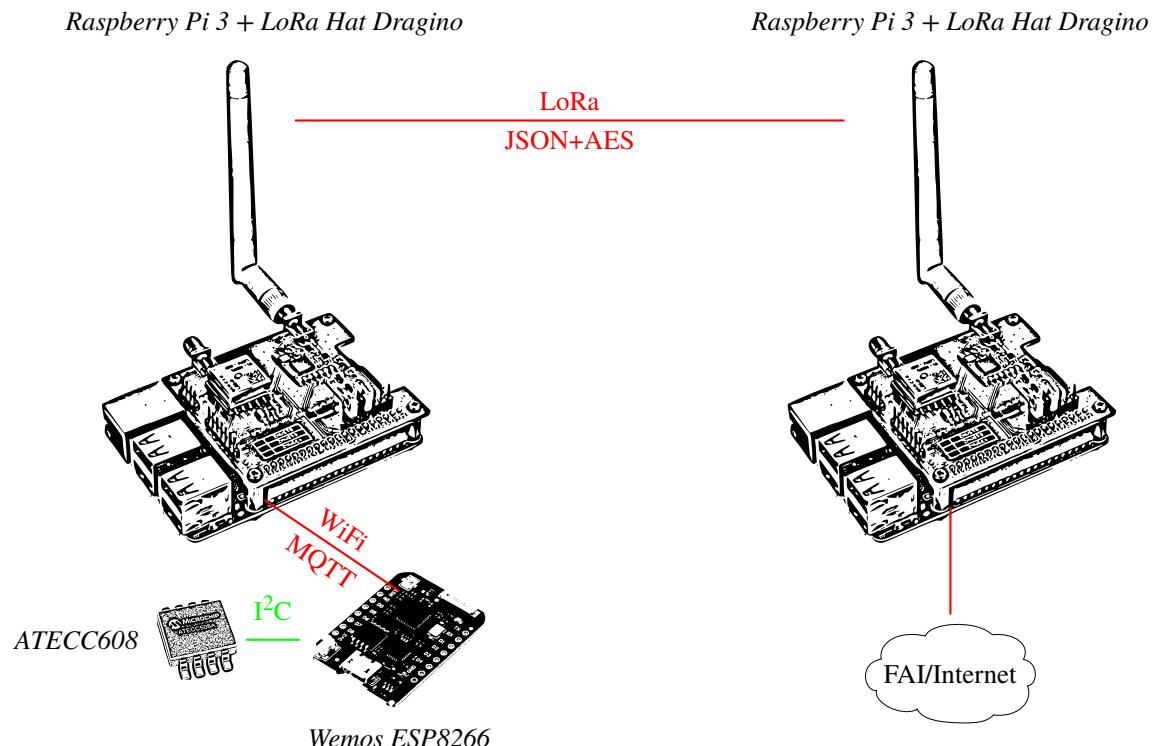


*IoT, LoRa, WiFi, MQTT, SSL, ATECC508, Mongoose OS, Raspberry Pi & ESP8266*

## ■ ■ ■ Présentation du projet



Le **but du projet** est de réaliser :

- un réseau de capteurs **connectés par WiFi** vers un concentrateur :
    - ◊ ils utilisent le **protocole MQTT** pour remonter des mesures vers le concentrateur au travers d'une connexion WiFi ;
    - ◊ chaque capteur correspond à un **ESP8266** intégré dans la carte de développement « *Wemos* » ;
    - ◊ un **Raspberry Pi** joue le rôle du concentrateur :
      - \* il exécute un « *broker* » MQTT : *logiciel mosquitto* ;
      - \* il sert de point d'accès WiFi : *logiciel hostapd* ;
  - chaque capteur exploite un circuit dédié à la manipulation de la **cryptographie sur courbe elliptique** ATECC608 connecté à l'ESP8266 par le bus I<sup>2</sup>C.
- Il réalise :
- ◊ l'authentification du serveur MQTT lors de la connexion en TLS ;
  - ◊ l'authentification du client auprès du serveur MQTT ;
  - le framework de développement est **Mongoose OS** permettant de :
    - ◊ programmer le système embarqué ESP8266 ;
    - ◊ disposer d'une implémentation de TLS ;
    - ◊ exploiter le composant ATECC608 pour réaliser les opérations de chiffrement/signature/vérification.
  - le concentrateur est relié vers une passerelle par l'utilisation des communications **LoRa** :
    - ◊ chaque Raspberry Pi est « *coiffé* » d'un dragino, intégrant un « *transceiver* » LoRa ainsi qu'un GPS ;
    - ◊ la communication d'une mesure est réalisée au travers de LoRa vers le Raspberry Pi connecté à Internet.

## ■■■ ■■■ ■■■ Raspberry Pi & WiFi

Vous installerez et configurerez sur le Raspberry Pi :

- ▷ pour la connexion WiFi des ESP8266 :
  - ◊ le point d'accès logiciel « hostapd » ;
  - ◊ le serveur DHCP permettant de prendre en charge les clients WiFi connectés avec dnsmasq ;
- ▷ le serveur « mosquitto » et la sécurité.

<https://www.rockingdlabs.com/exercises-experiments/ssl-client-certs-to-secure-mqtt>

## ■■■ ■■■ ■■■ Communications et sécurité

1. L'ESP8266 va se comporter comme un client MQTT en se connectant au serveur MQTT du Raspberry Pi jouant le rôle de concentrateur :

- ◊ il va transmettre à intervalle régulier une mesure, en tant que « *publisher* » ;  
*Ici, le message « Hello ! ».*
- ◊ sur le « *topic* » « /esp8266 » ;

*Vous pourrez contrôler que cela fonctionne en vous connectant sur le serveur MQTT en tant que « subscriber » :*

```
□ — xterm —
$ mosquitto_sub -h 10.20.30.1 -p 1883 -u toto -P secret -t '/esp8266'
Hello !
Hello !
```

- ◊ le Raspberry Pi pourra récupérer les mesures une par une, par exemple en sortie de la commande « *mosquitto\_sub* » ;
- ◊ **la connexion entre l'ESP8266 devra être faite en TLS avec authentification du serveur et du client par certificat/clé ECC.**

2. le Raspberry Pi concentrateur va transmettre la valeur par LoRa vers le Raspberry Pi assurant la passerelle Internet :

- ◊ la valeur sera **chiffrée par AES avec une clé partagée** entre les deux Raspberry Pis.

*Vous utiliserez les programmes « rf95\_client » et « rf95\_server » de démonstration du LoRa, pour les étendre et permettre le choix du message à envoyer correspondant à la valeur récupérée depuis MQTT*

- ◊ la valeur récupérée sera simplement affichée sur le Raspberry Pi recevant le message LoRa.  
*On pourrait imaginer de traiter les valeurs et permettre leur consultation sous forme d'un serveur Web ou bien en les transmettant sur un autre serveur MQTT situé sur Internet.*

*Une démonstration vous sera faite d'une application Android basée WebView embarquant une appli Web Angular+Client MQTT/Websockets permettant de récupérer les données.*

## ■■■ ■■■ Chiffrement ECC : clés et certificats

Pour la génération des différentes clés et certificats basés ECC :

```
□ — xterm —
$ openssl ecparam -out ecc.ca.key.pem -name prime256v1 -genkey
$ openssl ecparam -out ecc.key.pem -name prime256v1 -genkey

$ openssl req -config <(printf
" [req]\ndistinguished_name=dn\n[n[dn]]\n[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.pem

$ openssl req -config <(printf
" [req]\ndistinguished_name=dn\n[n[dn]]\n[n[ext]]\nbasicConstraints=CA:FALSE") -new -subj
"/C=FR/L=Limoges/O=TMC/OU=IOT/CN=esp8266" -reqexts ext -sha256 -key ecc.key.pem
-text -out ecc.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.csr.pem -text -out ecc.esp8266.pem -addtrust clientAuth
```

## ■ ■ ■ ■ ■ Mongoose OS : MQTT client et publication sécurisée par ECC

Pour l'utilisation de MQTT dans Mongoose OS :

- ▷ vous installerez une application vide Mongoose OS ;
- ▷ vous modifierez le fichier «`mos.yml`» avec le contenu suivant :

```
□ └── xterm └──
$ 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
filesystem:
- fs

build_vars:
    MGOSMBEDTLS_ENABLE_ATCA: 1

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", "p-fb.net:8883", {title: "MQTT server"}]
- ["mqtt.pub", "s", "/esp8266", {title: "Publish topic"}]
- ["mqtt.user", "s", "mqtt.tmc.com", {title: "User name"}]
- ["mqtt.pass", "s", "tmctmctmc", {title: "Password"}]
- ["mqtt.ssl_ca_cert", "s", "server_ca.pem", {title: "Verify server certificate using this CA bundle"}]
- ["mqtt.ssl_cert", "s", "ecc.crt.pem", {title: "Client certificate to present to the server"}]
- ["mqtt.ssl_key", "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: 2017-09-29
```

Dans le sous-répertoire «`fs`», correspondant aux fichiers installés dans l'ESP8266, vous installerez les certificats nécessaires au format pem : celui du serveur mosquitto et celui du client ESP8266.

*Vous n'oublierez pas de faire correspondre les noms des fichiers à la configuration donnée dans «`mos.yml`» (ici, les fichiers s'appellent «`server_ca.pem`» et «`server_ca.pem`»).*

## Le code source de l'application Mongoose OS :

```
└── xterm ━
$ cd src
$ cat main.c
#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;
}
```

## Pour compiler, flasher l'ESP8266 :

```
└── xterm ━
$ mos build --local --arch esp8266
$ mos flash; mos wifi TMC tmctmctmc; mos console
```

Ici, le réseau WiFi est de SSID « TMC » et le mot de passe « tmctmctmc ».

### Aide

- ▷ les certificats du serveur MQTT et du client MQTT/Mongoose OS doivent avoir la même CA pour faciliter l'authentification du serveur auprès du client et vice-versa ;
- ▷ le CN, « *Common Name* », certificat du serveur MQTT doit correspondre au nom DNS du serveur ;  
*Avec dnsmasq vous pouvez fournir une association IP/DNS :*

```
sudo dnsmasq -d -z -i $INTERFACE -F 10.20.30.100,10.20.30.150,255.255.255.0 \
-O 6,10.20.30.1,8.8.8.8 -A /mqtt.com/10.3.30.254
```

*Ici, le client reçoit le serveur DNS où réside le serveur dnsmasq qui va fournir l'association mqtt.com⇒10.3.30.254*

- ▷ le certificat du client pour être reconnu par Mongoose OS doit être entouré des lignes exactes :

```
-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----
```

- ▷ le projet est possible...je l'ai fait !

### Travail et évaluation

- le travail est à faire au minimum en **binôme**.
- une archive et/ou un github sera remis à bonnefoi@unilim.fr par l'intermédiaire d'un courrier (lien github) ou d'un FileSender (archive).
- l'évaluation tiendra compte :
  - ◊ des mises en place des **communications** :
    - \* WiFi et MQTT :
      - ▷ de l'ESP8266 vers le Raspberry Pi ;
      - ▷ la connexion du client MQTT de l'ESP8266 vers le serveur MQTT du Raspberry Pi ;
    - \* LoRa :
      - ▷ entre les deux Raspberry ;
  - ◊ du **chiffrement** :
    - \* par courbe elliptique tirant parti de l'ATECC608 :
      - ▷ authentification du serveur MQTT depuis l'ESP8266 ;
      - ▷ authentification du client ESP8266 auprès du serveur MQTT ;
    - \* par AES :
      - ▷ échange des valeurs entre les deux Raspberry Pis par LoRa.
      - ▷ utilisation du chiffrement AES ;
  - ◊ du « rapport » :
    - \* capture de trafic chiffré en sortie du Raspberry Pi concentrateur ;
    - \* capture des messages en sortie de l'ESP8266 ;
    - \* explication du travail réalisé et des programmes/scripts réalisés ;
    - \* copies d'écran ;
  - ◊ une vidéo sur **Youtube** avec une démo.