

# Rapport du projet : Terminaux mobiles communicants

Filière

CRYPTIS M2 : Info

---

Sujet

IoT, LoRa, Wi-Fi, MQTT, SSL, ATECC508, Mongoose OS,  
Raspberry Pi & ESP8266

---

Réalisé par :

EL-ASRI Loubna

BAHBAH Ayoub

LARHLIMI Hamza

Encadré par :

P-F. Bonnefoi

Année universitaire : 2020 - 2021



# Table des matières

|   |    |
|---|----|
| Table des matières.....                                 | 3  |
| Table des figures.....                                  | 5  |
| Introduction générale.....                              | 7  |
| Chapitre 1.....   | 8  |
| Installation du Raspberry Pi OS .....                   | 8  |
| 1. Introduction .....                                   | 8  |
| 2. Installation de l'OS .....                           | 9  |
| 3. Connection SSH .....                                 | 11 |
| Chapitre 2.....   | 13 |
| Installation des outils .....                           | 13 |
| 1. Installation de dnsmasq et hostapd.....              | 13 |
| 1.1. Dnsmasq .....                                      | 13 |
| 1.2. Hostapd .....                                      | 14 |
| 2. Génération des certificats ECC .....                 | 16 |
| 3. Installation du Mosquitto et Mosquitto-clients ..... | 17 |
| 3.1. Apt-get Mosquitto Mosquitto-clients.....           | 17 |
| 3.2. Tester la configuration.....                       | 18 |
| 4. Installation de mongoose-os.....                     | 20 |
| 4.1. L'outil mongoose-os/mos.....                       | 20 |
| 4.2. Cloner et modifier le projet Github .....          | 21 |
| Chapitre 3.....   | 23 |
| Connexion Wi-Fi .....                                   | 23 |

|                               |    |
|-------------------------------|----|
| 1. Installation Hardware..... | 23 |
| 2. Connexion Wi-Fi.....       | 24 |
| Conclusion générale.....      | 26 |
| Bibliographie.....            | 27 |

## Table des figures

|   |    |
|---|----|
| Figure 1 : Raspberry Pi 3 connecté .....                          | 8  |
| Figure 2 : Raspberry Pi Imager.....                               | 9  |
| Figure 3 : Raspberry Pi Imager : OS.....                          | 10 |
| Figure 4 : Raspberry Pi allumé .....                              | 10 |
| Figure 5 : Interface graphique du Raspberry Pi.....               | 11 |
| Figure 6 : Configuration Switch - Raspberry Pi - Ordinateur ..... | 11 |
| Figure 7 : Affichage de l'adresse IP RASPI .....                  | 12 |
| Figure 8 : Connexion SSH via CMD Windows .....                    | 12 |
| Figure 9 : Installation du dnsmasq.....                           | 13 |
| Figure 10 : Configuration du dnsmasq.....                         | 14 |
| Figure 11 : Installation hostapd .....                            | 14 |
| Figure 12 : Configuration hostapd.....                            | 15 |
| Figure 13 : Configuration du default/hostapd.....                 | 15 |
| Figure 14 : Activation du Forwarding IPv4.....                    | 16 |
| Figure 15 : Génération des certificats (2).....                   | 16 |
| Figure 16 : Génération des certificats (1).....                   | 16 |
| Figure 17 : Configuration mosquitto .....                         | 17 |
| Figure 18 : Configuration tls.conf et tcp.conf.....               | 17 |
| Figure 19 : Service Mosquitto active.....                         | 18 |
| Figure 20 : Test configuration TLS.....                           | 19 |
| Figure 21 : Test configuration TLS Handshake .....                | 19 |
| Figure 22 : Mosquitto_pub .....                                   | 20 |
| Figure 23 : Mosquitto_sub : Bonjour ! .....                       | 20 |
| Figure 24 : Installation de l'outil mos .....                     | 21 |

---

|   |    |
|---|----|
| Figure 25 : Modification du fichier yml.....                | 22 |
| Figure 26 : Modification du main.c .....                    | 22 |
| Figure 27 : Installation du matériel.....                   | 23 |
| Figure 28 : Mosquitto Active : Ayoub.....                   | 24 |
| Figure 29 : Lancement de mos console : connexion Wi-Fi..... | 25 |
| Figure 30 : Publish sur le topic .....                      | 25 |

# Introduction générale

Ce rapport détaille le processus pour réaliser une connexion Wi-Fi entre deux Raspberry Pi, en assurant une communication cryptée. Ce projet s'installe dans le cadre de la formation CRYPTIS M2, cadré par M. Bonnefoi dans le module TMC<sup>1</sup>.

Ce projet est réalisé par notre trinôme, EL-ASRI Loubna, BAHBAH Ayoub et LARHLIMI Hamza. Dans un premier temps, nous avons installé nos deux systèmes d'exploitation Debian Buster sur deux cartes mémoires (SD Cards). Par la suite, nous avons installé les outils nécessaires pour la réalisation du projet, notamment un service Mosquitto permettant l'implémentation du protocole MQTT. Ensuite, nous avons généré nos différents certificats (courbes elliptiques sur ATECC508), utilisés pour assurer la confidentialité du message transféré en l'air libre grâce au Wi-Fi. Dans un dernier temps, nous avons mis en place nos ESP8266, qui proposent à nos Raspberry Pi les services Wi-Fi grâce à Mongoose-OS. La connexion utilise le protocole TLS pour le transfert de message.

Ce rapport détail l'ensemble des commandes et nos choix pour aboutir à ce projet ainsi que quelques difficultés rencontrées. Le projet est réalisé en principale partie grâce aux cours disponibles sur le site de M. Bonnefoi (Voir bibliographie). Il existe également une version vidéo de ce projet sur le lien suivant :

<https://www.youtube.com/watch?v=iz5PhH7-MNk>

---

<sup>1</sup> Terminaux Mobiles Communicants

# Chapitre 1

## Installation du Raspberry Pi OS

### 1. Introduction

Lors de la réalisation de notre projet, nous avons utilisé des Raspberry Pi 3. Ces derniers, sont des « ordinateurs » low-cost d'une petite taille à faible coût d'une puissance et capacité assez élevée relativement à son prix qui ne dépasse pas les 35-40€.

Il est conçu pour initier les personnes à l'informatique, mais souvent utilisé dans des projets IoT<sup>2</sup>, notre cas. Il est capable d'utiliser des systèmes d'exploitation 32bit-64bit, et fonctionner, plus ou moins, comme un ordinateur avec la possibilité de parcourir internet, compiler et éventuellement faire tourner des programmes (jeu, calculs, etc). L'usage est assez simple, connecter à un écran via un câble HDMI, clavier et souris par port USB. Quant à l'OS<sup>3</sup>, il est installé sur une carte mémoire (16GB est largement suffisante).



Figure 1 : Raspberry Pi 3 connecté

---

<sup>2</sup> Internet of Things

<sup>3</sup> Operating System



## 2. Installation de l'OS

Pour utiliser un Raspberry Pi, plusieurs solutions sont possibles. Installer sur notre machine linux, un service NFS permettant de connecter le Raspberry Pi via câble Ethernet, et de faire en sorte que l'OS soit disponible sur notre ordinateur plutôt que sur le Raspberry Pi. Cette solution proposée par M. Bonnefoi permet de contourner l'usage des cartes SD. Cependant, elle s'est avérée difficile à mettre en place à cause des problèmes de virtualisations (Machine virtuelle). La solution standard, proposé par le constructeur, est d'installer un OS sur une carte SD grâce à un outil s'appelant Raspberry Pi Imager.

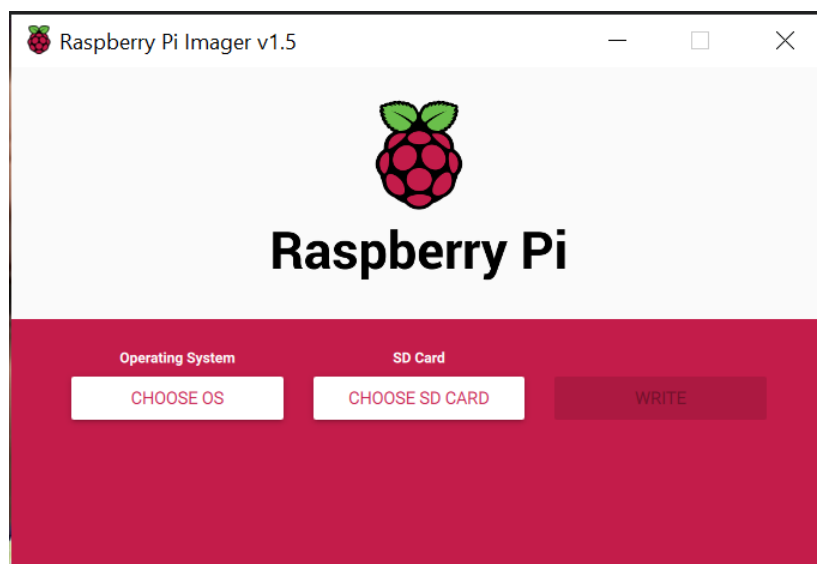


Figure 2 : Raspberry Pi Imager

Cet outil, assez simple, permet d'installer des OS sur notre carte SD. Il marche sur toute sorte d'OS, Windows en fait partie. Raspberry Pi Imager propose des OS préconfiguré et également la possibilité d'utiliser un OS customisé. Nous avons opté pour le choix d'utiliser un OS 64 bit.

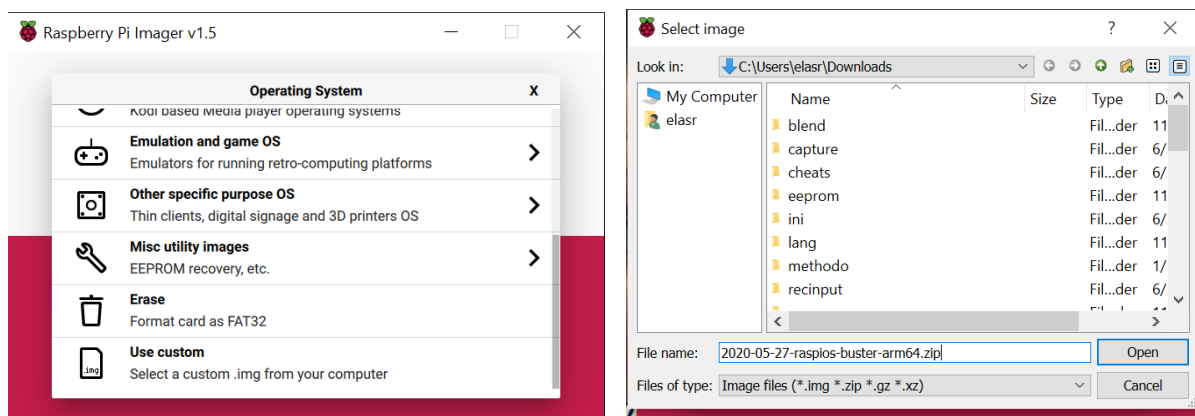


Figure 3 : Raspberry Pi Imager : OS

Une fois notre OS est installé, le Raspberry Pi est fonctionnel. Il comprend un assez large nombre d'outils. Il marche évidemment sous Linux, notamment Debian Buster pour notre projet. La figure suivante présente l'interface graphique de l'OS installé sur le Raspberry Pi 3. La diffusion se fait simplement grâce à une TV connectée par câble HDMI.

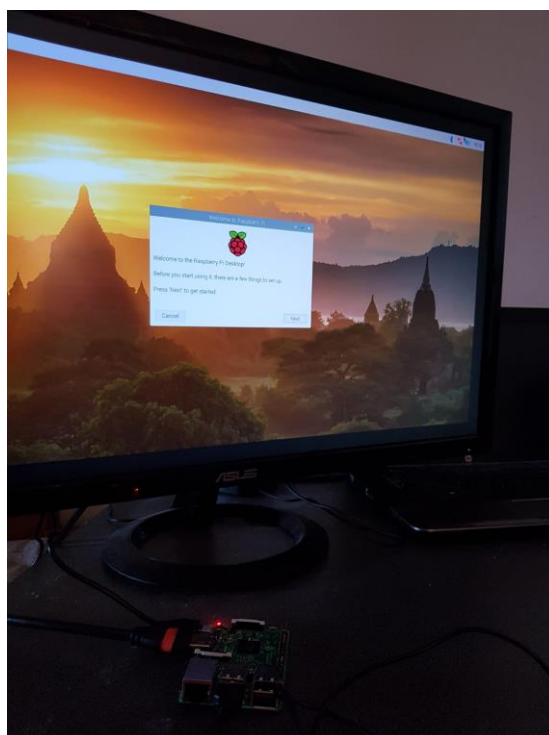


Figure 4 : Raspberry Pi allumé

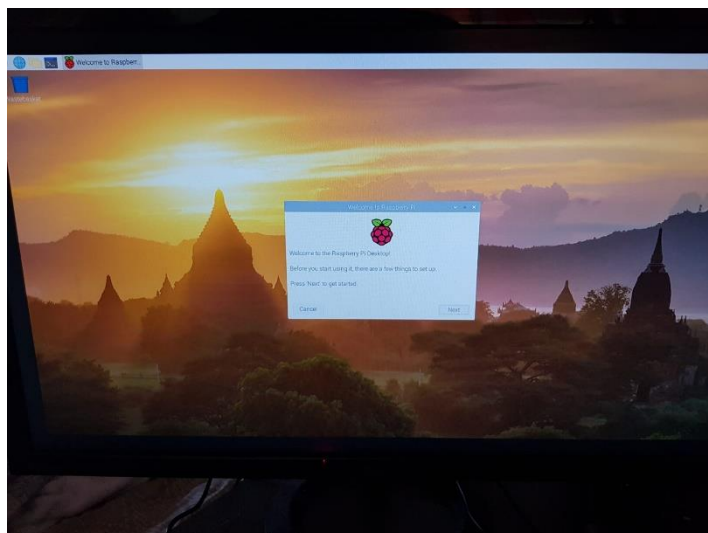


Figure 5 : Interface graphique du Raspberry Pi

### 3. Connection SSH

Afin de pouvoir configurer facilement notre Raspberry Pi, nous avons utilisé le SSH. Pour ce faire, nous avons connectés nos deux RASPI<sup>4</sup> et notre ordinateur via câble Ethernet.

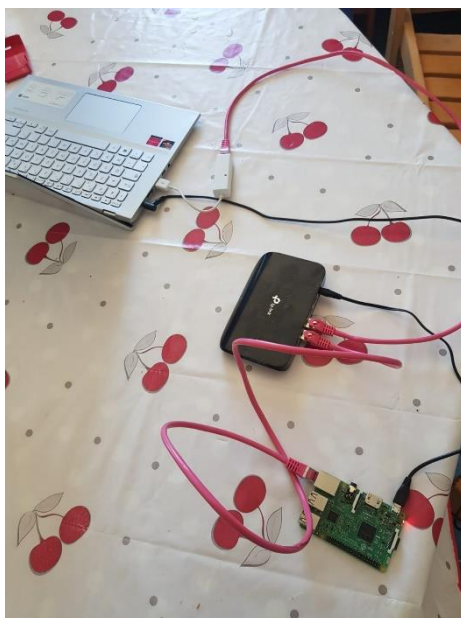


Figure 6 : Configuration Switch - Raspberry Pi - Ordinateur

---

<sup>4</sup> Raspberry Pi

Le switch s'occupe d'attribuer les adresses IP à nos machines, et également du transfert des messages. Il ne nécessite aucune configuration, il ne faut qu'afficher l'adresse IP et s'y connecter via SSH. Notre switch utilise le réseau 169.254.0.0/16

```

pi@raspberrypi:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 169.254.224.106 netmask 255.255.0.0 broadcast 169.254.255.255
    inet6 fe80::832f:5058:feaf:8fd prefixlen 64 scopeid 0x20<link>
    ether b8:27:eb:4e:e8:d1 txqueuelen 1000 (Ethernet)
    RX packets 225 bytes 61660 (60.2 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 365 bytes 41741 (40.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 130 bytes 12004 (11.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 130 bytes 12004 (11.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether b8:27:eb:1b:bd:84 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
  
```

Figure 7 : Affichage de l'adresse IP RASPI

```

pi@raspberrypi:~$ ifconfig
Wireless LAN adapter WiFi1:
    Connection-specific DNS Suffix . : numericable.fr
    Link-local IPv6 Address . . . . . : fe80::592d:4fba:8b5d:6305%
    IPv4 Address. . . . . : 192.168.0.42
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.0.1

Ethernet adapter Ethernet 2:
    Connection-specific DNS Suffix . :
    Link-local IPv6 Address . . . . . : fe80::f91e:204d:206f:a4e%13
    Autoconfiguration IPv4 Address. . : 169.254.10.78
    Subnet Mask . . . . . : 255.255.0.0
    Default Gateway . . . . . :

Ethernet adapter Bluetooth Network Connection:
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . :

C:\Users\elast>ssh pi@169.254.224.106
The authenticity of host '169.254.224.106 (169.254.224.106)' can't be established.
ECDSA key fingerprint is SHA256:LnR1RwVZP8/zH0Hw2jvNqGR5c2V3Ne60dHPhg.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '169.254.224.106' (ECDSA) to the list of known hosts.
pi@169.254.224.106's password:
Linux raspberrypi 5.4.42-v8+ #1319 SMP PREEMPT Wed May 20 14:18:56 BST 2020 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Feb 16 05:17:11 2021
pi@raspberrypi:~$
  
```

Figure 8 : Connexion SSH via CMD Windows

L'affichage de l'adresse IP se fait facilement avec la commande : **ifconfig**.

La connexion SSH se fait grâce à la commande **ssh pi@169.254.224.106**.

# Chapitre 2

## Installation des outils

Afin de permettre aux deux RASPI d'échanger, il est nécessaire d'installer un ensemble d'outils pour garantir le bon fonctionnement de l'ESP8266 et du ATECC508. Pour l'installation des packages, nous utilisons **apt-get**.

### 1. Installation de dnsmasq et hostapd

#### 1.1. Dnsmasq

Dnsmasq est un package qui permet d'instancier un serveur DNS<sup>5</sup> et DHCP<sup>6</sup> au sein de la configuration Raspberry Pi. Il facilitera l'utilisation de l'ESP8266. L'installation se fait avec la commande suivante : **apt-get install dnsmasq**.

La figure suivante présente l'installation du dnsmasq.

```
root@raspberrypi:/home/pi# apt-get install dnsmasq
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  dns-root-data dnsmasq-base
The following NEW packages will be installed:
  dns-root-data dnsmasq dnsmasq-base
0 upgraded, 3 newly installed, 0 to remove and 1 not upgraded.
Need to get 444 kB of archives.
After this operation, 1,007 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://deb.debian.org/debian buster/main arm64 dns-root-data all 2019031302 [5,396 B]
Get:2 http://archive.raspberrypi.org/debian buster/main arm64 dnsmasq-base arm64 2.80-1+rpt1 [422 kB]
Get:3 http://archive.raspberrypi.org/debian buster/main arm64 dnsmasq all 2.80-1+rpt1 [16.5 kB]
Fetched 444 kB in 1s (761 kB/s)
Selecting previously unselected package dns-root-data.
(Reading database ... 90394 files and directories currently installed.)
Preparing to unpack .../dns-root-data_2019031302_all.deb ...
Unpacking dns-root-data (2019031302) ...
Selecting previously unselected package dnsmasq-base.
Preparing to unpack .../dnsmasq-base_2.80-1+rpt1_arm64.deb ...
Unpacking dnsmasq-base (2.80-1+rpt1) ...
Selecting previously unselected package dnsmasq.
Preparing to unpack .../dnsmasq_2.80-1+rpt1_all.deb ...
Unpacking dnsmasq (2.80-1+rpt1) ...
Setting up dnsmasq-base (2.80-1+rpt1) ...
Setting up dns-root-data (2019031302) ...
Setting up dnsmasq (2.80-1+rpt1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/dnsmasq.service → /lib/systemd/system/dnsmasq.service.
Processing triggers for systemd (241-7-deb10u4) ...
Processing triggers for man-db (2.8.5-2) ...
Processing triggers for dbus (1.12.16-1) ...
root@raspberrypi:/home/pi#
```

Figure 9 : Installation du dnsmasq

<sup>5</sup> Domain Name System

<sup>6</sup> Dynamic Host Configuration Protocol

Par la suite, nous avons configuré l'interface du DNS, les adresses à utiliser dans le protocole DHCP. Nous avons également associé l'adresse 192.168.1.1 à un nom symbolique qui sera utilisé plus tard pour la relation client/serveur (ESP8266/RASPI).

```
root@raspberrypi:/home/pi# cat /etc/dnsmasq.conf
interface=wlan0
dhcp-range=192.168.1.2,192.168.1.20,255.255.255.0,24h
address=/mqtt.com/192.168.1.1

root@raspberrypi:/home/pi#
```

Figure 10 : Configuration du dnsmasq

## 1.2. Hostapd

Hostapd est un package disponible sur apt-get permettant la création d'un hotspot à l'aide du Raspberry Pi. Il est également installé d'une façon similaire que le dnsmasq.

**apt-get install hostapd**

```
root@raspberrypi:/home/pi# apt-get install hostapd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  hostapd
0 upgraded, 1 newly installed, 0 to remove and 1 not upgraded.
Need to get 724 kB of archives.
After this operation, 2,020 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian buster/main arm64 hostapd arm64 2:2.7+git20190128+0c1e29f-6+deb10u2 [724 kB]
Fetched 724 kB in 1s (805 kB/s)
Selecting previously unselected package hostapd.
(Reading database ... 90368 files and directories currently installed.)
Preparing to unpack .../hostapd_2%3a2.7+git20190128+0c1e29f-6+deb10u2_arm64.deb ...
Unpacking hostapd (2:2.7+git20190128+0c1e29f-6+deb10u2) ...
Setting up hostapd (2:2.7+git20190128+0c1e29f-6+deb10u2) ...
Created symlink /etc/systemd/system/multi-user.target.wants/hostapd.service → /lib/systemd/system/hostapd.service.
Job for hostapd.service failed because the control process exited with error code.
See "systemctl status hostapd.service" and "journalctl -xe" for details.
Created symlink /etc/systemd/system/hostapd.service → /dev/null.
Processing triggers for man-db (2.8.5-2) ...
Processing triggers for systemd (241-7~deb10u4) ...
root@raspberrypi:/home/pi#
```

Figure 11 : Installation hostapd

Comme hostapd concerne un Hotspot Wi-Fi, il est essentiel de configurer un SSID, un mot de passe, protocole à utiliser et d'autres paramètres (WPA, interface, etc). La figure ci-après présente l'ensemble des paramètres inscrits :



```

root@raspberrypi:/home/pi# cat /etc/hostapd/hostapd.conf
interface=wlan0
driver=nl80211
ssid=ayoub2bah
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=ayoubbahbah
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rns_pairwise=CCMP

root@raspberrypi:/home/pi#

```

Figure 12 : Configuration hostapd

Afin que notre fichier de configuration soit mis en considération par le RASPI, il est essentiel d'ajouter la ligne dans le fichier `/etc/default/hostapd` :

`DAEMON_CONF= '/etc/hostapd/hostapdconf'`

```

GNU nano 3.2 /etc/default/hostapd
# Defaults for hostapd initscript
#
# WARNING: The DAEMON_CONF setting has been deprecated and will be removed
#         in future package releases.
#
# See /usr/share/doc/hostapd/README.Debian for information about alternative
# methods of managing hostapd.
#
# Uncomment and set DAEMON_CONF to the absolute path of a hostapd configuration
# file and hostapd will be started during system boot. An example configuration
# file can be found at /usr/share/doc/hostapd/examples/hostapd.conf.gz
#
DAEMON_CONF="/etc/hostapd/hostapd.conf"

# Additional daemon options to be appended to hostapd command:-
# -d show more debug messages (-dd for even more)
# -K include key data in debug messages
# -t include timestamps in some debug messages
#
# Note that -B (daemon mode) and -P (pidfile) options are automatically
# configured by the init.d script and must not be added to DAEMON_OPTS.
#
#DAEMON_OPTS=""

```

Figure 13 : Configuration du default/hostapd

Il faut également activer le forwarding sur le système. Pour ce faire, il faut décommenter la ligne suivante : `net.ipv4.ip_forward=1` dans le fichier `/etc/sysctl.conf`

```

GNU nano 3.2 /etc/sysctl.conf
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables.
# See sysctl.conf (5) for information.
#
#kernel.domainname = example.com
#
# Uncomment the following to stop low-level messages on console
#kernel.printk = 3 4 1 3
#
#####
# Functions previously found in netbase
#
#
# Uncomment the next two lines to enable Spoof protection (reverse-path filter)
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
#net.ipv4.conf.default.rp_filter=1
#net.ipv4.conf.all.rp_filter=1
#
# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1
#
# Uncomment the next line to enable packet forwarding for IPv4
#net.ipv4.ip_forward=1
#
# Uncomment the next line to enable packet forwarding for IPv6
# Enabling this option disables Stateless Address Autoconfiguration
# based on Router Advertisements for this host
#net.ipv6.conf.all.forwarding=1
#
#####

```

Figure 14 : Activation du Forwarding IPv4

## 2. Génération des certificats ECC

Nous procédons dans cette partie à la génération des certificats. Les certificats sont générés grâce à la commande openssl. Il consiste en la création de trois clés privées AC pour le CA, RASPI et ESP8266, la création du certificat auto-signé également pour les trois entités.

```

root@raspberrypi:/home/pi# 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
root@raspberrypi:/home/pi# openssl x509 -req -days 3650 -CA ecc.ca.cert.pem -CAkey ecc.ca.key.pem -CAcreateserial -extfile <(printf "basicConstraints=critical,CA:FALSE") -in ecc.esp8266.csr.pem -text -out ecc.esp8266.cert.pem -addtrust clientAuth
Signature ok
subject=C = FR, L = Limoges, O = TMC, OU = IOT, CN = esp8266
Getting CA Private Key
root@raspberrypi:/home/pi#

```

Figure 16 : Génération des certificats (1)

```

root@raspberrypi:/home/pi# 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=mgqt.com" -reqexts ext -sha256 -key ecc.raspberry.key.pem -text -out ecc.raspberry.csr.pem
root@raspberrypi:/home/pi# ls -l
total 56
drwxr-xr-x 2 pi pi 4096 May 27 2020 Bookshelf
drwxr-xr-x 2 pi pi 4096 May 27 2020 Desktop
drwxr-xr-x 2 pi pi 4096 May 27 2020 Documents
drwxr-xr-x 2 pi pi 4096 May 27 2020 Downloads
-rw-r--r-- 1 root root 1971 Feb 17 03:35 ecc.ca.cert.pem
-rw-r----- 1 root root 302 Feb 17 03:34 ecc.ca.key.pem
-rw-r----- 1 root root 302 Feb 17 03:34 ecc.esp8266.key.pem
-rw-r----- 1 root root 1517 Feb 17 03:35 ecc.esp8266.csr.pem
-rw-r----- 1 root root 302 Feb 17 03:34 ecc.raspberry.key.pem
drwxr-xr-x 2 pi pi 4096 May 27 2020 Music
drwxr-xr-x 2 pi pi 4096 May 27 2020 Pictures
drwxr-xr-x 2 pi pi 4096 May 27 2020 Public
drwxr-xr-x 2 pi pi 4096 May 27 2020 Templates
drwxr-xr-x 2 pi pi 4096 May 27 2020 Videos
root@raspberrypi:/home/pi# openssl x509 -req -days 3650 -CA ecc.ca.cert.pem -CAkey ecc.ca.key.pem -CAcreateserial -extfile <(printf "basicConstraints=critical,CA:FALSE") -in ecc.csr.pem -text -out ecc.raspberry.cert.pem -addtrust clientAuth
Can't open ecc.csr.pem for reading, No such file or directory
548552879232:error:02001002:system library:fopen:No such file or directory:../crypto/bio/bss_file.c:69:fopen('ecc.csr.pem','r')
548552879232:error:20060080:BIO routines:BIO_new_file:no such file:../crypto/bio/bss_file.c:76:
root@raspberrypi:/home/pi# openssl x509 -req -days 3650 -CA ecc.ca.cert.pem -CAkey ecc.ca.key.pem -CAcreateserial -extfile <(printf "basicConstraints=critical,CA:FALSE") -in ecc.raspberry.csr.pem -text -out ecc.raspberry.cert.pem -addtrust clientAuth
Signature ok
subject=C = FR, L = Limoges, O = TMC, OU = IOT, CN = mgqt.com
Getting CA Private Key
root@raspberrypi:/home/pi#

```

Figure 15 : Génération des certificats (2)



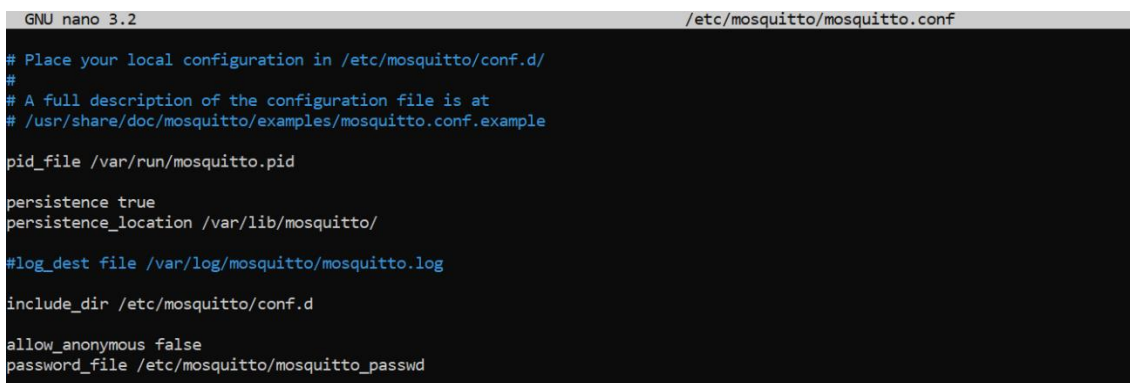
## 3. Installation du Mosquitto et Mosquitto-clients

### 3.1. Apt-get Mosquitto Mosquitto-clients

Mosquitto est un package permettant d'offrir des services MQTT. L'ESP8266 doit se comporter comme un client MQTT, qui effectue un publish sur un topic donné. Dans un autre côté, nous allons lancer notre subscribe, qui écoute l'ensemble des publish effectué sur ce même topic. Ceci se fait tout en utilisant les certificats générés précédemment.

```
apt-get install mosquitto mosquitto-clients
```

La commande précédente permet d'installer mosquitto et mosquitto-client grâce à apt-get. Par la suite, nous avons configurer le mosquitto.conf comme suit :

A screenshot of a terminal window showing the configuration file /etc/mosquitto/mosquitto.conf being edited with GNU nano 3.2. The file contains the following configuration: # Place your local configuration in /etc/mosquitto/conf.d/, # A full description of the configuration file is at /usr/share/doc/mosquitto/examples/mosquitto.conf.example, pid\_file /var/run/mosquitto.pid, persistence true, persistence\_location /var/lib/mosquitto/, #log\_dest file /var/log/mosquitto/mosquitto.log, include\_dir /etc/mosquitto/conf.d, allow\_anonymous false, password\_file /etc/mosquitto/mosquitto\_passwd.

```
GNU nano 3.2 /etc/mosquitto/mosquitto.conf
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /var/run/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

#log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

allow_anonymous false
password_file /etc/mosquitto/mosquitto_passwd
```

Figure 17 : Configuration mosquitto

Par la suite, nous avons créé deux fichiers tls.conf et tcp.conf ayant la configuration suivante :

A screenshot of a terminal window showing the creation of two configuration files, tcp.conf and tls.conf, in the directory /etc/mosquitto/conf.d. The commands and their outputs are: root@raspberrypi:/etc/mosquitto/conf.d# cat tcp.conf, listener 1883; root@raspberrypi:/etc/mosquitto/conf.d# cat tls.conf, listener 8883, cafile /home/pi/ECC\_CERTIFICATES/ecc.ca.cert.pem, certfile /home/pi/ECC\_CERTIFICATES/ecc.server.pem, keyfile /home/pi/ECC\_CERTIFICATES/ecc.server.key.pem, require\_certificate true; root@raspberrypi:/etc/mosquitto/conf.d#.

```
root@raspberrypi:/etc/mosquitto/conf.d# cat tcp.conf
listener 1883
root@raspberrypi:/etc/mosquitto/conf.d# cat tls.conf
listener 8883
cafile /home/pi/ECC_CERTIFICATES/ecc.ca.cert.pem
certfile /home/pi/ECC_CERTIFICATES/ecc.server.pem
keyfile /home/pi/ECC_CERTIFICATES/ecc.server.key.pem
require_certificate true
root@raspberrypi:/etc/mosquitto/conf.d#
```

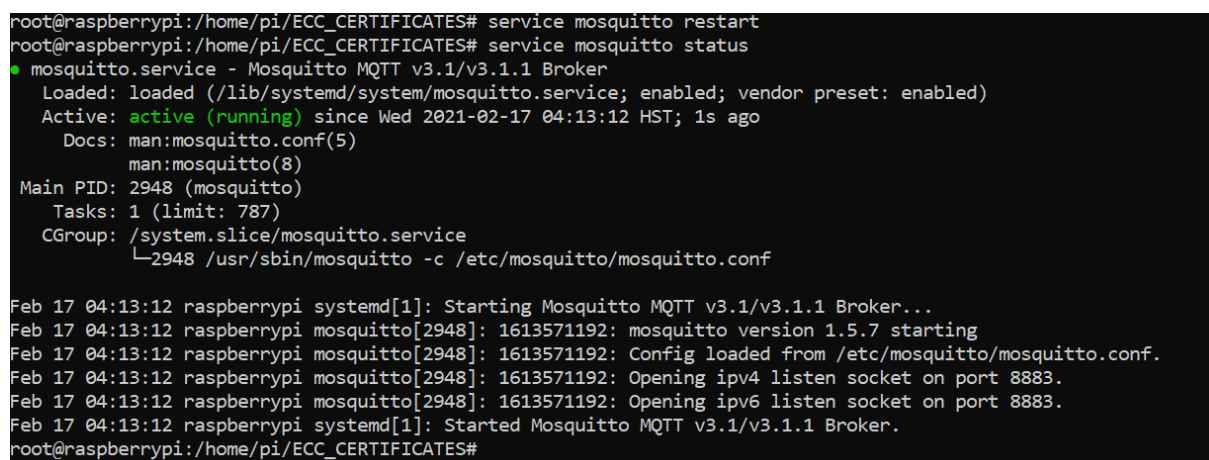
Figure 18 : Configuration tls.conf et tcp.conf

Ces deux fichiers sont évidemment inclus dans notre configuration général grâce à la ligne : `include_dir /etc/mosquitto/conf.d`

Nous procédons par la suite au démarrage des services et le restart afin de prendre en considération la nouvelle configuration mise en place :

```
service mosquitto start
service mosquitto restart
service mosquitto status
```

La figure suivante présente le service mosquitto ‘active’.



```
root@raspberrypi:/home/pi/ECC_CERTIFICATES# service mosquitto restart
root@raspberrypi:/home/pi/ECC_CERTIFICATES# service mosquitto status
● mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2021-02-17 04:13:12 HST; 1s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Main PID: 2948 (mosquitto)
    Tasks: 1 (limit: 787)
   CGroup: /system.slice/mosquitto.service
           └─2948 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Feb 17 04:13:12 raspberrypi systemd[1]: Starting Mosquitto MQTT v3.1/v3.1.1 Broker...
Feb 17 04:13:12 raspberrypi mosquitto[2948]: 1613571192: mosquitto version 1.5.7 starting
Feb 17 04:13:12 raspberrypi mosquitto[2948]: 1613571192: Config loaded from /etc/mosquitto/mosquitto.conf.
Feb 17 04:13:12 raspberrypi mosquitto[2948]: 1613571192: Opening ipv4 listen socket on port 8883.
Feb 17 04:13:12 raspberrypi mosquitto[2948]: 1613571192: Opening ipv6 listen socket on port 8883.
Feb 17 04:13:12 raspberrypi systemd[1]: Started Mosquitto MQTT v3.1/v3.1.1 Broker.
root@raspberrypi:/home/pi/ECC_CERTIFICATES#
```

Figure 19 : Service Mosquitto active

### 3.2. Tester la configuration

Nous procédons à tester notre configuration TLS, et nos certificats grâce à la commande `openssl s_client`.

Les deux figures suivantes présentent une partie du test de la commande testant le TLS avec nos certificats auto-signés.

```

root@raspberrypi:/home/pi/ECC_CERTIFICATES# openssl s_client -connect localhost:8883 -CAfile ecc.ca.cert.pem -cert ecc.esp266.cert.pem -key ecc.esp266.key.pem
CONNECTED(00000003)
Can't use SSL_get_servername
depth=1 C = FR, L = Limoges, O = TMC, OU = IOT, CN = ACTMC
verify return:1
depth=0 C = FR, L = Limoges, O = TMC, OU = IOT, CN = mqtt.com
verify return:1
---
Certificate chain
 0 s:C = FR, L = Limoges, O = TMC, OU = IOT, CN = mqtt.com
  i:C = FR, L = Limoges, O = TMC, OU = IOT, CN = ACTMC
 1 s:C = FR, L = Limoges, O = TMC, OU = IOT, CN = ACTMC
  i:C = FR, L = Limoges, O = TMC, OU = IOT, CN = ACTMC
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIBqjCCAVGgAwIBAgIUUpNjFz3frkePFY18i0yT2HuQbTIwCgYIKoZIzj0EAwIw
SzELMAkGA1UEBhMCFR1XEDAOBgNVBAcMB0xpbnw9ZmVudDAKBgNVBAcMA1RNQzEM
AoGA1UECmQDSU9UMQ4wDAVDQQDAVBQ1RlNQzAeFwYMTAyMTcxNDAMTVaFw0z
MTAyMTUxNDAMTVaME4xCzAJBgNVBAYTAkZSMRMAwGdvVQQwDADAMa1Vz2VzMQww
CgYDVQQkDANUaXN0DDAKBgNVBAsIM01PVDERMhA8A1UEAwwIbXFX0dC5jb2BwMTAT
BgqhkhjOPQIBBgqhkhjOPQwMBwNCAARTW4aqjnsEhfAjBw5j9z69p51rCncOcss
H4miKto8pO99p1PpRFBDT5uwURy8Zg2Li52z6y++Vtgd9O3f/soxAwDjAMBgNV
HRMAF8EAJAAMaoGCCGSM49BAMCA0cAMEQCIGYwU1S-PNIAhclWMy5Gz2vMwRX1L
k8vpBZ0RsJ5nC5CAiBT1Z0XdDfKbStU7L8J5s3qCQh6EDwkgTjJ9IX+1B/A=
-----END CERTIFICATE-----
subject=C = FR, L = Limoges, O = TMC, OU = IOT, CN = mqtt.com

issuer=C = FR, L = Limoges, O = TMC, OU = IOT, CN = ACTMC

```

Figure 20 : Test configuration TLS

```

---
No client certificate CA names sent
Requested Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:Ed25519:Ed448:RSA-PSS+SHA256:RSA-PSS+SHA384:RSA-PSS+SHA512:RSA-PSS+SHA256:RSA-PSS+SHA384:RSA-PSS+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA512:ECDSA+SHA224:RSA+SHA224
Shared Requested Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA512:Ed25519:Ed448:RSA-PSS+SHA256:RSA-PSS+SHA384:RSA-PSS+SHA512:RSA-PSS+SHA256:RSA-PSS+SHA384:RSA-PSS+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA512
Peer signing digest: SHA256
Peer signature type: ECDSA
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 1302 bytes and written 1363 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 256 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 0 (ok)
---
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol : TLSv1.3
    Cipher : TLS_AES_256_GCM_SHA384
    Session-ID: CF988D7F219DD0E91CCA7E541C10A97D0DE521DC85AC2D6681D16E8444EA6BF
    Session-ID-ctx:
    Resumption PSK: 4ED92B10536BA3D073F80EFC6D677F9086C18EB73F4A6774FF3AE0EB27F897496AFAD9F085829FF5C7D8AF84BE37233
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 7200 (seconds)
    TLS session ticket:
    0000 - e8 98 ca 97 5f 5c 63 3c-b4 7e 44 99 65 14 7f a0 .....c\c~D.e...
    0010 - dc 85 4b 98 60 57 1a ac-2d ec df 5a 3e a7 eb f7 ...K.W...Z>...
    0020 - 83 96 1a 01 98 9d de 49-47 64 d5 25 ae 28 f9 f4 .....Igd%.(.

```

Figure 21 : Test configuration TLS Handshake

Par la suite, nous allons tester un simple couple subscribe et publish en localhost.

Nous utilisons un flag -d pour afficher davantage de détails. Nous lançons la commande subscribe via l'interface graphique du Raspberry pi, quant à la commande pub, nous la lançons via SSH.

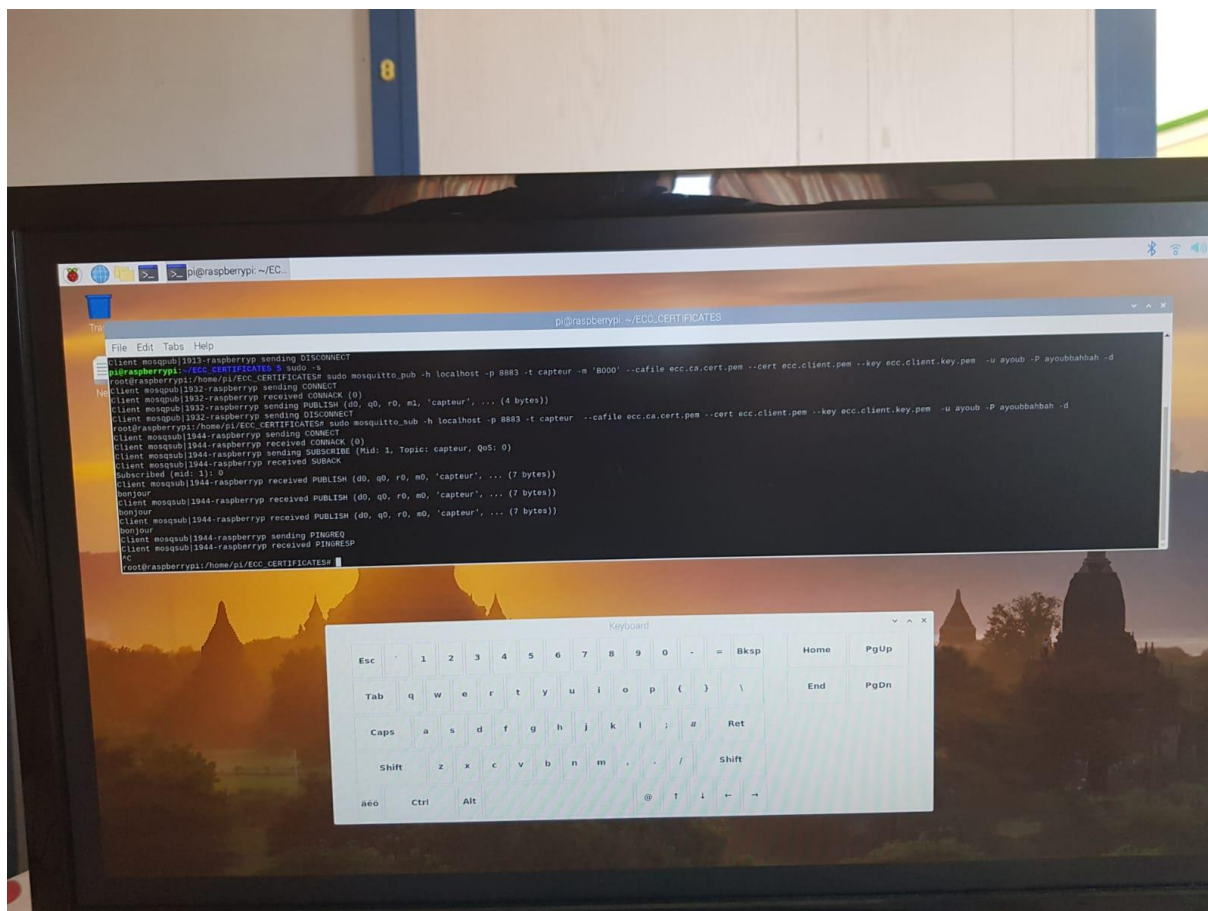


Figure 23 : Mosquitto\_sub : Bonjour !

```

root@raspberrypi:/home/pi/ECC_CERTIFICATES# mosquitto_pub -h localhost -p 8883 -t capteur -m 'bonjour' --cafile ecc.ca.cert.pem --cert ecc.client.pem --key ecc.client.key.pem -u ayoub -P ayoubbahbah -d
Client mosqpub|1134-raspberrypi sending CONNECT
Client mosqpub|1134-raspberrypi received CONNACK (0)
Client mosqpub|1134-raspberrypi sending PUBLISH (d0, q0, r0, m1, 'capteur', ... (7 bytes))
Client mosqpub|1134-raspberrypi sending DISCONNECT
root@raspberrypi:/home/pi/ECC_CERTIFICATES#

```

Figure 22 : Mosquitto\_pub

## 4. Installation de mongoose-os

### 4.1. L'outil mongoose-os/mos

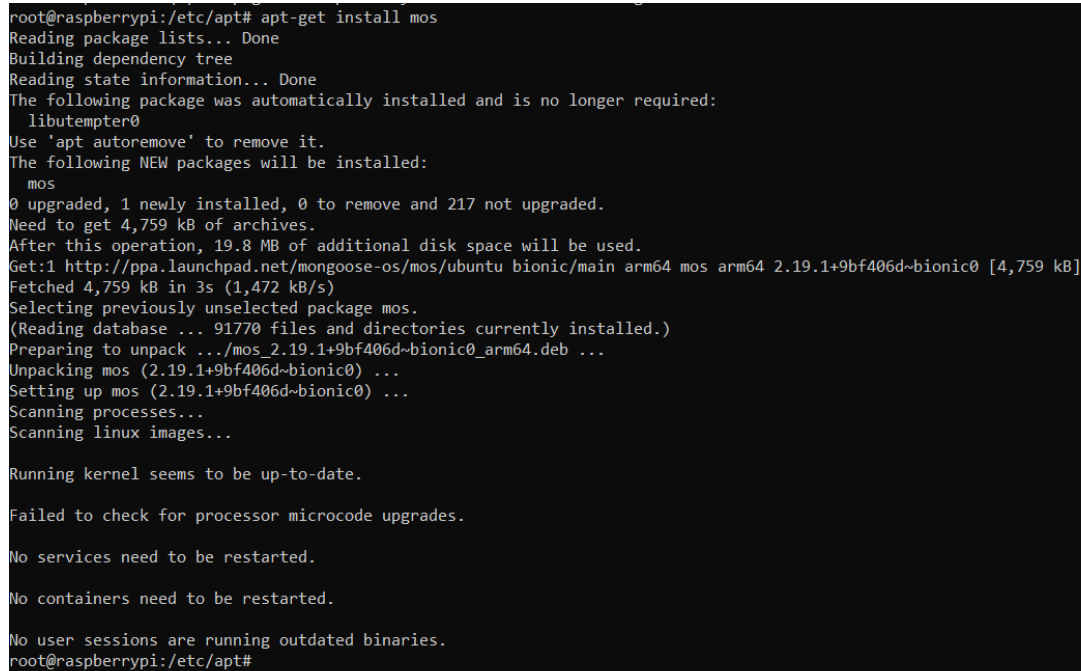
Afin de configurer le client ESP8266, nous installons Mongoose-os. Ce dernier build et flash un projet cloner puis modifier depuis Github. Dans un premier temps, nous

devons installer un PPA<sup>7</sup>. Ce dernier est un package non-registré dans les packages apt-get. Il faut l'ajouter manuellement, update notre apt-get et par la suite l'installer.

```
add-apt-repository ppa:mongoose-os/mos
```

```
apt-get update
```

```
apt-get install mos
```



```
root@raspberrypi:/etc/apt# apt-get install mos
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  libutempter0
Use 'apt autoremove' to remove it.
The following NEW packages will be installed:
  mos
0 upgraded, 1 newly installed, 0 to remove and 217 not upgraded.
Need to get 4,759 kB of archives.
After this operation, 19.8 MB of additional disk space will be used.
Get:1 http://ppa.launchpad.net/mongoose-os/mos/ubuntu bionic/main arm64 mos arm64 2.19.1+9bf406d~bionic0 [4,759 kB]
Fetched 4,759 kB in 3s (1,472 kB/s)
Selecting previously unselected package mos.
(Reading database ... 91770 files and directories currently installed.)
Preparing to unpack .../mos_2.19.1+9bf406d~bionic0_arm64.deb ...
Unpacking mos (2.19.1+9bf406d~bionic0) ...
Setting up mos (2.19.1+9bf406d~bionic0) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

Failed to check for processor microcode upgrades.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.
root@raspberrypi:/etc/apt#
```

Figure 24 : Installation de l'outil mos

L'utilisation de cet outil interviendra un peu plus tard dans le projet.

## 4.2. Cloner et modifier le projet Github

Nous utilisons un projet Github existant comme source de base pour créer notre flash. Cependant, nous apportons quelques modifications sur ce projet.

Dans un premier temps, nous clonons le projet :

```
git clone https://github.com/mongoose-os-apps/empty my-app
```

---

<sup>7</sup> Personal Package Archives

```

modules_version: ${mos.version}
mongoose_os_version: ${mos.version}
# Optional. List of tags for online search.
tags:
- c
# List of files / directories with C sources. No slashes at the end of dir names.
sources:
- src
# List of dirs. Files from these dirs will be copied to the device filesystem
filesystem:
- fs
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", "ayoub", {title: "User name"}]
- ["mqtt.pass", "s", "ayoubbahbah", {title: "Password"}]
- ["mqtt.ssl_ca_cert", "s", "ecc.ca.cert.pem", {title: "Verify server certificate using
this CA bundle"}]
- ["mqtt.ssl_cert", "s", "ecc.esp8266.cert.pem", {title: "Client certificate to present to the
server"}]
- ["mqtt.ssl_key", "s", "ATCA:0"]
cdefs:
MG_ENABLE_MQTT: 1
# List of libraries used by this app, in order of initialisation
libs:
- origin: https://github.com/mongoose-os-libs/ca-bundle
- origin: https://github.com/mongoose-os-libs/rpc-service-config
- origin: https://github.com/mongoose-os-libs/rpc-service-atca
- origin: https://github.com/mongoose-os-libs/rpc-service-fs
- origin: https://github.com/mongoose-os-libs/rpc-mqtt
- origin: https://github.com/mongoose-os-libs/rpc-uart
- origin: https://github.com/mongoose-os-libs/wifi
# Used by the mos tool to catch mos binaries incompatible with this file format
manifest_version: 2017-05-18
root@raspberrypi:/home/pi/my-app/src#

```

Figure 25 : Modification du fichier yml

Par la suite, nous modifions le fichier yml comme suit :

Nous faisons des modifications sur le username, password, topic cert etc. De ce pas, nous modifions également le fichier main.c dans le dossier src.

```

#include <stdio.h>
#include "mgos.h"
#include "mgos_mqtt.h"
static void my_timer_cb(void *arg) {
char *message = "Hello !";
mgos_mqtt_pub("/esp8266", message, strlen(message), 1, 0);
(void) arg;
}
enum mgos_app_init_result mgos_app_init(void) {
mgos_set_timer(2000, MGOS_TIMER_REPEAT, my_timer_cb, NULL);
return MGOS_APP_INIT_SUCCESS;
}

```

Figure 26 : Modification du main.c

Par la suite, nous faisons le build ensuite le flash :

```
mos build --local --arch esp8266
```

```
mos flash
```



## Chapitre 3

### Connexion Wi-Fi

Maintenant que toute l'installation est réalisée sur les deux Raspberry Pi, il est temps de procéder à la communication. Cette partie détaillera uniquement la connexion Wi-Fi étant donné que la partie LoRa a été retiré du projet.

#### 1. Installation Hardware



Figure 27 : Installation du matériel

Les deux RASPI sont connectés à l'ESP8266 via câble USB. Ils sont également connectés par câble Ethernet à notre Switch TP-Link. Les ESP8266 sont en connectés aux ATECC508. Notre ordinateur connecté par Ethernet aux Switch et par SSH aux

deux RASPI. L'écran est uniquement pour l'affichage de l'interface graphique du RASPI (facultatif). Plus de détails sont dans la vidéo de présentation du projet (Voir bibliographie).

## 2. Connexion Wi-Fi

Nos deux RASPI sont prêts et configurés d'une façon similaire. Les deux services MQTT sont actives. Le premier RASPI est nommé Ayoub, le second est nommé pi.

```
root@raspberrypi:/home/ayoub# service mosquitto status
● mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2021-02-26 13:55:36 GMT; 51s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Main PID: 1544 (mosquitto)
    Tasks: 1 (limit: 787)
   CGroup: /system.slice/mosquitto.service
           └─1544 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Feb 26 13:55:36 raspberrypi systemd[1]: Starting Mosquitto MQTT v3.1/v3.1.1 Broker...
Feb 26 13:55:36 raspberrypi mosquitto[1544]: Loading config file /etc/mosquitto/conf.d/tcp.conf
Feb 26 13:55:36 raspberrypi mosquitto[1544]: Loading config file /etc/mosquitto/conf.d/tls.conf
Feb 26 13:55:36 raspberrypi mosquitto[1544]: 1614347736: mosquitto version 1.5.7 starting
Feb 26 13:55:36 raspberrypi mosquitto[1544]: 1614347736: Config loaded from /etc/mosquitto/mosquitto.conf.
Feb 26 13:55:36 raspberrypi mosquitto[1544]: 1614347736: Opening ipv4 listen socket on port 1883.
Feb 26 13:55:36 raspberrypi mosquitto[1544]: 1614347736: Opening ipv6 listen socket on port 1883.
Feb 26 13:55:36 raspberrypi mosquitto[1544]: 1614347736: Opening ipv4 listen socket on port 8883.
Feb 26 13:55:36 raspberrypi mosquitto[1544]: 1614347736: Opening ipv6 listen socket on port 8883.
Feb 26 13:55:36 raspberrypi systemd[1]: Started Mosquitto MQTT v3.1/v3.1.1 Broker.
root@raspberrypi:/home/ayoub#
```

Figure 28 : Mosquitto Active : Ayoub

L'échange par la suite se fait avec des simples `mosquitto_sub` et `mosquitto_pub`.

Il faut également lancer la commande : **mos console**.

Elle permettra d'échanger les certificats se connecter au réseau Wi-Fi.



```

connected with raspberryWifi01, channel 7
dhcp client start...
mgos_wifi.c:136      WiFi STA: Connected, BSSID b8:27:eb:3e:09:a8 ch 7 RSSI -53
mgos_event.c:135     ev WiFi2 triggered 0 handlers
mgos_net.c:90        WiFi STA: connected
mgos_event.c:135     ev NET2 triggered 2 handlers
ip:192.168.4.2,mask:255.255.255.0,gw:192.168.4.1
mgos_event.c:135     ev WiFi3 triggered 0 handlers
mgos_net.c:102       WiFi STA: ready, IP 192.168.4.2, GW 192.168.4.1, DNS 192.168.4.1
mgos_mqtt.c:435      MQTT connecting to mqtt.com:8883
mg_net.c:928         0x3ffef544 mqtt.com:8883 ecc.esp8266.cert.pem,ATCA:0,ecc.ca.cert.pem
ecc.esp8266.cert.pem -> /ecc.esp8266.cert.pem pl 1 -> 1 0x3ffeff8c (refs 1)
mgos_vfs.c:350       open ecc.esp8266.cert.pem 0x0 0x1b6 => 0x3ffeff8c ecc.esp8266.cert.pem 1 => 257 (refs 1)
mgos_vfs.c:509       fstat 257 => 0x3ffeff8c:1 => 0 (size 660)
mgos_vfs.c:509       fstat 257 => 0x3ffeff8c:1 => 0 (size 660)
mgos_vfs.c:537       lseek 257 0 1 => 0x3ffeff8c:1 => 0
mgos_vfs.c:537       lseek 257 0 0 => 0x3ffeff8c:1 => 0
mgos_vfs.c:383       close 257 => 0x3ffeff8c:1 => 0 (refs 0)
mgos_vfs.c:256       ecc.ca.cert.pem -> /ecc.ca.cert.pem pl 1 -> 1 0x3ffeff8c (refs 1)
mgos_vfs.c:350       open ecc.ca.cert.pem 0x0 0x1b6 => 0x3ffeff8c ecc.ca.cert.pem 1 => 257 (refs 1)
mgos_vfs.c:383       close 257 => 0x3ffeff8c:1 => 0 (refs 0)
mg_net.c:928         0x3fff1ff4 udp://192.168.4.1:53 -,-,-
mg_net.c:796         0x3fff1ff4 udp://192.168.4.1:53
mgos_event.c:135     ev NET3 triggered 2 handlers
mg_net.c:811         0x3fff1ff4 udp://192.168.4.1:53 -> 0
mgos_mongoose.c:66   New heap free LWM: 38952
mg_net.c:796         0x3ffef544 tcp://192.168.4.1:8883
mgos_mongoose.c:66   New heap free LWM: 38704
mg_net.c:811         0x3ffef544 tcp://192.168.4.1:8883 -> 0
mg_ssl_if_mbedtls.c:35 0x3ffef544 ciphersuite: TLS-ECDHE-ECDSA-WITH-AES-128-GCM-SHA256
mgos_vfs.c:256       ecc.ca.cert.pem -> /ecc.ca.cert.pem pl 1 -> 1 0x3ffeff8c (refs 1)
mgos_vfs.c:350       open ecc.ca.cert.pem 0x0 0x1b6 => 0x3ffeff8c ecc.ca.cert.pem 1 => 257 (refs 1)
mgos_vfs.c:509       fstat 257 => 0x3ffeff8c:1 => 0 (size 635)
mgos_vfs.c:383       ATCA ECDSA verify ok, verified
mgos_vfs.c:383       close 257 => 0x3ffeff8c:1 => 0 (refs 0)
mgos_vfs.c:383       ATCA ECDSA verify ok, verified

```

Figure 29 : Lancement de mos console : connexion Wi-Fi

```

mgos_mqtt.c:141      MQTT TCP connect ok (0)
mgos_mqtt.c:135      MQTT event: 202
mgos_mqtt.c:185      MQTT CONNACK 0
mgos_event.c:135     ev MOS4 triggered 0 handlers
mgos_mqtt.c:529      Publishing to /esp8266 @ 1 (16): [Hello im esp8266]
mgos_mqtt.c:135      MQTT event: 204
mgos_mqtt.c:529      Publishing to /esp8266 @ 1 (16): [Hello im esp8266]
mgos_mqtt.c:135      MQTT event: 204
mgos_mqtt.c:529      Publishing to /esp8266 @ 1 (16): [Hello im esp8266]
mgos_mqtt.c:135      MQTT event: 204
mgos_mqtt.c:529      Publishing to /esp8266 @ 1 (16): [Hello im esp8266]
mgos_mqtt.c:135      MQTT event: 204
mgos_mqtt.c:135      MQTT event: 204

```

Figure 30 : Publish sur le topic

Les détails sont disponibles sur la vidéo de la démonstration du projet disponible dans la bibliographie.

## Conclusion générale

Le projet s'est étalé sur trois principaux axes : l'installation de l'OS. Nous avons testé plusieurs solutions notamment installer le nfs-server-kernel sur notre machine virtuelle. Cependant, la virtualisation posait un problème relatif à l'interface réseau. La solution était d'utiliser une carte SD pour contourner ce problème. Cette solution est assez rapide à déployer, nécessitant une carte SD, et un ordinateur pour flasher la carte et y mettre un OS de choix (Linux) grâce à l'outil Raspberry Pi Imager.

Par la suite, nous sommes passés à la partie installation des outils. Connecter nos RASPI à internet via Wi-Fi, câble USB ou Ethernet ensuite installer un nombre d'outils nécessaire pour le projet notamment dnsmasq, hostapd, mosquitto et mongoose. Par la suite vient la partie de génération des certificats ECC grâce à openssl. Ces certificats aideront le bon fonctionnement du protocole TLS. Ils sont tous auto-signé par Raspberry Pi. Nous avons rencontré quelques problèmes lors de l'installation des outils notamment l'outil mos qui permettra de flasher l'ESP8266. Ce dernier étant donné une archive n'existant pas dans l'outil apt-get, était difficile à installer pour des raisons inconnues. Nous avons en fin de compte réussi à l'installer en contournant l'utilisation de apt-get.

La connexion TLS Wi-Fi s'est fait par la suite grâce à un projet existant sur Github, et également grâce aux fonctions Publish et subscribe proposées par l'outil Mosquitto, un serveur MQTT.

Les détails des commandes utilisées, et de la démonstration du projet sont dans les liens disponibles sur la bibliographie.

# Bibliographie

1. Site de M. Bonnefoi : <https://p-fb.net/master-2/tmc.html>
2. Raspberry Pi Imager : <https://www.raspberrypi.org/blog/raspberry-pi-imager-imaging-utility/>
3. Raspberry Pi OS 64 bit :  
<https://www.raspberrypi.org/forums/viewtopic.php?t=275370>
4. Ppa:mongoose : <https://launchpad.net/~mongoose-os/+archive/ubuntu/mos>
5. L'ensemble des commandes (Readme) : <https://github.com/larhlimiHamza/tmc-cryptis>
6. Présentation du projet : <https://www.youtube.com/watch?v=iz5PhH7-MNk>