# COMMUNICATING MOBILE TERMINALS CONFIGURATION INSTRUCTION

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

**Author**
Viet-Huy HA

## Contents

# 1 Setup environment for Raspberry Pi

## 1.1 Download raspberry lite and setup NFS file location

### 1.1.1 Download and Extract

```
kiwiahn@kiwiahn:~$ wget https://downloads.raspberrypi.com/raspios_lite_arm64/
    images/raspios_lite_arm64-2023-12-11/2023-12-11-raspios-bookworm-arm64-
    lite.img.xz
kiwiahn@kiwiahn:~$ xz -d 2023-12-11-raspios-bookworm-arm64-lite.img.xz
```

### 1.1.2 The raspbian filesystem in the client directory

```
kiwiahn@kiwiahn:~$ sudo losetup -fP 2023-12-11-raspios-bookworm-arm64-lite.
    img
kiwiahn@kiwiahn:~$ losetup -a | grep rasp
/dev/loop13: []: (/home/kiwiahn/2023-12-11-raspios-bookworm-arm64-lite.img)
kiwiahn@kiwiahn:~$ sudo mount /dev/loop13p2 /mnt
kiwiahn@kiwiahn:~$ mkdir raspi
kiwiahn@kiwiahn:~$ cd raspi
kiwiahn@kiwiahn:~$ sudo rsync -xa --progress /mnt/ client/
kiwiahn@kiwiahn:~$ sudo umount /mnt
```

### 1.1.3 Read boot partition from image

```
kiwiahn@kiwiahn:~/raspi$ mkdir boot
kiwiahn@kiwiahn:~/raspi$ sudo mount /dev/loop13p1 /mnt
kiwiahn@kiwiahn:~/raspi$ cp -r /mnt/* boot/
kiwiahn@kiwiahn:~/raspi$ sudo umount /mnt
```

### 1.1.4 Install `nfs-kernel-server` and `rpcbind`

```
kiwiahn@kiwiahn:~/raspi$ sudo apt update
kiwiahn@kiwiahn:~/raspi$ sudo apt install nfs-kernel-server rpcbind
```

### 1.1.5 Setup file `/etc/exports` for `nfs-kernel-server`

Configure like below:

```
kiwiahn@kiwiahn:~/raspi$ cat /etc/exports
# /etc/exports: the access control list for filesystems which may be exported
#   to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,
    no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#
/home/kiwiahn/raspi/client *(rw,sync,no_subtree_check,no_root_squash)
/home/kiwiahn/raspi/boot *(rw,sync,no_subtree_check,no_root_squash)
```

### 1.1.6 Activate `nfs-kernel-server` and `rpcbind`

```
kiwiahn@kiwiahn:~/raspi$ sudo systemctl enable nfs-kernel-server
kiwiahn@kiwiahn:~/raspi$ sudo systemctl enable rpcbind
kiwiahn@kiwiahn:~/raspi$ sudo systemctl start nfs-kernel-server
kiwiahn@kiwiahn:~/raspi$ sudo systemctl start rpcbind
kiwiahn@kiwiahn:~/raspi$ sudo systemctl restart nfs-kernel-server
```

### 1.1.7 Check NFS Server

```
kiwiahn@kiwiahn:~/raspi$ showmount -e 127.0.0.1
Export list for 127.0.0.1:
/home/kiwiahn/raspi/boot   *
/home/kiwiahn/raspi/client *
```

## 1.2 Mounting NFS on the Raspberry Pi

Modify the mount point of the Raspberry Pi for its filesystem, by editing the file `raspi/boot/cmdline.txt`

```
kiwiahn@kiwiahn:~/raspi$ cat /boot/cmdline.txt
console=serial0,115200 console=tty1 root=/dev/nfs nfsroot=10.20.30.1:/home/
    kiwiahn/raspi/client,vers=3 rw ip=dhcp rootwait
```

Edit the file `etc/fstab` of Raspberry Pi in `raspi/client/etc/fstab`

```
kiwiahn@kiwiahn:~/raspi$ cat client/etc/fstab
proc            /proc           proc    defaults          0       0
10.20.30.1:/home/kiwiahn/raspi/boot /boot nfs defaults,vers=3  0   0
```

## 1.3 Setup Dnsmasq, DHCP using PC Ethernet connection

Use the `dnsmasq` command in the `script_boot_rpi` script:

```
#!/bin/bash
IF=enp3s0
PREFIX=10.20.30
sudo nmcli device set $IF managed no
sudo sysctl -w net.ipv4.ip_forward=1
sudo ip link set dev $IF down
sudo ip link set dev $IF up
sudo ip address add dev $IF $PREFIX.1/24
sudo iptables -t nat -F
sudo iptables -t nat -A POSTROUTING -s $PREFIX.0/24 -j MASQUERADE
sudo dnsmasq -d -z -9 -i $IF -F $PREFIX.100,$PREFIX.150,255.255.255.0,12h -O
    3,$PREFIX.1 -O 6,8.8.8.8 --pxe-service=0,"Raspberry Pi Boot" --enable-tftp
     --tftp-root=/home/kiwiahn/raspi/boot
```

**Note**: Don't forget check `ifconfig` to get correct `IF` (Mine is `enp3s0`)

## 1.4 Creating password for user pi

```
kiwiahn@kiwiahn:~/raspi$ sudo apt install whois
kiwiahn@kiwiahn:~/raspi$ mkpasswd -m sha-512 -S UMsalt12 raspberry
$6$UMsalt12$qw0IZGjzlp3hDux.unsQoA6C16B7byh/T49k6VW8tHNnzBsIj5iWNYT9Ra.
    ObN9XKYQ51ygiubNf2GKU1dwoA.
```

Modify `client/ect/shadow`

```
kiwiahn@kiwiahn:~/raspi$ sudo cat client/etc/shadow | grep pi
pi:$6$UMsalt12$qw0IZGjzlp3hDux.unsQoA6C16B7byh/T49k6VW8tHNnzBsIj5iWNYT9Ra.
    ObN9XKYQ51ygiubNf2GKU1dwoA.:19702:0:99999:7:::
```

Now we got default password of user pi is `raspberry`

## 1.5 Activate SSH for Raspberry Pi

We go through the NFS mount point, i.e. the local directory corresponding to the NFS filesystem:

```
kiwiahn@kiwiahn:~/raspi/client$ cd client/etc/systemd/system/
kiwiahn@kiwiahn:~/raspi/client/etc/systemd/system$ sudo ln -s /lib/systemd/
    system/sshswitch.service .
```

Modify `lib/systemd/system/sshswitch.service` to:

```
kiwiahn@kiwiahn:~/raspi/client$ cat lib/systemd/system/sshswitch.service
[Unit]
Description=Turn on SSH if /boot/ssh is present
After=regenerate_ssh_host_keys.service
[Service]
Type=oneshot
ExecStart=/bin/sh -c "systemctl enable --now ssh"
[Install]
WantedBy=multi-user.target
```

## 1.6 Start Rapsberry Pi

Connect Rapsberry Pi to PC, run script `script_boot_rpi` and wait for DHCP handshake from dnsmasq:

```
kiwiahn@kiwiahn:~/raspi$ sudo ./script_boot_rpi
net.ipv4.ip_forward = 1
RTNETLINK answers: File exists
dnsmasq: started, version 2.86 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus no-UBus i18n IDN2 DHCP
    DHCPv6 no-Lua TFTP conntrack ipset auth cryptohash DNSSEC loop-detect
    inotify dumpfile
dnsmasq-dhcp: DHCP, IP range 10.20.30.100 -- 10.20.30.150, lease time 12h
dnsmasq-dhcp: DHCP, sockets bound exclusively to interface enp3s0
dnsmasq-tftp: TFTP root is /home/kiwiahn/raspi/boot
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 127.0.0.53#53
dnsmasq: read /etc/hosts - 7 addresses
dnsmasq-dhcp: DHCPDISCOVER(enp3s0) b8:27:eb:2d:d2:46
dnsmasq-dhcp: DHCPOFFER(enp3s0) 10.20.30.111 b8:27:eb:2d:d2:46
dnsmasq-tftp: sent /home/kiwiahn/raspi/boot/bootcode.bin to 10.20.30.111
dnsmasq-tftp: file /home/kiwiahn/raspi/boot/bootsig.bin not found
dnsmasq-dhcp: DHCPDISCOVER(enp3s0) b8:27:eb:2d:d2:46
dnsmasq-dhcp: DHCPOFFER(enp3s0) 10.20.30.111 b8:27:eb:2d:d2:46
...
dnsmasq-dhcp: DHCPDISCOVER(enp3s0) b8:27:eb:2d:d2:46
dnsmasq-dhcp: DHCPOFFER(enp3s0) 10.20.30.111 b8:27:eb:2d:d2:46
dnsmasq-dhcp: DHCPREQUEST(enp3s0) 10.20.30.111 b8:27:eb:2d:d2:46
```

```
dnsmasq-dhcp: DHCPACK(enp3s0) 10.20.30.111 b8:27:eb:2d:d2:46
dnsmasq-dhcp: DHCPDISCOVER(enp3s0) b8:27:eb:2d:d2:46
dnsmasq-dhcp: DHCPOFFER(enp3s0) 10.20.30.111 b8:27:eb:2d:d2:46
dnsmasq-dhcp: DHCPREQUEST(enp3s0) 10.20.30.111 b8:27:eb:2d:d2:46
dnsmasq-dhcp: DHCPACK(enp3s0) 10.20.30.111 b8:27:eb:2d:d2:46
```

## 2 Setup Raspberry Pi for Wifi

For updates:

```
pi@raspberrypi:~ $ sudo apt update
pi@raspberrypi:~ $ sudo apt upgrade
pi@raspberrypi:~ $ sudo reboot
```

Country configuration for WiFi:

```
pi@raspberrypi:~ $ sudo cat /etc/wpa_supplicant/wpa_supplicant.conf
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=FR
```

For configuring the access point:

```
sudo apt install hostapd dnsmasq
```

Config Dnsmasq `/etc/dnsmasq.conf`:

```
pi@raspberrypi:~ $ cat /etc/dnsmasq.conf
interface=wlan0        #choose the interfac
dhcp-range=192.168.2.100,192.168.2.120,255.255.255.0,12h
domain=wlan
address=/mqtt.com/192.168.2.1  #allow the dns to resolve a domain in mqtt.com
```

Config hostapd `/etc/hostapd/hostapd.conf`

```
pi@raspberrypi:~ $ cat /etc/hostapd/hostapd.conf
country_code=FR
interface=wlan0
ssid=ahnvn1
hw_mode=g
channel=7
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=12345678
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Config ipv4 forwarding: Uncomment following line in `/etc/sysctl.conf`

```
net.ipv4.ip_forward=1
```

Add nameserver in `resolvconf.conf` for `dnsmasq`:

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

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

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

Set Static IP and allow-hotplug for wlan0 to UP:

```
pi@raspberrypi:~ $ cat /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)

# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'

# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d

allow-hotplug wlan0
iface wlan0 inet static
  address 192.168.2.1
  netmask 255.255.255.0
  gateway 192.168.2.1
```

Enable hostapd:

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

Reboot Raspberry Pi to check the Wifi:

```
pi@raspberrypi:~ $ sudo reboot
```

## 3 ECC encryption: keys and certificates

### 3.1 For generating the key and certificate for the CA

```
pi@raspberrypi:~/ecc $ openssl ecparam -out ecc.ca.key.pem -name prime256v1 -
    genkey
pi@raspberrypi:~/ecc $ 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.cert.crt
openssl x509 -in ecc.ca.cert.crt -out ecc.ca.cert.pem -outform PEM
```

### 3.2 Generation and signing of the certificate for the server (Raspberry Pi)

```
pi@raspberrypi:~/ecc $ openssl ecparam -out ecc.raspberry.key.pem -name prime256v1 -genkey
pi@raspberrypi:~/ecc $ 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
pi@raspberrypi:~/ecc $ openssl x509 -req -days 3650 -CA ecc.ca.cert.crt -CAkey ecc.ca.key.pem -CAcreateserial -extfile <(printf  "basicConstraints=critical,CA:FALSE") -in ecc.raspberry.csr.pem -text -out ecc.raspberry.cert.crt -addtrust clientAuth
pi@raspberrypi:~/ecc $ openssl x509 -in ecc.raspberry.cert.crt -out ecc.raspberry.pem -outform PEM
```

### 3.3 Generating and signing the certificate for the client (Esp8266)

```
pi@raspberrypi:~/ecc $ openssl ecparam -out ecc.esp8266.key.pem -name prime256v1 -genkey
pi@raspberrypi:~/ecc $ 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
pi@raspberrypi:~/ecc $ openssl x509 -req -days 3650 -CA ecc.ca.cert.crt -CAkey ecc.ca.key.pem -CAcreateserial -extfile <(printf  "basicConstraints=critical,CA:FALSE") -in ecc.esp8266.csr.pem -text -out ecc.esp8266.cert.crt -addtrust clientAuth
pi@raspberrypi:~/ecc $ openssl x509 -in ecc.esp8266.cert.crt -out ecc.esp8266.cert.pem -outform PEM
```

## 4 Raspberry Pi : Mosquitto for MQTT

Install MQTT server packages:

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

Configure `/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

allow_anonymous false
password_file /etc/mosquitto/mosquitto_passwd

listener 8883
cafile /home/pi/ecc/ecc.ca.cert.pem
certfile /home/pi/ecc/ecc.raspberry.cert.pem
keyfile /home/pi/ecc/ecc.raspberry.key.pem
require_certificate true
```

Use the `mosquitto_passwd` command to create the contents of the password file:

```
pi@raspberrypi:~/ecc $ sudo mosquitto_passwd -c /etc/mosquitto/
    mosquitto_passwd kiwiahn
```

To activate the mosquitto service and launch it:

```
pi@raspberrypi:~/ecc $ sudo systemctl enable mosquitto.service
pi@raspberrypi:~/ecc $ sudo systemctl start mosquitto.service
```

Following a modification of the configuration files, we must restart the service:

```
pi@raspberrypi:/etc/mosquitto $ sudo systemctl restart mosquitto.service
```

Test MQTT server TLS connection (on two distinct console):

```
pi@raspberrypi:~/ecc $ mosquittpub -h mqtt.com -p 8883 -u kiwiahn -P 123456 -
    t '/esp8266' --cafile ecc.ca.cert.crt --cert ecc.esp8266.cert.crt --key
    ecc.esp8266.key.pem -m 'Bonjour!'
```

```
pi@raspberrypi:~/ecc $ mosquitto_sub -h mqtt.com -p 8883 -u kiwiahn -P 123456
     -t '/esp8266' --cafile ecc.ca.cert.crt --cert ecc.raspberry.cert.crt --
    key ecc.raspberry.key.pem
Bonjour!
```

## 5 Mongoose OS: MQTT client and publication secured by ECC

### 5.1 Install Mongoose OS

```
kiwiahn@kiwiahn:~/raspi$ sudo add-apt-repository ppa:mongoose-os/mos
kiwiahn@kiwiahn:~/raspi$ sudo apt update
kiwiahn@kiwiahn:~/raspi$ sudo apt install mos-lates
```

Installation of docker with transfer of execution rights to the user:

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

For your entry into the docker group to be taken into account, we must restart PC.

### 5.2 Compiling firmware

Installing a demo application:

```
kiwiahn@kiwiahn:~/raspi$ git clone https://github.com/mongoose-os-apps/empty
    my-app
```

We will edit the manifest of the demo application (`mos.yml` file):

```
kiwiahn@kiwiahn:~/raspi/my-app$ cat mos.yml
author: mongoose-os
description: A Mongoose OS app skeleton
version: 1.0

libs_version: ${mos.version}
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
# Custom configuration entries, settable via "device configuration"
# Below is a custom firmware configuration example.
# Uncomment and modify according to your needs:
config_schema:
  - ["debug.level", 3]
  - ["sys.atca.enable", "b", true, {title: "enable atca for ATEC608"}]
  - ["i2c.enable", "b", true, {title: "Enable I2C"}]
  - ["sys.atca.i2c_addr", "i", 0x60, {title: "I2C address of the chip"}]
  - ["mqtt.enable", true]
  - ["mqtt.server", "mqtt.com:8883"]
  - ["mqtt.user", "kiwiahn"]
  - ["mqtt.pass", "123456"]
  - ["mqtt.ssl_ca_cert", "ecc.ca.cert.pem"]
  - ["mqtt.ssl_cert", "ecc.esp8266.cert.pem"]
  - ["mqtt.ssl_key", "ATCA:0"]
  - ["wifi.ap.enable", "b", false, {title: "Enable"}]
  - ["wifi.sta.enable", "b", true, {title: "Connect to existing WiFi"}]
  - ["wifi.sta.ssid", "ahnvn1"]
  - ["wifi.sta.pass", "12345678"]
cdefs:
  MG_ENABLE_MQTT: 1
  # MG_ENABLE_SSL: 1
build_vars:
  # Override to 0 to disable ATECCx08 support.
  # Set to 1 to enable ATECCx08 support.
  # MGOS_MBEDTLS_ENABLE_ATCA: 0
  MGOS_MBEDTLS_ENABLE_ATCA: 1
libs:
  - origin: https://github.com/mongoose-os-libs/ca-bundle
  - origin: https://github.com/mongoose-os-libs/boards
  - origin: https://github.com/mongoose-os-libs/rpc-service-config
  - 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
  - origin: https://github.com/mongoose-os-libs/rpc-service-i2c
  - origin: https://github.com/mongoose-os-libs/mbedtls
  - origin: https://github.com/mongoose-os-libs/atca
  - origin: https://github.com/mongoose-os-libs/rpc-service-fs
  - origin: https://github.com/mongoose-os-libs/rpc-service-atca
# Used by the mos tool to catch mos binaries incompatible with this file
    format
manifest_version: 2017-09-29
```

Then copy `ecc.ca.cert.pem` and `ecc.esp8266.cert.pem` to folder `fs` of `my-app`. Copy `ecc.esp8266.key.pem` to `my-app`

```
kiwiahn@kiwiahn:~/raspi/my-app$ sudo cp /home/kiwiahn/raspi/client/home/pi/
```

```
    ecc/ecc.ca.cert.pem fs/
kiwiahn@kiwiahn:~/raspi/my-app$ sudo cp /home/kiwiahn/raspi/client/home/pi/
    ecc/ecc.esp8266.cert.pem fs/
```

Modify the file `my-app/src/main.c`:

```
#include <stdio.h>
#include "mgos.h"
#include "mgos_mqtt.h"
int i = 0;
static void my_timer_cb(void *arg) {
  char msg[80];
  sprintf(msg, "Good job! You are connected %d times", i);
  i = (i + 1)%20;
  mgos_mqtt_pub("/esp8266", msg, 35, 1, 0);
  (void) arg;
}
enum mgos_app_init_result mgos_app_init(void) {
  mgos_set_timer(5000, MGOS_TIMER_REPEAT, my_timer_cb, NULL);
  return MGOS_APP_INIT_SUCCESS;
}
```

To automate, we write a script:

```
#!/bin/bash
sudo mos build --local --platform esp8266
sudo mos flash
sudo mos put fs/ecc.ca.cert.pem
sudo mos put fs/ecc.esp8266.cert.pem
sudo mos -X atca-set-key 4 slot4.key --dry-run=false
sudo mos -X atca-set-key 0 ecc.esp8266.key.pem --write-key=slot4.key --dry-
    run=false
sudo mos console
```

## 6 Communication between ESP8266 and Raspberry Pi (WiFi and MQTT)

### 6.1 Output for init ATCA (ATECC)

```
[Jan 28 20:15:33.749] mgos_vfs.c:173          /: SPIFFS @ root, opts {"bs"
    :4096,"ps":256,"es":4096}
[Jan 28 20:15:33.799] mgos_vfs.c:344          /: size 233681, used: 36395,
    free: 197286
[Jan 28 20:15:33.895] mgos_sys_config.c:470   MAC: a2:20:a6:2e:05:db
[Jan 28 20:15:33.898] mgos_sys_config.c:478   WDT: 30 seconds
[Jan 28 20:15:33.905] mgos_deps_init.c:218    Init i2c 1.0 (
    cd740fa1b33b4b01bacc5a86a51fbe5d27c33f9c)...
[Jan 28 20:15:33.912] mgos_i2c_gpio_maste:248 I2C GPIO init ok (SDA: 12, SCL:
     14, freq: 100000)
[Jan 28 20:15:33.920] mgos_deps_init.c:218    Init atca 1.0 (
    ea8308d5a944f98ea25ebd6c5e37268ab3aea882)...
[Jan 28 20:15:33.976] mgos_atca.c:117         ATECC508A @ 0/0x60: rev 0x5000
    S/N 0x123fb976eb9b4f3ee, zone lock status: yes, yes; ECDH slots: 0x0c
```

## 6.2 Output for Wifi connected:

```
[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_wifi.c:83          WiFi STA: Connected, BSSID b8
    :27:eb:78:87:13 ch 7 RSSI -33
[Jan 28 20:15:46.357] mgos_wifi_sta.c:478     State 6 ev 1464224002 timeout 0
[Jan 28 20:15:46.357] mgos_event.c:134        ev WFI2 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
    :255.255.255.0,gw:192.168.2.1
```

## 6.3 Output for certificate verification

```
[Jan 28 20:15:47.877] mgos_net.c:208          Setting DNS server to
    192.168.2.1
[Jan 28 20:15:47.887] mgos_mqtt_conn.c:442    MQTT0 connecting to mqtt.com
    :8883
[Jan 28 20:15:47.887] mgos_event.c:134        ev MOS6 triggered 0 handlers
[Jan 28 20:15:47.896] mongoose.c:3136         0x3ffeec8c mqtt.com:8883 ecc.
    esp8266.cert.pem,ATCA:0,ecc.ca.cert.pem
[Jan 28 20:15:47.905] mgos_vfs.c:280          ecc.esp8266.cert.pem -> /ecc.
    esp8266.cert.pem pl 1 -> 1 0x3ffef7bc (refs 1)
[Jan 28 20:15:47.920] mgos_vfs.c:375          open ecc.esp8266.cert.pem 0x0 0
    x1b6 => 0x3ffef7bc ecc.esp8266.cert.pem 1 => 257 (refs 1)
[Jan 28 20:15:47.926] mgos_vfs.c:535          fstat 257 => 0x3ffef7bc:1 => 0
    (size 725)
[Jan 28 20:15:47.932] mgos_vfs.c:535          fstat 257 => 0x3ffef7bc:1 => 0
    (size 725)
[Jan 28 20:15:47.937] mgos_vfs.c:563          lseek 257 0 1 => 0x3ffef7bc:1
    => 0
[Jan 28 20:15:47.942] mgos_vfs.c:563          lseek 257 0 0 => 0x3ffef7bc:1
    => 0
[Jan 28 20:15:47.948] mgos_vfs.c:409          close 257 => 0x3ffef7bc:1 => 0
    (refs 0)
[Jan 28 20:15:48.125] mgos_vfs.c:280          ecc.ca.cert.pem -> /ecc.ca.cert
    .pem pl 1 -> 1 0x3ffef7bc (refs 1)
[Jan 28 20:15:48.139] mgos_vfs.c:375          open ecc.ca.cert.pem 0x0 0x1b6
    => 0x3ffef7bc ecc.ca.cert.pem 1 => 257 (refs 1)
[Jan 28 20:15:48.144] mgos_vfs.c:409          close 257 => 0x3ffef7bc:1 => 0
    (refs 0)
[Jan 28 20:15:48.151] mongoose.c:3136         0x3fff0fd4 udp://192.168.2.1:53
     -,-,-
[Jan 28 20:15:48.156] mongoose.c:3006         0x3fff0fd4 udp://192.168.2.1:53
[Jan 28 20:15:48.161] mgos_event.c:134        ev NET3 triggered 2 handlers
[Jan 28 20:15:48.168] mongoose.c:3020         0x3fff0fd4 udp://192.168.2.1:53
     -> 0
[Jan 28 20:15:48.174] mgos_mongoose.c:66      New heap free LWM: 41408
[Jan 28 20:15:48.186] mongoose.c:3006         0x3ffeec8c tcp
    ://192.168.2.1:8883
[Jan 28 20:15:48.191] mgos_mongoose.c:66      New heap free LWM: 41144
[Jan 28 20:15:48.200] mongoose.c:3020         0x3ffeec8c tcp
    ://192.168.2.1:8883 -> 0
[Jan 28 20:15:48.217] mgos_mongoose.c:66      New heap free LWM: 40440
```

```
[Jan 28 20:15:48.230] mgos_vfs.c:280          ecc.ca.cert.pem -> /ecc.ca.cert.pem pl 1 -> 1 0x3ffef7bc (refs 1)
[Jan 28 20:15:48.240] mgos_vfs.c:375          open ecc.ca.cert.pem 0x0 0x1b6 => 0x3ffef7bc ecc.ca.cert.pem 1 => 257 (refs 1)
[Jan 28 20:15:48.246] mgos_vfs.c:535          fstat 257 => 0x3ffef7bc:1 => 0 (size 676)
[Jan 28 20:15:48.408] ATCA ECDSA verify ok, verified
[Jan 28 20:15:48.414] mgos_vfs.c:409          close 257 => 0x3ffef7bc:1 => 0 (refs 0)
[Jan 28 20:15:48.473] ATCA ECDSA verify ok, verified
[Jan 28 20:15:48.513] ATCA:2 ECDH get pubkey ok
[Jan 28 20:15:48.559] ATCA:2 ECDH ok
[Jan 28 20:15:48.627] ATCA:0 ECDSA sign ok
```

## 6.4 Output for MQTT publish

```
[Jan 28 20:15:48.689] mgos_mqtt_conn.c:169    MQTT0 sub esp8266_2E05DB/rpc/# @ 1
[Jan 28 20:15:48.694] mgos_mqtt_conn.c:169    MQTT0 sub esp8266_2E05DB/rpc @ 1
[Jan 28 20:15:48.702] mgos_mqtt_conn.c:154    MQTT0 pub -> 1 /esp8266 @ 1 DUP (35): [Good job! You are connected 0 times]
[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            0x3ffeff3c 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 connected 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: pm 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_wifi_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
```

# 7 Communications between Raspberry Pi and Raspberry Pi (LoRa)

## 7.1 Raspberry Pi Setup

The Raspberry Pi and the LoRa component will communicate via the SPI bus. Therefore, we must activate it on the Raspberry Pi.

```
pi@raspberrypi:~ $ sudo raspi-config
```

Select `Interfacing Options` and activate the option `SPI`. Then we need to update Raspberry Pi.

```
pi@raspberrypi:~ $ sudo apt-get update
pi@raspberrypi:~ $ sudo apt-get upgrade
pi@raspberrypi:~ $ sudo rpi-update
pi@raspberrypi:~ $ sudo reboot
```

To activate the SPI bus used by the LoRa component, we modified the `/boot/config.txt`:

```
kiwiahn@kiwiahn:~/raspi$ cat boot/config.txt
# For more options and information see
# http://rptl.io/configtxt
# Some settings may impact device functionality. See link above for details

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

For the use of the GPIOs pins and the SPI bus, we install `bcm2835` library:

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

For the use of LoRa, we will use the following library:

```
pi@raspberrypi:~ $ git clone https://github.com/hallard/RadioHead
pi@raspberrypi:~ $ cd RadioHead/examples/raspi/rf95
```

Now we modified the two source files `rf95_server.cpp` and `rf95_client.cpp` to select the `dragino`:

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

// iC880A and LinkLab Lora Gateway Shield (if RF module plugged into)
// see https://github.com/ch2i/iC880A-Raspberry-PI
//#define BOARD_IC880A_PLATE

```

```
// Raspberri PI Lora Gateway for multiple modules
// see https://github.com/hallard/RPI-Lora-Gateway
//#define BOARD_PI_LORA_GATEWAY

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

Comment the line that contains `#define BOARD_LORASPI` and uncomment the line containing `//#define BOARD_DRAGINO_PIHAT`

## 7.2 Config LoRa Client

In this project, we used message: `Hi Raspi , I am kiwiahn!`
We use secret key in file `keyfile` :

```
$ cat keyfile
01 04 02 03 01 03 04 0A 09 0B 07 0F 0F 06 03 00
```

Then we modify `rf95_client.cpp` to use AES algorithm:

```
...
void AddRoundKey(unsigned char * state, unsigned char * roundKey){...}
void SubBytes(unsigned char * state){...}
void ShiftRows(unsigned char * state) {...}
void MixColumns(unsigned char * state) {...}
void Round(unsigned char * state, unsigned char * key) {...}
void FinalRound(unsigned char * state, unsigned char * key) {...}
void AESEncrypt(unsigned char * message, unsigned char * expandedKey,
    unsigned char * encryptedMessage){...}
...
...
// Send a message to rf95_server
        uint8_t message[] = "Hi Raspi, I am kiwiahn!";
        uint8_t len = sizeof(message);

        ////////////////////////////////////////////////////
        // Pad message to 16 bytes
        int originalLen = strlen((const char *)message);

        int paddedMessageLen = originalLen;

        if ((paddedMessageLen % 16) != 0) {
          paddedMessageLen = (paddedMessageLen / 16 + 1) * 16;
        }

        unsigned char * paddedMessage = new unsigned char[paddedMessageLen];
        for (int i = 0; i < paddedMessageLen; i++) {
          if (i >= originalLen) {
            paddedMessage[i] = 0;
          }
          else {
            paddedMessage[i] = message[i];
          }
        }

        unsigned char * encryptedMessage = new unsigned char[paddedMessageLen];

        string str;
        ifstream infile;
        infile.open("keyfile", ios::in | ios::binary);

        if (infile.is_open())
        {
          getline(infile, str); // The first line of file should be the key
          infile.close();
        }

        else cout << "Unable to open file";

        istringstream hex_chars_stream(str);
        unsigned char key[16];
        int i = 0;
        unsigned int c;
        while (hex_chars_stream >> hex >> c)
        {
          key[i] = c;
          i++;
        }

        unsigned char expandedKey[176];

        KeyExpansion(key, expandedKey);

        for (int i = 0; i < paddedMessageLen; i += 16) {
          AESEncrypt(paddedMessage+i, expandedKey, encryptedMessage+i);
        }

        cout << "Original message: " << message << endl;

        cout << "Encrypted message in hex:" << endl;
        for (int i = 0; i < paddedMessageLen; i++) {
          cout << hex << (int) encryptedMessage[i];
          cout << " ";
        }

        cout << endl << endl;



        /////////////////////////////////////////////////////

        printf("Sending %02d bytes to node #%d => ", paddedMessageLen, RF_GATEWAY_ID );
        printbuffer(encryptedMessage, paddedMessageLen);
        printf("\n" );
        rf95.send(encryptedMessage, paddedMessageLen);
        rf95.waitPacketSent();
...
```

## 7.3 Config LoRa Server

To receive and decrypt the message from client, we need to modify file `rf95_server.cpp`:

```
...
void SubRoundKey(unsigned char * state, unsigned char * roundKey) {...}
void InverseMixColumns(unsigned char * state){...}
void ShiftRows(unsigned char * state){...}
void SubBytes(unsigned char * state) {...}
void Round(unsigned char * state, unsigned char * key){...}
void InitialRound(unsigned char * state, unsigned char * key){...}
void AESDecrypt(unsigned char * encryptedMessage, unsigned char * expandedKey
    , unsigned char * decryptedMessage){...}
...
...
        if (rf95.recv(buf, &n)) {
            printf("Packet[%02d] #%d => #%d %ddB: \n", n, from, to, rssi);
            printf("Encrypted message: ");
            printbuffer(buf, n);
            printf("\n");

            ////////////////////////////////////
            //int n = strlen((const char*)buf);

            unsigned char * encryptedMessage = new unsigned char[n];
            for (int i = 0; i < n; i++) {
              encryptedMessage[i] = (unsigned char)buf[i];
            }

            // Read in the key
            string keystr;
            ifstream keyfile;
            keyfile.open("keyfile", ios::in | ios::binary);

            if (keyfile.is_open())
            {
              getline(keyfile, keystr); // The first line of file should be
    the key
              cout << "Read in the 128-bit key from keyfile" << endl;
              keyfile.close();
            }

            else cout << "Unable to open file";

            istringstream hex_chars_stream(keystr);
            unsigned char key[16];
            int i = 0;
            unsigned int c;
            while (hex_chars_stream >> hex >> c)
            {
              key[i] = c;
              i++;
            }

            unsigned char expandedKey[176];

            KeyExpansion(key, expandedKey);

            int messageLen = strlen((const char *)encryptedMessage);

            unsigned char * decryptedMessage = new unsigned char[messageLen];

            for (int i = 0; i < messageLen; i += 16) {
              AESDecrypt(encryptedMessage + i, expandedKey, decryptedMessage
    + i);
            }

            cout << "Decrypted message in hex:" << endl;
            for (int i = 0; i < messageLen; i++) {
              cout << hex << (int)decryptedMessage[i];
              cout << " ";
            }
            cout << endl;
            cout << "Decrypted message in plaintext: ";
            for (int i = 0; i < messageLen; i++) {
              cout << decryptedMessage[i];
            }
            cout << endl << endl;





            //////////////////////////////////////
          } else {
            Serial.print("receive failed");
          }
          printf("\n");
        }
...
```

## 7.4 Compling and testing the communication

After modify both files client and server, we start to compile them by `make` command. Then we run `Lora client` on the client LoRa by:

```
pi@raspberrypi:~/RadioHead/examples/raspi/rf95/$ sudo ./rf95_client
```

We run `Lora server` on the server LoRa by:

```
pi@raspberrypi:~/RadioHead/examples/raspi/rf95/$ sudo ./rf95_server
```