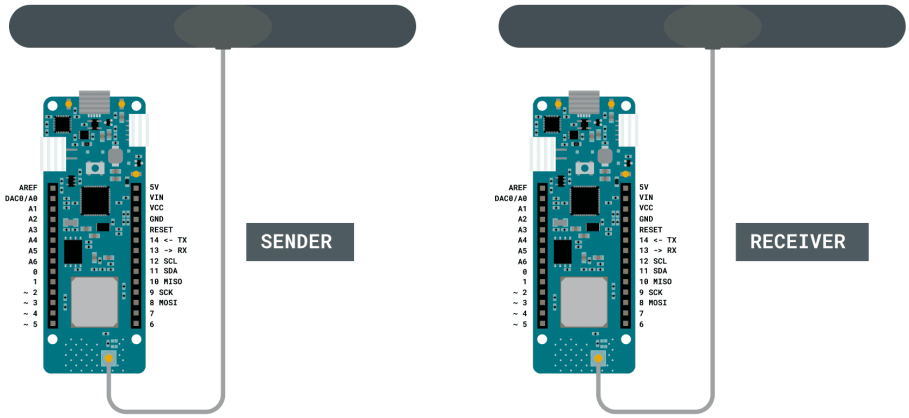
TP 4
CHIFFREMENT D'UNE TRANSMISSION LORA PAIR À PAIR.

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/Tp3Lora2Lora/LoRaSender-encrypt.ino

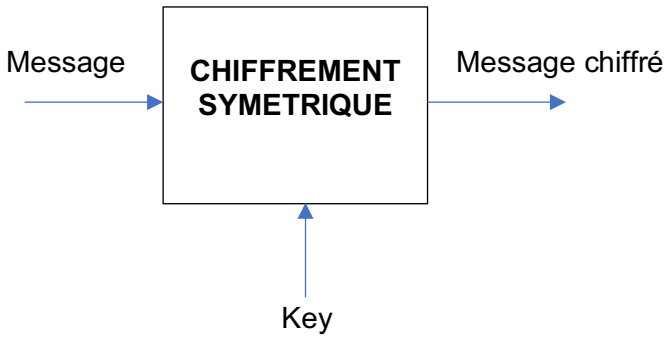
2.4 Modifier le fichier de la manière suivante :

- 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>

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

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

  Serial.println("Initialisation de l'émetteur 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

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

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

  // Envoi du message
  LoRa.beginPacket();
  LoRa.write(encryptedMessage, sizeof(encryptedMessage));
  LoRa.endPacket();

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

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

Explication du chiffrement :



1. Clé AES-128 :

A. Clé de chiffrement :

```
const byte key[16] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05,
0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F };
```

- C'est un tableau de 16 octets (128 bits), nécessaire pour l'algorithme **AES-128**.
- La clé est utilisée à la fois par l'émetteur et le récepteur pour le chiffrement et le déchiffrement. Elle doit rester secrète.

B. Initialisation de l'objet AES

```
AES128 aesEncryptor;
aesEncryptor.setKey(key, sizeof(key));
```

1. Création de l'objet aesEncryptor :

- La classe AES128 est fournie par la bibliothèque **Crypto**.
- Elle est spécialisée pour le chiffrement et le déchiffrement AES en mode ECB (Electronic Codebook) par blocs de 16 octets.

2. Configuration de la clé :

- setKey() configure la clé utilisée pour le chiffrement.
- Paramètres :
  - key : La clé secrète (tableau de 16 octets).
  - sizeof(key) : Taille de la clé (16 octets ici).

C. Préparation du message

```
char message[] = "Hello LoRa!";
byte encryptedMessage[16];
```

1. Message clair :

- Le message "Hello LoRa!" est un tableau de caractères à chiffrer.

- Sa longueur est inférieure à 16 octets, donc il s'insère parfaitement dans un seul bloc AES (16 octets).

2. Buffer pour le résultat chiffré :

- encryptedMessage est un tableau de 16 octets qui stockera le résultat du chiffrement.

D. Chiffrement

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

1. encryptBlock :

- Cette méthode chiffre un **bloc unique** de 16 octets.
- Paramètres :
  - encryptedMessage : Tableau de sortie où le résultat chiffré sera stocké.
  - (byte*)message : Le message clair à chiffrer, converti en un tableau d'octets.

2. Principe du chiffrement AES :

- AES (Advanced Encryption Standard) est un algorithme de chiffrement symétrique qui transforme un bloc de données (16 octets) en un autre bloc de données de la même taille.
- La transformation utilise :
  - La clé secrète (configurée avec setKey()).
  - Une série de substitutions, permutations et opérations mathématiques sur les données.

3. Bloc unique :

- Ici, le mode ECB (Electronic Codebook) est utilisé : chaque bloc est chiffré indépendamment.
- Si le message dépasse 16 octets, vous devez gérer plusieurs blocs.

E. Envoi des données chiffrées

```
LoRa.beginPacket();
LoRa.write(encryptedMessage, sizeof(encryptedMessage));
LoRa.endPacket();
```

1. Envoi des données :

- Les 16 octets chiffrés sont envoyés via LoRa.
- write(encryptedMessage, sizeof(encryptedMessage)) transmet directement les données binaires.

Élève 2 : carte réceptrice :

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

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

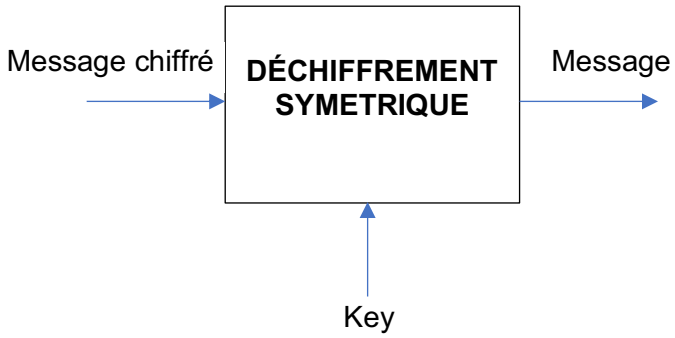
2.7 Créer un fichier LoraReceiver avec à partir du fichier :
https://github.com/bouhenic/FormationIOT/blob/main/Tp3Lora2Lora/LoRaReceiver-encrypt.ino

2.8 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>

// Clé de déchiffrement AES (identique à l'émetteur)
const byte key[16] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                       0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F };

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

  Serial.println("Initialisation du récepteur LoRa...");

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

  LoRa.setSpreadingFactor(8);
  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 decryptedMessage[16]; // Buffer pour le message déchiffré

    // Lire les données reçues
    for (int i = 0; i < packetSize && i < sizeof(encryptedMessage); i++) {
      encryptedMessage[i] = LoRa.read();
    }

    // Déchiffrement AES
    AES128 aesDecryptor;
    aesDecryptor.setKey(key, sizeof(key));
    aesDecryptor.decryptBlock(decryptedMessage, encryptedMessage);

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

Explication du chiffrement :



A. Lecture des données reçues

```
for (int i = 0; i < packetSize && i < sizeof(encryptedMessage); i++) {
  encryptedMessage[i] = LoRa.read();
}
```

1. **LoRa.parsePacket**() :
   - Cette fonction détecte si un paquet a été reçu.
   - **packetSize** contient la taille du paquet (en octets).

2. Lecture des données :
   - Une boucle lit les données du paquet byte par byte avec **LoRa.read()**.
   - Les données reçues sont stockées dans le tableau **encryptedMessage**.

3. Limitation à 16 octets :
   - Le tableau **encryptedMessage** est défini pour 16 octets, car l'algorithme AES fonctionne avec des blocs de cette taille.
   - Si le message dépasse 16 octets, seuls les 16 premiers octets seront pris en compte.

B. Initialisation de l'objet AES

AES128 aesDecryptor;
aesDecryptor.setKey(key, sizeof(key));

1. Création de l'objet aesDecryptor :
   - Cet objet représente un déchiffreur AES-128.

2. Configuration de la clé :
   - La méthode setKey() configure la clé secrète utilisée pour le déchiffrement.
   - La clé (key) doit être identique à celle utilisée lors du chiffrement côté émetteur.

C. Déchiffrement du message

```
aesDecryptor.decryptBlock(decryptedMessage, encryptedMessage);
```

1. **decryptBlock** :
   - Cette méthode déchiffre un bloc unique de 16 octets.

- Paramètres :
    - **decryptedMessage** : Tableau où sera stocké le message déchiffré.
    - **encryptedMessage** : Tableau contenant les données chiffrées reçues.

2. Processus de déchiffrement :
- L'algorithme AES utilise la clé configurée pour inverser les transformations appliquées lors du chiffrement.
- Le tableau decryptedMessage contiendra les données en clair une fois le déchiffrement terminé.

D. Affichage du message déchiffré

```
Serial.print("Message déchiffré : ");
Serial.println((char*)decryptedMessage);
```

1. Conversion en chaîne de caractères :
    - (char*)decryptedMessage interprète les données déchiffrées comme une chaîne de caractères.
    - Cela fonctionne si le message d'origine est une chaîne ASCII.

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