Communicating Mobile Terminals

Project Report

Faculty of Science and Technology
University of Limoges

presented by

Nguyen Van Tien
Nguyen Thi Mai Phuong
Doan Thi Van Thao

Master 2 CRYPTIS

Instructor: Prof. Pierre-François Bonnefoi

April 14, 2022

Contents

List of Figures

1 Project's objective

The goal of the project is setting up the communication between a MQTT client and MQTT server (Raspberry Pis) via LoRa, where the former has connections to ESP8266, while the latter is enable to gain access to the Internet. Before communication, the authentication of both the client and server needs to be undertaken. That is why ATECC608 is integrated, which allows to apply elliptic-curve cryptography to authenticate with their credentials. Moreover, to ensure the security of exchanged data of two parties after a successful identity verification, AES encryption is related. The whole system architecture is illustrated on Figure 1.

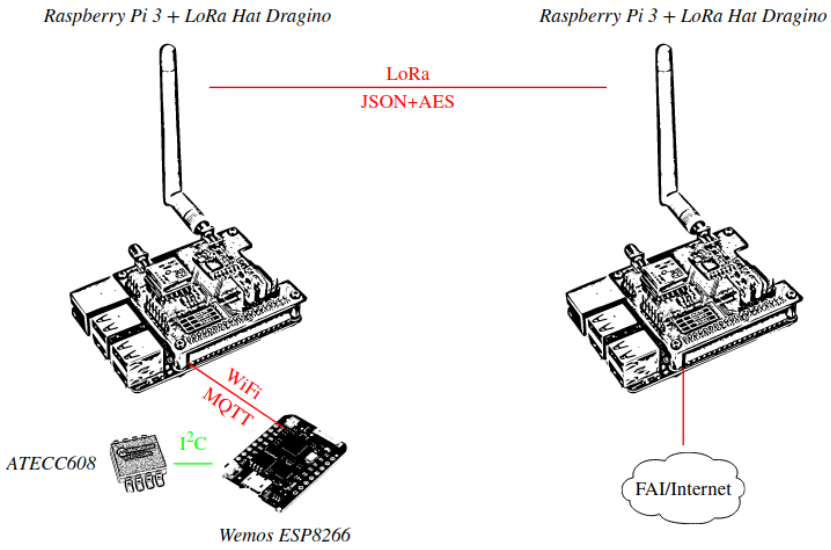

Figure 1: Graph of system architecture

Based on this architecture, the setting up of devices is as follows:

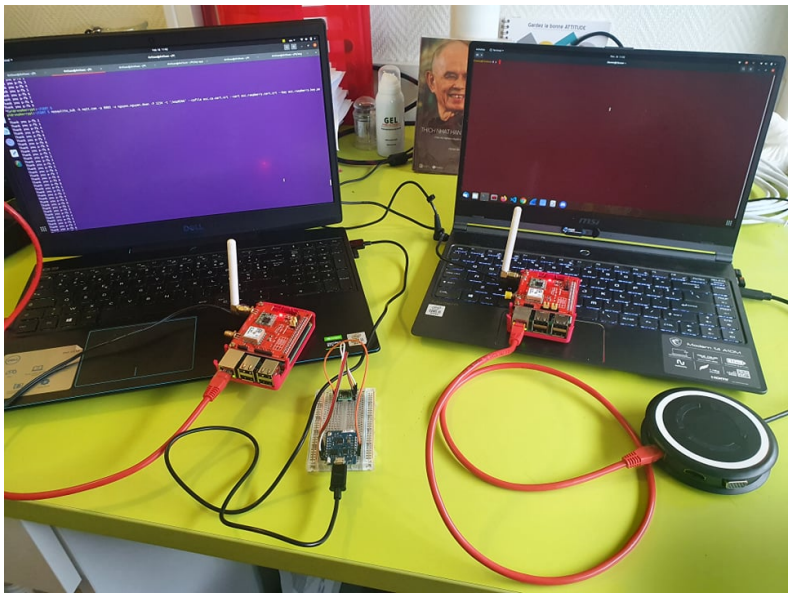

Figure 2: Setting up the complete system

Figure 3: Setting up the ESP8266

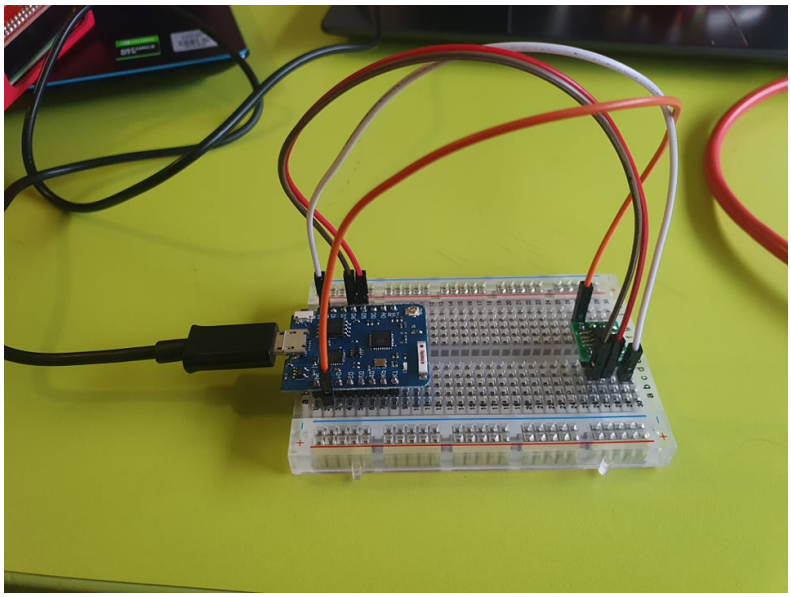

Figure 4: Setting up the Raspberry Pi

In the scope of the report, we only present the result of communication setting, encryption, and data exchanges as the requirements. For details of implementation, please refer to file `README.md` at Github.

You can find the video demonstration on Youtube here.

2 Communication setup

2.1 WiFi and MQTT

2.1.1 WiFi connection from ESP8266 to Raspberry Pi

At first, we set up a WiFi access point on the Raspberry Pi by installing the `hostapd` and `dnsmasq` packages, where `hostapd` package permits us to create a wireless hotspot, while `dnsmasq` facilitates to launch a DNS and DHCP server.

```
[Feb 18 02:41:52.271] mgos_wifi_sta.c:475     State 6 ev -1 timeout 0
[Feb 18 02:41:52.276] mgos_net.c:89           WiFi STA: connecting
[Feb 18 02:41:52.281] mgos_event.c:134        ev NET1 triggered 1 handlers
[Feb 18 02:41:52.359] esp_main.c:137          SDK: scandone
[Feb 18 02:41:53.247] esp_main.c:137          SDK: state: 0 -> 2 (b0)
[Feb 18 02:41:53.254] esp_main.c:137          SDK: state: 2 -> 3 (0)
[Feb 18 02:41:53.278] esp_main.c:137          SDK: state: 3 -> 5 (10)
[Feb 18 02:41:53.284] esp_main.c:137          SDK: add 0
[Feb 18 02:41:53.284] esp_main.c:137          SDK: aid 1
[Feb 18 02:41:53.288] esp_main.c:137          SDK: cnt
[Feb 18 02:41:53.306] esp_main.c:137          SDK:
[Feb 18 02:41:53.306] esp_main.c:137          SDK: connected with vn1, channel 7
[Feb 18 02:41:53.311] esp_main.c:137          SDK: dhcp client start...
[Feb 18 02:41:53.320] mgos_wifi.c:82          WiFi STA: Connected, BSSID b8:27:eb:fe:fb:91 ch 7 RSSI -65
[Feb 18 02:41:53.325] mgos_wifi_sta.c:475     State 6 ev 1464224002 timeout 0
[Feb 18 02:41:53.325] mgos_event.c:134        ev WFI2 triggered 0 handlers
[Feb 18 02:41:53.339] mgos_net.c:93           WiFi STA: connected
```

Figure 5: ESP8266 connects to WiFi

2.1.2 Connection from MQTT client to MQTT server

MQTT client is ESP8266 which is supposed to connect MQTT server, a Raspberry Pi. The MQTT client sends messages to the topic *esp8266* with content *"Thank you p-fb"* every 5 seconds. Meanwhile, the MQTT server subscribes to the same topic to receive the data.

```
[Feb 18 03:55:35.807] mgos_mqtt_conn.c:117    MQTT0 ack 1
[Feb 18 03:55:35.814] mgos_mqtt_conn.c:153    MQTT0 pub -> 2 /esp8266 @ 1 DUP (16): [Thank you p-fb 1]
[Feb 18 03:55:35.829] mgos_mqtt_conn.c:179    MQTT0 event: 204
[Feb 18 03:55:35.829] mgos_mqtt_conn.c:117    MQTT0 ack 2
[Feb 18 03:55:35.836] mgos_mqtt_conn.c:153    MQTT0 pub -> 3 /esp8266 @ 1 DUP (16): [Thank you p-fb 2]
[Feb 18 03:55:35.848] mgos_mqtt_conn.c:179    MQTT0 event: 204
[Feb 18 03:55:35.848] mgos_mqtt_conn.c:117    MQTT0 ack 3
[Feb 18 03:55:35.855] mgos_mqtt_conn.c:153    MQTT0 pub -> 4 /esp8266 @ 1 DUP (16): [Thank you p-fb 3]
[Feb 18 03:55:35.869] mgos_mqtt_conn.c:179    MQTT0 event: 204
[Feb 18 03:55:35.869] mgos_mqtt_conn.c:117    MQTT0 ack 4
[Feb 18 03:55:35.873] mgos_mqtt_conn.c:315    MQTT0 queue drained
[Feb 18 03:55:40.038] mgos_mqtt_conn.c:153    MQTT0 pub -> 7 /esp8266 @ 1 (16): [Thank you p-fb 4]
[Feb 18 03:55:40.053] mgos_mqtt_conn.c:179    MQTT0 event: 204
[Feb 18 03:55:40.053] mgos_mqtt_conn.c:117    MQTT0 ack 7
[Feb 18 03:55:43.297] esp_main.c:137          SDK: pm open,type:0 0
[Feb 18 03:55:45.038] mgos_mqtt_conn.c:153    MQTT0 pub -> 8 /esp8266 @ 1 (16): [Thank you p-fb 5]
[Feb 18 03:55:45.053] mgos_mqtt_conn.c:179    MQTT0 event: 204
```

Figure 6: ESP8266 publishes to the topic *esp8266*

```
pi@raspberrypi:~/CERT $ mosquitto_sub -h mqtt.com -p 8883 -u nguyen.nguyen.doan -P 1234 -t '/esp8266'
--cafile ecc.ca.cert.crt --cert ecc.raspberry.cert.crt --key ecc.raspberry.key.pem
Thank you p-fb 5
Thank you p-fb 6
Thank you p-fb 7
Thank you p-fb 8
Thank you p-fb 9
Thank you p-fb 0
Thank you p-fb 1
Thank you p-fb 2
Thank you p-fb 3
Thank you p-fb 4
Thank you p-fb 5
Thank you p-fb 6
Thank you p-fb 7
Thank you p-fb 8
Thank you p-fb 9
```

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

2.2 LoRa

For LoRa communication, we have a LoRa client and server which are two Raspberry Pis. The LoRa client is the Raspberry Pi connecting with ESP8266 and it will send the messages received from the topic *esp8266* to the LoRa server. The communication between the two is secured by AES encryption which will be presented specifically in subsection 3.2.

3 Encryption

3.1 By Elliptic-curve cryptography

Nowadays, ECC is applied widely in IoT based on one of the most advantages that with a smaller key, ECC has equivalent security compared to RSA. In this project, the TLS is facilitated by ciphersuite `TLS-ECDHE-ECDSA-WITH-AES-128-GCM-SHA256` where:

- Transport Layer Security (TLS) is protocol.
- Key Exchange is Elliptic Curve Diffie-Hellman Ephemeral (`ECDHE`).
- Elliptic Curve Digital Signature Algorithm (`ECDSA`) is used for authentication.
- Encryption is Advanced Encryption Standard with 128bit key in Galois/Counter mode (AES 128 GCM).
- Hash function is Secure Hash Algorithm 256 (SHA256).

Following the architecture in Figure 1, the ESP8266 is connected to the ATECC608 component which is a secure element with advanced Elliptic Curve Cryptography (ECC) capabilities. This helps to perform encryption/signature/verification operations.

As the requirements, the connection between the ESP8266 and Raspberry Pi must be made with TLS protocol after the authentication process which includes the certificate/ECC key exchanges.

```
[Feb 18 02:41:57.462] mgos_mqtt_conn.c:435    MQTT0 connecting to mqtt.com:8883
[Feb 18 02:41:57.462] mgos_event.c:134        ev MOS6 triggered 0 handlers
[Feb 18 02:41:57.471] mongoose.c:3136         0x3ffef034 mqtt.com:8883 ecc.esp8266.cert.pem,ATCA:0,ecc.ca.cert.pem
[Feb 18 02:41:57.480] mgos_vfs.c:280          ecc.esp8266.cert.pem -> /ecc.esp8266.cert.pem pl 1 -> 1 0x3ffefb04 (refs 1)
[Feb 18 02:41:57.494] mgos_vfs.c:375          open ecc.esp8266.cert.pem 0x0 0x1b6 => 0x3ffefb04 ecc.esp8266.cert.pem 1 => 257 (refs 1)
[Feb 18 02:41:57.501] mgos_vfs.c:535          fstat 257 => 0x3ffefb04:1 => 0 (size 639)
[Feb 18 02:41:57.517] mgos_vfs.c:535          fstat 257 => 0x3ffefb04:1 => 0 (size 639)
[Feb 18 02:41:57.517] mgos_vfs.c:563          lseek 257 0 1 => 0x3ffefb04:1 => 0
[Feb 18 02:41:57.517] mgos_vfs.c:563          lseek 257 0 0 => 0x3ffefb04:1 => 0
[Feb 18 02:41:57.523] mgos_vfs.c:409          close 257 => 0x3ffefb04:1 => 0 (refs 0)
[Feb 18 02:41:57.678] mgos_vfs.c:280          ecc.ca.cert.pem -> /ecc.ca.cert.pem pl 1 -> 1 0x3ffefb04 (refs 1)
[Feb 18 02:41:57.692] mgos_vfs.c:375          open ecc.ca.cert.pem 0x0 0x1b6 => 0x3ffefb04 ecc.ca.cert.pem 1 => 257 (refs 1)
[Feb 18 02:41:57.698] mgos_vfs.c:409          close 257 => 0x3ffefb04:1 => 0 (refs 0)
[Feb 18 02:41:57.704] mongoose.c:3136         0x3fff09e4 udp://192.168.4.1:53 -,-,-
[Feb 18 02:41:57.709] mongoose.c:3006         0x3fff09e4 udp://192.168.4.1:53
[Feb 18 02:41:57.714] mgos_event.c:134        ev NET3 triggered 2 handlers
[Feb 18 02:41:57.721] mongoose.c:3020         0x3fff09e4 udp://192.168.4.1:53 -> 0
[Feb 18 02:41:57.727] mgos_mongoose.c:66      New heap free LWM: 40760
[Feb 18 02:41:57.738] mongoose.c:3006         0x3ffef034 tcp://192.168.4.1:8883
[Feb 18 02:41:57.744] mgos_mongoose.c:66      New heap free LWM: 40496
[Feb 18 02:41:57.751] mongoose.c:3020         0x3ffef034 tcp://192.168.4.1:8883 -> 0
[Feb 18 02:41:57.768] mongoose.c:4906         0x3ffef034 ciphersuite: TLS-ECDHE-ECDSA-WITH-AES-128-GCM-SHA256
[Feb 18 02:41:57.780] mgos_vfs.c:280          ecc.ca.cert.pem -> /ecc.ca.cert.pem pl 1 -> 1 0x3ffefb04 (refs 1)
[Feb 18 02:41:57.789] mgos_vfs.c:375          open ecc.ca.cert.pem 0x0 0x1b6 => 0x3ffefb04 ecc.ca.cert.pem 1 => 257 (refs 1)
[Feb 18 02:41:57.795] mgos_vfs.c:535          fstat 257 => 0x3ffefb04:1 => 0 (size 635)
[Feb 18 02:41:57.953] ATCA ECDSA verify ok, verified
[Feb 18 02:41:57.958] mgos_vfs.c:409          close 257 => 0x3ffefb04:1 => 0 (refs 0)
[Feb 18 02:41:58.017] ATCA ECDSA verify ok, verified
[Feb 18 02:41:58.060] ATCA:2 ECDH get pubkey ok
[Feb 18 02:41:58.107] ATCA:2 ECDH ok
```

Figure 8: Authentication using TLS protocol

After TCP establishment, the TLS protocol is implemented to secure the exchanged data. Figure 9 shows the packets of connection between the server and client.

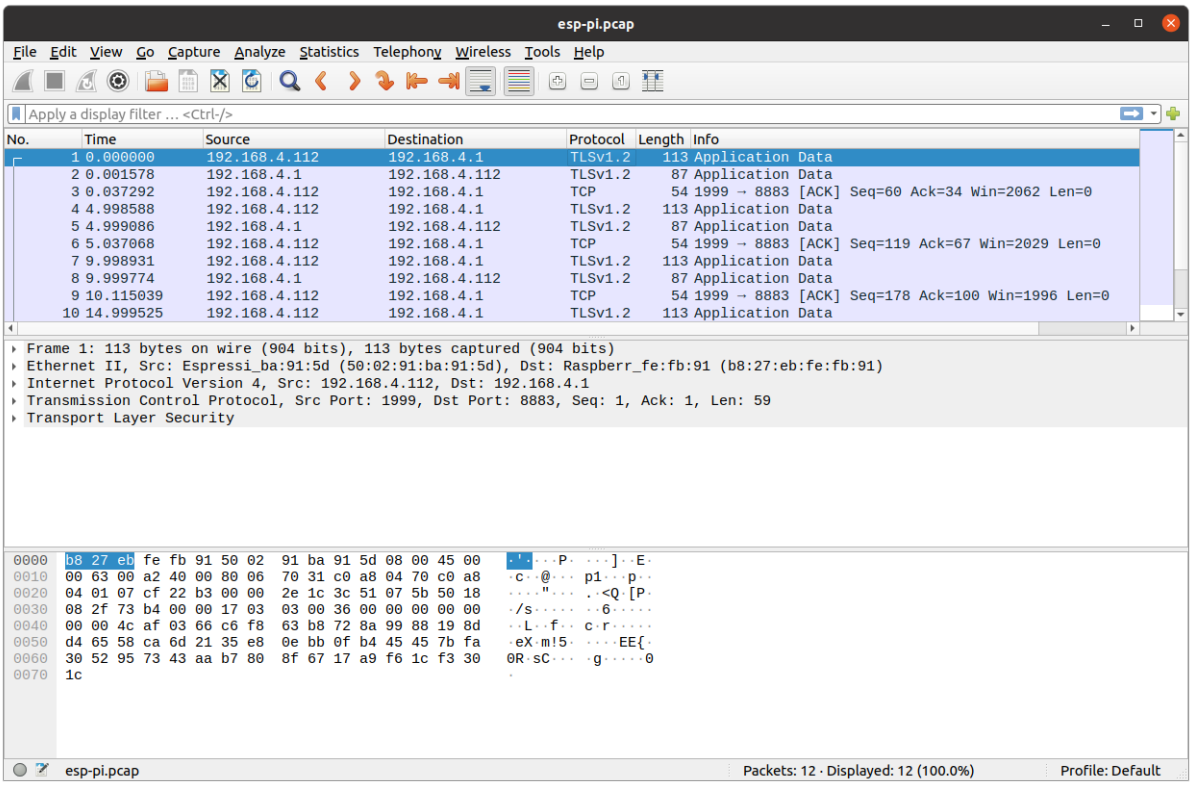

Figure 9: Connection between the server and client

3.2 By AES

The message sent by the LoRa client to the LoRa server will be encrypted by AES with a shared key between them. After that, the server decrypts the received data and then displays them as shown on Figure 11.

```
pi@raspberrypi:~/LoRa/RadioHead/examples/raspi/rf95 $ python3 mqtt_to_lora.py
/esp8266 0 b'Thank you p-fb 5'
rf95_client
RF95 CS=GPIO25, IRQ=GPIO4, RST=GPIO17, LED=GPIO255
RF95 module seen OK!
RF95 node #10 init OK @ 868.00MHz
Sending 129 bytes: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoiYXJ5QWx3MDIyOWNSUFhkMldpS0RiQT09In0.l_pgJOmPswJebdcP9RJrM8ccWCNaIRXLUBF
xZhBSvdo


/esp8266 0 b'Thank you p-fb 6'
rf95_client
RF95 CS=GPIO25, IRQ=GPIO4, RST=GPIO17, LED=GPIO255
RF95 module seen OK!
RF95 node #10 init OK @ 868.00MHz
Sending 129 bytes: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoiREY4SHNNMDVIb0czckhvYXBWakdCZz09In0.fJ5Lu0AIIFqcuqmRE2yK-3-2ixPWB7R-nBd
3WIQZqt0
```

Figure 10: LoRa client sends the messages

```
pi@tiennv:~/PI/RadioHead/examples/raspi/rf95 $ sudo ./rf95_server
rf95_server
RF95 CS=GPIO25, IRQ=GPIO4, RST=GPIO17, LED=GPIO255 OK NodeID=1 @ 868.00MHz
Listening packet...
RSSI = -38dB;
LoRa Raw Data: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoiYXJ5QWx3MDIyOWNSUFhkMldpS0RiQT09In0.l_pgJOmPswJebdcP9RJrM8ccWCNaIRX
LUBFxZhBSvdo
AES encrypted data: aryAlw0229cRPXd2WiKDbA==
AES Decrypted data : Thank you p-fb 5


RSSI = -28dB;
LoRa Raw Data: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoiREY4SHNNMDVIb0czckhvYXBWakdCZz09In0.fJ5Lu0AIIFqcuqmRE2yK-3-2ixPWB7R
-nBd3WIQZqt0
AES encrypted data: DF8HsM05HoG3rHoapVjGBg==
AES Decrypted data : Thank you p-fb 6
```

Figure 11: LoRa servers receives the messages

4 Conclusion

In the project, we have successfully implemented the required architecture which supports the secured communication between ESP8266 and Raspberry Pi, as well as between two Raspberry Pi. In the aim of simulating a real IoT structure, the messages from the embedded devices can be transferred to a MQTT server located on the Internet via an intermediate server.

Although facing with a lot of technical obstacles, we completed all requirements within the limited time. However, thanks to these difficulties, we have gained much more knowledge which will help us to make substantial progress in both our personal career and professional experiences.