Université de Limoges

Faculté des Sciences et Techniques

Master 2 CRYPTIS

UE : TMC

IoT, LoRa, WiFi, MQTT, SSL, ATECC508, Mongoose OS, Raspberry Pi & ESP8266

Presenté par :

Claudio ANTONIO
Nhat BUI
Yawavi Latevi

Enseignant : **P-F. Bonnefoi**

Limoges , Février 2023

Table des matières

I Introduction

Ce rapport donnera une brève description de notre projet dont le but est de réaliser : un réseau de capteurs connectés par WiFi vers un concentrateur , dans lequel chaque capteur exploite un circuit dédié à la manipulation de la cryptographie sur courbe elliptique et le concentrateur est relié à une passerelle par l'utilisation des communications LoRa.

Le projet est disponible sur github à l'adresse suivante : `https://github.com/MonaQuimbamba/IOT`.

II Raspberry Pi & WiFi

Sur le Raspberry Pi nous allons configurer :

- La connexion WiFi de l'ESP8266 :

  Pour faire cela, nous allons configurer le hostapd et le dnsmasq avec les commandes suivantes :

  – L'installation du point d'accès logiciel « hostapd »

```
pi@raspberrypi:~ $ sudo apt update
pi@raspberrypi:~ $ sudo apt-get install hostapd
```

  On modifie le fichier /etc/hostapd/hostapd.conf du raspberry Pi

```
pi@raspberrypi:~ $ cat /etc/hostapd/hostapd.conf
country_code=FR
interface=wlan0
ssid=iot_claudio
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=123456789
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

– L'installation du serveur DHCP permettant de prendre en charge les clients WiFi connectés avec dnsmasq

```
pi@raspberrypi:~ $ sudo apt update
pi@raspberrypi:~ $ sudo apt-get install dnsmasq
```

On modifie le fichier /etc/dnsmasq.conf du raspberry Pi

```
pi@raspberrypi:~ $ sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.bak
pi@raspberrypi:~ $ sudo vim /etc/dnsmasq.conf
pi@raspberrypi:~ $ cat /etc/dnsmasq.conf
interface=wlan0        #choose the interfac
dhcp-range=10.33.33.100,10.33.33.150,255.255.255.0,12h
domain=wlan
address=/mqtt.com/10.33.33.101
```

On ajoute le nameserver dans le fichier resolvconf.conf du raspberry Pi

```
pi@raspberrypi:~ $ cat /etc/resolvconf.conf
# Configuration for resolvconf(8)
# See resolvconf.conf(5) for details

resolv_conf=/etc/resolv.conf
# If you run a local name server, you should uncomment the below line and
# configure your subscribers configuration files below.
name_servers=127.0.0.56

# Mirror the Debian package defaults for the below resolvers
# so that resolvconf integrates seemlessly.
dnsmasq_resolv=/var/run/dnsmasq/resolv.conf
pdnsd_conf=/etc/pdnsd.conf
unbound_conf=/var/cache/unbound/resolvconf_resolvers.conf

```

On définit un IP statique sur l'interface wlan0.

```
pi@raspberrypi:~ $ cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
# Include files from /etc/network/interfaces.d:
source /etc/network/interfaces.d
#/*
allow-hotplug wlan0
iface wlan0 inet static
   address 10.33.33.101
   netmask 255.255.255.0
   gateway 10.33.33.101

```

On démarre l'hostapd et le dnsmasq.

```
pi@raspberrypi:~ $ sudo systemctl unmask hostapd
pi@raspberrypi:~ $ sudo systemctl enable hostapd
pi@raspberrypi:~ $ sudo systemctl enable dnsmasq
```

On vérifie que le DNS marche bien en faisant un ping au serveur mqtt.com

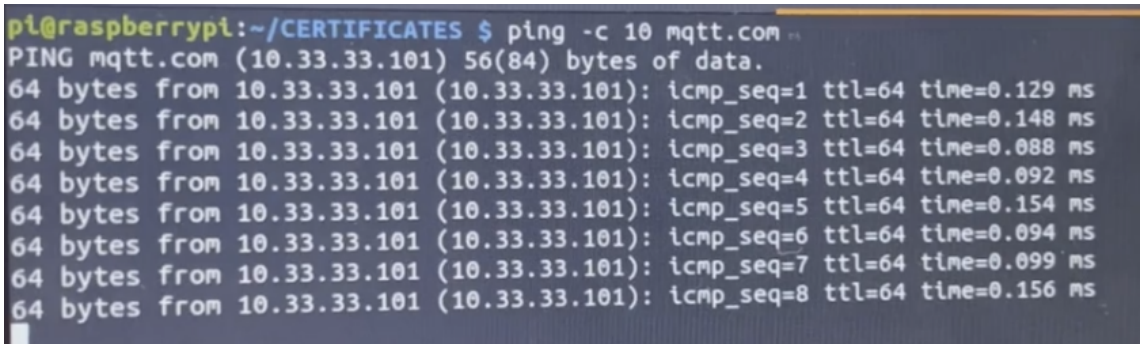

FIGURE 1 –

- Le serveur « mosquitto » et sa sécurité

  Avant de configurer le serveur mosquitto, on crée d'abord les certificats avec les commandes suivantes :

  Génération de clés privées pour l'AC, le serveur et le client.

```
pi@raspberrypi:~/CERTIFICATES $ openssl ecparam -out ecc.ca.key.pem -name prime256v1 -genkey

pi@raspberrypi:~/CERTIFICATES $ openssl ecparam -out ecc.raspberry.key.pem -name prime256v1 -genkey

pi@raspberrypi:~/CERTIFICATES $ openssl ecparam -out ecc.esp8266.key.pem -name prime256v1 -genkey
```

  Génération du certificat auto-signé de l'AC qui servira à signer ceux du serveur et du client

```
pi@raspberrypi:~/CERTIFICATES $
openssl req -config <(printf "[req]\ndistinguished_name=dn\n[dn]\n[ext]\nbasicConstraints=CA:TRUE") -new -nodes -subj "/C=FR/L=Limoges/O=TMC/OU=IOT/CN=ACTMC" -x509 -extensions ext -sha256 -key ecc.ca.key.pem -text -out ecc.ca.pem
```

  Génération et signature du certificat pour le serveur (Raspberry Pi)

```
pi@raspberrypi:~/CERTIFICATES $
$ openssl req -config <(printf "[req]\ndistinguished_name=dn\n[dn]\n[ext]\nbasicConstraints=CA:FALSE") -new -subj "/C=FR/L=Limoges/O=TMC/OU=IOT/CN=mqtt.com" -reqexts ext -sha256 -key ecc.raspberry.key.pem -text -out ecc.raspberry.csr.pem

$ openssl x509 -req -days 3650 -CA ecc.ca.pem -CAkey ecc.ca.key.pem -CAcreateserial -extfile <(printf "basicConstraints=critical,CA:FALSE") -in
ecc.raspberry.csr.pem -text -out ecc.raspberry.pem -addtrust clientAuth
```

Génération et signature du certificat pour le client (Esp8266)

```
pi@raspberrypi:~/CERTIFICATES $
$ openssl req -config <(printf "[req]\ndistinguished_name=dn\n[dn]\n[ext]\
    nbasicConstraints=CA:FALSE") -new -subj "/C=FR/L=Limoges/O=TMC/OU=IOT/CN=esp8266" -
    reqexts ext -sha256 -key ecc.esp8266.key.pem -text -out ecc.esp8266.csr.pem

$ openssl x509 -req -days 3650 -CA ecc.ca.pem -CAkey ecc.ca.key.pem -CAcreateserial -
    extfile <(printf "basicConstraints=critical,CA:FALSE") -in ecc.esp8266.csr.pem -text -
    out ecc.esp8266.pem -addtrust clientAuth
```

Ensuite on suit les étapes suivantes pour la configuration du serveur mosquitto :

L'installation du serveur et client mosquitto

```
pi@raspberrypi:~ $ sudo apt-get install mosquitto
pi@raspberrypi:~ $ sudo apt-get install mosquitto-clients

```

On copie les fichiers ecc.ca.pem ecc.raspberry.pem et ecc.raspberry.key.pem dans les répertoires /etc/-mosquitto/

```
pi@raspberrypi:~ $
$ sudo cp /CERTIFICATES/ecc.ca.pem /etc/mosquitto/ca_certificates/
$ sudo cp /CERTIFICATES/ecc.raspberry.pem /etc/mosquitto/certs/
$ sudo cp /CERTIFICATES/ecc.raspberry.key.pem /etc/mosquitto/certs/
```

et nous changeons la propriété des certificats en l'utilisateur "mosquitto" et le groupe "mosquitto".

```
sudo chown mosquitto:mosquitto  /etc/mosquitto/ca_certificates/ecc.ca.pem
sudo chown mosquitto:mosquitto  /etc/mosquitto/certs/ecc.raspberry.pem
sudo chown mosquitto:mosquitto  /etc/mosquitto/certs/ecc.raspberry.key.pem
```

On modifie le fichier /etc/mosquitto/mosquitto.conf

```
pi@raspberrypi:~ $  cat /etc/mosquitto/mosquitto.conf
allow_anonymous false
password_file /etc/mosquitto/mosquitto_passwd
listener 8883
cafile /etc/mosquitto/ca_certificates/ecc.ca.pem
certfile /etc/mosquitto/certs/ecc.raspberry.pem
keyfile /etc/mosquitto/certs/ecc.raspberry.key.pem
require_certificate true
```

Ensuite, nous utilisons `mosquitto_passwd` pour générer l'utilisateur `tmc` dans le fichier `mosquitto_passwd` et puis on redémarre le serveur mosquitto.

```
pi@raspberrypi:~ $ sudo mosquitto_passwd -c /etc/mosquitto/mosquitto_passwd tmc
pi@raspberrypi:~ $ sudo systemctl restart mosquitto.service
```

III Mongoose OS : MQTT client et publication sécurisée par ECC

L'installation du Mongoose OS

```
$ sudo add-apt-repository ppa:mongoose-os/mos
$ sudo apt-get update
$ sudo apt-get install mos
```

Pour générer un flash pour l'ESP8266, on installe un docker et on lui donne le droit d'exécution

```
$ sudo apt install docker.io
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
```

On installe une nouvelle application et on modifie le fichier de configuration mos.yml comme suit :

```
$ git clone https://github.com/mongoose-os-apps/empty my-app
$ cd my-app
$ cat mos.yml

.....

build_vars:
        MGOS_MBEDTLS_ENABLE_ATCA: 1
config_schema:
    - ["debug.level", 3]
    - ["sys.atca.enable", "b", true, {title: "Enable the chip"}]
    - ["i2c.enable", "b", true, {title: "Enable I2C"}]
    - ["sys.atca.i2c_addr", "i", 0x60, {title: "I2C address of the chip"}]
    - ["mqtt.enable", "b", true, {title: "Enable MQTT"}]
    - ["mqtt.server", "s", "mqtt.com:8883", {title: "MQTT server"}]
    - ["mqtt.pub", "s", "/esp8266", {title: "Publish topic"}]
    - ["mqtt.user", "s", "tmc", {title: "User name"}]
    - ["mqtt.pass", "s", "iot", {title: "Password"}]
    - ["mqtt.ssl_ca_cert", "s", "ecc.ca.pem", {title: "Verify server certificate using this CA
     bundle"}]
    - ["mqtt.ssl_cert", "s", "ecc.esp8266.pem", {title: "Client certificate to present to the
         server"}]
    - ["mqtt.ssl_key", "ATCA:0"]
    - ["wifi.ap.enable", "b", false, {title: "Enable"}]
    - ["wifi.sta.enable", "b", true, {title: "Connect to existing WiFi"}]
    - ["wifi.sta.ssid", "s", "iot_claudio", {title: "SSID"}]
    - ["wifi.sta.pass", "s", "123456789", {title: "Password", type: "password"}]

....
```

Nous allons modifier les certificats de l'ESP8266 de la façon suivante :

```
cat ecc.ca.pem
-----BEGIN CERTIFICATE-----
MIIBqTCCAU6gAwIBAgIUZvV4MFZwpjQOQDY2+VAo854viQgwCgYIKoZIzj0EAwIw
SzELMAkGA1UEBhMCRlIxEDAOBgNVBAcMB0xpbW9nZXMxDDAKBgNVBAoMA1RNQzEM
MAoGA1UECwwDSU9UMQ4wDAYDVQQDDAVBQ1RNQzAeFw0yMjEyMjYxMjE4MTlaFw0y
MzAxMjUxMjE4MTlaMEsxCzAJBgNVBAYTAkZSMRAwDgYDVQQHDAdMaW1vZ2VzMQww
CgYDVQQKDANUTUMxDDAKBgNVBAsMA0lPVDEOMAwGA1UEAwwFQUNUTUMwWTATBgcq
hkjOPQIBBggqhkjOPQMBBwNCAAQfPmdpqouUrgoa2dh3SkCN1hsidembc0weGKvH
X8M165stzB4BZr7pOMngEmR4tXVfp3SeeS/y5Vp9SzOCqENVoxAwDjAMBgNVHRME
BTADAQH/MAoGCCqGSM49BAMCA0kAMEYCIQCr7I5ji3TlyUnoFwsHhLKWNZoW8RjU
2QCXciWSx+r/dAIhANii/EhpqvvFEZpMCVezmxuVotnO8FEJg+VhEfyeIEIJ
-----END CERTIFICATE-----
```

```
cat ecc.esp8266.pem
-----BEGIN CERTIFICATE-----
MIIBqjCCAVCgAwIBAgIULt5GDXJYM/gs+AGfgJA+XXI1wUswCgYIKoZIzj0EAwIw
SzELMAkGA1UEBhMCRlIxEDAOBgNVBAcMB0xpbW9nZXMxDDAKBgNVBAoMA1RNQzEM
MAoGA1UECwwDSU9UMQ4wDAYDVQQDDAVBQ1RNQzAeFw0yMjEyMjYxMjM5MjdaFw0z
MjEyMjMxMjM5MjdaME0xCzAJBgNVBAYTAkZSMRAwDgYDVQQHDAdMaW1vZ2VzMQww
CgYDVQQKDANUTUMxDDAKBgNVBAsMA0lPVDEQMA4GA1UEAwwHZXNwODI2NjBZMBMG
ByqGSM49AgEGCCqGSM49AwEHA0IABNg6ah44kFNPH08liyiMY10sWaqFAROup1fr
928AusR2NRWTH0VoeimrPITvUfos1WvqkzWB/GzKSuycc5odzSOjEDAOMAwGA1Ud
EwEB/wQCMAAwCgYIKoZIzj0EAwIDSAAwRQIhAN+B80BkA9ybWaPTfc823sNgeDaJ
rBMYkwyc7wxV3+QBAiASLd6hUbaLmFHwN45w68npFrvlIBRrv6fq8WrJf7OzdzAM
MAoGCCsGAQUFBwMC
-----END CERTIFICATE-----
```

On copie les certificats ecc.ca.pem, ecc.esp8266.pem, et la clé ecc.ca.key.pem dans le dossier my-app de la nouvelle application.

Ensuite on place les deux certificats vers le dossier fs à l'intérieur du dossier my-app.

```
$cp ecc.esp8266.pem  ecc.ca.pem fs
```

On ajoute le Mongoose os dans l'ESP8266 avec les commandes suivantes :

```
my-app $ sudo mos build --local --platform esp8266
my-app $ sudo mos flash
```

On installe la clé privée dans ATECC508 avec les commandes suivantes :

```
$ openssl rand -hex 32 > slot4.key
$ sudo mos -X atca-set-key 4 slot4.key --dry-run=false
$ sudo mos -X atca-set-key 0 ecc.esp8266.key.pem --write-key=slot4.key --dry-run=false
```

On vérifie que le certificat est bien pris en compte et que la communication est sécurisée avec la commande suivante :

```
$ mos console
```

La connexion au accés point

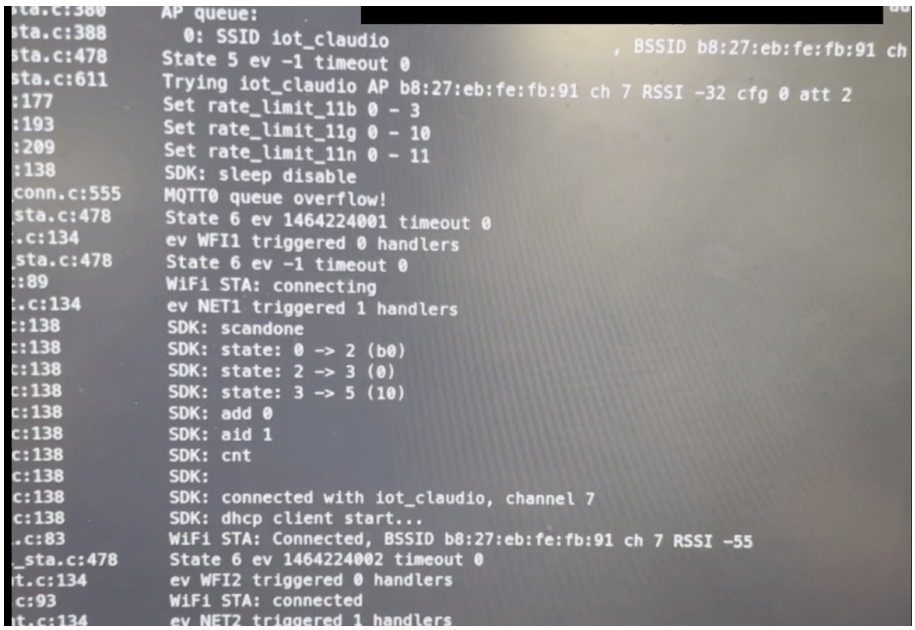

Figure 2 –

La configuration du certificat

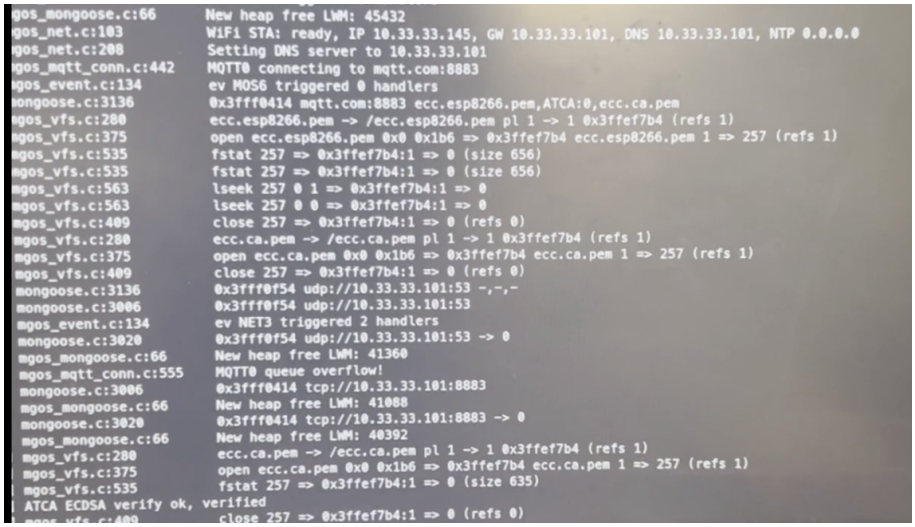

FIGURE 3 –

La publication des message par le client MQTT

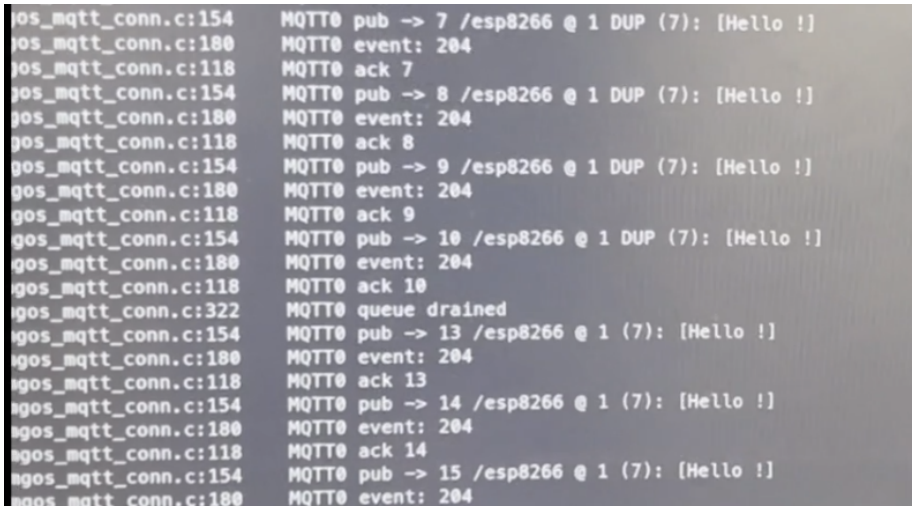

FIGURE 4 –

IV Communication LoRa sur le Raspberry Pi

Pour activer le bus SPI utilisé par le composant LoRa, nous allons modifier le fichier RASPI/boot/config.txt de la façon suivante.

```
...
# Uncomment some or all of these to enable
the optional hardware interfaces
#dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on
```

```
...
# Uncomment some or all of these to en
able the optional hardware interfaces
#dtparam=i2c_arm=on
#dtparam=i2s=on
dtparam=spi=on
dtoverlay=gpio-no-irq
```

FIGURE 5 –

Pour l'utilisation des pins GPIOs et du bus SPI , on a besoin de la librairie bcm2835 :

```
$ wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.71.tar.gz
$ tar zxvf bcm2835-1.71.tar.gz
$ cd bcm2835-1.71
$ ./configure
$ make
$ sudo make check
$ sudo make install
```

Pour l'utilisation du LoRa, nous utiliserons la librairie suivante :

```
$ git clone https://github.com/hallard/RadioHead
```

On se deplace dans le dossier suivant :

```
$ cd RadioHead/examples/raspi/rf95
```

On modifie les deux fichiers source : `rf95_server.cpp` et `rf95_client.cpp`, pour utiliser le dragino :

```
// LoRasPi board
// see https://github.com/hallard/LoRasPI
#define BOARD_LORASPI
```

```
// LoRasPi board
// see https://github.com/hallard/LoRas
PI
//#define BOARD_LORASPI
```

On désactive ce `#define` en premier. ⟹

```
// Dragino Raspberry PI hat
// see https://github.com/dragino/Lora
//#define BOARD_DRAGINO_PIHAT
```

```
// Dragino Raspberry PI hat
// see https://github.com/dragino/Lora
#define BOARD_DRAGINO_PIHAT
```

FIGURE 6 –

Modification du fichier rf95-client.cpp pour l'envoyé des données

```
/* A 256 bit key */
unsigned char *key = (unsigned char *)"01234567890123456789012345678901";
/* A 128 bit IV */
unsigned char *iv = (unsigned char *)"0123456789012345";
// get data from input
const char* data;
unsigned char * plaintext =(unsigned char *) argv[1];

/* Buffer for the decrypted text */
unsigned char ciphertext[128];
int decryptedtext_len, ciphertext_len;
/* Encrypt the plaintext */
ciphertext_len = encrypt (plaintext, strlen ((char *)plaintext), key, iv, ciphertext);
uint8_t* data2send = new uint8_t [ciphertext_len];
for(int i =0 ; i < ciphertext_len; i++) data2send[i]=ciphertext[i];
printf("Sending %02d bytes to node #%d => ", ciphertext_len, RF_GATEWAY_ID );
printbuffer(data2send, ciphertext_len);
printf("\n\n\n" );
rf95.send(data2send, ciphertext_len);
rf95.waitPacketSent();
exit(1);
```

Modification du fichier rf95-server.cpp pour pour la réception des données

```
if (rf95.recv(buf, &len)) {
printf("Packet[%02d] #%d => #%d %ddB:", len, from, to, rssi);
printbuffer(buf, len);
printf(" \n ");
unsigned char decryptedtext[128];
int decryptedtext_len;
/* Decrypt the ciphertext */
decryptedtext_len = decrypt(buf,len, key, iv,decryptedtext);
/* Add a NULL terminator. We are expecting printable text */
decryptedtext[decryptedtext_len] = '\0';
 /* Show the decrypted text */
printf("Decrypted text is:\n");
printf("%s\n", decryptedtext);
std::string convert;
convert.assign(decryptedtext, decryptedtext+decryptedtext_len);
char buffer[512];
std::string result = "";
std::string str = "python3 server_LORA.py "+convert;
const char * command = str.c_str();
FILE* pipe = popen(command, "r");
if (!pipe) throw std::runtime_error("popen() failed!");
try {
while (fgets(buffer, sizeof buffer, pipe) != NULL) {
       result += buffer;
}
} catch (std::string const& chaine){
    pclose(pipe);
    throw;
}
std::cout << result << std::endl;
pclose(pipe);
```

IV.I Chiffrement AES

Pour la réalisation du chiffrement des données entre les deux LORA , on a utilisé les fonctions EVP de la librairie OpenSSL qui fournissent une interface de haut niveau aux fonctions cryptographiques en langage c++. Et donc on a ajouté les fonctions suivantes aux fichiers rf95_client.cpp et rf95_server.cpp :

Fonction de chiffrement AES au rf95_client.cpp

```
int encrypt(unsigned char *plaintext, int plaintext_len, unsigned char *key,
        unsigned char *iv, unsigned char *ciphertext)
{
 EVP_CIPHER_CTX *ctx;

 int len;

 int ciphertext_len;

 /* Create and initialise the context */
 if(!(ctx = EVP_CIPHER_CTX_new()))
     handleErrors();

 /*
  * Initialise the encryption operation. IMPORTANT - ensure you use a key
  * and IV size appropriate for your cipher
  * In this example we are using 256 bit AES (i.e. a 256 bit key). The
  * IV size for *most* modes is the same as the block size. For AES this
  * is 128 bits
  */
 if(1!= EVP_EncryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv))
     handleErrors();

 /*
  * Provide the message to be encrypted, and obtain the encrypted output.
  * EVP_EncryptUpdate can be called multiple times if necessary
  */
 if(1!= EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext_len))
     handleErrors();
 ciphertext_len = len;

 /*
  * Finalise the encryption. Further ciphertext bytes may be written at
  * this stage.
  */
 if(1!= EVP_EncryptFinal_ex(ctx, ciphertext + len, &len))
     handleErrors();
 ciphertext_len += len;

 /* Clean up */
 EVP_CIPHER_CTX_free(ctx);

 return ciphertext_len;
}
```

Fonction de déchiffrement AES au rf95_server.cpp

```
1
2
3  int decrypt(unsigned char *ciphertext, int ciphertext_len, unsigned char *key,
4           unsigned char *iv, unsigned char *plaintext)
5  {
6   EVP_CIPHER_CTX *ctx;
7
8   int len;
9
10  int plaintext_len;
11
12  /* Create and initialise the context */
13  if(!(ctx = EVP_CIPHER_CTX_new()))
14      handleErrors();
15  /*
16   * Initialise the decryption operation. IMPORTANT - ensure you use a key
17   * and IV size appropriate for your cipher
18   * In this example we are using 256 bit AES (i.e. a 256 bit key). The
19   * IV size for *most* modes is the same as the block size. For AES this
20   * is 128 bits
21   */
22  if(1!= EVP_DecryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv))
23      handleErrors();
24  /*
25   * Provide the message to be decrypted, and obtain the plaintext output.
26   * EVP_DecryptUpdate can be called multiple times if necessary.
27   */
28  if(1!= EVP_DecryptUpdate(ctx, plaintext, &len, ciphertext, ciphertext_len))
29      handleErrors();
30  plaintext_len = len;
31  /*
32   * Finalise the decryption. Further plaintext bytes may be written at
33   * this stage.
34   */
35  if(1!= EVP_DecryptFinal_ex(ctx, plaintext + len, &len))
36      handleErrors();
37  plaintext_len += len;
38
39  /* Clean up */
40  EVP_CIPHER_CTX_free(ctx);
41  return plaintext_len;
42 }
```

Ensuite, pour compilation on a ajouté les flags suivant :

```
CC            = g++
CFLAGS        = -DRASPBERRY_PI -DBCM2835_NO_DELAY_COMPATIBILITY -D__BASEFILE__=\"$*\"
LIBS          = -lbcm2835 -lcrypto -lssl
RADIOHEADBASE = ../../..
INCLUDE       = -I$(RADIOHEADBASE)
```

FIGURE 7 –

Compilation et exécution :

```
$ make
$ sudo ./rf95_client
```

IV.II JSON Format

Avant d'envoyer les données via le LORA nous plaçons les données dans le format JSON selon l'extrait de code suivant.

```
## use rf95 programm to send the Data to onther endPoint LORA
data =jwt.encode( {'data':data.decode('utf-8') }, "TMC", algorithm='HS256')
cmd = subprocess.Popen("sudo ./RadioHead/examples/raspi/rf95/rf95_client %s "%data, shell=True
    ,stdout=subprocess.PIPE)
(resultat , ignorer) = cmd.communicate()
```

Nous récupérons les données au format JSON selon extrait du code suivant.

```
encoded =jwt.decode(data, "TMC", algorithms=['HS256'])
print(Fore.WHITE+" data decoded from JSON  => ",encoded)
data=encoded['data']
```

L'envoi d'une donnée par LORA

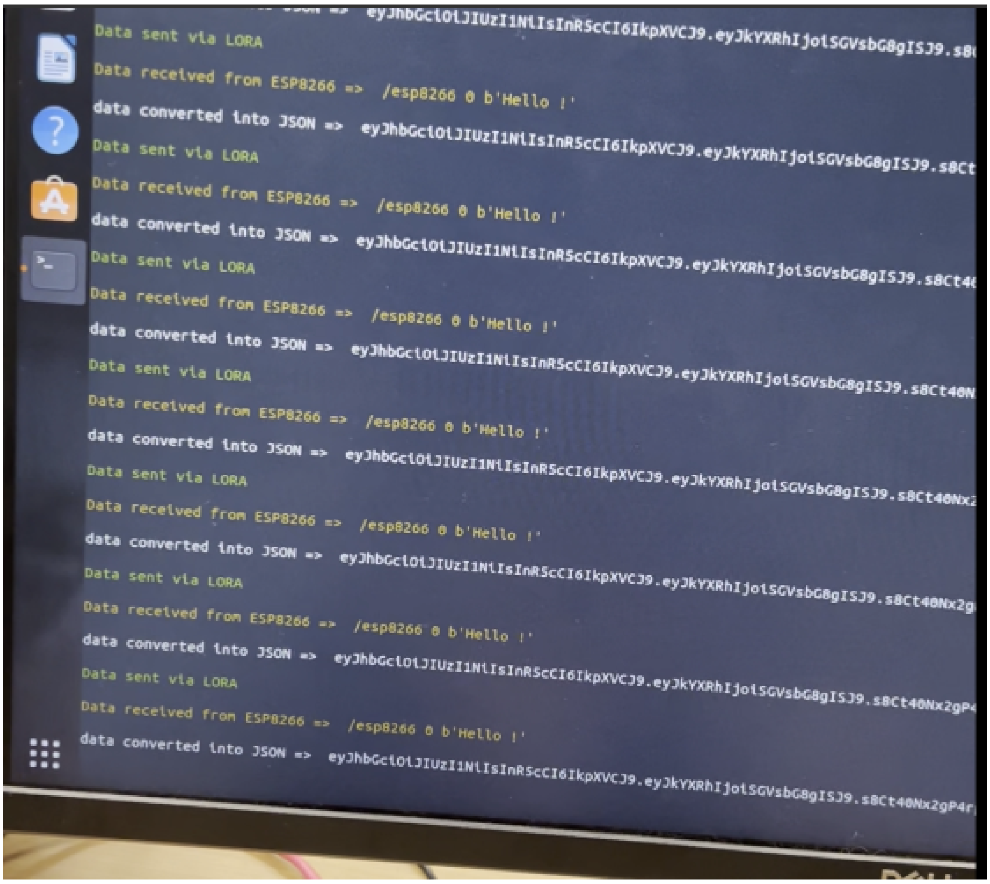

FIGURE 8 –

La réception d'une donnée par LORA

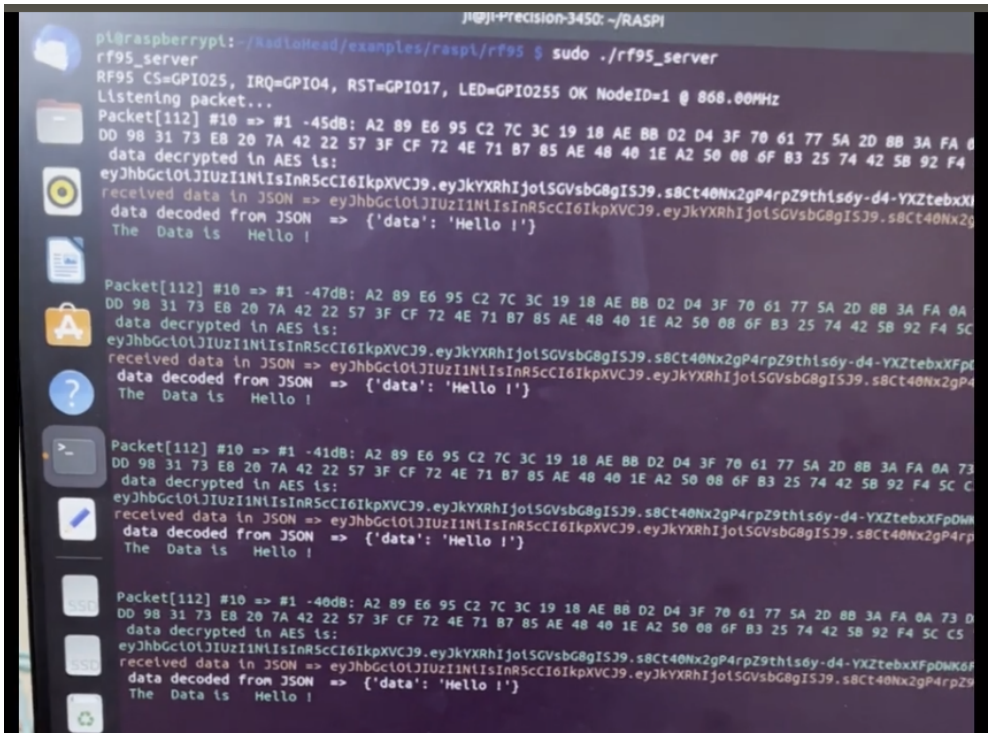

Figure 9 –

V Video Youtube

https://www.youtube.com/watch?v=i1hnMEcURoM