.
  - **Signature RSA ou ECDSA** pour garantir que le firmware n'a pas été modifié.
  - **Vérification avant installation** : l'appareil IoT doit refuser un firmware dont la signature ne correspond pas.



Firmware Blob *(ou* ***binary blob****) est un* ***microprogramme précompilé*** *livré sous forme de* ***fichier binaire*** *(et non sous forme de code source lisible)*

Exemple avec OpenSSL pour signer un firmware :

```
# Générer une clé privée RSA 2048 bits
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048

# Extraire la clé publique
openssl rsa -in private_key.pem -pubout -out public_key.pem

# Signer le firmware
openssl dgst -sha256 -sign private_key.pem -out firmware.sig firmware.bin

# Vérifier la signature sur l'appareil
openssl dgst -sha256 -verify public_key.pem -signature firmware.sig firmware.bin
```

Cette commande permet de **vérifier l'authenticité et l'intégrité** du firmware (**firmware.bin**) en utilisant une **clé publique** (**public_key.pem**).

Cette commande permet de **signer numériquement** un fichier firmware (**firmware.bin**) en utilisant une clé privée RSA (**private_key.pem**).

2.3. Protection contre le rollback (retour à une version vulnérable)

- Un attaquant pourrait tenter d'installer une ancienne version du firmware contenant une faille connue.
- Il faut donc **empêcher l'appareil de rétrograder vers un firmware plus ancien**.

1. Utilisation d'un compteur OTP (One-Time Programmable) :

- L'appareil conserve **un numéro de version du firmware** dans une mémoire **OTP (One-Time Programmable)** ou **eFuse**.
- **Si une mise à jour tente d'installer un firmware plus ancien, l'appareil refuse l'installation.**

Exemple sur ESP32 avec eFuse :

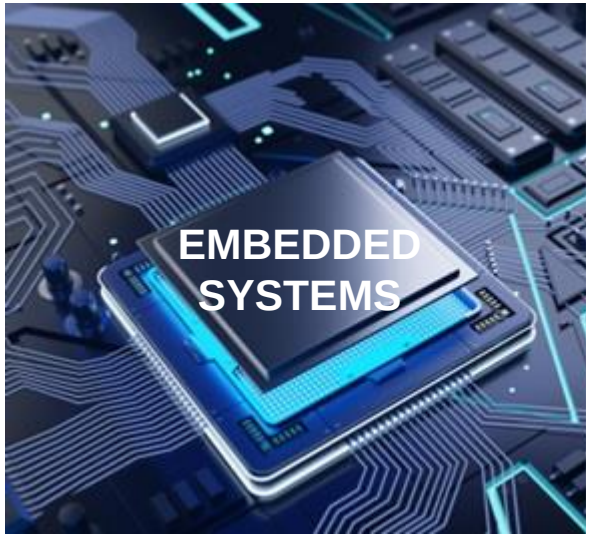
```
#include "esp_efuse.h"
#include "esp_efuse_table.h"

// Lire la version actuelle
uint32_t version = 0;
esp_efuse_read_field_blob(ESP_EFUSE_SECURE_VERSION, &version, 32);

// Comparer avec la version du nouveau firmware
if (new_firmware_version > version) {
    // Mise à jour autorisée : on met à jour l'eFuse
    esp_efuse_write_field_blob(ESP_EFUSE_SECURE_VERSION, &new_firmware_version, 32);
} else {
    // Rollback détecté : mise à jour refusée
}
```

2. Vérification de la signature et de la version avant exécution:

- Lors du démarrage, le bootloader vérifie :
  - **La signature du firmware** (RSA, ECDSA).
  - **Le numéro de version** dans le firmware.
- Si la version est inférieure à la dernière version enregistrée, **le firmware ne démarre pas**.

Exemple de validation dans un bootloader sécurisé :

```
if (firmware_version < stored_version) {
    printf("Firmware trop ancien, rollback détecté !");
    reboot_to_factory();
}
```



FIN !
Questions ?