

---

# COMMUNICATING MOBILE TERMINALS REPORT

---

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

**Author**  
Viet-Huy HA

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Communication</b>	<b>4</b>
2.1	WiFi and MQTT . . . . .	4
2.1.1	WiFi connection from ESP8266 to Raspberry Pi . . . . .	4
2.1.2	Connection between MQTT client and MQTT server . . . . .	5
2.2	LoRa . . . . .	6
<b>3</b>	<b>Encryption</b>	<b>6</b>
3.1	Elliptic-curve cryptography . . . . .	6
3.2	AES . . . . .	7
<b>4</b>	<b>Conclusion</b>	<b>8</b>

# 1 Introduction

The objective of this project is to establish communication between an MQTT client and an MQTT server (both implemented on Raspberry Pis) using LoRa technology. The MQTT client is connected to an ESP8266 module, and the MQTT server is configured to access the Internet. A critical step before initiating communication is the authentication process for both the client and server. For this purpose, the ATECC608 chip is employed to facilitate elliptic-curve cryptography, enabling secure authentication using their respective credentials. Furthermore, to safeguard the data exchanged between the two entities after successful authentication, AES encryption is utilized. The entire system's structure is depicted in Figure 1.

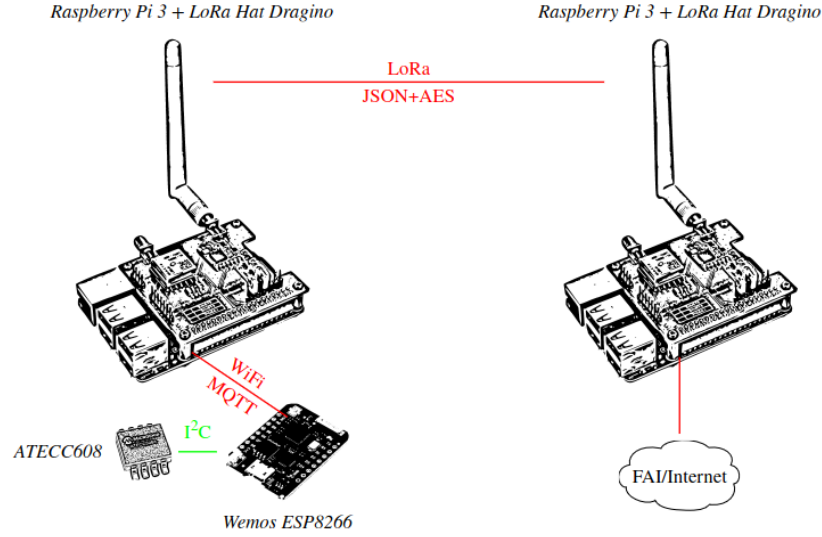


Figure 1: System Architecture

Then we set up devices as following:



Figure 2: System Architecture

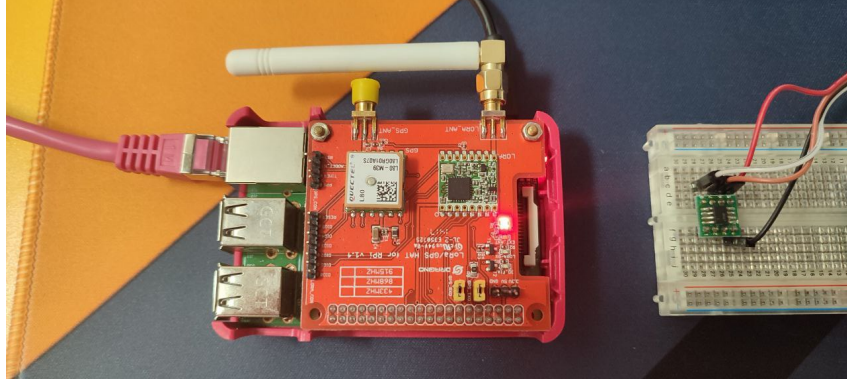


Figure 3: Raspberry Pi

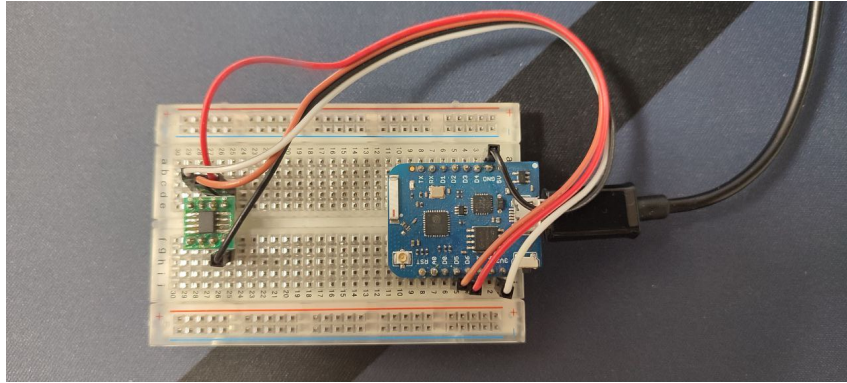


Figure 4: ESP8266

Within the confines of this report, we focus solely on the outcomes related to setting up communication, encryption, and data exchanges, as specified by the project requirements. For in-depth information on the implementation process, please consult the document titled `TMCPProject_ConfigurationInstruction.pdf` and the demo video on Youtube [here](#).

## 2 Communication

### 2.1 WiFi and MQTT

#### 2.1.1 WiFi connection from ESP8266 to Raspberry Pi

Initially, we configured a WiFi access point on the Raspberry Pi by installing the `hostapd` and `dnsmasq` packages. The `hostapd` package enables the creation of a wireless hotspot, and `dnsmasq` allows for the setting up of a DNS and DHCP server.

```

[Jan 28 20:15:46.157] mgos_wlfl_sta.c:611 Trying ahnvn1 AP b8:27:eb:78:87:13 ch 7 RSSI -31 cfg 0 att 2
[Jan 28 20:15:46.170] esp_wlfl.c:177 Set rate_limit_11b 0 - 3
[Jan 28 20:15:46.170] esp_wlfl.c:193 Set rate_limit_11g 0 - 10
[Jan 28 20:15:46.170] esp_wlfl.c:209 Set rate_limit_11n 0 - 11
[Jan 28 20:15:46.175] esp_main.c:138 SDK: sleep disable
[Jan 28 20:15:46.181] mgos_wlfl_sta.c:478 State 6 ev 1464224001 timeout 0
[Jan 28 20:15:46.186] mgos_event.c:134 ev WiFi triggered 0 handlers
[Jan 28 20:15:46.191] mgos_wlfl_sta.c:478 State 6 ev -1 timeout 0
[Jan 28 20:15:46.195] mgos_net.c:89 WiFi STA: connecting
[Jan 28 20:15:46.195] mgos_event.c:134 ev NET1 triggered 1 handlers
[Jan 28 20:15:46.279] esp_main.c:138 SDK: scandone
[Jan 28 20:15:46.284] esp_main.c:138 SDK: state: 0 -> 2 (b0)
[Jan 28 20:15:46.289] esp_main.c:138 SDK: state: 2 -> 3 (0)
[Jan 28 20:15:46.306] esp_main.c:138 SDK: state: 3 -> 5 (10)
[Jan 28 20:15:46.313] esp_main.c:138 SDK: add 0
[Jan 28 20:15:46.313] esp_main.c:138 SDK: add 1
[Jan 28 20:15:46.316] esp_main.c:138 SDK: cnt
[Jan 28 20:15:46.328] esp_main.c:138 SDK:
[Jan 28 20:15:46.328] esp_main.c:138 SDK: connected with ahnvn1, channel 7
[Jan 28 20:15:46.338] esp_main.c:138 SDK: dhcp client start...
[Jan 28 20:15:46.347] mgos_wlfl.c:83 WiFi STA: Connected, BSSID b8:27:eb:78:87:13 ch 7 RSSI -33
[Jan 28 20:15:46.357] mgos_wlfl_sta.c:478 State 6 ev 1464224002 timeout 0
[Jan 28 20:15:46.357] mgos_event.c:134 ev WiFi2 triggered 0 handlers
[Jan 28 20:15:46.363] mgos_net.c:93 WiFi STA: connected
[Jan 28 20:15:46.367] mgos_event.c:134 ev NET2 triggered 1 handlers
[Jan 28 20:15:47.379] esp_main.c:138 SDK: ip:192.168.2.111,mask:255.255.0,gw:192.168.2.1
[Jan 28 20:15:47.384] mgos_wlfl_sta.c:478 State 7 ev 1464224003 timeout 0
[Jan 28 20:15:47.390] mgos_wlfl_sta.c:698 Saving AP ahnvn1 b8:27:eb:78:87:13 ch 7
[Jan 28 20:15:47.398] mgos_vfs.c:280 conf1.json.try -> /conf1.json.try pl 1 -> 1 0x3ffef7bc (refs 1)
[Jan 28 20:15:47.411] mgos_vfs.c:506 stat conf1.json.try => 0x3ffef7bc conf1.json.try => -1 (size 0)
[Jan 28 20:15:47.418] mgos_vfs.c:280 conf1.json -> /conf1.json.pl 1 -> 1 0x3ffef7bc (refs 1)
[Jan 28 20:15:47.429] mgos_vfs.c:506 stat conf1.json => 0x3ffef7bc conf1.json => -1 (size 0)
[Jan 28 20:15:47.437] mgos_vfs.c:280 conf2.json.try -> /conf2.json.try pl 1 -> 1 0x3ffef7bc (refs 1)
[Jan 28 20:15:47.450] mgos_vfs.c:506 stat conf2.json.try => 0x3ffef7bc conf2.json.try => -1 (size 0)
[Jan 28 20:15:47.457] mgos_vfs.c:280 conf2.json -> /conf2.json.pl 1 -> 1 0x3ffef7bc (refs 1)

```

Figure 5: ESP8266 connects to WiFi

### 2.1.2 Connection between MQTT client and MQTT server

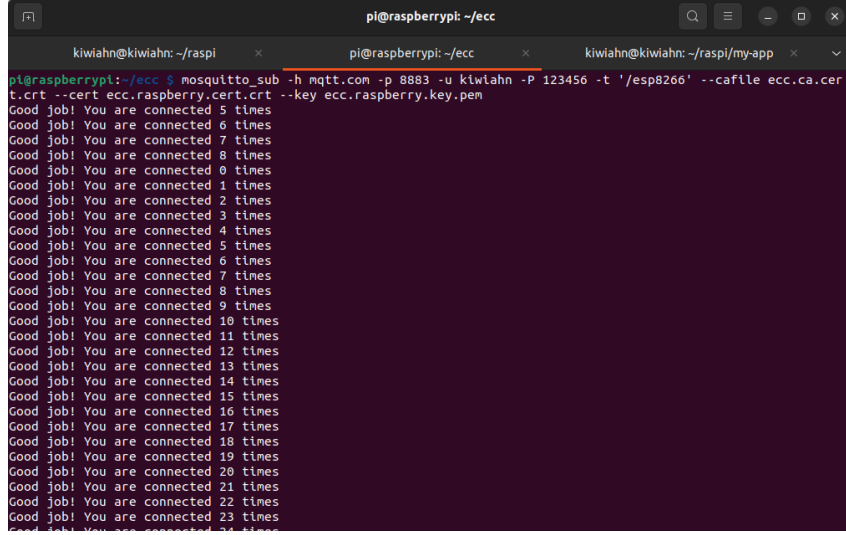
The MQTT client is an ESP8266 module designed to connect to an MQTT server, which is a Raspberry Pi in this scenario. The client is programmed to send messages with the content “Good job! You are connected” to the topic ‘esp8266’ every 5 seconds. Concurrently, the MQTT server is subscribed to the same topic to receive these messages.

```

[Jan 28 20:15:48.716] mgos_mqtt_conn.c:180 MQTT0 event: 209
[Jan 28 20:15:48.755] mgos_mqtt_conn.c:180 MQTT0 event: 209
[Jan 28 20:15:48.760] mg_rpc.c:500 0x3ffef73c CHAN OPEN (MQTT)
[Jan 28 20:15:48.764] mgos_event.c:134 ev RPC0 triggered 0 handlers
[Jan 28 20:15:48.772] mgos_mqtt_conn.c:180 MQTT0 event: 204
[Jan 28 20:15:48.772] mgos_mqtt_conn.c:118 MQTT0 ack 1
[Jan 28 20:15:48.782] mgos_mqtt_conn.c:154 MQTT0 pub -> 2 /esp8266 @ 1 DUP (35): [Good job! You are connecte
1 times]
[Jan 28 20:15:48.799] mgos_mqtt_conn.c:180 MQTT0 event: 204
[Jan 28 20:15:48.799] mgos_mqtt_conn.c:118 MQTT0 ack 2
[Jan 28 20:15:48.804] mgos_mqtt_conn.c:322 MQTT0 queue drained
[Jan 28 20:15:49.125] mgos_mqtt_conn.c:154 MQTT0 pub -> 5 /esp8266 @ 1 (35): [Good job! You are connected 2
times]
[Jan 28 20:15:49.142] mgos_mqtt_conn.c:180 MQTT0 event: 204
[Jan 28 20:15:49.142] mgos_mqtt_conn.c:118 MQTT0 ack 5
[Jan 28 20:15:54.125] mgos_mqtt_conn.c:154 MQTT0 pub -> 6 /esp8266 @ 1 (35): [Good job! You are connected 3
times]
[Jan 28 20:15:54.141] mgos_mqtt_conn.c:180 MQTT0 event: 204
[Jan 28 20:15:54.141] mgos_mqtt_conn.c:118 MQTT0 ack 6
[Jan 28 20:15:56.317] esp_main.c:138 SDK: pin open,type:0 0
[Jan 28 20:15:59.125] mgos_mqtt_conn.c:154 MQTT0 pub -> 7 /esp8266 @ 1 (35): [Good job! You are connected 4
times]
[Jan 28 20:15:59.141] mgos_mqtt_conn.c:180 MQTT0 event: 204
[Jan 28 20:15:59.141] mgos_mqtt_conn.c:118 MQTT0 ack 7
[Jan 28 20:16:01.180] mgos_wlfl_sta.c:478 State 8 ev -1 timeout 1
[Jan 28 20:16:04.125] mgos_mqtt_conn.c:154 MQTT0 pub -> 8 /esp8266 @ 1 (35): [Good job! You are connected 5
times]
[Jan 28 20:16:04.140] mgos_mqtt_conn.c:180 MQTT0 event: 204
[Jan 28 20:16:04.140] mgos_mqtt_conn.c:118 MQTT0 ack 8
[Jan 28 20:16:09.125] mgos_mqtt_conn.c:154 MQTT0 pub -> 9 /esp8266 @ 1 (35): [Good job! You are connected 6
times]
[Jan 28 20:16:09.141] mgos_mqtt_conn.c:180 MQTT0 event: 204
[Jan 28 20:16:09.141] mgos_mqtt_conn.c:118 MQTT0 ack 9
[Jan 28 20:16:14.125] mgos_mqtt_conn.c:154 MQTT0 pub -> 10 /esp8266 @ 1 (35): [Good job! You are connected 7
times]

```

Figure 6: ESP8266 publishes to the topic `esp8266`

A terminal window titled 'pi@raspberrypi: ~/ecc' is shown. The command 'mosquitto\_sub -h mqtt.com -p 8883 -u kiwiahn -P 123456 -t '/esp8266' --cafile ecc.ca.cer --key ecc.raspberry.key.pem' has been executed. The output consists of a series of 'Good job! You are connected X times' messages, where X ranges from 5 to 24, indicating successful connections to the MQTT broker on the topic 'esp8266'.

```
pi@raspberrypi:~/ecc$ mosquitto_sub -h mqtt.com -p 8883 -u kiwiahn -P 123456 -t '/esp8266' --cafile ecc.ca.cer --key ecc.raspberry.key.pem
Good job! You are connected 5 times
Good job! You are connected 6 times
Good job! You are connected 7 times
Good job! You are connected 8 times
Good job! You are connected 9 times
Good job! You are connected 10 times
Good job! You are connected 11 times
Good job! You are connected 12 times
Good job! You are connected 13 times
Good job! You are connected 14 times
Good job! You are connected 15 times
Good job! You are connected 16 times
Good job! You are connected 17 times
Good job! You are connected 18 times
Good job! You are connected 19 times
Good job! You are connected 20 times
Good job! You are connected 21 times
Good job! You are connected 22 times
Good job! You are connected 23 times
Good job! You are connected 24 times
```

Figure 7: Raspberry Pi subscribes to the topic `esp8266`

## 2.2 LoRa

In the setup for LoRa communication, there are a LoRa client and a LoRa server, both of which are Raspberry Pi. The LoRa client is the Raspberry Pi that connects to the ESP8266. This client receives messages from the topic `esp8266` and forwards them to the LoRa server. The communication between these two devices is secured with AES encryption, details of which will be elaborated in the next section.

## 3 Encryption

### 3.1 Elliptic-curve cryptography

According to the project's architecture illustrated in Figure 1, the ESP8266 module is connected to the ATECC608 component. The ATECC608 is a secure element equipped with advanced ECC capabilities, facilitating encryption, signature, and verification operations efficiently.

In line with the project requirements, the connection between the ESP8266 and the Raspberry Pi is established using the TLS protocol after completing the authentication process, which involves certificate and ECC key exchanges. This ensures a secure communication channel between the devices, leveraging the strengths of ECC for IoT security.

```

mgos_vfs.c:280      conf9.json -> /conf9.json pl 1 -> 1 0x3ffef7bc (refs 2)
mgos_vfs.c:631      rename tmp -> conf9.json => 0x3ffef7bc tmp -> conf9.json => 0
mgos_sys_config.c:323 Saved to conf9.json
mgos_wifi_sta.c:380 AP history:
mgos_wifi_sta.c:388 0: SSID ahnvn1 , BSSID b8:27:eb:78:87:13 ch 7 RSSI

mgos_event.c:134    ev WIFI3 triggered 0 handlers
mgos_net.c:103      WiFi STA: ready, IP 192.168.2.111, GW 192.168.2.1, DNS 192.168.2.1, NTP 0.0.0.0
mgos_net.c:208      Setting DNS server to 192.168.2.1
mgos_mqtt_conn.c:442 MQTT0 connecting to mqtt.com:8883
mgos_event.c:134    ev MOS6 triggered 0 handlers
mgos_vfs.c:280      0x3fff0404 mqtt.com:8883 ecc.esp8266.cert.pem,ATCA:0,ecc.ca.cert.pem
mgos_vfs.c:375      ecc.esp8266.cert.pem -> /ecc.esp8266.cert.pem pl 1 -> 1 0x3ffef7bc (refs 1)
mgos_vfs.c:535      fstat 257 => 0x3ffef7bc:1 => 0 (size 725)
mgos_vfs.c:535      fstat 257 => 0x3ffef7bc:1 => 0 (size 725)
mgos_vfs.c:563      lseek 257 0 1 => 0x3ffef7bc:1 => 0
mgos_vfs.c:563      lseek 257 0 => 0x3ffef7bc:1 => 0
mgos_vfs.c:409      close 257 => 0x3ffef7bc:1 => 0 (refs 0)
mgos_vfs.c:280      ecc.ca.cert.pem -> /ecc.ca.cert.pem pl 1 -> 1 0x3ffef7bc (refs 1)
mgos_vfs.c:375      open ecc.ca.cert.pem 0x0 0x1b6 => 0x3ffef7bc ecc.ca.cert.pem 1 => 257 (refs 1)
mgos_vfs.c:409      close 257 => 0x3ffef7bc:1 => 0 (refs 0)
mongoose.c:3136     0x3fff1904 udp://192.168.2.1:53 -,-,-
mongoose.c:3006     0x3fff1904 udp://192.168.2.1:53
mgos_event.c:134    ev NET3 triggered 2 handlers
mongoose.c:3020     0x3fff1904 udp://192.168.2.1:53 -> 0
mgos_mongoose.c:66  New heap free LWM: 41488
mongoose.c:3006     0x3fff0404 tcp://192.168.2.1:8883
mgos_mongoose.c:66  New heap free LWM: 41224
mongoose.c:3020     0x3fff0404 tcp://192.168.2.1:8883 -> 0
mgos_mongoose.c:66  New heap free LWM: 40520
mgos_vfs.c:280      ecc.ca.cert.pem -> /ecc.ca.cert.pem pl 1 -> 1 0x3ffef7bc (refs 1)
mgos_vfs.c:375      open ecc.ca.cert.pem 0x0 0x1b6 => 0x3ffef7bc ecc.ca.cert.pem 1 => 257 (refs 1)
mgos_vfs.c:535      fstat 257 => 0x3ffef7bc:1 => 0 (size 676)
ATCA ECDSA verify ok, verified
mgos_vfs.c:409      close 257 => 0x3ffef7bc:1 => 0 (refs 0)
ATCA ECDSA verify ok, verified
ATCA:2 ECDH get pubkey ok
ATCA:2 ECDH ok
ATCA:0 ECDSA sign ok
mgos_mongoose.c:66  New heap free LWM: 36208
mgos_mqtt_conn.c:187 MQTT0 TCP connected ok (0)
mgos_mqtt_conn.c:180 MQTT0 event: 202

```

Figure 8: Authentication using TLS protocol

### 3.2 AES

The communication process between the LoRa client and the LoRa server involves encrypting messages sent by the LoRa client using Advanced Encryption Standard (AES) with a key shared between the two devices. Once the LoRa server receives the encrypted message, it proceeds to decrypt the data using the same shared key. After decryption, the server displays the received information. This method ensures that the data exchanged over the LoRa network remains confidential and secure from unauthorized access.

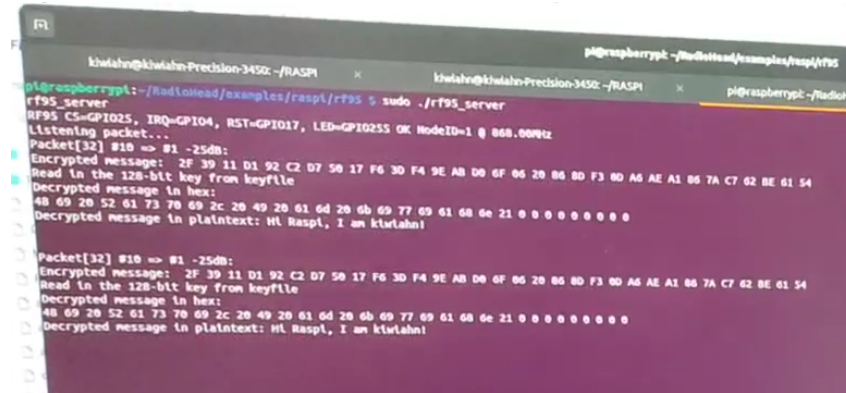
```

khalid@khalid:~/raspi
pi@raspberrypi:~/khalidhead/examples/raspi/rf95 $ sudo ./rf95_client
rf95_client
RF95 CS=GPIO25, IRQ=GPIO4, RST=GPIO17, LED=GPIO255
RF95 module seen OK!
RF95 mode #10 init OK @ 0x0.0000
Original message: Hi Raspi!, I am khalid!
Encrypted message to base:
2F 39 11 d1 92 c2 d7 30 17 F6 3d F4 9c 4b 4b d7 4 2b 86 8d F3 d 66 6e a1 86 7a c7 62 be 61 54
Sending 32 bytes to node 01 => 2F 39 11 d1 92 c2 d7 30 17 F6 3d F4 9c 4b d7 4 2b 86 8d F3 d 66 6e a1 86 7a c7 62 be 61 54

```

Figure 9: Message from LoRa client



A terminal window on a Raspberry Pi showing the output of the rf95\_server program. The terminal has a dark background with light-colored text. The output shows the server listening for packets, receiving a packet, and then displaying the encrypted message in hex, the key used for decryption, and the resulting plaintext message. The plaintext message is "Hi Raspi, I am ktulahn!".

```
ktulahn@ktulahn-Precision-3450: ~/RASPI
pi@raspberrypi: ~/Radiohead/examples/raspi/rf95 $ sudo ./rf95_server
rf95_server
RF95 CS=GPIO25, IRQ=GPIO4, RST=GPIO17, LED=GPIO255 OK ModeID=1 @ 868.00MHz
Listening packet...
Packet[32] #10 => #1 -25dB:
Encrypted message: 2f 39 11 d1 92 c2 d7 50 17 f6 30 f4 9e ab d0 6f 06 20 06 0d f3 00 a6 ae a1 06 7a c7 02 be 01 54
Read in the 128-bit key from keyfile
Decrypted message in hex:
48 69 20 52 61 73 70 69 2c 20 49 20 61 6d 20 6b 69 77 69 61 68 6e 21 00 00 00 00 00
Decrypted message in plaintext: Hi Raspi, I am ktulahn!

Packet[32] #10 => #1 -25dB:
Encrypted message: 2f 39 11 d1 92 c2 d7 50 17 f6 30 f4 9e ab d0 6f 06 20 06 0d f3 00 a6 ae a1 06 7a c7 02 be 01 54
Read in the 128-bit key from keyfile
Decrypted message in hex:
48 69 20 52 61 73 70 69 2c 20 49 20 61 6d 20 6b 69 77 69 61 68 6e 21 00 00 00 00 00
Decrypted message in plaintext: Hi Raspi, I am ktulahn!
```

Figure 10: Received message on LoRa server

## 4 Conclusion

In this project, we have successfully implemented the specified architecture, facilitating secure communication between the ESP8266 and a Raspberry Pi, as well as between two Raspberry Pi. This setup simulates a real-world IoT structure, where messages from embedded devices are relayed to an MQTT server on the Internet through an intermediate server. Despite encountering numerous technical challenges, we managed to fulfill all project requirements within the constrained timeframe. These difficulties, however, contributed significantly to our learning, enhancing our knowledge and skills in a way that will greatly benefit our personal development and professional careers.