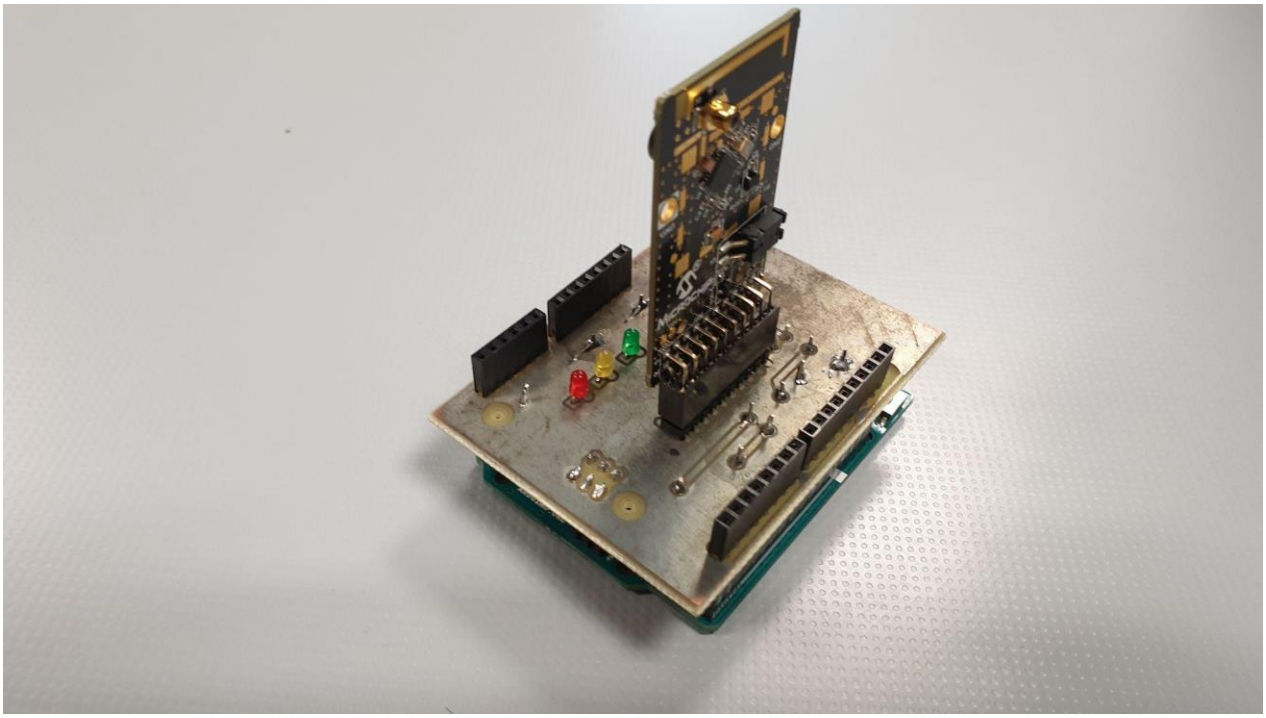
SAE Alternants :
Projet ZigBee



Eloi Tourangin Lucas Auger
G2300

Enseignant tuteur : Philippe Lucidarme

Ce rapport présente le travail réalisé par un groupe d'étudiants dans le cadre d'un projet pédagogique. Les auteurs et l'Université d'Angers ne garantissent pas que l'information, les documents, la méthodologie et le matériel présentés dans ce document soient complets, conformes à l'état de l'art et exacts, ni n'assurent en toutes circonstances la sécurité des biens, des personnes et des utilisateurs.
Les auteurs et l'Université d'Angers ne seront pas tenus responsables des dommages éventuels qui pourraient résulter de l'utilisation du contenu du présent rapport.

Eloi Tourangin et Lucas Auger, auteurs du présent rapport, cédons gratuitement à l'Université d'Angers à titre non-exclusif pour le monde et pendant la durée légale de protection, les droits de reproduire, représenter, conserver et d'adapter le présent rapport sur son site Internet ou tout autre média numérique de l'Université d'Angers. Nous garantissons la jouissance entière, paisible et libre des droits cédés contre tout trouble, revendication ou éviction.

Sommaire :

Remerciements :

Nous souhaitons remercier notre professeur tuteur, M.Lucidarme, ainsi que l'IUT d'Angers pour la confiance qu'ils ont placés en nous, nous avoir épaulé tout au long du projet, le matériel prêté ainsi que les locaux mis à notre disposition.

I - Introduction du projet

Pour commencer notre projet, nous avons reçu deux capteurs de température et humidité, un bouton poussoir ainsi qu'une base principale communicante avec les capteurs et le bouton. Notre but était d'intercepter ces communications par l'intermédiaire d'un module Arduino puis de les interpréter avec un code python.

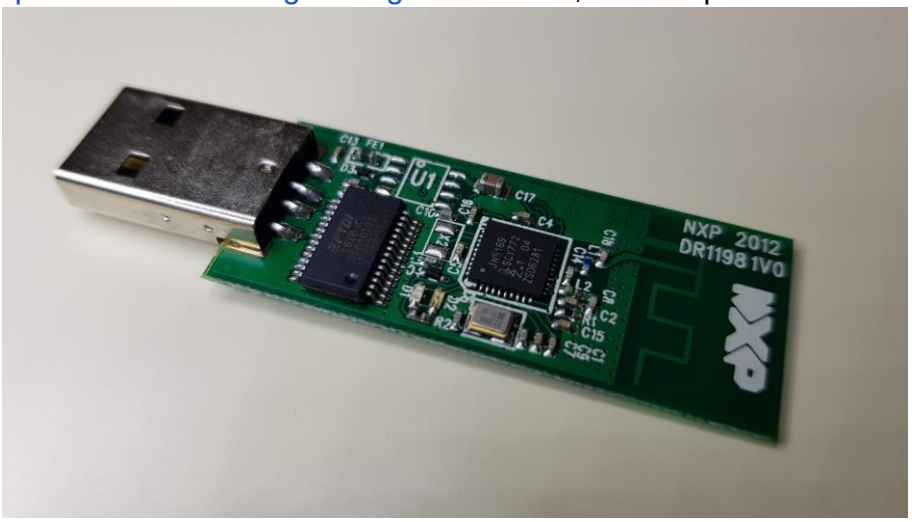
II - Cahier des charges

- Enregistrer une séquence complète avec Ubiqua (pour analyser leurs compositions)
- Enregistrer une trame complète avec Arduino
- Faire un tuto sur le déchiffrement dans Ubiqua
- Communiquer sur le canal avec Arduino
- Déchiffrer avec Python (Récupérer les informations intéressantes des trames)

III - Réalisation

1. Déchiffrement ZigBee

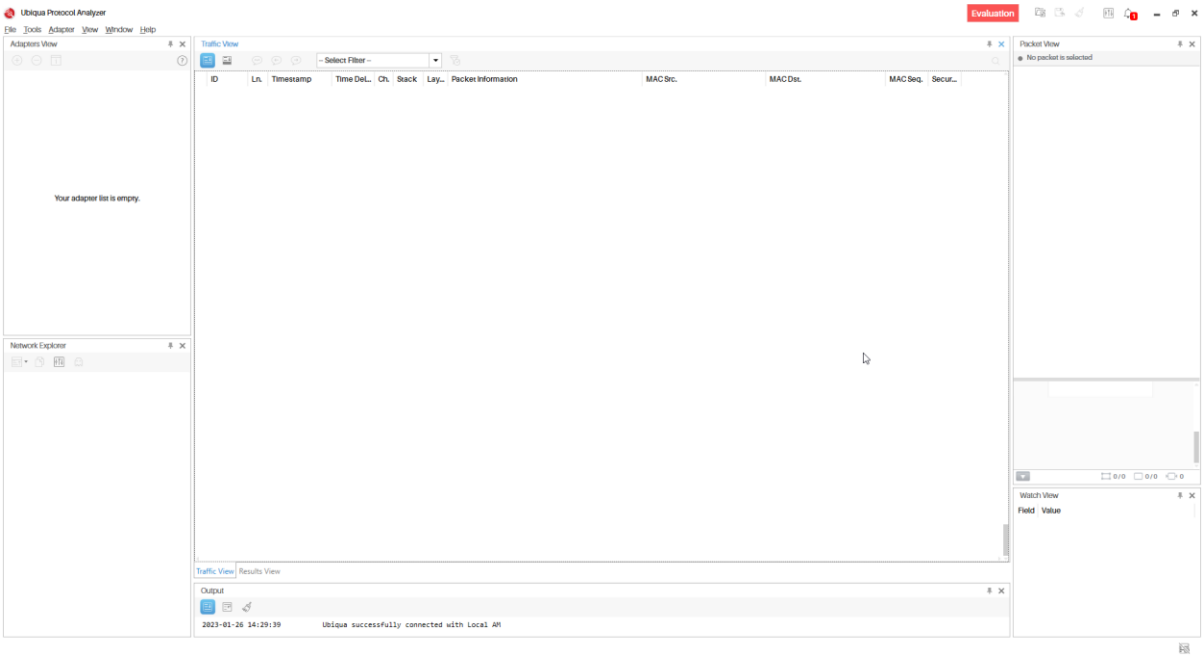
Le logiciel Ubiqua permet une analyse détaillée des trames Zigbee reçues par le dongle USB (pour plus d'info sur le dongle, vous pouvez aller voir sur ce page internet: https://lucidar.me/fr/zigbee/zigbee-sniffer/ , dans la partie matérielle) :



Pour commencer, il faut créer un compte sur le site officiel et activer l'essai gratuit de 7 jours, puis télécharger l'installateur et suivre les étapes.
Une fois le logiciel bien installé, lancé et le dongle branché, il faut sélectionner le port utilisé et l'application que vous souhaitez utiliser.

Pour cela :

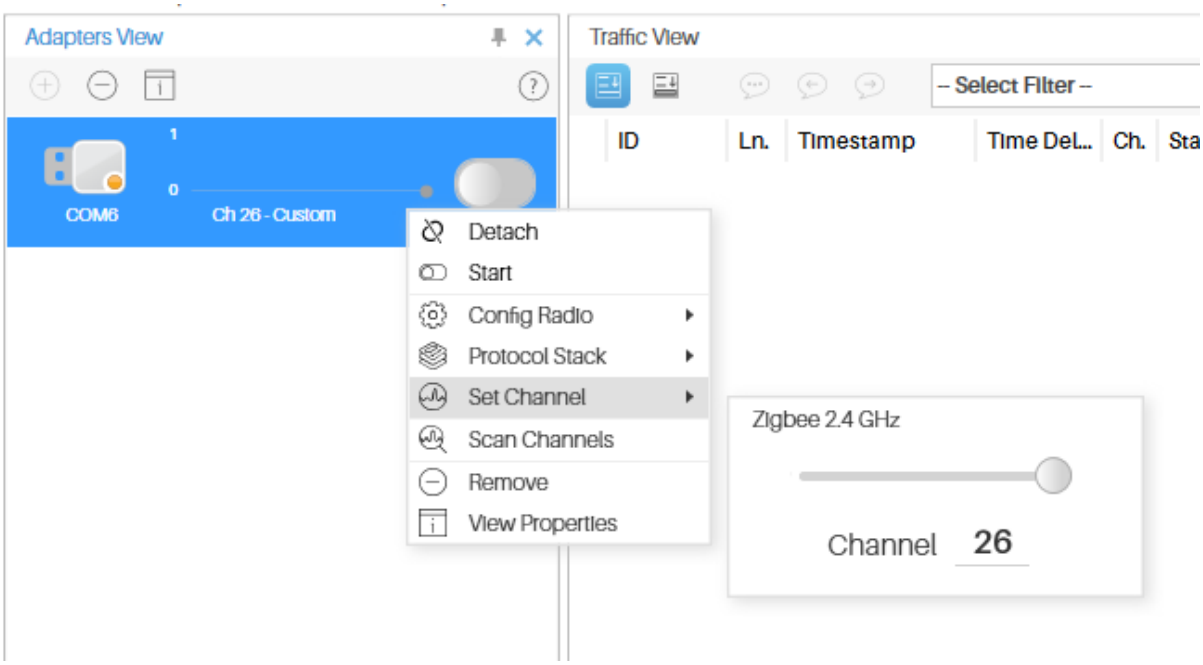
- Cliquer sur Adapter>Add Adapter



- Puis sur la roue crantée

- Dans le menu Connector type sélectionner UART. Dans le menu App, sélectionner Jennic Sniffer et pour le Connector Name, celui correspondant au port du dongle (si c'est le seul périphérique USB connecté à votre machine, il n'y aura qu'un choix).

- Suite à cette manipulation, un icône de clé USB à dû apparaître dans la colonne de gauche. Faites clic droit sur votre nouveau périphérique et choisissez le bon canal (channel) en fonction de ce que vous sniffez (Pour notre passerelle Zigbee : 26).

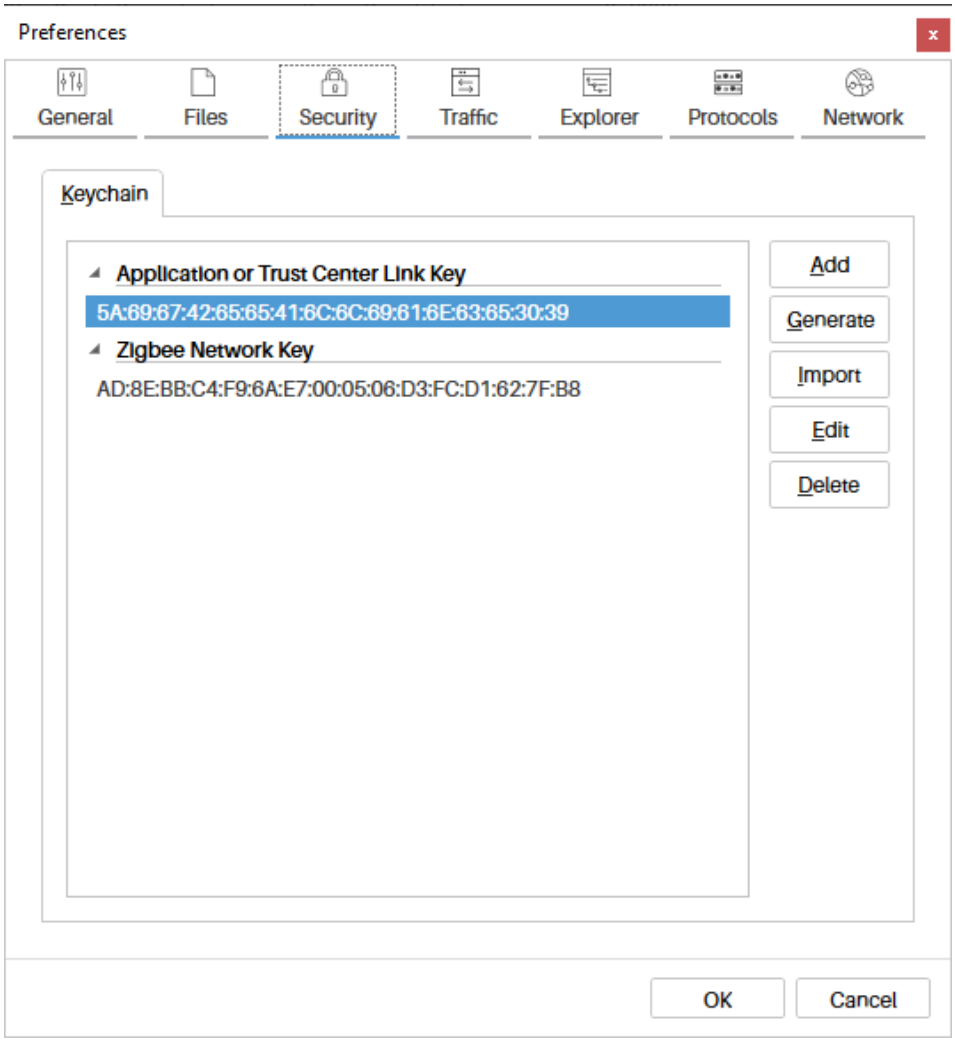


- Pour ajouter les clés de déchiffrement, allez dans Tools>Préférences>Security

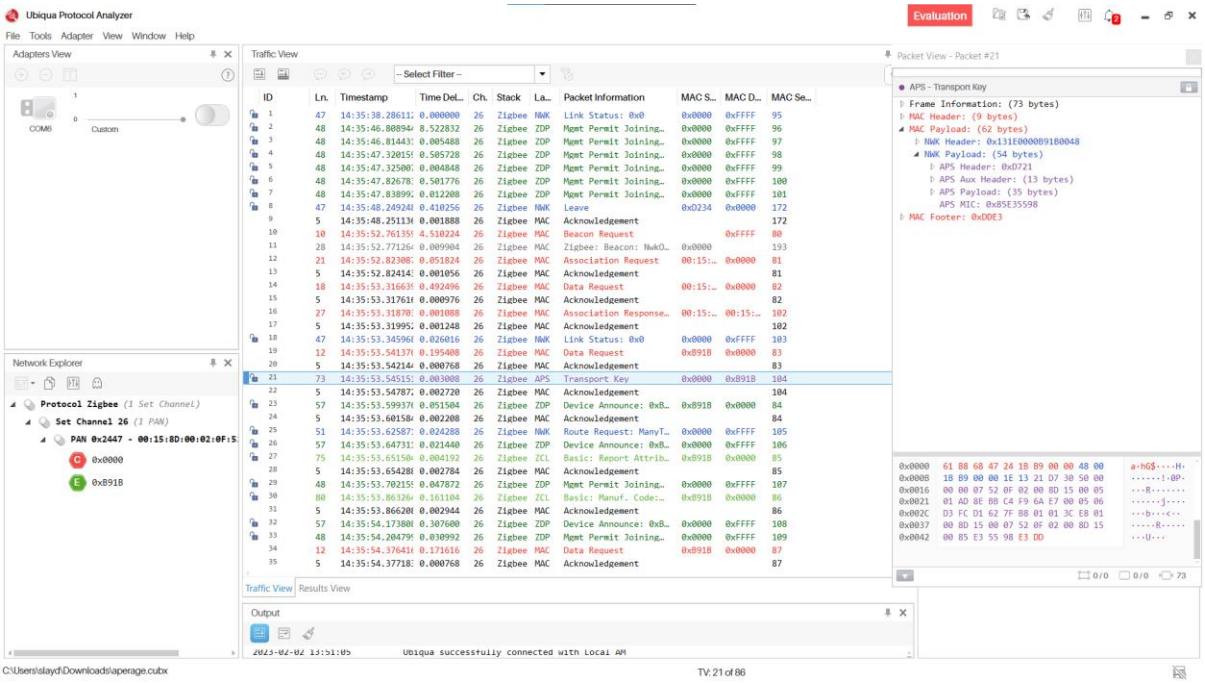
Trust Center Link Key :
5A:69:67:42:65:65:41:6C:6C:69:61:6E:63:65:30:39

La TCL key est la même pour presque tous les appareils Zigbee (source : faire-ca-sois-meme.fr) elle nous sert à déchiffrer les premières trames d'appairage, nous pourrons trouver la network key dans une de ces trames.
ZigBee Network Key :
AD:8E:BB:C4:F9:6A:E7:00:05:06:D3:FC:D1:62:7F:B8

Cette clé va nous servir à déchiffrer toutes les trames Zigbee entre la passerelle et les différents appareils connectés.



Vous êtes maintenant prêt à sniffer la communication entre vos appareils connectés.



Pour enclencher la récupération de trames, il faut cliquer sur le bouton en haut à gauche à côté de la bar annoté custom

1	47	14:35:38.28611	0.000000	26	Zigbee	NWK	Link Status: 0x0	0x0000	0xFFFF	95
2	48	14:35:46.80894	8.522832	26	Zigbee	ZDP	Mgmt Permit Joining…	0x0000	0xFFFF	96
3	48	14:35:46.81443	0.005488	26	Zigbee	ZDP	Mgmt Permit Joining…	0x0000	0xFFFF	97
4	48	14:35:47.32015	0.505728	26	Zigbee	ZDP	Mgmt Permit Joining…	0x0000	0xFFFF	98
5	48	14:35:47.32500	0.004848	26	Zigbee	ZDP	Mgmt Permit Joining…	0x0000	0xFFFF	99
6	48	14:35:47.82678	0.501776	26	Zigbee	ZDP	Mgmt Permit Joining…	0x0000	0xFFFF	100
7	48	14:35:47.83899	0.012208	26	Zigbee	ZDP	Mgmt Permit Joining…	0x0000	0xFFFF	101
8	47	14:35:48.24924	0.410256	26	Zigbee	NWK	Leave	0xD234	0x0000	172
9	5	14:35:48.25113	0.001888	26	Zigbee	MAC	Acknowledgement			172
10	10	14:35:52.76135	4.510224	26	Zigbee	MAC	Beacon Request		0xFFFF	80
11	28	14:35:52.77126	0.009904	26	Zigbee	MAC	Zigbee: Beacon: NwkO…	0x0000		193
12	21	14:35:52.82308	0.051824	26	Zigbee	MAC	Association Request	00:15:…	0x0000	81
13	5	14:35:52.82414	0.001056	26	Zigbee	MAC	Acknowledgement			81
14	18	14:35:53.31663	0.492496	26	Zigbee	MAC	Data Request	00:15:…	0x0000	82
15	5	14:35:53.31761	0.000976	26	Zigbee	MAC	Acknowledgement			82
16	27	14:35:53.31870	0.001088	26	Zigbee	MAC	Association Response…	00:15:…	00:15:…	102
17	5	14:35:53.31995	0.001248	26	Zigbee	MAC	Acknowledgement			102
18	47	14:35:53.34596	0.026016	26	Zigbee	NWK	Link Status: 0x0	0x0000	0xFFFF	103
19	12	14:35:53.54137	0.195408	26	Zigbee	MAC	Data Request	0xB91B	0x0000	83
20	5	14:35:53.54214	0.000768	26	Zigbee	MAC	Acknowledgement			83
21	73	14:35:53.54515	0.003008	26	Zigbee	APS	Transport Key	0x0000	0xB91B	104

Sur ce screen, on peut voir les premières trames envoyées entre la passerelle et un ou plusieurs appareils. Ces trames sont les trames d'appairage. On retrouve parmi elles, une trame appelée Transport Key qui contient la clé de chiffrement, clé qui, comme son nom l'indique, sert à chiffrer et à déchiffrer les trames pour la passerelle ou par les appareils appairés.

2. Déchiffrement sous python

Les fonctions suivantes sont toutes dans le fichier cryptage.py . Ces fonctions sont séparées pour éviter de surcharger le main.py.

```
def printhex(x, sep = ' '):
  str = ''
  for b in x:
    byte = hex(b)[2:]
    if (len(byte)<2)  :
      str += '0' + byte + sep
    else:
      str += hex(b)[2:] + sep
  print (str[:-1].upper())
```

La fonction ‘printhex’ permet d'afficher une chaîne de caractère avec les lettres en majuscules à partir du tableau contenant des valeurs hexadécimales.

```
def returnhex(x, sep = ' '):
  str = ''
  for b in x:
    byte = hex(b)[2:]
    if (len(byte)<2):
      str += '0' + byte + sep
    else:
      str += hex(b)[2:] + sep
  return (str[:-1].upper())
```

La fonction ‘returnhex’ permet de retourner une chaîne de caractère avec les lettres en majuscules à partir du tableau contenant des valeurs hexadécimales.

```
def pad(x):
  n=(16-len(x)%16)%16
  return x + bytes([0x00]*n)
```

La fonction ‘pad’ permet d'ajouter des octets nuls pour garantir les longueurs des trames.

```
def decrypt(trame):
  print(trame)
  key =   bytes([0xAD, 0x8E, 0xBB, 0xC4, 0xF9, 0x6A, 0xE7, 0x00, 0x05, 0x06, 0xD3, 0xFC,
0xD1, 0x62, 0x7F, 0xB8])
  # printhex (key)

  L = 2
  M=4

  #Recuperation de NwkHeader depuis la trame recue
  NwkHeader = bytes.fromhex(trame[18:34])

  #Recuperation de AuxHeader depuis la trame recue
  AuxiliaryHeader = bytes.fromhex('2D'+trame[36:62])

  #Formation du nonce à partir de differentes partie de la trame recue
  nonce = bytes.fromhex(trame[44:60]+trame[36:44]+'2D')


  # Octet string a
  a = NwkHeader + AuxiliaryHeader

  # Octet string m
  #m = bytes([0x83, 0x6A, 0xAB, 0xF6, 0xFB, 0x86, 0x72, 0xAD, 0x54, 0x16, 0xC1, 0xB0, 0x9C,
0xAF, 0xCC, 0x7E])
  m = bytes.fromhex(trame[62:94])

  # Right-concatenate the octet string L(a) with the octet string a itself.
  AddAuthData = len(a).to_bytes(2, byteorder = 'big') + a
  # Form the padded message AddAuthData
  AddAuthData = pad(AddAuthData)



  PlaintextData = m
  # Padding (not necessary here, because length(m)=16)
  PlaintextData = pad(PlaintextData)

  #print (len(PlaintextData), 'bytes')
  #printhex (PlaintextData)

  AuthData = AddAuthData + PlaintextData
```

```
  Flags = bytes([0x49]) # = ([0b01001001])

  B0 = Flags + nonce + len(m).to_bytes(2, byteorder = 'big')

  X0 = bytes([0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00])

  cipher = AES.new(key, AES.MODE_CBC, X0)
  X1 = cipher.encrypt(B0 + AuthData)
  T = X1[-16:-12]

  Flags = bytes ([0b00000001])

  A0 = Flags + nonce + bytes([0x00, 0x00])
  A1 = Flags + nonce + bytes([0x00, 0x01])

  from Crypto.Util import Counter
  ctr = Counter.new(128, initial_value = int.from_bytes(A1, "big"))
  cipher = AES.new(key, AES.MODE_CTR, counter = ctr)
  Ciphertext = cipher.encrypt(PlaintextData)

  # Encryption: S0:= E(Key, A0)
  cipher = AES.new(key, AES.MODE_ECB)
  S0 = cipher.encrypt(A0)

  # Perform S0[0:4] XOR T
  U = bytes(a ^ b for (a, b) in zip(S0[0:4], T))

  return returnhex(Ciphertext)
```

La fonction ‘decrypt’ permet de décrypter les trames à l'aide d'une clé de déchiffrage. Elle commence par récupérer chaque partie de la trame séparément (NwkHeader, AuxiliaryHeader, nonce,...) . Dans le but de la retourner décryptée en hexadécimal.

```
def decoupeTrame(trame):
    a = ""
    for i in range(len(trame)):
        if (i%2) == 0 and i>1:
            a = a + " " + trame[i]
        else:
            a = a + "" +trame[i]
    list_trame = a.split()
    for j in range(len(list_trame)):
        if j < len(list_trame):
            list_trame[j] += " "
    return a , list_trame
```

La fonction decoupeTrame prend en entrée une trame sous forme de chaîne de caractères et retourne un tuple constitués d'une chaîne de caractère et d'une liste de sous-chaîne.
Pour chaque caractère dans 'trame', s'il est à un index pair et supérieur à 1, il est ajouté à 'a' avec un espace. Sinon, il est ajouté à 'a' sans aucun autre caractère.
Exemple: pour trame = "01E21E52A2S5D2", a = "01 E2 1E 52 A2 S5 D2" et list_trame = ['01', 'E2','1E','52','A2','S5','D2']

```
def incrementTrame(trame):
  T_trame,list_trame = decoupeTrame(trame)
  tab = [2,16,18,38,40]
  for j in range(len(tab)):
    for i in range(len(list_trame)):
      if (i == tab[j]):
        list_trame[i] = int(list_trame[i],16)
        list_trame[i]+= 0x01
        list_trame[i] = str(hex(list_trame[i]))
        list_trame[i] = list_trame[i].replace("0x","")
        list_trame[i] = list_trame[i].upper()
      else:
        list_trame[i] = list_trame[i].replace(" ","")
  return list_trame
```

La fonction ‘incrementTrame’ utilise la fonction ‘decoupeTrame’ pour obtenir la liste de sous-chaîne. Ensuite, pour chaque élément de la liste à l'indice spécifié par tab, elle convertit l'élément en entier, incrémente l’élément de 1 puis le re-convertit en chaîne hexadécimale et en enfin le replace dans la liste. Finalement, la fonction retourne la liste modifiée.

```
def concatenageTrame(trame):
  tramestr = ""
  for i in range(len(trame)):
    tramestr += trame[i]
  return tramestr
```

La fonction ‘concatenageTrame’ concatène les éléments d'une liste de chaînes de caractères en une seule chaîne de caractères. La boucle for parcourt la liste trame et concatène chaque élément de la liste à la chaîne 'tramestr'. La fonction retourne finalement la chaîne concaténée 'tramestr'.

Les fonctions suivantes proviennent du main.py .

```
def clean_frame(frame):
    frame = frame.decode("utf-8") # enlever le b'
    frame = frame.replace("\r\n", "") # enlever le '\r\n'
    return frame
```

La fonction ‘clean_frame’ décode les données binaires reçues en utilisant l'encodage "utf-8", enlevant ainsi le préfixe 'b' qui peut parfois être ajouté aux données lors de la transmission.
Supprime les retours à la ligne '\r\n' présents dans les données en utilisant la méthode replace()

```
def recup_data(trame):
    trame1 = trame[45:47]
    trame1 += trame[42:44] # extraire les 4 derniers octets de la trame
    #print(trame1)
    data = int(trame1, 16) # convertir la trame en décimal et diviser par 100 pour obtenir la
température en degrés Celsius
    return data
```

La fonction ‘recup_data’ a pour but de récupérer les données utiles à partir d'une trame de données reçue via une communication série. Elle effectue les actions suivantes :
Extrait les 4 derniers octets de la trame en utilisant la concaténation de chaînes de caractères et la découpe de sous-chaînes de la trame originale (trame[45:47] et trame[42:44]).
Convertit la sous-chaîne obtenue en décimal en utilisant la fonction int(trame1,16), où '16' représente la base hexadécimale.
La fonction retourne le résultat de la conversion en décimal, qui représente les données utiles extraites de la trame.

```
def recup_info(trame):
    trame1 = trame[9:11]
    trame1 += trame[6:8] # extraire les 4 derniers octets de la trame
    #print("info1 :", trame1)
    info = int(trame1, 16)
    #print("info2 :", info)
    return info
```

La fonction ‘recup_info' a pour but de récupérer des informations utiles à partir d'une trame de données reçue via une communication série. Elle effectue les actions suivantes :
Extrait les 4 octets d'informations de la trame en utilisant la concaténation de chaînes de caractères et la découpe de sous-chaînes de la trame originale (trame[9:11] et trame[6:8]).
Convertit la sous-chaîne obtenue en décimal en utilisant la fonction int(trame1,16), où 16 représente la base hexadécimale.
La fonction retourne le résultat de la conversion en décimal, qui représente les informations utiles extraites de la trame.

```
def sendTrame(trame):
  ser.write(trame.encode())#'utf-8'))
```

La fonction ‘sendTrame’ encode la trame en utf-8 et l'envoie à l'interface série en utilisant la méthode write() de l'objet 'ser' qui est une instance de la classe Serial.

La fonction ne retourne rien, elle envoie simplement la trame de données via la communication série.

```
def paytotrame(trame,payload):
  clean_payload = payload.replace(" ","")
  #returnhex(trame)
  trame_dec=trame[0:62]+clean_payload+trame[94:106]
  # print("trame :",trame)
  # print("payload :",clean_payload)
  # print("trame_dec :",trame_dec)
  return trame_dec
```

La fonction ‘pattotrame’ a pour but de construire une nouvelle trame de données à partir d'une trame existante et d'un payload (charge utile) fourni en entrée. Elle effectue les actions suivantes : Supprime les espaces dans le payload en utilisant la méthode replace() pour obtenir une version nettoyée du payload (clean_payload). Concatène les parties de la trame existante avec le payload nettoyé pour former une nouvelle trame (trame_dec).
La fonction retourne la nouvelle trame construite.

```python
while True:
  data = ser.readline()
  print ("Trame Brut :",data)
  trame = clean_frame(data)
  #print("longueur :",len(trame))
  if len(trame) == 106:
    payload = decrypt(trame)
    print ('Payload = ', payload)

    packet_info = recup_info(payload)
    if packet_info == 1026:
      print("La temperature est de ", recup_data(payload)/100,"°C")
    if packet_info == 1029:
      print("L'Humidité est de ", recup_data(payload)/100,"%")
    elif packet_info == 18:
      print("Bouton !!!!  ", recup_data(payload), "clic")
      last_button_dec = paytotrame(trame,payload)
    print("\n")

  elif len(trame) == 10:
    print("Acknowledgement")
    print("\n")

  elif len(trame) == 7:
    print("Bouton")
    print("Last button:",last_button_dec)
    send_trame = incrementTrame(last_button_dec)
    send_trame = concatenageTrame(send_trame)
    #send_trame = decrypt(send_trame)
    print("Trame incr :",send_trame)
    sendTrame(send_trame)
    #time.sleep(1)
    print("\n")
```

Ce bloc de code décode et analyse ces données, puis envoie des réponses si nécessaire.

1. Utilise la méthode readline() de l'objet ‘ser’ pour lire les données reçues via la communication série.
2. Affiche les données brutes reçues (Trame_Brut).
3. Nettoie les données en utilisant la fonction clean_frame().
4. Vérifie la longueur de la trame nettoyée.

En fonction des longueurs de trames, nous arrivons à les différencier pour mieux les décrypter :

a. Si la longueur est de 106, le code décrypte la trame en utilisant la fonction decrypt() pour obtenir le payload. Affiche le payload (Payload =).
b. Extrait les informations de la trame en utilisant la fonction recup_info().

c. Vérifie la valeur de l'information extraite pour déterminer le type de données contenu dans la trame (température, humidité ou bouton).

i. Si la valeur est 1026, affiche la température en utilisant la fonction recup_data().

ii. Si la valeur est 1029, affiche l'humidité en utilisant la fonction recup_data().

iii. Si la valeur est 18, affiche le nombre de clics du bouton en utilisant la fonction recup_data() et enregistre la trame dans une variable (last_button_dec).

1. Si la longueur de la trame est de 10, le code affiche un message indiquant que c'est un Acknowledgement.
2. Si la longueur de la trame est de 7, le code affiche un message indiquant qu'il s'agit d'un bouton et affiche la dernière trame de bouton enregistrée (last_button_dec).

a. Construit une nouvelle trame en utilisant la fonction paytotrame() pour incrémenter la trame de bouton enregistrée.
b. Concatène la trame incrémentée en utilisant la fonction concatenageTrame().
c. Envoie la trame concaténée via la communication série en utilisant la fonction sendtrame().

3. Envoi de la température au serveur

Nous voulions que la température interceptée par notre module Arduino puis interpréter par notre programme python puisse être directement envoyé sur un serveur.
Pour cela, nous avons utilisé une requête post :

url = 'https://eloitourangin.fr/SAE/post.php?temp='+str(temp)
requests.post(url)

Sur le serveur, le fichier post.php s'occupe d'écrire la valeur **temp** dans un fichier txt et enfin un fichier index.php s'occuper d'afficher en continu la valeur présente dans le fichier txt.

Conclusion :

Pour conclure, nous dirons que ce projet est une bonne clé d'entrée au protocole ZigBee. Il permet d'en comprendre les bases et d'imaginer les possibilités de développement. Vis-à-vis de l'avancée du projet, nous avons pensé à quelques pistes d'amélioration que voici :

- Tout d'abord, il serait envisageable de développer une transmission depuis le code Arduino vers les appareils connectés (capteurs, bouton) et ainsi rendre le système indépendant de la base d'appairage principale.

- Ensuite, nous avons pensé qu'il serait pratique de développer une IHM (Interface Homme Machine) pour faciliter la compréhension des informations reçues. Et, à termes, cette même IHM pourrait être utilisée pour les transmissions d'informations.

En résumé, lors de ce projet, nous avons eu à identifier les trames via l'outil Ubiqua, puis nous avons développé un code python pour les analyser et en extraire les informations importantes. Et enfin, nous avons exporté les informations extraites via python sur un serveur web.

In summary, during this project, we had to identify the frames via the Ubiqua tool, then we developed a python code to analyze them and extract important information. And finally, we exported the extracted information via python to a web server.