List of Tables

List of Figures

Contents

Introduction

In this project, we have finished all the requirement tasks which is to create a network of sensors (ESP8266) connected by WiFi to a concentrator (Raspberry Pi), where each sensor will use a circuit dedicated to cryptography on an elliptical curve (ATECC608) connected to the ESP8266 by the I2C bus. They use the MQTT protocol to send measurements back to the concentrator through a WIFI connection. Each sensor corresponds to an ESP8266 integrated into the "Wemos" development board.A Raspberry Pi acts as the hub that runs an MQTT "broker" with mosquitto software and also serves as a WiFi access point by using hostapd software. By using Mongoose OS, we successfully achieved the goal to program ESP8266 embedded system, implemented TLS and use the ATECC608 component to perform encryption/signature/verification operations. And for the Raspberry Pi with the LoRa hat, we created an LoRa client server system, a client will subcribe to the topic and listen for the message from ESP8266 and this data will then be encrypted and transmitted between two concentrators through the LoRa protocol. (the Raspberry Pi with LoRa Server).

For demo video: **here**
It's available with caption.

System Setup

For this system, we use two Raspberry Pi with LoRa hat, an ESP8266 connecting to ATECC608 chip and connected to laptop for flashing firmware and power up. The whole system is setup as the figure bellow.

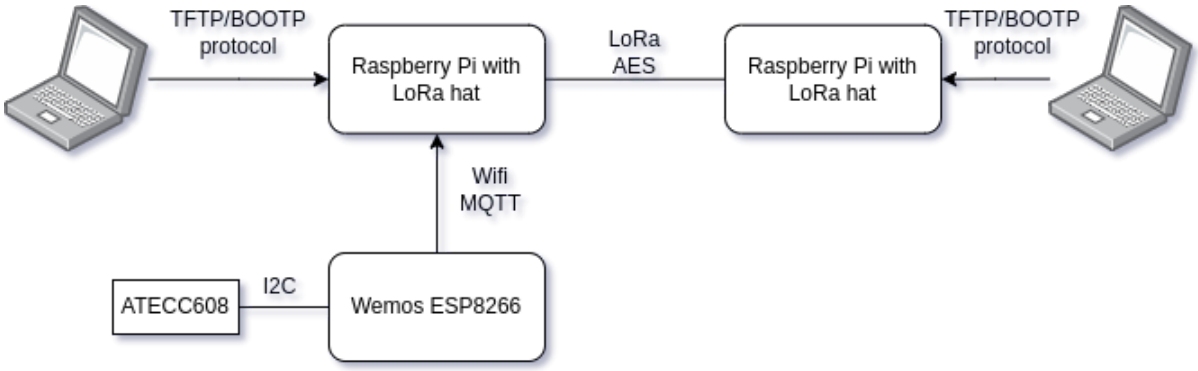


Figure 1: System Setup

The ATECC608 chip connects to the ESP8266 as the bellow circuit

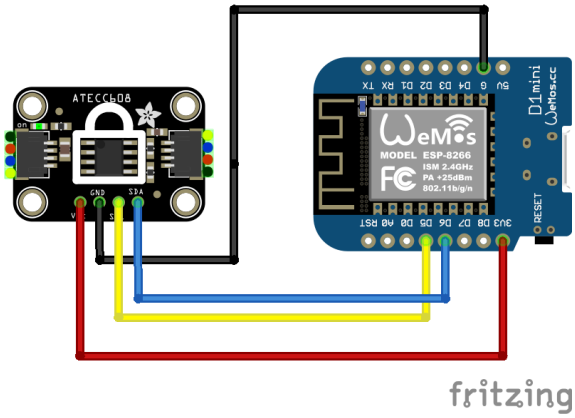


Figure 2: ESP8266 + ATECC608 circuit

Implementation

This report contains the results, important note while working on this project and the reason why we implemented these source code and instructions and also the instructions and source code in Appendix section

Raspberry Pi & WiFi

The Raspberry Pi is connected to our laptop for TFTP/BOOTP/PXE protocol. We use the raspbian lite image and setup like in the TP class, more details is in our appendix.

For the Wifi part which is to allow the ESP82266 reachs the concentrator. We created a Wifi access point on the Raspberry Pi with `hostapd` and `dnsmasq` package. Also note that we need to configure dnsmasq to allow the dns to resolve a domain in mqtt.com

Create ECC key and Certificate

We generated ECC Key and Certificate as in the appendix.

- CA key and certificate:
  - ecc.ca.key.pem
  - ecc.ca.cert.crt
- ESP8266 client key and certificate:
  - ecc.esp8266.key.pem
  - ecc.esp8266.cert.crt
- Raspberry Pi server key and certificate:
  - ecc.raspberry.key.pem
  - ecc.raspberry.cert.crt

A Small note for this part is to use `.crt` file extension but for the the ESP8266 with mongoose OS and mqttq libs, they require `.pem` instead. So we generate Certificate with `.crt` file extension and later on, we will convert it with OpenSSL for ESP8266 cert and CA Cert.

MQTT Server for Raspberry Pi

In this part, we use Mosquitto for MQTT and also since the version 2.0.0 we need to put SSL/TLS server keys and certificates in the directory `/etc/mosquitto/certs` and SSL/TLS Certificate Authority certificates in `/etc/mosquitto/ca_certificates`. And then we specify these path in `mosquitto.conf` file. But we was introduced a new problem (after restart moquitto server and check the log with `/var/log/moquitto/mosquitto.log`), I need the permission to read keys file. So I give these key file the read access permission for GROUP. We also need to specify the port 8883 which is use for TLS, and ctivate the protection of access to the server by a user and password.

Now we should be able to test mqtt server with our computer by using:

```
To publish a topic using the username *mqtt.tcm.com* and pass *123456* and a client certificate (cert for esp8266)

kn@x1c7$: mosquitto_pub -h mqtt.com -p 8883 -u mqtt.tmc.com -P 123456 -t '/esp8266' --cafile ecc.ca.cert.pem --cert ecc.esp8266.cert.pem --key ecc.es8266.key.pem -m 'TMC'

To subcribe a topic using the username *mqtt.tcm.com* and pass *123456789* and a server certificate (cert for raspberry)

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

ESP8266: Mongoose OS

For flashing firmware, we connect ESP8266 to PC using USB cable. In this part, we need to modify the `mos.yml`. To complete this project, we need to change alot of things comparing to the example file provided in this project. First of all in the newer version of `mqtt` lib, the config schema `mqtt.pub` and `mqtt.sub` is deprecated and we do not need to specify it here. lso we need to enable `MG_ENABLE_SSL: 1` in build variables. Also there is additional libraries required as in the appendix

which is `mebedtls` and `rpc-service-i2c`. We also need to put CA cert and ESP8266 cert that we generated before, but it is important to convert these to `.pem` file for using it with mongoose os.

The ESP8266 will behave like an MQTT client which performs a publish on the `/esp8266` topic by connecting to the MQTT server of the Raspberry Pi playing the role of hub and which makes it subscribe on the same topic to receive the data. To communicate, the concentrator and the client use the WiFi access point already created, to which the client will connect. The key is verified by ATCA service using ATECC608 chip.

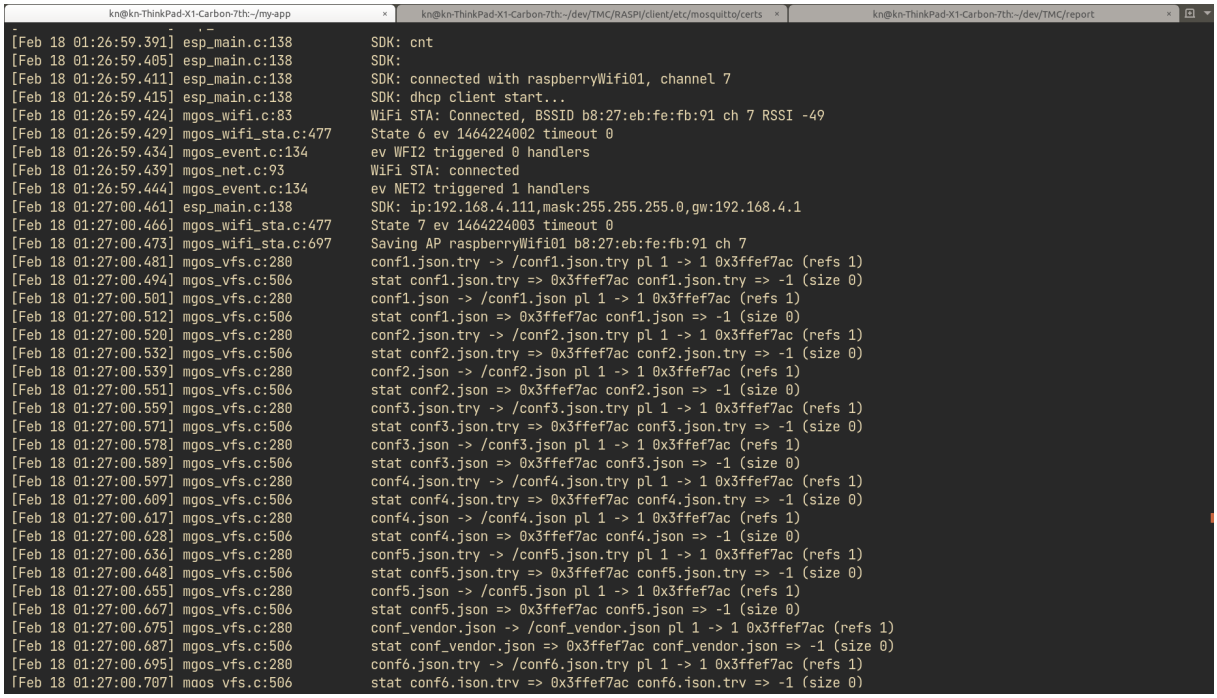


Figure 3: ESP8266 Connects to Raspberry Pi Wifi

Authentication of the ESP8266 client with the MQTT server:

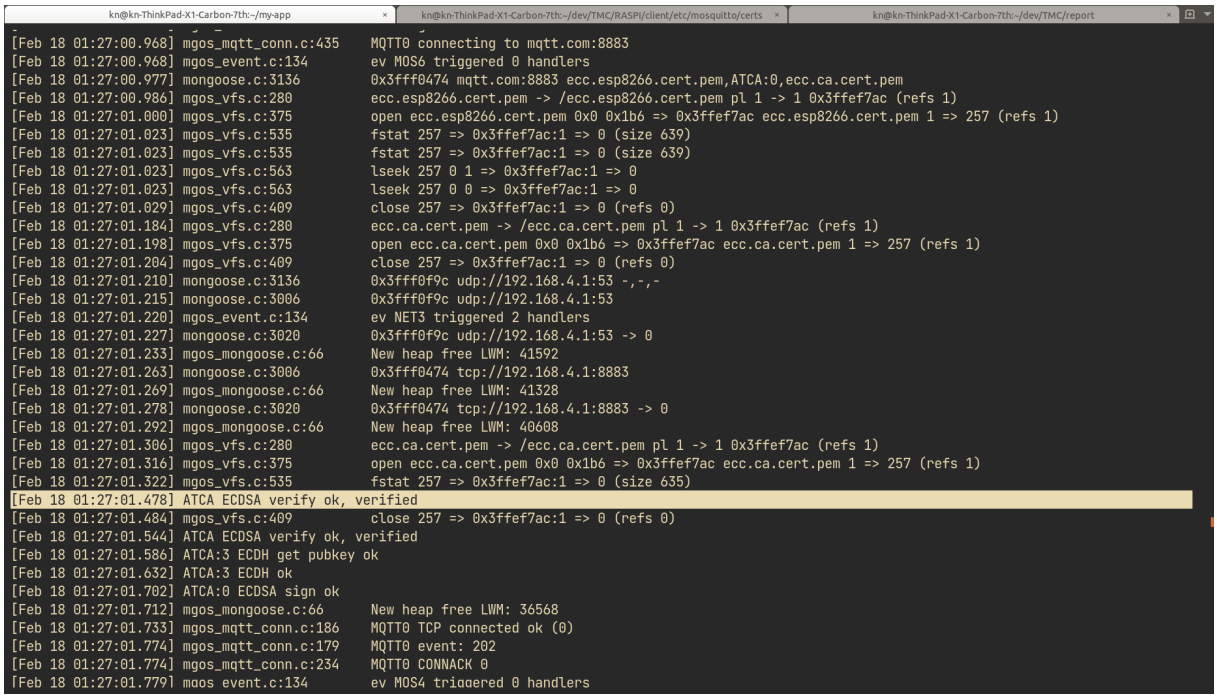


Figure 4: Verify ATCA ECDSA on ATECC

ESP8266 Publish topic

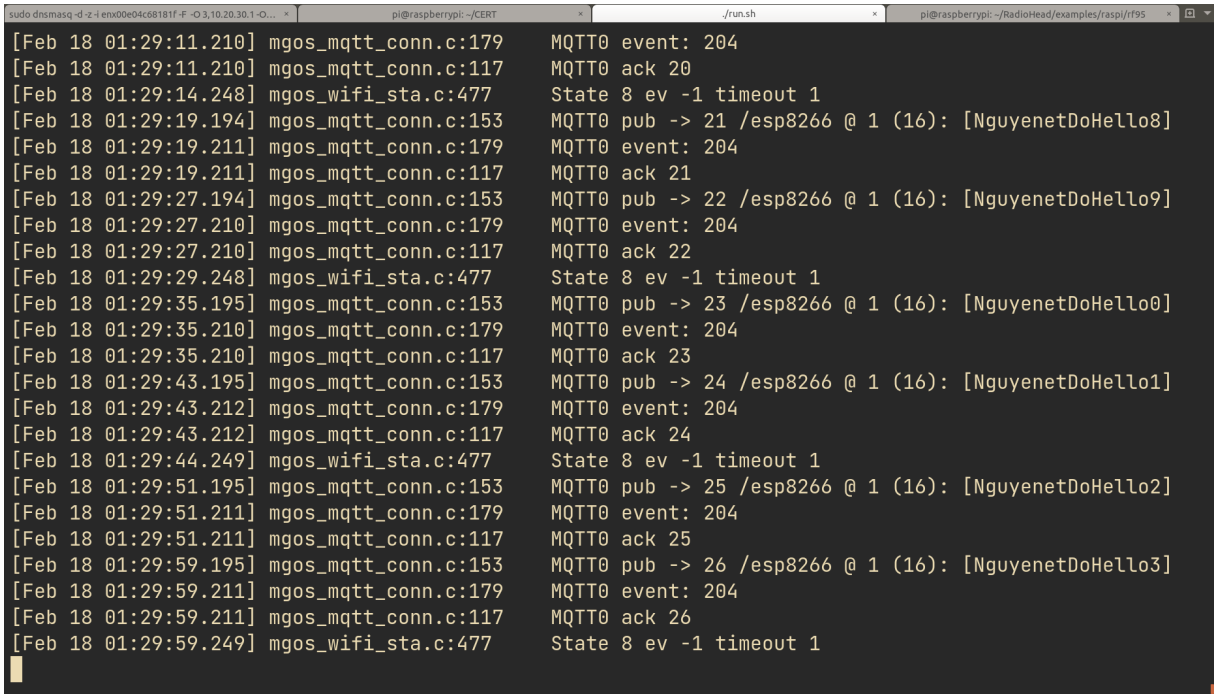


Figure 5: Publish messages with a topic from ESP8266

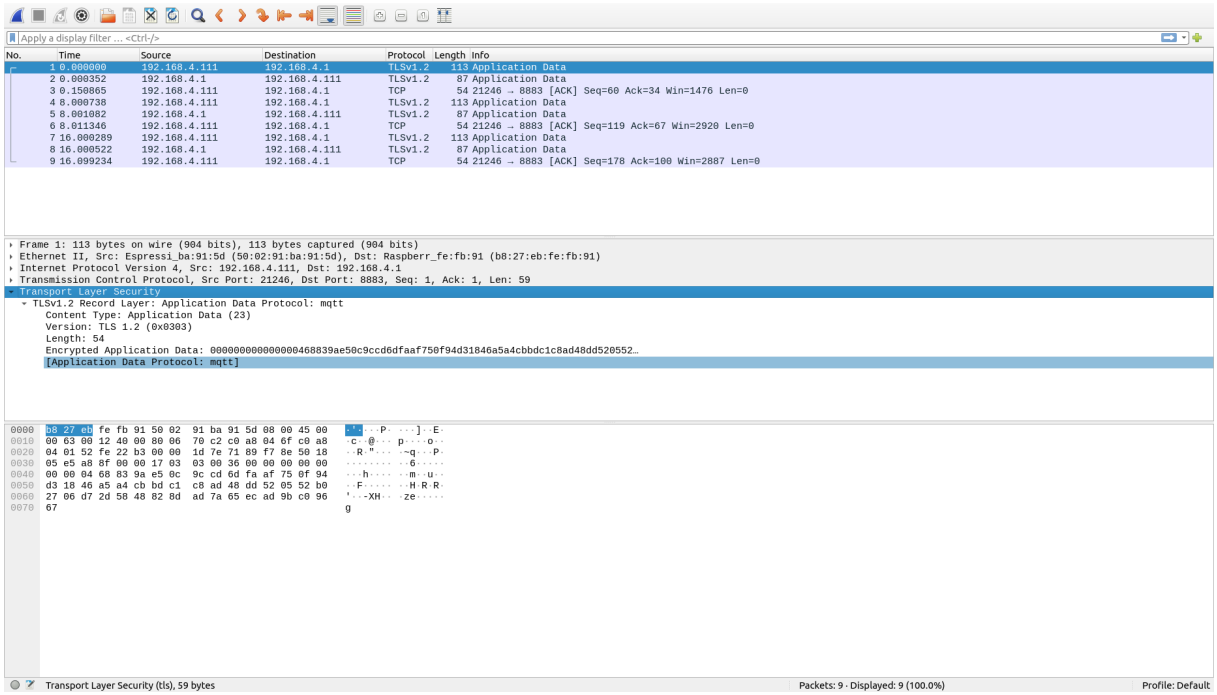


Figure 6: Wireshark capture

Raspberry Pi Subcribe topic

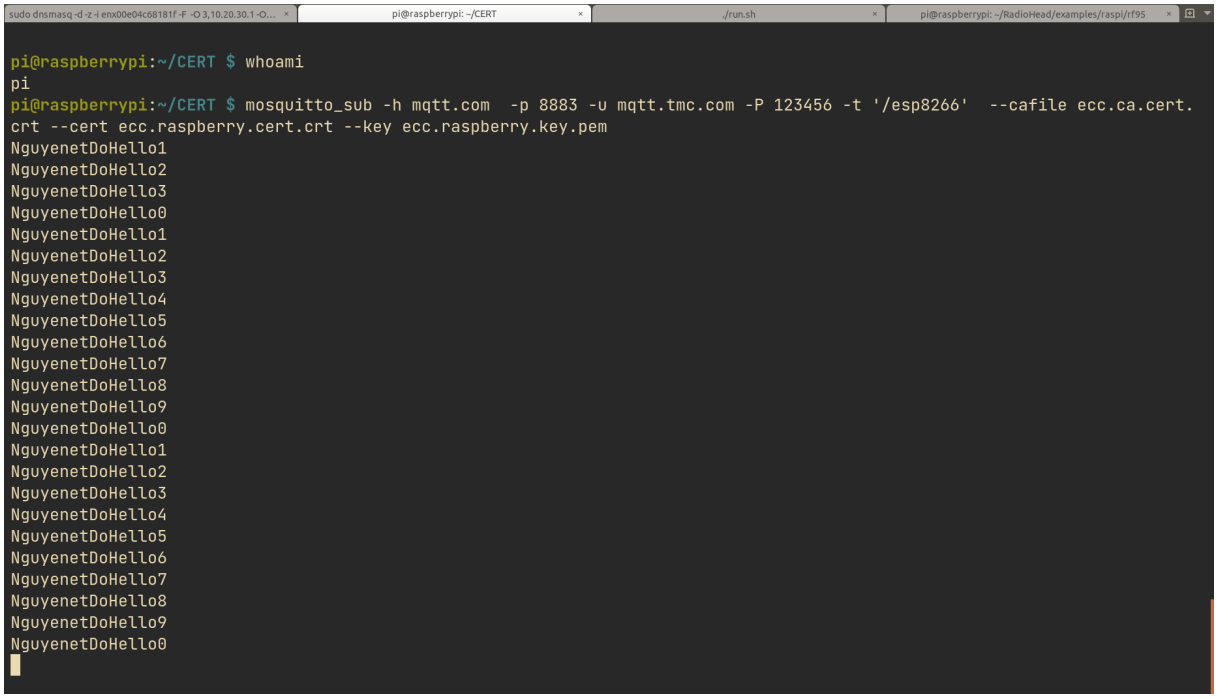
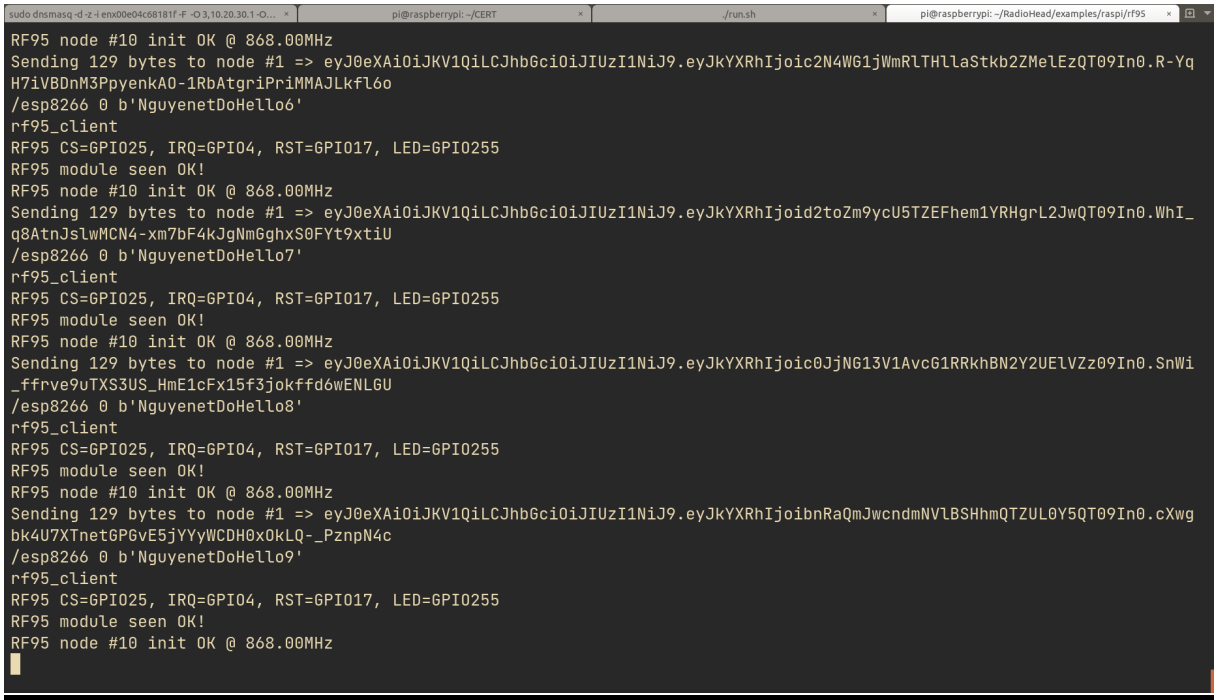
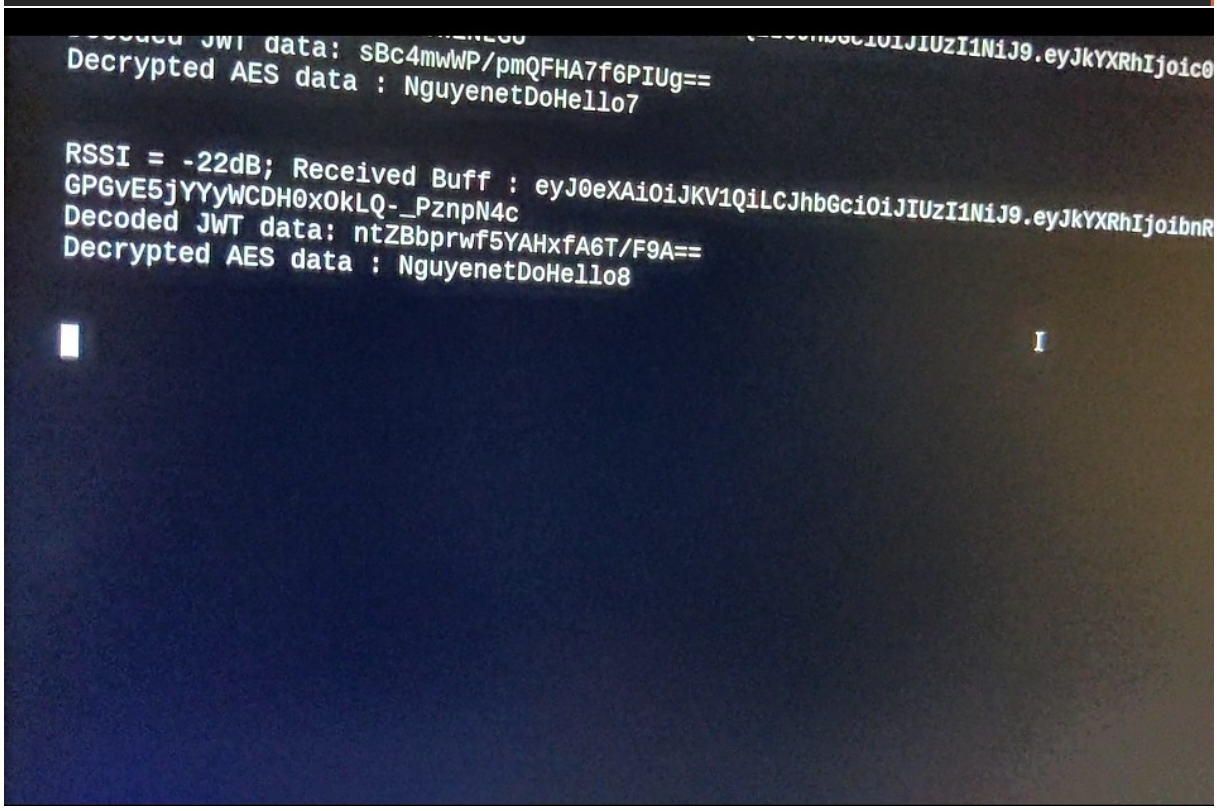


Figure 7: Subscribe a topic on Raspberry Pi

AES


sudo dnsmasq -d -z -i enx00e04c68181f -F -O 3,10.20.30.1 -O...
pi@raspberrypi: ~/CERT
./run.sh
pi@raspberrypi: ~/RadioHead/examples/raspi/rf95
RF95 node #10 init OK @ 868.00MHz
Sending 129 bytes to node #1 => eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoic2N4WG1jWmRlTHllaStkb2ZMelEzQT09In0.R-YqH7iVBDnM3PpyenkAO-1RbAtgriPriMMAJLkfl6o
/esp8266 0 b'NguyenetDoHello6'
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 to node #1 => eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoid2toZm9ycU5TZEFhem1YRHgrL2JwQT09In0.WhI_q8AtnJslwMCN4-xm7bF4kJgNmGghxS0FYt9xtiU
/esp8266 0 b'NguyenetDoHello7'
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 to node #1 => eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoic0JjNG13V1AvcG1RRkhBN2Y2UElVZz09In0.SnWi_ffrve9uTXS3US_HmE1cFx15f3jokffd6wENLGU
/esp8266 0 b'NguyenetDoHello8'
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 to node #1 => eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoibnRaQmJwcndmNVlBSHhmQTZUL0Y5QT09In0.cXwgbk4U7XTnetGPGvE5jYYyWCDH0xOkLQ-_PznpN4c
/esp8266 0 b'NguyenetDoHello9'
rf95_client
RF95 CS=GPIO25, IRQ=GPIO4, RST=GPIO17, LED=GPIO255
RF95 module seen OK!
RF95 node #10 init OK @ 868.00MHz


Decrypted AES data : NguyenetDoHello7
RSSI = -22dB; Received Buff : eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjoibnR
GPGvE5jYYyWCDH0xOkLQ-_PznpN4c
Decoded JWT data: ntZBbprwf5YAHxfA6T/F9A==
Decrypted AES data : NguyenetDoHello8

Conclusion and Future Work

In this project, we successfully implemented and achieved all the project requirement and also learn how to setup LoRa, MQTT server and how to use IoT with mongoose, ESP8266 and ATECC608 crypto chip. For the future work, we will optimize the code and improve the AES and JWT.

Appendix

Raspberry Pi & WiFi

```
kn@x1c7$ mkdir RASPI
kn@x1c7$ cd RASPI
kn@x1c7$ wget https://downloads.raspberrypi.org/raspbian_lite_latest
kn@x1c7$ unzip raspbian_lite_latest
```

First we need to check available loop device with `losetup`, in my computer it is loop25.

```
kn@x1c7$ sudo losetup -P /dev/loop25 2019-09-26-raspbian-buster-lite.img
kn@x1c7$ sudo mount /dev/loop25p2 /mnt
kn@x1c7$ mkdir client
kn@x1c7$ sudo rsync -xa --progress /mnt/ client/
kn@x1c7$ sudo umount /mnt
```

Read boot partition from image

```
kn@x1c7$ mkdir boot
kn@x1c7$ sudo mount /dev/loop25p1 /mnt
kn@x1c7$ cp -r /mnt/* boot/
kn@x1c7$ sudo umount /mnt
```

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

```
kn@x1c7$ sudo apt update
kn@x1c7$ sudo apt install nfs-kernel-server rpcbind
```

Configuring the NFS share in the `/etc/exports` file:

```
kn@x1c7$ 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) # might cause nfs to fail to start
#
# 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/kn/RASPI/client *(rw,sync,no_subtree_check,no_root_squash)
/home/kn/RASPI/boot *(rw,sync,no_subtree_check,no_root_squash)
```

Enable the NFS and RPCBind service:

```
kn@x1c7$ sudo systemctl enable nfs-kernel-server
kn@x1c7$ sudo systemctl enable rpcbind
```

Start the services:

```
kn@x1c7$ sudo systemctl start nfs-kernel-server
kn@x1c7$ sudo systemctl start rpcbind
```

If you modify the configuration of an export, you must restart the NFS service:

To see the mount points offered by an NFS server:

```
kn@x1c7$ showmount -e 127.0.0.1
Export list for 127.0.0.1:
/home/kn/RASPI/boot *
/home/kn/RASPI/client *
```

Edit mount point

Edit mount point for filesystem Raspberry Pi in `RASPI/boot/cmdline.txt`

```
kn@x1c7$ cat RASPI/boot/cmdline.txt
dwc_otg.lpm_enable=0 console=serial0,115200 console=tty1 root=/dev/nfs
    nfsroot=10.20.30.1:/home/kn/RASPI/client,vers=3 rw ip=dhcp rootwait
    elevator=deadline
```

Edit mount point for boot in RASPI/client/etc/fstab

```
kn@x1c7$ cat RASPI/client/etc/fstab
proc            /proc           proc        defaults          0         0
10.20.30.1:/home/kn/RASPI/boot /boot nfs rsize=8192,wsize=8192,timeo=14,intr,
    noauto,x-systemd.automount 0 0
```

Enable SSH for Raspberry Pi

```
kn@x1c7$ cat RASPI/client/lib/systemd/system/sshswitch.service
[Unit]
Description=Turn on SSH if /boot/ssh is present
#ConditionPathExistsGlob=/boot/ssh{,.txt}
After=regenerate_ssh_host_keys.service
[Service]
Type=oneshot
ExecStart=/bin/sh -c "update-rc.d ssh enable && invoke-rc.d ssh start && rm -
    f /boot/ssh; rm -f /boot/ssh.txt"
[Install]
WantedBy=multi-user.target
```

Setup Dnsmasq, DHCP using PC Ethernet connection:

kn@x1c7:$ IF=enx00e04c68181f kn@x1c7:$ PREFIX=10.20.30

```
kn@x1c7:$ sudo sysctl -w net.ipv4.ip_forward=1
kn@x1c7:$ sudo ip link set dev $IF down
kn@x1c7:$ sudo ip link set dev $IF address aa:aa:aa:aa:aa:aa
kn@x1c7:$ sudo ip link set dev $IF up
kn@x1c7:$ sudo ip address add dev $IF $PREFIX.1/24
kn@x1c7:$ sudo iptables -t nat -A POSTROUTING -s $PREFIX.0/24 -j MASQUERADE
```

In case IP address does not hold and keep changing. Configure static IP in file: /etc/network/ interfaces

```
kn@x1c7$ cat /etc/network/interfaces
...
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

```

```
auto enx00e04c68181f
iface enx00e04c68181f inet static
        address 10.20.30.1
        netmask 255.255.255.0
        dns-nameservers 8.8.8.8
        dns-nameservers 8.8.4.4
```

and config nameserver in `/etc/resolv.conf` for dnsmasq to use.

```
kn@x1c7$ cat /etc/resolv.conf
...
# See man:systemd-resolved.service(8) for details about the supported modes of
# operation for /etc/resolv.conf.

#options edns0
nameserver 127.0.0.53
```

Start Dnsmasq:

```
kn@x1c7$
sudo dnsmasq -d -z -i enx00e04c68181f -F 10.20.30.100,10.20.30.150,255.255.255.0,12h -O 3,10.20.30.1 -O 6,8.8.8.8,8.8.4.4 --pxe-service=0,"Raspberry Pi Boot" --enable-tftp --tftp-root=/home/kn/RASPI/boot
```

Connect Rapsberry Pi to PC, and wait for DHCP handshake from dnsmasq:

```
kn@x1c7$
...
dnsmasq-dhcp: DHCPDISCOVER(enx00e04c68181f) b8:27:eb:73:ff:53
dnsmasq-dhcp: DHCPOFFER(enx00e04c68181f) 10.20.30.129 b8:27:eb:73:ff:53
dnsmasq-tftp: file /home/kn/RASPI/boot/bootsig.bin not found
dnsmasq-tftp: sent /home/kn/RASPI/boot/bootcode.bin to 10.20.30.129
dnsmasq-dhcp: DHCPDISCOVER(enx00e04c68181f) b8:27:eb:73:ff:53
dnsmasq-dhcp: DHCPOFFER(enx00e04c68181f) 10.20.30.129 b8:27:eb:73:ff:53
dnsmasq-tftp: file /home/kn/RASPI/boot/2073ff53/start.elf not found
dnsmasq-tftp: file /home/kn/RASPI/boot/autoboot.txt not found
dnsmasq-tftp: sent /home/kn/RASPI/boot/config.txt to 10.20.30.129
dnsmasq-tftp: file /home/kn/RASPI/boot/recovery.elf not found
dnsmasq-tftp: sent /home/kn/RASPI/boot/start.elf to 10.20.30.129
dnsmasq-tftp: sent /home/kn/RASPI/boot/fixup.dat to 10.20.30.129
dnsmasq-tftp: file /home/kn/RASPI/boot/recovery.elf not found
```

```
dnsmasq-tftp: sent /home/kn/RASPI/boot/config.txt to 10.20.30.129
dnsmasq-tftp: file /home/kn/RASPI/boot/dt-blob.bin not found
dnsmasq-tftp: file /home/kn/RASPI/boot/recovery.elf not found
dnsmasq-tftp: sent /home/kn/RASPI/boot/config.txt to 10.20.30.129
dnsmasq-tftp: file /home/kn/RASPI/boot/bootcfg.txt not found
dnsmasq-tftp: sent /home/kn/RASPI/boot/bcm2710-rpi-3-b.dtb to 10.20.30.129
dnsmasq-tftp: sent /home/kn/RASPI/boot/config.txt to 10.20.30.129
dnsmasq-tftp: sent /home/kn/RASPI/boot/cmdline.txt to 10.20.30.129
dnsmasq-tftp: file /home/kn/RASPI/boot/recovery8.img not found
dnsmasq-tftp: file /home/kn/RASPI/boot/recovery8-32.img not found
dnsmasq-tftp: file /home/kn/RASPI/boot/recovery7.img not found
dnsmasq-tftp: file /home/kn/RASPI/boot/recovery.img not found
dnsmasq-tftp: file /home/kn/RASPI/boot/kernel8-32.img not found
dnsmasq-tftp: error 0 Early terminate received from 10.20.30.129
dnsmasq-tftp: failed sending /home/kn/RASPI/boot/kernel8.img to 10.20.30.129
dnsmasq-tftp: file /home/kn/RASPI/boot/armstub8-32.bin not found
dnsmasq-tftp: error 0 Early terminate received from 10.20.30.129
dnsmasq-tftp: failed sending /home/kn/RASPI/boot/kernel7.img to 10.20.30.129
dnsmasq-tftp: sent /home/kn/RASPI/boot/kernel7.img to 10.20.30.129
dnsmasq-dhcp: DHCPDISCOVER(enx00e04c68181f) b8:27:eb:73:ff:53
dnsmasq-dhcp: DHCPOFFER(enx00e04c68181f) 10.20.30.129 b8:27:eb:73:ff:53
dnsmasq-dhcp: DHCPREQUEST(enx00e04c68181f) 10.20.30.129 b8:27:eb:73:ff:53
dnsmasq-dhcp: DHCPACK(enx00e04c68181f) 10.20.30.129 b8:27:eb:73:ff:53
```

Wifi AP

```
kn@x1c7$ ssh pi@10.20.30.129
```

Run `raspi-config` and config the wifi country to avoid rfkill block wifi.

```
pi@raspberrypi:~ $ sudo rfkill unblock all
```

Install dnsmasq and hostapd for wifi hotspot in RaspPi

```
pi@raspberrypi$ sudo apt update
pi@raspberrypi$ sudo apt-get install hostapd dnsmasq
```

Config Dnsmasq/`etc`/`dnsmasq.conf`

```
sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.bak
sudo vim /etc/dnsmasq.conf
```

```
3
4  interface=wlan0
5  dhcp-range=192.168.4.100,192.168.4.120,255.255.255.0,12h
6  domain=wlan
7  address=/mqtt.com/192.168.4.1
```

Config hostapd `/etc/hostapd/hostapd.conf` for Wifi AP

```
1  pi@raspberrypiberrypi:~ $ cat /etc/hostapd/hostapd.conf
2  country_code=FR
3  interface=wlan0
4  ssid=raspberryWifi01
5  hw_mode=g
6  channel=7
7  wmm_enabled=0
8  macaddr_acl=0
9  auth_algs=1
10 ignore_broadcast_ssid=0
11 wpa=2
12 wpa_passphrase=raspberry01
13 wpa_key_mgmt=WPA-PSK
14 wpa_pairwise=TKIP
15 rsn_pairwise=CCMP
```

Config ipv4 forwarding

```
1  $ sudo vim /etc/sysctl.conf
2
3  net.ipv4.ip_forward=1     #uncomment this line
```

Static and manual config

Add nameserver in resolvconf.conf for dnsmasq

```
1  pi@raspberrypiberrypi:~ $ cat /etc/resolvconf.conf
2  # Configuration for resolvconf(8)
3  # See resolvconf.conf(5) for details
4
5  resolv_conf=/etc/resolv.conf
6  # If you run a local name server, you should uncomment the below line and
7  # configure your subscribers configuration files below.
```

```
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@raspberrypiberrypi:~ $ 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 192.168.4.1
    netmask 255.255.255.0
    gateway 192.168.4.1
```

Enable hostapd

```
pi@raspberrypi$ sudo systemctl unmask hostapd
pi@raspberrypi$ sudo systemctl enable hostapd
pi@raspberrypi$ sudo systemctl start hostapd
pi@raspberrypi$ sudo systemctl enable dnsmasq
```

ECC encryption: keys and certificates

Generation of private keys for the CA, the server and the client.

```
openssl ecparam -out ecc.ca.key.pem -name prime256v1 -genkey
openssl ecparam -out ecc.raspberry.key.pem -name prime256v1 -genkey
openssl ecparam -out ecc.esp8266.key.pem -name prime256v1 -genkey
```

Generation self-signed certificate of the CA which will be used to sign those of

the server and client

```
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
```

Generation and signing of the certificate for the server (Raspberry 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=
    mqtt.com" -reqexts ext -sha256 -key ecc.raspberry.key.pem -text -out ecc.
    raspberry.csr.pem

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.crt -addtrust
    clientAuth
```

Generating and signing the certificate for the client - ESP8266

```
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

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.crt -addtrust
    clientAuth
```

MQTT Server

First to fix network:

```
sudo -s
echo "nameserver 8.8.8.8" >> /etc/resolv.conf
chmod 644 /etc/resolv.conf
exit
ping google.com
```

or check ip tables or ip route on the Raspberry Pi

Installation of the MQTT server packages

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

Activate the protection of access to the server by a password To activate the protection of access to the MQTT server by password, we add in the file `/etc/mosquitto/mosquitto.conf`

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

listener 8883
cafile /etc/mosquitto/ca_certificates/ecc.ca.cert.crt
certfile /etc/mosquitto/certs/ecc.raspberry.cert.crt
keyfile /etc/mosquitto/certs/ecc.raspberry.key.pem
require_certificate true
```

NOTE for version >2.0, you need to put Pi cert to /etc/mosquitto/certs and also CA cert in /etc/mosquitto/ca_certificates

It will enable the user authentication by password and certificate for mosquitto.

Then we use `mosquitto_passwd` to generate the user `mqtt.tmc.com` in file `mosquitto_passwd`:

```
sudo mosquitto_passwd -c /etc/mosquitto/mosquitto_passwd mqtt.tmc.com
```

Restart Mosquitto service

```
sudo systemctl restarts mosquitto
```

ESP8266: Mongoose OS + ATECC608

```
kn@latop$
sudo add-apt-repository ppa:mongoose-os/mos
sudo apt-get update
sudo apt-get install mos
```

To generate a flash, install docker and set the execution right

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

MQTT client application

Convert Cert to use in Mongoose OS

```
openssl x509 -in ecc.esp8266.cert.crt -out ecc.esp8266.cert.pem -outform PEM
openssl x509 -in ecc.ca.cert.crt -out ecc.ca.cert.pem -outform PEM
```

Install new app and config file `mos.yml` like following:

```
kn@x1c7$
$ git clone https://github.com/mongoose-os-apps/empty my-app
$ cd my-app
$ cat mos.yml

author: mongoose-os
description: A Mongoose OS app skeleton
version: 2.19.1
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

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"}]
  - ["wifi.ap.enable", "b", false, {title: "Enable"}]
  - ["wifi.sta.enable", "b", true, {title: "Connect to existing WiFi"}]
```

```
27    - ["wifi.sta.ssid", "raspberryWifi01"]
28    - ["wifi.sta.pass", "raspberry01"]
29    - ["mqtt.enable", true]
30    - ["mqtt.server", "mqtt.com:8883"]
31    - ["mqtt.user", "mqtt.tmc.com"]
32    - ["mqtt.pass", "123456"]
33    - ["mqtt.ssl_ca_cert", "ecc.ca.cert.pem"]
34    - ["mqtt.ssl_cert", "ecc.esp8266.cert.pem"]
35    - ["mqtt.ssl_key", "ATCA:0"]
36
37  cdefs:
38    MG_ENABLE_MQTT: 1
39    # MG_ENABLE_SSL: 1
40
41  build_vars:
42    # Override to 0 to disable ATECCx08 support.
43    # Set to 1 to enable ATECCx08 support.
44    # MGOS_MBEDTLS_ENABLE_ATCA: 0
45    MGOS_MBEDTLS_ENABLE_ATCA: 1
46
47
48  libs:
49    - origin: https://github.com/mongoose-os-libs/ca-bundle
50    - origin: https://github.com/mongoose-os-libs/boards
51    - origin: https://github.com/mongoose-os-libs/rpc-service-config
52    - origin: https://github.com/mongoose-os-libs/rpc-mqtt
53    - origin: https://github.com/mongoose-os-libs/rpc-uart
54    - origin: https://github.com/mongoose-os-libs/wifi
55    - origin: https://github.com/mongoose-os-libs/rpc-service-i2c
56    - origin: https://github.com/mongoose-os-libs/mbedtls
57    - origin: https://github.com/mongoose-os-libs/atca
58    - origin: https://github.com/mongoose-os-libs/rpc-service-fs
59    - origin: https://github.com/mongoose-os-libs/rpc-service-atca
60
61  # Used by the mos tool to catch mos binaries incompatible with this file
        format
62  manifest_version: 2017-05-18
```

Client ESP8266 code

```
1  kn@x1c7:my-app/$
2  $ cd src
```

```
$ cat main.c
#include <stdio.h>
#include "mgos.h"
#include "mgos_mqtt.h"
int i = 0;
static void my_timer_cb(void *arg) {
  if (i == 26) i = 0;
  char message[] = {'N', 'g', 'u', 'y' , 'e' , 'n', 'e', 't', 'D', 'o', 'H',
      'e', 'l', 'l', 'o', '0'+i};
  i++;
  mgos_mqtt_pub("/esp8266", message, 16, 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;
}
```

Build and Flash firmware for ESP8266

```
kn@x1c7:my-app/$
mos build --local --arch esp8266
mos flash
```

Install private key into ATECC608:

```
kn@x1c7:my-app/$
$ openssl rand -hex 32 > slot4.key
$ mos -X atca-set-key 4 slot4.key --dry-run=false

AECC508A rev 0x5000 S/N 0x012352aad1bbf378ee, config is locked, data is
    locked
Slot 4 is a non-ECC private key slot
SetKey successful.
```

Then add certificate key to ATECC608

```
kn@x1c7:my-app/$
$ mos -X atca-set-key 0 ecc.esp8266.key.pem --write-key=slot4.key --dry-run=
    false

```

```
Using port /dev/ttyUSB0
ATECC508A rev 0x5000 S/N 0x0123fb976eb9b4f3ee, config is locked, data is
    locked
Slot 0 is a ECC private key slot
Parsed EC PRIVATE KEY
Data zone is locked, will perform encrypted write using slot 4 using slot4.
    key
SetKey successful.
```

You can also install the certificate in the "filesystem" of the ESP8266 without recompilation:

```
mos put ecc.ca.cert.pem
mos put ecc.esp8266.cert.pem
```

Start `mos console` to see the LOG from esp8266

AES

For the use of the GPIOs pins and the SPI bus you will need the bcm2835 library:

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

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

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

and then we modify the two source files: "rf95_server.cpp" and "rf95_client.cpp", to select the dragino as the project description.

For the LoRa Client Python script:

```
#!/bin/python3
```

```
import paho.mqtt.client as mqtt
import os, ssl, json, binascii, base64, jwt, subprocess
from urllib.parse import urlparse
from Crypto import Random
from Crypto.Cipher import AES

cafile ="/home/pi/CERT/ecc.ca.cert.crt"
cert = "/home/pi/CERT/ecc.raspberry.cert.crt"
key = "/home/pi/CERT/ecc.raspberry.key.pem"

def encrypt(message, passphrase):
    aes = AES.new(passphrase, AES.MODE_CBC, '0123456789abcdef')
    return base64.b64encode(aes.encrypt(message))

def on_message(client, obj, msg):
    print(msg.topic + " " + str(msg.qos) + " " + str(msg.payload))
    data=encrypt(msg.payload,"VietNamVoDich123")
    command ="sudo ./rf95_client "+jwt.encode( {'data':data.decode('utf-8')
        }, "MQTT", algorithm='HS256')
    os.system("%s"%(command))

mqttc = mqtt.Client()

# Assign event callbacks
mqttc.on_message = on_message

url_str = os.environ.get('CLOUDMQTT_URL', 'mqtt://mqtt.com:8883//esp8266')
url = urlparse(url_str)
topic = url.path[1:] or '/esp8266'

# Connect
mqttc.username_pw_set("mqtt.tmc.com", "123456")
mqttc.tls_set(ca_certs=cafile, certfile=cert, keyfile=key, cert_reqs=ssl.
    CERT_REQUIRED, tls_version=ssl.PROTOCOL_TLS, ciphers=None)
mqttc.connect(url.hostname, url.port)

# Start subscribe, with QoS level 0
mqttc.subscribe(topic, 0)

rc = 0
while rc == 0:
    rc = mqttc.loop()
```

For the LoRa Server Python script:

```python
#!/bin/python3
import jwt, subprocess, sys, binascii,os, ssl, base64
from Crypto.Cipher import AES
data = sys.argv[1]
#print("Received encoded JWT: " +data)
encoded = ""
try:
        encoded = jwt.decode(data, "MQTT")
        print("AES encrypted data: " + encoded['data'])
except:
        print("Error decoding JWT")
        exit(1)
decryption_suite = AES.new('VietNamVoDich123', AES.MODE_CBC, '0123456789abcdef')
try:
        plain_text = decryption_suite.decrypt(base64.b64decode(encoded['data']))
        print("AES Decrypted data : " + plain_text.decode('utf-8'))
except:
        print("Error AES decryption")
        exit(1)
```

For the RF95 server:

```
// rf95_server.cpp
//
// Example program showing how to use RH_RF95 on Raspberry Pi
// Uses the bcm2835 library to access the GPIO pins to drive the RFM95 module
// Requires bcm2835 library to be already installed
// http://www.airspayce.com/mikem/bcm2835/
// Use the Makefile in this directory:
// cd example/raspi/rf95
// make
// sudo ./rf95_server
//
// Contributed by Charles-Henri Hallard based on sample RH_NRF24 by Mike Poublon

#include <bcm2835.h>
#include <stdio.h>
```

```
16 #include <signal.h>
17 #include <unistd.h>
18 #include <iostream>
19 #include <string>
20 #include <stdexcept>
21 #include <RH_RF69.h>
22 #include <RH_RF95.h>
23
24 // define hardware used change to fit your need
25 // Uncomment the board you have, if not listed
26 // uncommment custom board and set wiring tin custom section
27
28 // LoRasPi board
29 // see https://github.com/hallard/LoRasPI
30 //#define BOARD_LORASPI
31
32 // iC880A and LinkLab Lora Gateway Shield (if RF module plugged into)
33 // see https://github.com/ch2i/iC880A-Raspberry-PI
34 //#define BOARD_IC880A_PLATE
35
36 // Raspberri PI Lora Gateway for multiple modules
37 // see https://github.com/hallard/RPI-Lora-Gateway
38 //#define BOARD_PI_LORA_GATEWAY
39
40 // Dragino Raspberry PI hat
41 // see https://github.com/dragino/Lora
42 #define BOARD_DRAGINO_PIHAT
43
44 // Now we include RasPi_Boards.h so this will expose defined
45 // constants with CS/IRQ/RESET/on board LED pins definition
46 #include "../RasPiBoards.h"
47
48 // Our RFM95 Configuration
49 #define RF_FREQUENCY  868.00
50 #define RF_NODE_ID    1
51
52 // Create an instance of a driver
53 RH_RF95 rf95(RF_CS_PIN, RF_IRQ_PIN);
54 //RH_RF95 rf95(RF_CS_PIN);
55
56 //Flag for Ctrl-C
57 volatile sig_atomic_t force_exit = false;
```

```
58
59  void sig_handler(int sig)
60  {
61    printf("\n%s Break received, exiting!\n", __BASEFILE__);
62    force_exit=true;
63  }
64
65  //Main Function
66  int main (int argc, const char* argv[] )
67  {
68    unsigned long led_blink = 0;
69
70    signal(SIGINT, sig_handler);
71    printf( "%s\n", __BASEFILE__);
72
73    if (!bcm2835_init()) {
74      fprintf( stderr, "%s bcm2835_init() Failed\n\n", __BASEFILE__ );
75      return 1;
76    }
77
78    printf( "RF95 CS=GPIO%d", RF_CS_PIN);
79
80  #ifdef RF_LED_PIN
81    pinMode(RF_LED_PIN, OUTPUT);
82    digitalWrite(RF_LED_PIN, HIGH );
83  #endif
84
85  #ifdef RF_IRQ_PIN
86    printf( ", IRQ=GPIO%d", RF_IRQ_PIN );
87    // IRQ Pin input/pull down
88    pinMode(RF_IRQ_PIN, INPUT);
89    bcm2835_gpio_set_pud(RF_IRQ_PIN, BCM2835_GPIO_PUD_DOWN);
90    // Now we can enable Rising edge detection
91    bcm2835_gpio_ren(RF_IRQ_PIN);
92  #endif
93
94  #ifdef RF_RST_PIN
95    printf( ", RST=GPIO%d", RF_RST_PIN );
96    // Pulse a reset on module
97    pinMode(RF_RST_PIN, OUTPUT);
98    digitalWrite(RF_RST_PIN, LOW );
99    bcm2835_delay(150);
```

```
100   digitalWrite(RF_RST_PIN, HIGH );
101   bcm2835_delay(100);
102 #endif
103
104 #ifdef RF_LED_PIN
105   printf( ", LED=GPIO%d", RF_LED_PIN );
106   digitalWrite(RF_LED_PIN, LOW );
107 #endif
108
109   if (!rf95.init()) {
110     fprintf( stderr, "\nRF95 module init failed, Please verify wiring/module\
          n" );
111   } else {
112     // Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf =
          128chips/symbol, CRC on
113
114     // The default transmitter power is 13dBm, using PA_BOOST.
115     // If you are using RFM95/96/97/98 modules which uses the PA_BOOST
          transmitter pin, then
116     // you can set transmitter powers from 5 to 23 dBm:
117     //  driver.setTxPower(23, false);
118     // If you are using Modtronix inAir4 or inAir9,or any other module which
          uses the
119     // transmitter RFO pins and not the PA_BOOST pins
120     // then you can configure the power transmitter power for -1 to 14 dBm
          and with useRFO true.
121     // Failure to do that will result in extremely low transmit powers.
122     // rf95.setTxPower(14, true);
123
124
125     // RF95 Modules don't have RFO pin connected, so just use PA_BOOST
126     // check your country max power useable, in EU it's +14dB
127     rf95.setTxPower(14, false);
128
129     // You can optionally require this module to wait until Channel Activity
130     // Detection shows no activity on the channel before transmitting by
          setting
131     // the CAD timeout to non-zero:
132     //rf95.setCADTimeout(10000);
133
134     // Adjust Frequency
135     rf95.setFrequency(RF_FREQUENCY);
```

```
136
137     // If we need to send something
138     rf95.setThisAddress(RF_NODE_ID);
139     rf95.setHeaderFrom(RF_NODE_ID);
140
141     // Be sure to grab all node packet
142     // we're sniffing to display, it's a demo
143     rf95.setPromiscuous(true);
144
145     // We're ready to listen for incoming message
146     rf95.setModeRx();
147
148     printf( " OK NodeID=%d @ %3.2fMHz\n", RF_NODE_ID, RF_FREQUENCY );
149     printf( "Listening packet...\n" );
150
151     //Begin the main body of code
152     while (!force_exit) {
153
154 #ifdef RF_IRQ_PIN
155       // We have a IRQ pin ,pool it instead reading
156       // Modules IRQ registers from SPI in each loop
157
158       // Rising edge fired ?
159       if (bcm2835_gpio_eds(RF_IRQ_PIN)) {
160         // Now clear the eds flag by setting it to 1
161         bcm2835_gpio_set_eds(RF_IRQ_PIN);
162         //printf("Packet Received, Rising event detect for pin GPIO%d\n",
              RF_IRQ_PIN);
163 #endif
164
165         if (rf95.available()) {
166 #ifdef RF_LED_PIN
167           led_blink = millis();
168           digitalWrite(RF_LED_PIN, HIGH);
169 #endif
170           // Should be a message for us now
171           uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
172           uint8_t len  = sizeof(buf);
173           uint8_t from = rf95.headerFrom();
174           uint8_t to   = rf95.headerTo();
175           uint8_t id   = rf95.headerId();
176           uint8_t flags= rf95.headerFlags();;
```

```
177           int8_t rssi  = rf95.lastRssi();
178
179           if (rf95.recv(buf, &len)) {
180             printf("RSSI = %ddB; Received JWT Data: ", rssi);
181             printbuffer(buf, len);
182         printf("\n");
183         std::string convert;
184         convert.assign(buf, buf+len);
185
186         char buffer[512];
187         std::string result = "";
188         std::string str = "python3 aes_decrypt.py "+convert;
189         const char * command = str.c_str();
190         FILE* pipe = popen(command, "r");
191         if (!pipe) throw std::runtime_error("popen() failed!");
192         try {
193                 while (fgets(buffer, sizeof buffer, pipe) != NULL) {
194                         result += buffer;
195                     }
196             } catch (std::string const& chaine){
197                 pclose(pipe);
198                 throw;
199             }
200             std::cout << result << std::endl;
201             pclose(pipe);
202           } else {
203             Serial.print("receive failed");
204           }
205           printf("\n");
206         }
207
208 #ifdef RF_IRQ_PIN
209       }
210 #endif
211
212 #ifdef RF_LED_PIN
213       // Led blink timer expiration ?
214       if (led_blink && millis()-led_blink>200) {
215         led_blink = 0;
216         digitalWrite(RF_LED_PIN, LOW);
217       }
218 #endif
```

```
219       // Let OS doing other tasks
220       // For timed critical appliation you can reduce or delete
221       // this delay, but this will charge CPU usage, take care and monitor
222       bcm2835_delay(5);
223     }
224   }
225
226 #ifdef RF_LED_PIN
227   digitalWrite(RF_LED_PIN, LOW );
228 #endif
229   printf( "\n%s Ending\n", __BASEFILE__ );
230   bcm2835_close();
231   return 0;
232 }
```

For the RF95 Client:

```
1  // rf95_client.cpp
2  //
3  // Example program showing how to use RH_RF95 on Raspberry Pi
4  // Uses the bcm2835 library to access the GPIO pins to drive the RFM95 module
5  // Requires bcm2835 library to be already installed
6  // http://www.airspayce.com/mikem/bcm2835/
7  // Use the Makefile in this directory:
8  // cd example/raspi/rf95
9  // make
10 // sudo ./rf95_client
11 //
12 // Contributed by Charles-Henri Hallard based on sample RH_NRF24 by Mike
       Poublon
13
14 #include <bcm2835.h>
15 #include <stdio.h>
16 #include <signal.h>
17 #include <unistd.h>
18 #include <iostream>
19 #include <string>
20 #include <stdexcept>
21
22 #include <RH_RF69.h>
23 #include <RH_RF95.h>
24
```

```
25 // define hardware used change to fit your need
26 // Uncomment the board you have, if not listed
27 // uncommment custom board and set wiring tin custom section
28
29 // LoRasPi board
30 // see https://github.com/hallard/LoRasPI
31 //#define BOARD_LORASPI
32
33 // iC880A and LinkLab Lora Gateway Shield (if RF module plugged into)
34 // see https://github.com/ch2i/iC880A-Raspberry-PI
35 //#define BOARD_IC880A_PLATE
36
37 // Raspberri PI Lora Gateway for multiple modules
38 // see https://github.com/hallard/RPI-Lora-Gateway
39 //#define BOARD_PI_LORA_GATEWAY
40
41 // Dragino Raspberry PI hat
42 // see https://github.com/dragino/Lora
43 #define BOARD_DRAGINO_PIHAT
44
45 // Now we include RasPi_Boards.h so this will expose defined
46 // constants with CS/IRQ/RESET/on board LED pins definition
47 #include "../RasPiBoards.h"
48
49 // Our RFM95 Configuration
50 #define RF_FREQUENCY  868.00
51 #define RF_GATEWAY_ID 1
52 #define RF_NODE_ID    10
53
54 // Create an instance of a driver
55 RH_RF95 rf95(RF_CS_PIN, RF_IRQ_PIN);
56 //RH_RF95 rf95(RF_CS_PIN);
57
58 //Flag for Ctrl-C
59 volatile sig_atomic_t force_exit = false;
60
61 void sig_handler(int sig)
62 {
63   printf("\n%s Break received, exiting!\n", __BASEFILE__);
64   force_exit=true;
65 }
66
```

```
67  //Main Function
68  int main (int argc, const char* argv[] )
69  {
70    static unsigned long last_millis;
71    static unsigned long led_blink = 0;
72
73    signal(SIGINT, sig_handler);
74    printf( "%s\n", __BASEFILE__);
75
76    if (!bcm2835_init()) {
77      fprintf( stderr, "%s bcm2835_init() Failed\n\n", __BASEFILE__ );
78      return 1;
79    }
80
81    printf( "RF95 CS=GPIO%d", RF_CS_PIN);
82
83  #ifdef RF_LED_PIN
84    pinMode(RF_LED_PIN, OUTPUT);
85    digitalWrite(RF_LED_PIN, HIGH );
86  #endif
87
88  #ifdef RF_IRQ_PIN
89    printf( ", IRQ=GPIO%d", RF_IRQ_PIN );
90    // IRQ Pin input/pull down
91    pinMode(RF_IRQ_PIN, INPUT);
92    bcm2835_gpio_set_pud(RF_IRQ_PIN, BCM2835_GPIO_PUD_DOWN);
93  #endif
94
95  #ifdef RF_RST_PIN
96    printf( ", RST=GPIO%d", RF_RST_PIN );
97    // Pulse a reset on module
98    pinMode(RF_RST_PIN, OUTPUT);
99    digitalWrite(RF_RST_PIN, LOW );
100   bcm2835_delay(150);
101   digitalWrite(RF_RST_PIN, HIGH );
102   bcm2835_delay(100);
103 #endif
104
105 #ifdef RF_LED_PIN
106   printf( ", LED=GPIO%d", RF_LED_PIN );
107   digitalWrite(RF_LED_PIN, LOW );
108 #endif
```

```
109
110   if (!rf95.init()) {
111     fprintf( stderr, "\nRF95 module init failed, Please verify wiring/module\
          n" );
112   } else {
113     printf( "\nRF95 module seen OK!\r\n");
114
115 #ifdef RF_IRQ_PIN
116     // Since we may check IRQ line with bcm_2835 Rising edge detection
117     // In case radio already have a packet, IRQ is high and will never
118     // go to low so never fire again
119     // Except if we clear IRQ flags and discard one if any by checking
120     rf95.available();
121
122     // Now we can enable Rising edge detection
123     bcm2835_gpio_ren(RF_IRQ_PIN);
124 #endif
125
126     // Defaults after init are 434.0MHz, 13dBm, Bw = 125 kHz, Cr = 4/5, Sf =
          128chips/symbol, CRC on
127
128     // The default transmitter power is 13dBm, using PA_BOOST.
129     // If you are using RFM95/96/97/98 modules which uses the PA_BOOST
          transmitter pin, then
130     // you can set transmitter powers from 5 to 23 dBm:
131     //rf95.setTxPower(23, false);
132     // If you are using Modtronix inAir4 or inAir9,or any other module which
          uses the
133     // transmitter RFO pins and not the PA_BOOST pins
134     // then you can configure the power transmitter power for -1 to 14 dBm
          and with useRFO true.
135     // Failure to do that will result in extremely low transmit powers.
136     //rf95.setTxPower(14, true);
137
138     rf95.setTxPower(14, false);
139
140     // You can optionally require this module to wait until Channel Activity
141     // Detection shows no activity on the channel before transmitting by
          setting
142     // the CAD timeout to non-zero:
143     //rf95.setCADTimeout(10000);
144
```

```
145     // Adjust Frequency
146     rf95.setFrequency( RF_FREQUENCY );
147
148     // This is our Node ID
149     rf95.setThisAddress(RF_NODE_ID);
150     rf95.setHeaderFrom(RF_NODE_ID);
151
152     // Where we're sending packet
153     rf95.setHeaderTo(RF_GATEWAY_ID);
154
155     printf("RF95 node #%d init OK @ %3.2fMHz\n", RF_NODE_ID, RF_FREQUENCY );
156
157     last_millis = millis();
158
159     //Begin the main body of code
160     while (!force_exit) {
161
162       //printf( "millis()=%ld last=%ld diff=%ld\n", millis() , last_millis,
              millis() - last_millis );
163
164       // Send every 5 seconds
165       if ( millis() - last_millis > 5000 ) {
166         last_millis = millis();
167
168 #ifdef RF_LED_PIN
169         led_blink = millis();
170         digitalWrite(RF_LED_PIN, HIGH);
171 #endif
172
173         // Send a message to rf95_server
174
175         //printf("Sending %02d bytes to node #%d => ", len, RF_GATEWAY_ID );
176         //printbuffer(data, len);
177         //printf("\n" );
178         //rf95.send(data, len);
179         //rf95.waitPacketSent();
180      const char* msg1;
181      std::string str = argv[1];
182      msg1 = str.c_str();
183      size_t length = strlen(msg1) + 1;
184
185      const char* beg = msg1;
```

```
186       const char* end = msg1 + length;
187       uint8_t* msg2 = new uint8_t[length];
188
189       size_t i = 0;
190       for (; beg != end; ++beg, ++i){
191         msg2[i] = (uint8_t)(*beg);
192       }
193       uint8_t data[] = "hi";
194       uint8_t len = sizeof(data);
195
196       printf("Sending %02d bytes to node #%d => ", length, RF_GATEWAY_ID );
197       printbuffer(msg2, length);
198       printf("\n" );
199       rf95.send(msg2, length);
200       rf95.waitPacketSent();
201       exit(1);
202  /*
203          // Now wait for a reply
204          uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
205          uint8_t len = sizeof(buf);
206
207          if (rf95.waitAvailableTimeout(1000)) {
208            // Should be a reply message for us now
209            if (rf95.recv(buf, &len)) {
210              printf("got reply: ");
211              printbuffer(buf,len);
212              printf("\nRSSI: %d\n", rf95.lastRssi());
213            } else {
214              printf("recv failed");
215            }
216          } else {
217            printf("No reply, is rf95_server running?\n");
218          }
219  */
220
221       }
222
223  #ifdef RF_LED_PIN
224       // Led blink timer expiration ?
225       if (led_blink && millis()-led_blink>200) {
226         led_blink = 0;
227         digitalWrite(RF_LED_PIN, LOW);
```

```
228       }
229 #endif
230 
231       // Let OS doing other tasks
232       // Since we do nothing until each 5 sec
233       bcm2835_delay(100);
234     }
235   }
236 
237 #ifdef RF_LED_PIN
238   digitalWrite(RF_LED_PIN, LOW );
239 #endif
240   printf( "\n%s Ending\n", __BASEFILE__ );
241   bcm2835_close();
242   return 0;
243 }
```