



UNIVERSITÉ DE LIMOGES
FACULTÉ DES SCIENCES ET TECHNIQUES

MASTER 2 CRYPTIS
UE : TMC

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

Présenté par :

CLAUDIO ANTONIO
NHAT BUI
YAWAVI LATEVI

Enseignant : P-F. Bonnefoi

Limoges , Février 2023

Table des matières

I	Introduction	3
II	Raspberry Pi & WiFi	3
III	Mongoose OS : MQTT client et publication sécurisée par ECC	7
IV	Communication LoRa sur le Raspberry Pi	11
IV.I	Chiffrement AES	13
IV.II	JSON Format	16
V	Video Youtube	17

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 »

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

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

```
1 pi@raspberrypi:~ $ cat /etc/hostapd/hostapd.conf
2 country_code=FR
3 interface=wlan0
4 ssid=iot_claudio
5 hw_mode=g
6 channel=7
7 wmm_enabled=0
8 macaddr_acl=0
9 auth_algs=1
10 ignore_broadcast_ssid=0
11 wpa=2
12 wpa_passphrase=123456789
13 wpa_key_mgmt=WPA-PSK
14 wpa_pairwise=TKIP
15 rsn_pairwise=CCMP
```

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

```

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

```

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

```

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

```

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

```

1 pi@raspberrypi:~ $ cat /etc/resolvconf.conf
2 # Configuration for resolvconf(8)
3 # See resolvconf.conf(5) for details
4
5 resolv_conf=/etc/resolv.conf
6 # If you run a local name server, you should uncomment the below line and
7 # configure your subscribers configuration files below.
8 name_servers=127.0.0.56
9
10 # Mirror the Debian package defaults for the below resolvers
11 # so that resolvconf integrates seemlessly.
12 dnsmasq_resolv=/var/run/dnsmasq/resolv.conf
13 pdnsd_conf=/etc/pdnsd.conf
14 unbound_conf=/var/cache/unbound/resolvconf_resolvers.conf
15

```

On définit un IP statique sur l'interface wlan0.

```

1 pi@raspberrypi:~ $ cat /etc/network/interfaces
2 # interfaces(5) file used by ifup(8) and ifdown(8)
3 # Include files from /etc/network/interfaces.d:
4 source /etc/network/interfaces.d
5 /*
6 allow-hotplug wlan0
7 iface wlan0 inet static
8   address 10.33.33.101
9   netmask 255.255.255.0
10  gateway 10.33.33.101
11

```

On démarre l'hostapd et le dnsmasq.

```

1 pi@raspberrypi:~ $ sudo systemctl unmask hostapd
2 pi@raspberrypi:~ $ sudo systemctl enable hostapd
3 pi@raspberrypi:~ $ sudo systemctl enable dnsmasq

```

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

```
pi@raspberrypi:~/CERTIFICATES $ ping -c 10 mqtt.com
PING mqtt.com (10.33.33.101) 56(84) bytes of data.
64 bytes from 10.33.33.101 (10.33.33.101): icmp_seq=1 ttl=64 time=0.129 ms
64 bytes from 10.33.33.101 (10.33.33.101): icmp_seq=2 ttl=64 time=0.148 ms
64 bytes from 10.33.33.101 (10.33.33.101): icmp_seq=3 ttl=64 time=0.088 ms
64 bytes from 10.33.33.101 (10.33.33.101): icmp_seq=4 ttl=64 time=0.092 ms
64 bytes from 10.33.33.101 (10.33.33.101): icmp_seq=5 ttl=64 time=0.154 ms
64 bytes from 10.33.33.101 (10.33.33.101): icmp_seq=6 ttl=64 time=0.094 ms
64 bytes from 10.33.33.101 (10.33.33.101): icmp_seq=7 ttl=64 time=0.099 ms
64 bytes from 10.33.33.101 (10.33.33.101): icmp_seq=8 ttl=64 time=0.156 ms
```

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.

```
1 pi@raspberrypi:~/CERTIFICATES $ openssl ecparam -out ecc.ca.key.pem -name prime256v1 -
genkey
2
3 pi@raspberrypi:~/CERTIFICATES $ openssl ecparam -out ecc.raspberry.key.pem -name
prime256v1 -genkey
4
5 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

```
1 pi@raspberrypi:~/CERTIFICATES $
2 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)

```
1 pi@raspberrypi:~/CERTIFICATES $
2 $ 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
3
4 $ openssl x509 -req -days 3650 -CA ecc.ca.pem -CAkey ecc.ca.key.pem -CAcreateserial -
extfile <(printf "basicConstraints=critical,CA:FALSE") -in
5 ecc.raspberry.csr.pem -text -out ecc.raspberry.pem -addtrust clientAuth
```

Génération et signature du certificat pour le client (Esp8266)

```
1 pi@raspberrypi:~/CERTIFICATES $  
2 $ openssl req -config <(printf "[req]\nsubject_dn=%s\nbasicConstraints=CA:FALSE") -new -subj "/C=FR/L=Limoges/O=TMC/OU=IOT/CN=esp8266" -  
3 reqexts ext -sha256 -key ecc.esp8266.key.pem -text -out ecc.esp8266.csr.pem  
4 $ 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

```
1 pi@raspberrypi:~ $ sudo apt-get install mosquitto  
2 pi@raspberrypi:~ $ sudo apt-get install mosquitto-clients  
3
```

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

```
1 pi@raspberrypi:~ $  
2 $ sudo cp /CERTIFICATES/ecc.ca.pem /etc/mosquitto/ca_certificates/  
3 $ sudo cp /CERTIFICATES/ecc.raspberry.pem /etc/mosquitto/certs/  
4 $ 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".

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

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

```
1 pi@raspberrypi:~ $ cat /etc/mosquitto/mosquitto.conf  
2 allow_anonymous false  
3 password_file /etc/mosquitto/mosquitto_passwd  
4 listener 8883  
5 cafile /etc/mosquitto/ca_certificates/ecc.ca.pem  
6 certfile /etc/mosquitto/certs/ecc.raspberry.pem  
7 keyfile /etc/mosquitto/certs/ecc.raspberry.key.pem  
8 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.

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

III Mongoose OS : MQTT client et publication sécurisée par ECC

L'installation du Mongoose OS

```
1 $ sudo add-apt-repository ppa:mongoose-os/mos
2 $ sudo apt-get update
3 $ sudo apt-get install mos
```

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

```
1 $ sudo apt install docker.io
2 $ sudo groupadd docker
3 $ sudo usermod -aG docker $USER
```

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

```
1 $ git clone https://github.com/mongoose-os-apps/empty my-app
2 $ cd my-app
3 $ cat mos.yml
4
5 ....
6
7 build_vars:
8     MGOSMBEDTLS_ENABLE_ATCA: 1
9 config_schema:
10    - ["debug.level", 3]
11    - ["sys.atca.enable", "b", true, {title: "Enable the chip"}]
12    - ["i2c.enable", "b", true, {title: "Enable I2C"}]
13    - ["sys.atca.i2c_addr", "i", 0x60, {title: "I2C address of the chip"}]
14    - ["mqtt.enable", "b", true, {title: "Enable MQTT"}]
15    - ["mqtt.server", "s", "mqtt.com:8883", {title: "MQTT server"}]
16    - ["mqtt.pub", "s", "/esp8266", {title: "Publish topic"}]
17    - ["mqtt.user", "s", "tmc", {title: "User name"}]
18    - ["mqtt.pass", "s", "iot", {title: "Password"}]
19    - ["mqtt.ssl_ca_cert", "s", "ecc.ca.pem", {title: "Verify server certificate using this CA bundle"}]
20    - ["mqtt.ssl_cert", "s", "ecc.esp8266.pem", {title: "Client certificate to present to the server"}]
21    - ["mqtt.ssl_key", "ATCA:0"]
22    - ["wifi.ap.enable", "b", false, {title: "Enable"}]
23    - ["wifi.sta.enable", "b", true, {title: "Connect to existing WiFi"}]
24    - ["wifi.sta.ssid", "s", "iot_claudio", {title: "SSID"}]
25    - ["wifi.sta.pass", "s", "123456789", {title: "Password", type: "password"}]
26
27 ....
```

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

```
1 cat ecc.ca.pem
2 -----BEGIN CERTIFICATE-----
3 MIIBqTCCAU6gAwIBAgIUZvV4MFZwpjQOQDY2+VAo854viQgwCgYIKoZIzj0EAwIw
4 SzELMAkGA1UEBhMCRIIxEDAOBgNVBAcMB0xpB9nZXmxDDAKBgNVBAoMA1RNQzEM
5 MAoGA1UECwwDSU9UMQ4wDAYDVQQDDAVBQ1RNQzAeFwOyMjEyMjYxMjE4MT1aFwOy
6 MzAxMjUxMjE4MT1aMEsxCzAJBgNVBAYTAkZSMRAwDgYDVQQHDAadMaW1vZ2VzMQw
7 CgYDVQQKDANUTUMxDDAKBgNVBAsMA01PVDEOMAwGA1UEAwxFQUNUTUMwWTATBgcq
8 hkjOPQIBBggqhkJOPQMBBwNCAAQfPmdpqouUrgoa2dh3SkCN1hsidembc0weGKvh
9 X8M165stzB4BZr7pOMngEmR4tXVfp3SeeS/y5Vp9Sz0CqENVoxAwDjAMBgNVHRME
10 BTADAQH/MAoGCCqGSM49BAMCA0kAMEYCIQCrtI5ji3TlyUnoFwsHhLKNZoW8Rju
11 2QCXciWSx+r/dAIhAnii/EhpqvvFEZpMCVezmxuVotn08FEJg+VhEfyeIEIJ
12 -----END CERTIFICATE-----
```

```
1 cat ecc.esp8266.pem
2 -----BEGIN CERTIFICATE-----
3 MIIBqjCCAVCgAwIBAgIULT5GDXJYM/gs+AGfgJA+XXI1wUswCgYIKoZIzj0EAwIw
4 SzELMAkGA1UEBhMCRIIxEDAOBgNVBAcMB0xpB9nZXmxDDAKBgNVBAoMA1RNQzEM
5 MAoGA1UECwwDSU9UMQ4wDAYDVQQDDAVBQ1RNQzAeFwOyMjEyMjYxMjM5MjdaFwOz
6 MjEyMjMxMjM5MjdaME0xCzAJBgNVBAYTAkZSMRAwDgYDVQQHDAadMaW1vZ2VzMQw
7 CgYDVQQKDANUTUMxDDAKBgNVBAsMA01PVDEQMA4GA1UEAwHZXNwODI2NjBZMBMG
8 ByqGSM49AgEGCCqGSM49AWEHA0IABNg6ah44kFNPH081iyiMY10sWaqFAROup1fr
9 928AusR2NRWTH0VoeimrPITvUfos1WvqkzWB/GzKSuycc5odzS0jEDAOMAwGA1Ud
10 EwEB/wQCMAAwCgYIKoZIzj0EAwIDSAAwRQIhAN+B80BkA9ybWaPTfc823sNgeDaJ
11 rBMYkwyc7wxV3+QBAiASLd6hUbaLmFHN45w68npFrVlIBRrv6fq8WrJf70zdzAM
12 MAoGCCsGAQUFBwMC
13 -----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.

```
1 $cp ecc.esp8266.pem ecc.ca.pem fs
```

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

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

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

```
1 $ openssl rand -hex 32 > slot4.key
2 $ sudo mos -X atca-set-key 4 slot4.key --dry-run=false
3 $ 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 :

```
1 $ mos console
```

La connexion au accés point

```
sta.c:388 AP queue: [REDACTED]
sta.c:388   0: SSID iot_claudio , BSSID b8:27:eb:fe:fb:91 ch
sta.c:478 State 5 ev -1 timeout 0
sta.c:611 Trying iot_claudio AP b8:27:eb:fe:fb:91 ch 7 RSSI -32 cfg 0 att 2
:c:177 Set rate_limit_11b 0 - 3
:c:193 Set rate_limit_11g 0 - 10
:c:209 Set rate_limit_11n 0 - 11
:c:138 SDK: sleep disable
conn.c:555 MQTT0 queue overflow!
sta.c:478 State 6 ev 1464224001 timeout 0
:c:134 ev WiFi triggered 0 handlers
sta.c:478 State 6 ev -1 timeout 0
:c:89 WiFi STA: connecting
:c:134 ev NET1 triggered 1 handlers
:c:138 SDK: scandone
:c:138 SDK: state: 0 -> 2 (b0)
:c:138 SDK: state: 2 -> 3 (0)
:c:138 SDK: state: 3 -> 5 (10)
:c:138 SDK: add 0
:c:138 SDK: aid 1
:c:138 SDK: cnt
:c:138 SDK:
:c:138 SDK: connected with iot_claudio, channel 7
:c:138 SDK: dhcp client start...
:c:83 WiFi STA: Connected, BSSID b8:27:eb:fe:fb:91 ch 7 RSSI -55
_sta.c:478 State 6 ev 1464224002 timeout 0
:t.c:134 ev WiFi2 triggered 0 handlers
:c:93 WiFi STA: connected
:t.c:134 ev NET2 triggered 1 handlers
```

FIGURE 2 –

La configuration du certificat

```

gos_mongoose.c:66      New heap free LWM: 45432
gos_net.c:183          WiFi STA ready, IP 10.33.33.145, GW 10.33.33.101, DNS 10.33.33.101, NTP 0.0.0.0
gos_net.c:208
gos_mqtt_conn.c:442    Setting DNS server to 10.33.33.101
MQTT0 connecting to mqtt.com:8883
gos_event.c:134        ev M056 triggered 2 handlers
mongoose.c:3136
gos_vfs.c:280
gos_vfs.c:375
gos_vfs.c:535
gos_vfs.c:535
gos_vfs.c:563
gos_vfs.c:563
gos_vfs.c:409
gos_vfs.c:280
gos_vfs.c:375
gos_vfs.c:409
mongoose.c:3136
mongoose.c:3006
mongoose.c:3028
gos_event.c:134
mongoose.c:3028
mongoose.c:66          New heap free LWM: 41360
gos_mqtt_conn.c:555    MQTT0 queue overflow!
mongoose.c:3006
mongoose.c:66          New heap free LWM: 41088
mongoose.c:3028
gos_mongoose.c:66      New heap free LWM: 40392
gos_vfs.c:280
gos_vfs.c:375
gos_vfs.c:535
gos_vfs.c:535          fstat 257 => 0x3ffef7b4:1 => 0 (size 635)
ATCA ECDSA verify ok, verified
gos_vfs.c:409          close 257 => 0x3ffef7b4:1 => 0 (refs 0)

```

FIGURE 3 –

La publication des message par le client MQTT

```

gos_mqtt_conn.c:154    MQTT0 pub -> 7 /esp8266 @ 1 DUP (7): [Hello !]
gos_mqtt_conn.c:180    MQTT0 event: 204
gos_mqtt_conn.c:118    MQTT0 ack 7
gos_mqtt_conn.c:154    MQTT0 pub -> 8 /esp8266 @ 1 DUP (7): [Hello !]
gos_mqtt_conn.c:180    MQTT0 event: 204
gos_mqtt_conn.c:118    MQTT0 ack 8
gos_mqtt_conn.c:154    MQTT0 pub -> 9 /esp8266 @ 1 DUP (7): [Hello !]
gos_mqtt_conn.c:180    MQTT0 event: 204
gos_mqtt_conn.c:118    MQTT0 ack 9
gos_mqtt_conn.c:154    MQTT0 pub -> 10 /esp8266 @ 1 DUP (7): [Hello !]
gos_mqtt_conn.c:180   MQTT0 event: 204
gos_mqtt_conn.c:118   MQTT0 ack 10
gos_mqtt_conn.c:322   MQTT0 queue drained
gos_mqtt_conn.c:154   MQTT0 pub -> 13 /esp8266 @ 1 (7): [Hello !]
gos_mqtt_conn.c:180   MQTT0 event: 204
gos_mqtt_conn.c:118   MQTT0 ack 13
gos_mqtt_conn.c:154   MQTT0 pub -> 14 /esp8266 @ 1 (7): [Hello !]
gos_mqtt_conn.c:180   MQTT0 event: 204
gos_mqtt_conn.c:118   MQTT0 ack 14
gos_mqtt_conn.c:154   MQTT0 pub -> 15 /esp8266 @ 1 (7): [Hello !]
gos_mqtt_conn.c:180   MQTT0 event: 204

```

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  
...  
#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 :

```
1 $ wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.71.tar.gz  
2 $ tar zxvf bcm2835-1.71.tar.gz  
3 $ cd bcm2835-1.71  
4 $ ./configure  
5 $ make  
6 $ sudo make check  
7 $ sudo make install
```

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

```
1 $ git clone https://github.com/hallard/RadioHead
```

On se déplace dans le dossier suivant :

```
1 $ 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  
On désactive ce #define en premier. ==> // LoRasPi board  
// see https://github.com/hallard/LoRasPI  
// #define BOARD_LORASPI  
// 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

```
1 /* A 256 bit key */
2 unsigned char *key = (unsigned char *)"01234567890123456789012345678901";
3 /* A 128 bit IV */
4 unsigned char *iv = (unsigned char *)"0123456789012345";
5 // get data from input
6 const char* data;
7 unsigned char * plaintext =(unsigned char *) argv[1];
8
9 /* Buffer for the decrypted text */
10 unsigned char ciphertext[128];
11 int decryptedtext_len, ciphertext_len;
12 /* Encrypt the plaintext */
13 ciphertext_len = encrypt(plaintext, strlen ((char *)plaintext), key, iv, ciphertext);
14 uint8_t* data2send = new uint8_t [ciphertext_len];
15 for(int i = 0 ; i < ciphertext_len; i++) data2send[i]=ciphertext[i];
16 printf("Sending %02d bytes to node #%d => ", ciphertext_len, RF_GATEWAY_ID );
17 printbuffer(data2send, ciphertext_len);
18 printf("\n\n\n");
19 rf95.send(data2send, ciphertext_len);
20 rf95.waitPacketSent();
21 exit(1);
```

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

```
1 if (rf95.recv(buf, &len)) {
2 printf("Packet[%02d] #%d => #%d %dB:", len, from, to, rssi);
3 printbuffer(buf, len);
4 printf(" \n ");
5 unsigned char decryptedtext[128];
6 int decryptedtext_len;
7 /* Decrypt the ciphertext */
8 decryptedtext_len = decrypt(buf,len, key, iv,decryptedtext);
9 /* Add a NULL terminator. We are expecting printable text */
10 decryptedtext[decryptedtext_len] = '\0';
11 /* Show the decrypted text */
12 printf("Decrypted text is:\n");
13 printf("%s\n", decryptedtext);
14 std::string convert;
15 convert.assign(decryptedtext, decryptedtext+decryptedtext_len);
16 char buffer[512];
17 std::string result = "";
18 std::string str = "python3 server_LORA.py "+convert;
19 const char * command = str.c_str();
20 FILE* pipe = popen(command, "r");
21 if (!pipe) throw std::runtime_error("popen() failed!");
22 try {
23 while (fgets(buffer, sizeof buffer, pipe) != NULL) {
24     result += buffer;
25 }
26 } catch (std::string const& chaine){
27     pclose(pipe);
28     throw;
29 }
30 std::cout << result << std::endl;
31 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

```
1 int encrypt(unsigned char *plaintext, int plaintext_len, unsigned char *key,
2             unsigned char *iv, unsigned char *ciphertext)
3 {
4     EVP_CIPHER_CTX *ctx;
5
6     int len;
7
8     int ciphertext_len;
9
10    /* Create and initialise the context */
11    if(!(ctx = EVP_CIPHER_CTX_new()))
12        handleErrors();
13
14    /*
15     * Initialise the encryption operation. IMPORTANT - ensure you use a key
16     * and IV size appropriate for your cipher
17     * In this example we are using 256 bit AES (i.e. a 256 bit key). The
18     * IV size for *most* modes is the same as the block size. For AES this
19     * is 128 bits
20     */
21    if(1!= EVP_EncryptInit_ex(ctx, EVP_aes_256_cbc(), NULL, key, iv))
22        handleErrors();
23
24    /*
25     * Provide the message to be encrypted, and obtain the encrypted output.
26     * EVP_EncryptUpdate can be called multiple times if necessary
27     */
28    if(1!= EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext, plaintext_len))
29        handleErrors();
30    ciphertext_len = len;
31
32    /*
33     * Finalise the encryption. Further ciphertext bytes may be written at
34     * this stage.
35     */
36    if(1!= EVP_EncryptFinal_ex(ctx, ciphertext + len, &len))
37        handleErrors();
38    ciphertext_len += len;
39
40    /* Clean up */
41    EVP_CIPHER_CTX_free(ctx);
42
43    return ciphertext_len;
44 }
```

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__=\"$*\"
LIBS        = -l bcm2835 -lcrypto -lssl
RADIOHEADBASE = .../...
INCLUDE     = -I$(RADIOHEADBASE)
```

FIGURE 7 –

Compilation et exécution :

```
1 $ make
2 $ 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.

```

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

```

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

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

L'envoi d'une donnée par LORA

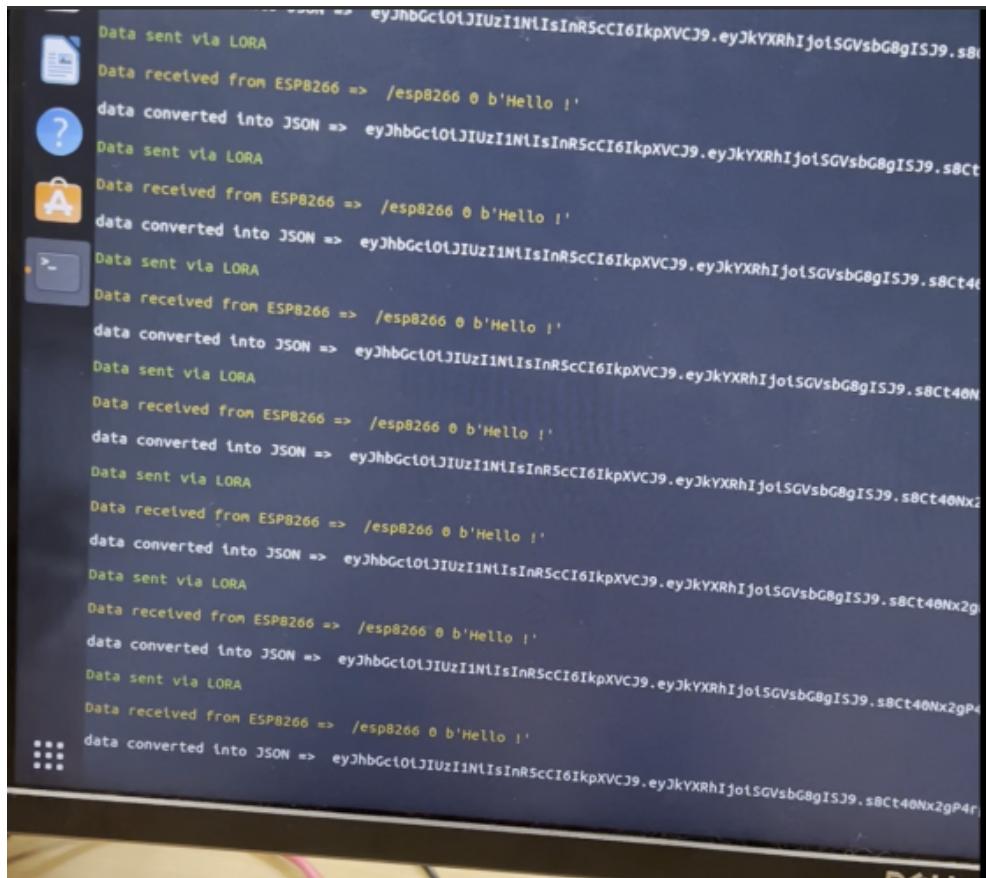


FIGURE 8 -

La réception d'une donnée par LORA

```
pi@raspberrypi:~/RadioHead/examples/raspi/rf95 $ sudo ./rf95_server
RF95 CS=GPIO25, IRQ=GPIO4, RST=GPIO17, LED=GPIO255 OK NodeID=1 @ 868.00MHz
Listening packet...
Packet[112] #10 => #1 -45dB: A2 89 E6 95 C2 7C 3C 19 18 AE BB D2 D4 3F 70 61 77 5A 2D 8B 3A FA 0A
DD 98 31 73 E8 20 7A 42 22 57 3F CF 72 4E 71 B7 85 AE 48 40 1E A2 50 08 6F B3 25 74 42 5B 92 F4
data decrypted in AES is:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJXRHijoLSGVsbG8gISJ9.s8Ct40Nx2gP4rpZ9this6y-d4-YXZtebxX
data decoded from JSON => {'data': 'Hello !'}
The Data is Hello !

Packet[112] #10 => #1 -47dB: A2 89 E6 95 C2 7C 3C 19 18 AE BB D2 D4 3F 70 61 77 5A 2D 8B 3A FA 0A
DD 98 31 73 E8 20 7A 42 22 57 3F CF 72 4E 71 B7 85 AE 48 40 1E A2 50 08 6F B3 25 74 42 5B 92 F4 5C
data decrypted in AES is:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJXRHijoLSGVsbG8gISJ9.s8Ct40Nx2gP4rpZ9this6y-d4-YXZtebxXfpD
data decoded from JSON => {'data': 'Hello !'}
The Data is Hello !

Packet[112] #10 => #1 -41dB: A2 89 E6 95 C2 7C 3C 19 18 AE BB D2 D4 3F 70 61 77 5A 2D 8B 3A FA 0A 73
DD 98 31 73 E8 20 7A 42 22 57 3F CF 72 4E 71 B7 85 AE 48 40 1E A2 50 08 6F B3 25 74 42 5B 92 F4 5C C5
data decrypted in AES is:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJXRHijoLSGVsbG8gISJ9.s8Ct40Nx2gP4rpZ9this6y-d4-YXZtebxXfpDwF
data decoded from JSON => {'data': 'Hello !'}
The Data is Hello !

Packet[112] #10 => #1 -40dB: A2 89 E6 95 C2 7C 3C 19 18 AE BB D2 D4 3F 70 61 77 5A 2D 8B 3A FA 0A 73 D
DD 98 31 73 E8 20 7A 42 22 57 3F CF 72 4E 71 B7 85 AE 48 40 1E A2 50 08 6F B3 25 74 42 5B 92 F4 5C
data decrypted in AES is:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJXRHijoLSGVsbG8gISJ9.s8Ct40Nx2gP4rpZ9this6y-d4-YXZtebxXfpDwF6
data decoded from JSON => {'data': 'Hello !'}
The Data is Hello !
```

FIGURE 9 –

V Video Youtube

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