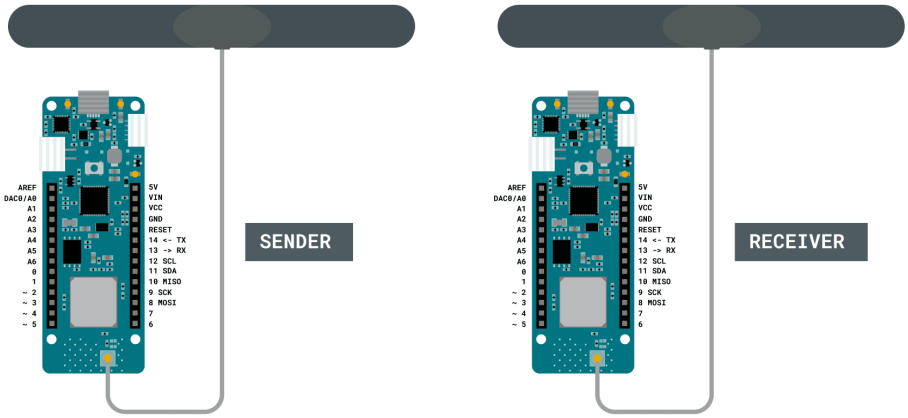
TP 5
AUTHENTIFICATION D'UNE TRANSMISSION LORA PAIR À PAIR.

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/TP5Lora2Lora/emitter-hmac-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/TP5Lora2Lora/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/TP5Lora2Lora/HMAC.cpp**

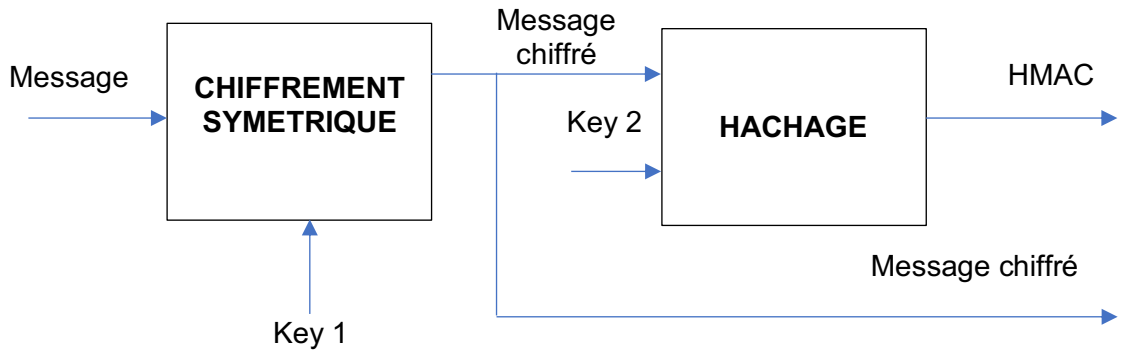
2.6 Modifier le fichier de la manière suivante :

- 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 };

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 encryptedMessage[16];          // Buffer pour le message chiffré
    byte hmac[32];                      // Buffer pour le HMAC calculé
```

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

    // Chiffrement AES
    aesEncryptor.encryptBlock(encryptedMessage, (byte*)message);

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

    // Envoi des données via LoRa
    LoRa.beginPacket();
    LoRa.write(encryptedMessage, sizeof(encryptedMessage)); // Envoi du message chiffré
    LoRa.write(hmac, 8); // Envoi des 8 premiers octets du HMAC
    LoRa.endPacket();

    // Affichage pour débogage
    Serial.print("Message chiffré envoyé : ");
    for (int i = 0; i < sizeof(encryptedMessage); i++) {
        Serial.print(encryptedMessage[i], HEX);
        Serial.print(" ");
    }
    Serial.println();

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

    delay(2000); // Pause avant d'envoyer un nouveau message
}
```

Explication du chiffrement et de l'authentification :



Fonctionnement d'un HMAC

1. **Entrées principales** :
   - Un **message** (les données que vous voulez envoyer).
   - Une **clé secrète** (connue seulement par l'émetteur et le récepteur).
   - Une **fonction de hachage cryptographique** (par exemple SHA-256, SHA-1, etc.).
2. **Étapes de calcul du HMAC** :
   - On applique la fonction de hachage deux fois, en combinant la clé secrète et le message :
     1. **Clé et remplissage :** La clé est formatée (ou tronquée) pour s'adapter à la taille du bloc de la fonction de hachage.
     2. **Première étape :** Hachage de la clé combinée avec un "ipad" (un motif de remplissage interne).

3. **Deuxième étape :** Hachage de la sortie précédente combinée avec un "opad" (un motif de remplissage externe).
   - Le résultat final est le **HMAC**, une signature unique pour le message.

Dans notre code, voici comment le HMAC est calculé :

1. **Clé HMAC :**

```
const byte hmacKey[32] = { 0x10, 0x11, 0x12, ... };
```

- Cette clé secrète est partagée entre l'émetteur et le récepteur.
- Elle garantit que seul le détenteur de la clé peut calculer le bon HMAC.

2. **Message chiffré :**

```
aesEncryptor.encryptBlock(encryptedMessage, (byte*)message);
```

- Le message est chiffré avec AES avant le calcul du HMAC. Cela garantit la confidentialité.

3. **Calcul du HMAC :**

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

La fonction **calculateHMAC** prend en entrée la clé HMAC, le message chiffré, et calcule un code d'authentification.

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/receiver-hmac-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/TP5Lora2Lora/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/TP5Lora2Lora/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.

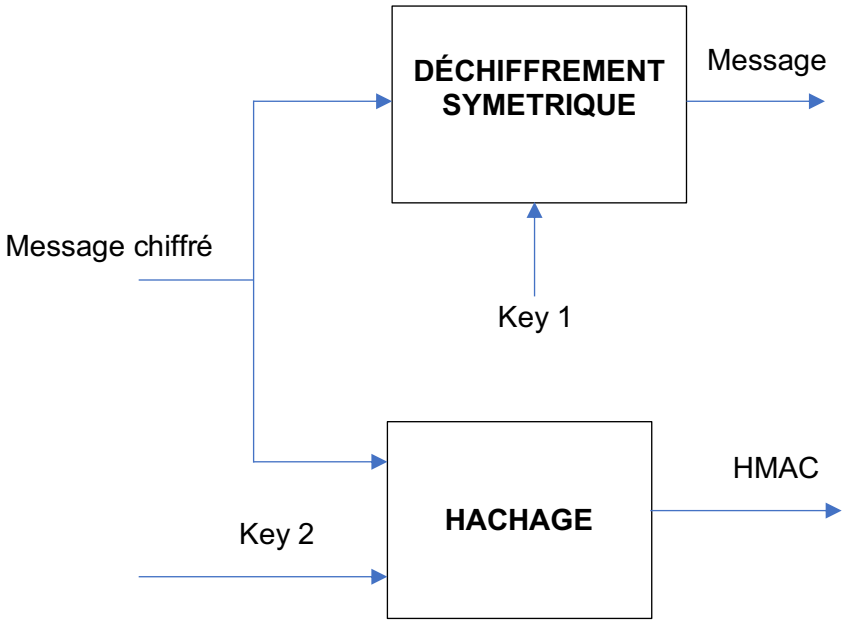
```
#include <LoRa.h>
#include <Crypto.h>
#include <AES.h>
#include "HMAC.h"

// Clé de déchiffrement 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 };

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

  Serial.println("Initialisation du récepteur 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);

  Serial.println("LoRa prêt !");
}

void loop() {
  int packetSize = LoRa.parsePacket();
  if (packetSize) {
    Serial.println("Message reçu :");

    byte encryptedMessage[16];  // Buffer pour les données chiffrées reçues
    byte receivedHMAC[8];       // Buffer pour les 8 premiers octets du HMAC reçu
    byte calculatedHMAC[32];    // Buffer pour le HMAC recalculé
    byte decryptedMessage[17];  // Buffer pour le message déchiffré (16 + 1 pour '\0')

    // Lire les données chiffrées
    for (int i = 0; i < 16; i++) {
      if (LoRa.available()) {
        encryptedMessage[i] = LoRa.read();
      }
    }

    // Lire le HMAC reçu
    for (int i = 0; i < 8; i++) {
      if (LoRa.available()) {
        receivedHMAC[i] = LoRa.read();
      }
    }

    // Recalculer le HMAC pour authentification
    HMAC::calculateHMAC(hmacKey, sizeof(hmacKey), encryptedMessage, sizeof(encryptedMessage),
calculatedHMAC);

    // Comparer le HMAC reçu avec le HMAC recalculé
    bool hmacValid = true;
    for (int i = 0; i < 8; i++) {
      if (receivedHMAC[i] != calculatedHMAC[i]) {
        hmacValid = false;
        break;
      }
    }

    if (!hmacValid) {
      Serial.println("Authentification échouée : HMAC invalide");
      return; // Arrêter le traitement si le HMAC est incorrect
    }

    // Déchiffrer les données
    AES128 aesDecryptor;
    aesDecryptor.setKey(aesKey, sizeof(aesKey));
    aesDecryptor.decryptBlock(decryptedMessage, encryptedMessage);
```

```
    // Ajouter le caractère nul pour terminer la chaîne
    decryptedMessage[16] = '\0';

    // Afficher le message déchiffré
    Serial.print("Message déchiffré : ");
    Serial.println((char*)decryptedMessage);
  }
}
```

Explication du chiffrement et vérification de l'authentification:

Sur le message reçu, le hash est recalculé. Après vérification, le message est déchiffré en cas de calcul positif et rejeté dans le cas contraire



1. Réception du message et du HMAC

- **Lecture des données chiffrées :**

```
for(int i = 0; i < 16; i++) {
if(LoRa.available()){ encryptedMessage[i] = LoRa.read();}
}
```

- Les 16 premiers octets reçus via LoRa sont stockés dans encryptedMessage. Ce sont les données chiffrées avec AES.

- **Lecture du HMAC reçu :**

```
   for (int i = 0; i < 8; i++) { if (LoRa.available())
{receivedHMAC[i] = LoRa.read(); }}
```

- Les 8 octets suivants correspondent à une version tronquée du HMAC envoyé avec le message. Ils servent à vérifier l'authenticité.

2. RECALCUL DU HMAC

- **Calcul du HMAC attendu :**

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

- Le HMAC est recalculé côté réception en utilisant :
  - La clé HMAC secrète **hmacKey**.
  - Les données chiffrées **encryptedMessage**.

  - La fonction **calculateHMAC** produit un HMAC complet (32 octets) et le stocke dans **calculatedHMAC**.

3.Vérification de l'intégrité et de l'authenticité

- **Comparaison des HMAC :**

```
bool hmacValid = true;
for (int i = 0; i < 8; i++) { if (receivedHMAC[i] !=
calculatedHMAC[i]) {
hmacValid = false;break;
}
}
```

  - Les 8 premiers octets du HMAC recalculé (**calculatedHMAC**) sont comparés avec le HMAC reçu (**receivedHMAC**).
  - Si une seule différence est détectée, le HMAC est invalidé, et le message est rejeté.

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