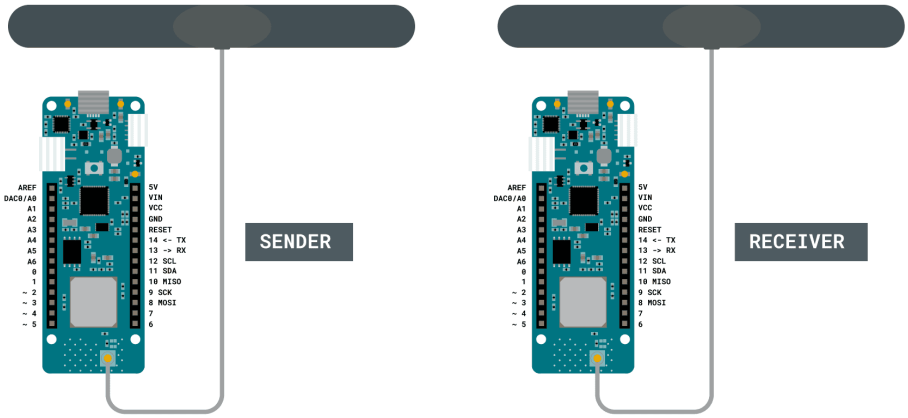
TP 5
PROTECTION ANTI-REJOUE D'UNE TRANSMISSION LORA PAIR À PAIR.

Objectifs :

- Faire communiquer plusieurs équipements LoRa de manière sécurisée



1. TABLEAU DE REPARTITION DES BINOMES

1.1. Cocher et noter le numéro de binôme que le professeur vous a attribué, relever la fréquence et le facteur d'étalement de votre binôme. Il faudra les remplacer dans les programmes.

binôme	Fréquence	Facteur d'étalement (Spreading Factor)
1 ❑	867100000	7
2 ❑	867100000	8
3 ❑	867500000	7
4 ❑	867500000	8
5 ❑	867900000	7
6 ❑	867900000	8
7 ❑	868300000	7
8 ❑	868300000	8

2. TRAVAIL A EFFECTUER EN BINÔME **CHAQUE ELEVE REALISE SA PARTIE**

Les cartes utilisées sont des Arduino MKR 1310.

Élève 1 : carte émettrice :

2.1. Lancer Arduino choisir le type de carte **Arduino SAMD Boards / Arduino MKR WAN 1310.**

2.2. Installer les bibliothèques **lora Sandeep Mistry** et **Crypto**

2.3 Créer un fichier LoraSender avec à partir du fichier :
https://github.com/bouhenic/FormationIOT/blob/main/TP6Lora2Lora/ noReplay-emitter-lora.ino

2.4 Créer un fichier HMAC.h dans le même dossier que LoraReceiver à partir du fichier : **https://github.com/bouhenic/FormationIOT/blob/main/TP6Lora2Lora/HMAC.h**

2.5 Créer un fichier HMAC.cpp dans le même dossier que LoraReceiver à partir du fichier : **https://github.com/bouhenic/FormationIOT/blob/main/TP6Lora2Lora/HMAC.cpp**

2.6 Modifier le fichier de la manière suivante :

- En jaune les lignes à modifier
- En vert la valeur du facteur d'étalement de votre binôme.

```
#include <LoRa.h>
#include <Crypto.h>
#include <AES.h>
#include "HMAC.h"

// Clé de chiffrement AES (16 octets pour AES-128)
const byte aesKey[16] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                          0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F };

// Clé HMAC (32 octets ici)
const byte hmacKey[32] = { 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
                           0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
                           0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
                           0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F };

// Compteur global pour identifier les messages (initialisé à 0)
unsigned long frameCounter = 0;

void setup() {
    Serial.begin(9600);
    while (!Serial);

    Serial.println("Initialisation de l'émetteur LoRa...");

    // Initialisation LoRa
    if (!LoRa.begin(867.5E6)) {
        Serial.println("Erreur lors de l'initialisation LoRa");
        while (1);
    }

    LoRa.setSpreadingFactor(7);
    LoRa.setSignalBandwidth(125E3);
    LoRa.setCodingRate4(5);
    LoRa.setTxPower(14);

    Serial.println("LoRa prêt !");
}

void loop() {
```

```
    char message[] = "Hello LoRa!"; // Message à envoyer
    byte plainMessage[32] = {0};    // Message en clair incluant le compteur
    byte encryptedMessage[32] = {0}; // Buffer pour le message chiffré
    byte hmac[32];                  // Buffer pour le HMAC calculé

    // Construction du message clair
    memset(plainMessage, 0, sizeof(plainMessage)); // Effacer le buffer
    memcpy(plainMessage, message, strlen(message)); // Copier le texte du message
    plainMessage[28] = frameCounter & 0xFF;           // Byte 0 du compteur
    plainMessage[29] = (frameCounter >> 8) & 0xFF;    // Byte 1
    plainMessage[30] = (frameCounter >> 16) & 0xFF;   // Byte 2
    plainMessage[31] = (frameCounter >> 24) & 0xFF;   // Byte 3

    // Afficher le message clair (avec compteur)
    Serial.print("Message clair (avec compteur) : ");
    for (int i = 0; i < 32; i++) {
        Serial.print(plainMessage[i], HEX);
        Serial.print(" ");
    }
    Serial.println();

    // Initialisation de l'objet AES
    AES128 aesEncryptor;
    aesEncryptor.setKey(aesKey, sizeof(aesKey));

    // Chiffrement par blocs de 16 octets
    aesEncryptor.encryptBlock(encryptedMessage, plainMessage);       // Bloc 1
    aesEncryptor.encryptBlock(encryptedMessage + 16, plainMessage + 16); // Bloc 2

    // Afficher le message chiffré
    Serial.print("Message chiffré : ");
    for (int i = 0; i < 32; i++) {
        Serial.print(encryptedMessage[i], HEX);
        Serial.print(" ");
    }
    Serial.println();

    // Calcul du HMAC
    HMAC::calculateHMAC(hmacKey, sizeof(hmacKey), encryptedMessage, sizeof(encryptedMessage), hmac);

    // Préparation du payload (chiffrement + HMAC court)
    byte payload[32 + 8]; // Chiffrement + 8 octets du HMAC
    memcpy(payload, encryptedMessage, sizeof(encryptedMessage));
    memcpy(payload + sizeof(encryptedMessage), hmac, 8); // HMAC court

    // Envoi des données via LoRa
    LoRa.beginPacket();
    LoRa.write(payload, sizeof(payload));
    LoRa.endPacket();

    // Affichage du HMAC envoyé
    Serial.print("HMAC envoyé (8 octets) : ");
    for (int i = 0; i < 8; i++) {
        Serial.print(hmac[i], HEX);
        Serial.print(" ");
    }
    Serial.println();

    // Affichage du compteur
    Serial.print("Compteur : ");
    Serial.println(frameCounter);

    frameCounter++; // Incrémentation du compteur
    delay(2000);    // Pause avant d'envoyer un nouveau message
}
```

Objectif de la protection anti-rejoue

- **Empêcher les attaques par rejeu :** L'attaquant ne peut pas retransmettre un message valide intercepté pour tromper le récepteur.

- **Assurer l'unicité des messages :** Chaque trame envoyée doit être identifiable de manière unique par le récepteur.

Analyse du code émetteur

Déclaration et initialisation du compteur

```
unsigned long frameCounter = 0;
```

- frameCounter est une variable globale de type unsigned long (4 octets) qui sert de **nonce** (**number used once)** unique pour chaque message.
- Elle est initialisée à 0 au démarrage de l'émetteur.

Ajout du compteur au message clair

```
plainMessage[28] = frameCounter & 0xFF;        // Byte 0 du compteur
plainMessage[29] = (frameCounter >> 8) & 0xFF; // Byte 1
plainMessage[30] = (frameCounter >> 16) & 0xFF;// Byte 2
plainMessage[31] = (frameCounter >> 24) & 0xFF;// Byte 3
```

- Le compteur (frameCounter) est ajouté à la fin du tableau plainMessage aux indices 28 à 31 (4 derniers octets du message clair).
- Chaque octet du compteur est extrait avec un **décalage binaire** et stocké individuellement :
  - `frameCounter & 0xFF` : Octet de poids faible (LSB).
  - `(frameCounter >> 8) & 0xFF` : Octet suivant.
  - `(frameCounter >> 16) & 0xFF` : Troisième octet.
  - `(frameCounter >> 24) & 0xFF` : Octet de poids fort (MSB).

  - Cela permet de représenter le compteur sur 4 octets.

Incrémentation du compteur

```
frameCounter++;
```

- Après chaque envoi, le compteur est incrémenté de 1 pour garantir l'unicité des messages.
- Si un attaquant capture un message, il ne pourra pas rejouer celui-ci, car le récepteur n'acceptera pas un compteur déjà vu ou inférieur au dernier compteur.

Inclusion dans le chiffrement

```
aesEncryptor.encryptBlock(encryptedMessage, plainMessage);
aesEncryptor.encryptBlock(encryptedMessage + 16, plainMessage + 16);
```

- Le message clair (plainMessage), qui inclut le texte et le compteur, est chiffré avec AES en deux blocs de 16 octets.
- Cela garantit que :
  - Le compteur est protégé contre les modifications.
  - Si un attaquant modifie le message ou le compteur, le HMAC calculé ne correspondra pas, et le récepteur rejettera la trame.

Authentification avec le HMAC

```
HMAC::calculateHMAC(hmacKey, sizeof(hmacKey), encryptedMessage, sizeof(encryptedMessage), hmac);
```

- Le HMAC est calculé sur le message chiffré, qui inclut le compteur.
- Si un attaquant tente de rejouer ou de modifier un message, le HMAC recalculé par le récepteur ne correspondra pas, et le message sera rejeté.

Envoi du message avec le compteur intégré

```
byte payload[32 + 8];
memcpy(payload, encryptedMessage, sizeof(encryptedMessage));
memcpy(payload + sizeof(encryptedMessage), hmac, 8);
LoRa.beginPacket();
LoRa.write(payload, sizeof(payload));
LoRa.endPacket();
```

- Le message final envoyé contient :

1. **Les 32 octets chiffrés :** Incluent le texte et le compteur.
2. **Les 8 premiers octets du HMAC :** Permettent au récepteur de vérifier l'intégrité et l'authenticité du message.

- Le compteur est maintenant protégé à la fois par le chiffrement et l'HMAC.

Élève 2 : carte réceptrice :

2.7 Lancer Arduino choisir le type de carte **Arduino SAMD Boards / Arduino MKR WAN 1310.**

2.8 Installer les bibliothèques **lora Sandeep Mistry** et **Crypto**

2.9 Créer un fichier LoraReceiver avec à partir du fichier :
https://github.com/bouhenic/FormationIOT/blob/main/TP5Lora2Lora/ noReplay-receiver-lora.ino

2.8 Créer un fichier HMAC.h dans le même dossier que LoraReceiver à partir du fichier : `https://github.com/bouhenic/FormationIOT/blob/main/TP6Lora2Lora/HMAC.h`

2.9 Créer un fichier HMAC.cpp dans le même dossier que LoraReceiver à partir du fichier : `https://github.com/bouhenic/FormationIOT/blob/main/TP6Lora2Lora/HMAC.cpp`

2.10 Modifier le fichier de la manière suivante :

- En jaune les lignes à modifier
- En vert la valeur du facteur d'étalement de votre binôme.

```
#include <LoRa.h>
#include <Crypto.h>
#include <AES.h>
#include "HMAC.h"

// Clé de chiffrement AES (16 octets pour AES-128)
const byte aesKey[16] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                          0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F };

// Clé HMAC (32 octets ici)
const byte hmacKey[32] = { 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
                           0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
                           0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
```

```
                          0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F };

// Compteur global pour identifier les messages (initialisé à 0)
unsigned long frameCounter = 0;

void setup() {
    Serial.begin(9600);
    while (!Serial);

    Serial.println("Initialisation de l'émetteur LoRa...");

    // Initialisation LoRa
    if (!LoRa.begin(867.5E6)) {
        Serial.println("Erreur lors de l'initialisation LoRa");
        while (1);
    }

    LoRa.setSpreadingFactor(7);
    LoRa.setSignalBandwidth(125E3);
    LoRa.setCodingRate4(5);
    LoRa.setTxPower(14);

    Serial.println("LoRa prêt !");
}

void loop() {
    char message[] = "Hello LoRa!"; // Message à envoyer
    byte plainMessage[32] = {0};    // Message en clair incluant le compteur
    byte encryptedMessage[32] = {0}; // Buffer pour le message chiffré
    byte hmac[32];                  // Buffer pour le HMAC calculé

    // Construction du message clair
    memset(plainMessage, 0, sizeof(plainMessage)); // Effacer le buffer
    memcpy(plainMessage, message, strlen(message)); // Copier le texte du message
    plainMessage[28] = frameCounter & 0xFF;           // Byte 0 du compteur
    plainMessage[29] = (frameCounter >> 8) & 0xFF;    // Byte 1
    plainMessage[30] = (frameCounter >> 16) & 0xFF;   // Byte 2
    plainMessage[31] = (frameCounter >> 24) & 0xFF;   // Byte 3

    // Afficher le message clair (avec compteur)
    Serial.print("Message clair (avec compteur) : ");
    for (int i = 0; i < 32; i++) {
        Serial.print(plainMessage[i], HEX);
        Serial.print(" ");
    }
    Serial.println();

    // Initialisation de l'objet AES
    AES128 aesEncryptor;
    aesEncryptor.setKey(aesKey, sizeof(aesKey));

    // Chiffrement par blocs de 16 octets
    aesEncryptor.encryptBlock(encryptedMessage, plainMessage);       // Bloc 1
    aesEncryptor.encryptBlock(encryptedMessage + 16, plainMessage + 16); // Bloc 2

    // Afficher le message chiffré
    Serial.print("Message chiffré : ");
    for (int i = 0; i < 32; i++) {
        Serial.print(encryptedMessage[i], HEX);
        Serial.print(" ");
    }
    Serial.println();

    // Calcul du HMAC
    HMAC::calculateHMAC(hmacKey, sizeof(hmacKey), encryptedMessage, sizeof(encryptedMessage), hmac);

    // Préparation du payload (chiffrement + HMAC court)
    byte payload[32 + 8]; // Chiffrement + 8 octets du HMAC
    memcpy(payload, encryptedMessage, sizeof(encryptedMessage));
    memcpy(payload + sizeof(encryptedMessage), hmac, 8); // HMAC court

    // Envoi des données via LoRa
    LoRa.beginPacket();
    LoRa.write(payload, sizeof(payload));
```

```
    LoRa.endPacket();

    // Affichage du HMAC envoyé
    Serial.print("HMAC envoyé (8 octets) : ");
    for (int i = 0; i < 8; i++) {
        Serial.print(hmac[i], HEX);
        Serial.print(" ");
    }
    Serial.println();

    // Affichage du compteur
    Serial.print("Compteur : ");
    Serial.println(frameCounter);

    frameCounter++; // Incrémentation du compteur
    delay(2000);    // Pause avant d'envoyer un nouveau message
}
```

Objectif de la protection anti-rejoue

Empêcher un attaquant de rejouer une trame interceptée (même valide) en s'assurant que :

- Chaque trame reçue est unique grâce à un compteur.
- Le récepteur rejette automatiquement toute trame avec un compteur inférieur ou égal au dernier reçu.

Parties clés du code récepteur pour l'anti-rejoue

1. Stockage du dernier compteur reçu

`unsigned long lastFrameCounter = 0;` // Dernier compteur reçu

- Cette variable garde en mémoire le dernier compteur reçu (et accepté) par le récepteur.
- Toute trame avec un compteur inférieur ou égal à lastFrameCounter sera rejetée comme une attaque par rejeu.

2. Extraction du compteur depuis le message déchiffré

```
receivedFrameCounter = (decryptedMessage[28]) |
                       (decryptedMessage[29] << 8) |
                       (decryptedMessage[30] << 16) |
                      (decryptedMessage[31] << 24);
```

- Les 4 derniers octets du message déchiffré (indices 28 à 31) contiennent le compteur envoyé par l'émetteur.
- Chaque octet est extrait et combiné pour reconstruire un entier de 32 bits (unsigned long).

Exemple :
Si les octets sont :
decryptedMessage[28] = 0x01
decryptedMessage[29] = 0x00
decryptedMessage[30] = 0x00
decryptedMessage[31] = 0x00

Le compteur sera reconstruit comme :
`receivedFrameCounter = 0x00000001 = 1`

3. Vérification du compteur

```
if (receivedFrameCounter <= lastFrameCounter) {
Serial.println("Rejeu détecté : compteur invalide");
return; // Rejeter la trame
}
```

- Si le compteur reçu (**`receivedFrameCounter`**) est inférieur ou égal au dernier compteur stocké (`lastFrameCounter`), la trame est rejetée.
- Cela signifie que la trame a déjà été acceptée ou qu'elle est invalide (potentiellement une attaque par rejeu).

4. Mise à jour du compteur

```
lastFrameCounter = receivedFrameCounter;
```

- Après avoir validé une trame, le récepteur met à jour lastFrameCounter pour refléter le compteur le plus récent.
- Cela garantit que les trames futures avec des compteurs inférieurs ou égaux seront rejetées.

Protection contre les attaques par rejeu : Exemple détaillé

Cas 1 : Message valide

1. Dernier compteur reçu (lastFrameCounter) = 5.
2. Message reçu avec compteur (receivedFrameCounter) = 6.
3. Condition :

```
if (6 <= 5) // Faux
```

- Le message est accepté.
- lastFrameCounter est mis à jour à 6.

Cas 2 : Attaque par rejeu

1. Dernier compteur reçu (lastFrameCounter) = 6.
2. Message rejoué avec compteur (receivedFrameCounter) = 5.
3. Condition :

```
if (5 <= 6) // Vrai
```

- Le message est rejeté.
- Aucune mise à jour du compteur.

Cas 3 : Message désordonné ou invalide

1. Dernier compteur reçu (lastFrameCounter) = 10.
2. Message reçu avec compteur (receivedFrameCounter) = 9.
3. Condition :

```
if (9 <= 10) // Vrai
```

- Le message est rejeté.

Autres parties liées à l'anti-rejoue

Authentification du message avec HMAC

Avant même de vérifier le compteur, le récepteur s'assure que le message n'a pas été modifié en validant le HMAC :

```
if (!hmacValid) {
    Serial.println("Authentification échouée : HMAC invalide");
    return; // Rejeter la trame
}
```

- Cela protège contre les attaques où l'attaquant tente de modifier le compteur ou d'autres parties du message.

2.11 Compiler et téléverser le programme, observer les serial monitor des émetteurs et récepteurs.