# Internet of Things (IoT) / Internet des Objets (IdO)

# Application Protocols : MQTT

Gilles Menez
University Nice Côte d'Azur
email : menez@unice.fr
www : www.i3s.unice.fr/~menez

March 28, 2024: V 1.0

# Table of Contents

## List of Figures

## What you should learn ...

1. We use another application protocol : MQTT
2. The paradigm is different !

## Two remarks

Si vous rencontrez des problèmes de stabilité, il se peut que cela soit dû à la cohabitation du serveur http asynchrone et du client mqtt.

- Essayez de revenir en synchrone ... quitte à simplifier le nombre d'URLs et de routes.

When the code is given ... YOU ARE SUPPOSED TO study IT !

not only to make it run ... I already KNOW it is running ;-)

---

Sections titled "TODO : The final step !" explain what you have to experiment or create by yourself from examples.

# 1 Mosquitto

## 1.1 Installation

On pourrait n'installer qu'un client MQTT sur votre machine et utiliser un des nombreux brokers disponibles sur Internet.

Mais pour réduire votre dépendance à la disponibilité d'un réseau Internet et comme c'est assez simple à faire, vous pouvez installer client **et** serveur/broker sur votre machine.

https://mosquitto.org/download/

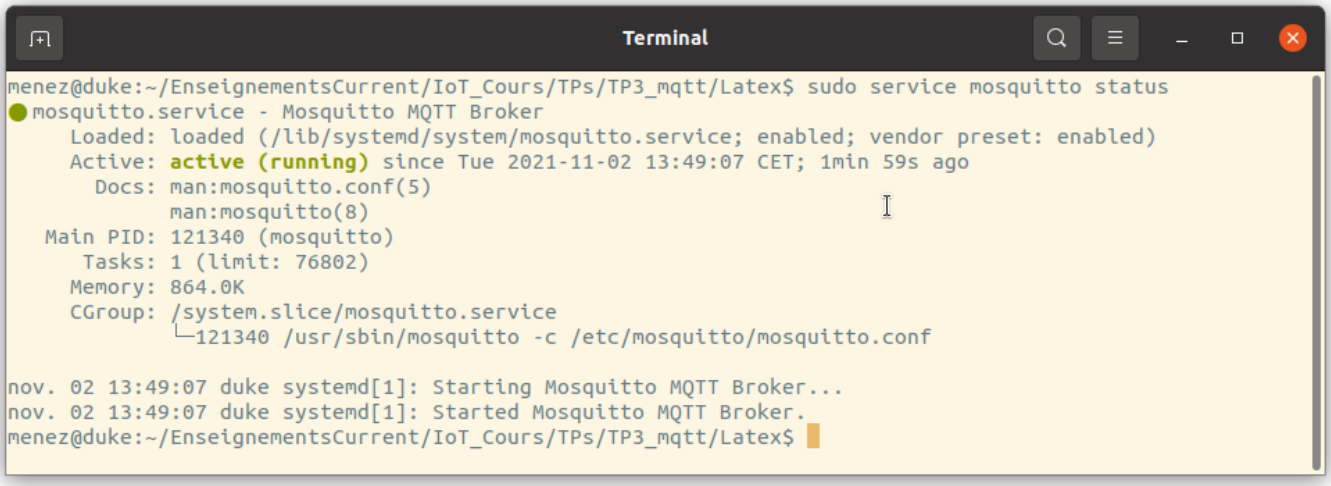
*** Vous pourriez aussi passer par un container ... moi j'apt-get ;-) ***

➢ Le broker : `sudo apt-get install mosquitto`

➢ Le client (pub, sub et passw) : `sudo apt-get install mosquitto-clients`

## 1.2 Ca tourne ?

Sous Linux, vous pouvez vérifier que le service "tourne" : `sudo service mosquitto status`




et si il le faut, éventuellement manipuler le service (post reconfiguration ?) :

```
sudo service mosquitto start
sudo service mosquitto stop
sudo service mosquitto restart
```

Vous remarquez sur la console que le serveur MQTT **s'exécute "avec" le fichier de configuration** :

/etc/mosquitto/mosquitto.conf

## 1.3 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 /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

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

include_dir /etc/mosquitto/conf.d
```

Ce fichier

- ➢ active la persistence, c'est à dire la sauvegarde des données reçues / envoyées en MQTT.
  On précise le répertoire ("/var/lib/mosquitto") contenant le "persistence_file" ("mosquitto.db")
- ➢ puis "appelle" (include) le fichier personnalisé (par moi au plus simple) suivant :

```
#File : /etc/mosquitto/conf.d/gm.conf
#  Changement  => sudo service mosquitto restart
#the default config will only bind to localhost as a move to
#a more secure default posture compared to previous MQTT versions.

#Section Extra Listener  :
#Running the broker with a listener defined will bind by default to
#0.0.0.0 / :: and so will be accessible from any interface.
#It is still possible to bind to a specific address/interface !.
listener 1883
protocol mqtt

#Section Connection :
#By default it will also only allow anonymous connections (without
#username/password) from localhost, to allow anonymous from REMOTE
#add:
allow_anonymous true
```

**Quelques remarques :**

- ➢ La ligne listener 1883 crée un nouveau "listener" qui va écouter sur le port 1883 pour toutes les interfaces réseaux (et non pas seulement sur l´hôte local (localhost) comme par défaut)

➢ On indique avec le paramètre protocol qu'il s'agit de mqtt.

➢ On est loin d'être "secure" avec ce type de configuration !

On s'essayera de s'occuper de ce problème plus tard.

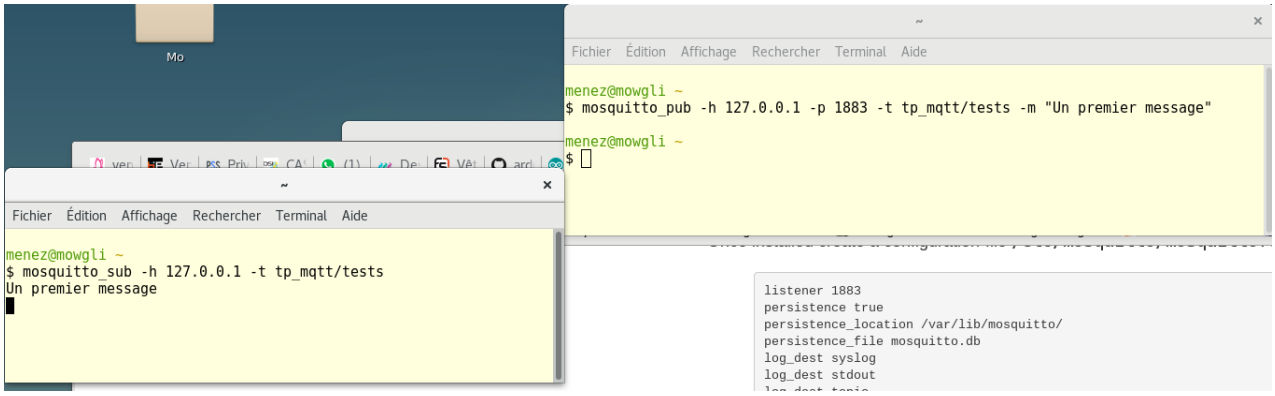
## 1.4 Test fonctionnel du client et du serveur

On utilise le serveur que vous venez d'installer sur votre machine (127.0.0.1) :

① Un client souscrit à un topic `"tpmqtt_tests"`


② Un autre publie un message sur ce même topic.

③ Le broker (sur le localhost) fait immédiatement suivre




Les options de verbose (-v) et debug (-d) peuvent apporter des informations . . . en cas de problème . . . cela va venir !

## 1.5 Dump du serveur

Avec la commande qui suit, vous aurez un dump des méta-informations contenues dans le serveur (broker) :

```
mosquitto_sub -h localhost -v -t \$SYS/#
```

- ✔ $SYS/broker/load/bytes/received :

  The total number of bytes received since the broker started.

- ✔ $SYS/broker/load/bytes/sent :

  The total number of bytes sent since the broker started.

- ✔ $SYS/broker/clients/connected :

  The number of currently connected clients

- ✔ $SYS/broker/clients/disconnected :

  The total number of persistent clients (with clean session disabled) that are registered at the broker but are currently disconnected.

- ✔ $SYS/broker/clients/maximum :

  The maximum number of active clients that have been connected to the broker. This is only calculated when the $SYS topic tree is updated, so short lived client connections may not be counted.

- ✔ $SYS/broker/clients/total :

  The total number of connected and disconnected clients with a persistent session currently connected and registered on the broker.

- ✔ $SYS/broker/messages/received :

  The total number of messages of any type received since the broker started.

- ✔ $SYS/broker/messages/sent :

  The total number of messages of any type sent since the broker started.

- ✔ $SYS/broker/messages/publish/dropped :

  The total number of publish messages that have been dropped due to inflight/queuing limits.

- ✔ $SYS/broker/messages/publish/received :

  The total number of PUBLISH messages received since the broker started.

- ✔ $SYS/broker/messages/publish/sent :

  The total number of PUBLISH messages sent since the broker started.

- ✔ $SYS/broker/messages/retained/count :

  The total number of retained messages active on the broker.

- ✔ $SYS/broker/subscriptions/count :

  The total number of subscriptions active on the broker.

- ✔ $SYS/broker/time :

  The current time on the server.

- ✔ $SYS/broker/uptime :

  The amount of time in seconds the broker has been online.

- ✔ $SyS/broker/version :

  The version of the broker. Static.

## 1.6 Un serveur public ?

Si vous voulez utiliser un serveur/broker visible sur Internet (dans le cloud), il y a des solutions commerciales et aussi gratuites (parfois avec des restrictions).

On mentionnera tout particulièrement :

➢ https://mosquitto.org/, avec son broker

test.mosquitto.org

➢ https://www.hivemq.com/mqtt-demo/, avec son broker

broker.hivemq.com

"Our public HiveMQ MQTT broker is open for anyone to use. Feel free to write an MQTT client that connects with this broker. We have a dashboard so you can see the amount of traffic on this broker. We also keep a list of MQTT client libraries that can be used to connect to HiveMQ."

➢ https://iot.eclipse.org/getting-started/, avec son broker

iot.eclipse.org

➢ https://docs.shiftr.io/interfaces/mqtt/, avec son broker

broker.shiftr.io

# 2 API MQTT pour ESP32

Pour que l'ESP puisse manipuler MQTT il y a plusieurs possibilités parmi lesquelles :

1. L'API d'Espressif (les concepteurs de l'ESP).

   https://github.com/espressif/esp-mqtt

2. Les bibliothèques de l'IDE Arduino, et tout particulièrement la bibliothèque : "pubsubclient" (by O'Leary) qui propose une approche **synchrone** :

   https://github.com/knolleary/pubsubclient

Il faut noter que d'autres API clients MQTT existent parmi lesquelles :

- ➢ https://github.com/256dpi/arduino-mqtt
- ➢ https://www.esp8266.com/viewtopic.php?t=8172,
- ➢ https://github.com/marvinroger/async-mqtt-client qui propose une **bibliothèque asynchrone** !

  Pas certain qu'on puisse avoir plusieurs "serveurs asynchrones" (HTTP et MQTT) en même temps ?

- ➢ ...

## 2.1 Installation de l'API "pubsubclient"

Pour l'installer à partir de l'IDE Arduino :

**Croquis/Inclure une bibliothèque/Gérer les bibliothèques**

et chercher "pubsubclient".

La **documentation** est plutôt bien :

- https://pubsubclient.knolleary.net/api.html
- http://www.steves-internet-guide.com/using-arduino-pubsub-mqtt-client/

**ATTENTION de bien lire les limitations** ... notamment sur la taille d'un message standard !!!

- https://github.com/knolleary/pubsubclient/issues/501

**Si votre payload est plus grande, il faudra le spécifier** ! Par exemple, pour 512 octets :

client.setBufferSize(512);

---

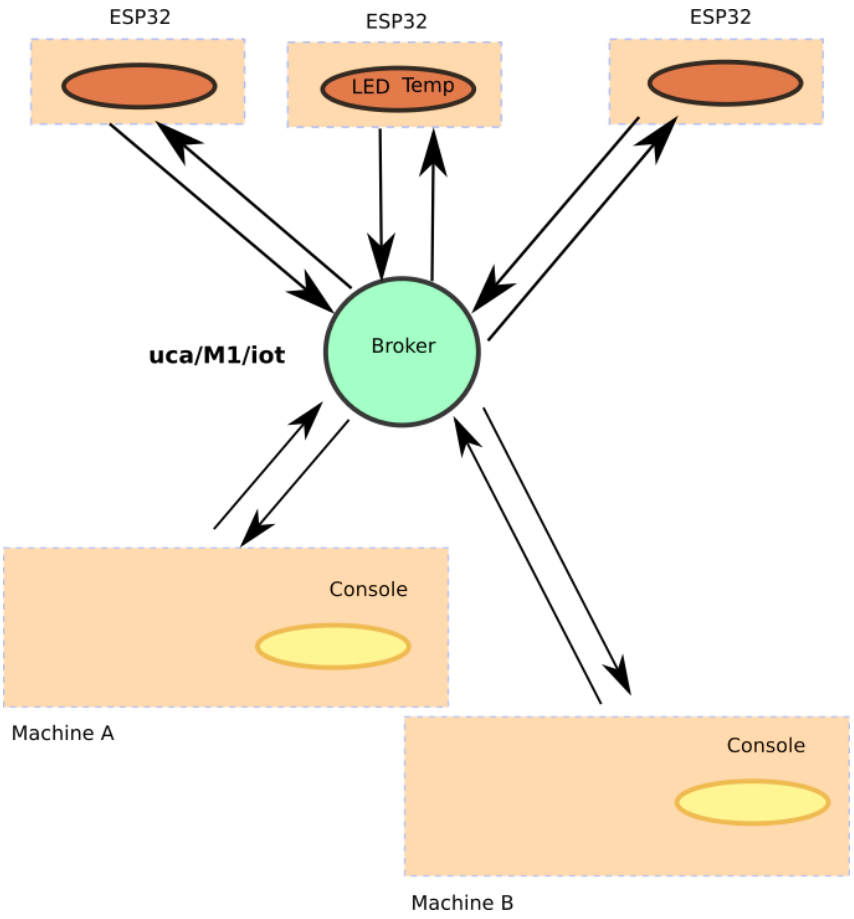
Pour les MacOs ... merci Robin :

```
Pour le problème sur MacOS du PubSubClient.h introuvable il faut l'installer manuellement :
1. Télécharger https://github.com/knolleary/pubsubclient/releases/tag/v2.7
2. unzip ~/Downloads/pubsubclient-2.7.zip
3. mv ~/Downloads/pubsubclient-2.7 ~/Documents/Arduino/libraries
```

# 3 Expérimentation : Un premier exemple

Pour appréhender MQTT, vous allez expérimenter une utilisation simple.

L'exemple qui suit et que je vous demande d'étudier montre comment un ESP peut utiliser MQTT pour diffuser ses résultats et aussi recevoir des informations.




L'ESP publie sa température auprès d'un broker.

- ➢ Il utilise les topics : **"uca/M1/iot/temp"** et **"uca/M1/iot/led"**
- ➢ Il peut aussi recevoir, via ce broker, des informations ... par exemple les températures des "autres" ?

Si les deux API (HTTP et MQTT) sont "théoriquement" compatibles sur l'ESP, il serait souhaitable que l'adresse du broker soit toujours fournie (à l'ESP) par le formulaire de configuration HTML.

Mais mes tentatives pour faire cohabiter l'asynchrone du HTTPd et le MQTT ne sont pas forcément des succès : on verra cela plus tard !

## 3.1 Coté ESP32

Ce code **est FOURNI SUR LE SITE !** ... vous devez le comprendre et utiliser pour cela la documentation de l'API "pubsubclient" :

```
/*********
          Based on Rui Santos work : https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/
          File mqtt_full/mqtt_full.ino
          Modified by GM

          Test with CLI :
          mosquitto_pub  -h test.mosquitto.org -t "uca/M1/iot/led" -m on -q 1
*********/
#include <WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>
#include "wifi_utils.h"
#include "OneWire.h"
#include "DallasTemperature.h"

/*============== GPIO ==============*/
const int ledPin = 19; // LED Pin
/* ---- TEMP ---- */
OneWire oneWire(23); // Pour utiliser une entite oneWire sur le port 23
DallasTemperature tempSensor(&oneWire) ; // Cette entite est utilisee par le capteur de temperature
float temperature = 0;
float light = 0;

#define MQTT_HOST IPAddress(192, 168, 1, XXX)

/*===== MQTT broker/server =========*/
//const char* mqtt_server = "192.168.1.101";
//const char* mqtt_server = "public.cloud.shiftr.io"; // Failed in 2021
// need login and passwd (public,public) mqtt://public:public@public.cloud.shiftr.io
//const char* mqtt_server = "broker.hivemq.com"; // anynomous Ok in 2021
const char* mqtt_server = "test.mosquitto.org"; // anynomous Ok in 2021
//const char* mqtt_server = "mqtt.eclipseprojects.io"; // anynomous Ok in 2021
/*===== MQTT TOPICS ================*/
#define TOPIC_TEMP "uca/M1/iot/temp"
#define TOPIC_LED  "uca/M1/iot/led"
/*===== ESP is a MQTT Client ========*/
WiFiClient espClient;           // Wifi
PubSubClient mqttclient(espClient); // MQTT client
String hostname = "Mon petit objet ESP32";

#define USE_SERIAL Serial

/*===== Arduino IDE paradigm : setup+loop =====*/
void setup() {
  // Serial port
  Serial.begin(9600);
  while (!Serial); // wait for a serial connection. Needed for native USB port only

  // GPIOs configuration
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);// Set outputs to LOW
```

```
52    // Init temperature sensor
53    tempSensor.begin();
54
55    // Wifi connection
56    wifi_connect_multi(hostname);
57    wifi_printstatus(0);
58
59    // set server of our MQTT client
60    mqttclient.setServer(mqtt_server, 1883);
61    // set callback when publishes arrive for the subscribed topic
62    mqttclient.setCallback(mqtt_pubcallback);
63  }
64
65  /*============= TO COMPLETE ? ====================*/
66  void set_LED(int v){
67
68  }
69
70  float get_Temperature(){
71    return 22.5;
72  }
73
74  /*============== CALLBACK ====================*/
75  void mqtt_pubcallback(char* topic,
76                        byte* payload,
77                        unsigned int length) {
78    /*
79     * Callback when a message is published on a subscribed topic.
80     */
81    USE_SERIAL.print("Message arrived on topic : ");
82    USE_SERIAL.println(topic);
83    USE_SERIAL.print("=> ");
84
85    // Byte list (of the payload) to String and print to Serial
86    String message;
87    for (int i = 0; i < length; i++) {
88      //USE_SERIAL.print((char)payload[i]);
89      message += (char)payload[i];
90    }
91    USE_SERIAL.println(message);
92
93    /*
94    char msg[length + 1];
95    memcpy(msg, payload, length);
96    msg[length] = NULL;
97    message = String(msg);
98    */
99
100   // Feel free to add more if statements to control more GPIOs with MQTT
101
102   // If a message is received on the topic,
103   // you check if the message is either "on" or "off".
104   // Changes the output state according to the message
105   if (String(topic) == TOPIC_LED) {
106     USE_SERIAL.print("so ... changing output to ");
107     if (message == "on") {
```

```
108         USE_SERIAL.println("on");
109         set_LED(HIGH);
110       }
111       else if (message == "off") {
112         USE_SERIAL.println("off");
113         set_LED(LOW);
114       }
115     }
116   }
117
118   /*============= SUBSCRIBE to TOPICS ===================*/
119   void mqtt_subscribe_mytopics() {
120     /*
121      * Subscribe to MQTT topics
122      * There is no way on checking the subscriptions from a client.
123      * But you can also subscribe WHENEVER you connect.
124      * Then it is guaranteed that all subscriptions are existing.
125      * => If the client is already connected then we have already subscribe
126      * since connection and subscriptions go together
127      */
128     // Checks whether the client is connected to the MQTT server
129     while (!mqttclient.connected()) { // Loop until we're reconnected
130       USE_SERIAL.print("Attempting MQTT connection...");
131
132       // Attempt to connect => https://pubsubclient.knolleary.net/api
133
134       // Create a client ID from MAC address .. should be unique ascii string and different from all other devices using the b
135       String mqttclientId = "ESP32-";
136       mqttclientId += WiFi.macAddress(); // if we need random : String(random(0xffff), HEX);
137       if (mqttclient.connect(mqttclientId.c_str(), // Mqttclient Id when connecting to the server : 8-12 alphanumeric charact
138                              NULL,   /* No credential */
139                              NULL))
140         {
141         USE_SERIAL.println("connected");
142
143         // then Subscribe topics
144         mqttclient.subscribe(TOPIC_LED,1);
145         // mqttclient.subscribe(anothertopic ?);
146       }
147       else { // Connection to broker failed : retry !
148         USE_SERIAL.print("failed, rc=");
149         USE_SERIAL.print(mqttclient.state());
150         USE_SERIAL.println(" try again in 5 seconds");
151         delay(5000); // Wait 5 seconds before retrying
152       }
153     } // end while
154   }
155
156   /*================= LOOP ======================*/
157   void loop() { /*--- Publish Temperature periodically   */
158     int32_t period = 5000; // 5 sec
159
160     /*--- subscribe to TOPIC_LED if not yet ! */
161     mqtt_subscribe_mytopics();
162
163     /*--- make payload ... too badly ! */
```

```
164     delay(period);
165     temperature = get_Temperature();
166     // Convert the value to a char array
167     char payload[100];
168     sprintf(payload,"{\"temperature\" : \"%.2f\"}",temperature);
169     // Serial info
170     USE_SERIAL.print("Publish payload : "); USE_SERIAL.print(payload);
171     USE_SERIAL.print(" on topic : "); USE_SERIAL.println(TOPIC_TEMP);
172
173     /*--- Publish payload on TOPIC_TEMP  */
174     mqttclient.publish(TOPIC_TEMP, payload);
175
176     /* Process MQTT ... une fois par loop() ! */
177     mqttclient.loop(); // Process MQTT event/action
178   }
179
180   #if 0
181   /*
182    *  Old fashion payload and publishing
183    *
184    */
185     char payload[100];
186     char topic[150];
187     // Space to store values to send
188     char str_sensor[10];
189     char str_lat[6];
190     char str_lng[6];
191
192     sprintf(topic, "%s%s", "/v1.6/devices/", DEVICE_LABEL);
193     sprintf(payload, "%s", ""); // Cleans the payload
194     sprintf(payload, "{\"%s\":", VARIABLE_LABEL); // Adds the variable label
195
196     float sensor = analogRead(SENSOR);
197     float lat = 6.101;
198     float lng= -1.293;
199
200     /* 4 is mininum width, 2 is precision; float value is copied onto str_sensor*/
201     dtostrf(sensor, 4, 2, str_sensor);
202     dtostrf(lat, 4, 2, str_lat);
203     dtostrf(lng, 4, 2, str_lng);
204
205     sprintf(payload, "%s {\"value\": %s", payload, str_sensor); // Adds the value
206     sprintf(payload, "%s, \"context\":{\"lat\": %s, \"lng\": %s}", payload, str_lat, str_lng); // Adds coordinates
207     sprintf(payload, "%s } }", payload); // Closes the dictionary brackets
208
209     client.publish(topic, payload);
210   #endif
```

### 3.1.1 Analyse du code

① Une phase préparatoire au dialogue et aux échanges.

Ligne 31, on déclare le broker et on définit les topics avec notamment le topic de température à la ligne 34.

On choisit d'utiliser un topic par capteur . . . pas certain de l'optimalité de cette solution ? Mais on se focalise pour l'instant sur les topics.

A la ligne 38, l'ESP est un "client" WiFi et ce lien réseau sera utilisé par le client MQTT.

② Ligne 60, on associe (mais **sans le connecter**) le broker à notre ESP (qui est client).

③ Ligne 62 : on associe aussi un "callback" au client qui sera invoqué à la réception d'une publication sur un des topics auxquels l'ESP pourrait avoir souscris.

④ Dans ce callback (Ligne 75), on analyse le message MQTT recu pour en déduire le topic concerné.

On peut avoir souscris à plusieurs topics !

⑤ La connexion au broker est réalisée par la fonction mqtt_subscribe_mytopics à la ligne 112.

Cette fonction permet de plus de souscrire aux topics qui nous intéresse : Ligne 136

Pourquoi coupler les deux (connexion et topic) ?

Deux raisons, la première est que le client n'a pas de moyen de savoir si il a déjà souscrit et la seconde raison est que si il y a eu une déconnexion du réseau physique, le comportement du broker n'est pas garanti et rien n'assure que la souscription a persisté.

En liant les deux on peut savoir que si on est connecté alors ou a souscrit

Dans cet exemple, l'utilisation de la fonction "connect" (Ligne 111) est **simpliste** :

- ✔ un identifiant construit l'adresse MAC de l'ESP ;
  Le premier paramètre ("esp32") est l'identifiant (clientId) du client (l'ESP) au niveau du broker.
  Ce n'est pas le "UserId" dont on verra plus loin qu'il peut être utilisé pour contrôler les connexions au broker.
  Il est néanmoins souhaitable que **cet identificateur soit unique** au niveau du broker ... gare à la recopie multiple de ce code sans comprendre ni lire ! surtout si vous utilisez le même broker.
- ✔ pas de sécurisation de la connexion (No credential):
  Le User Id et son mot de passe ne sont pas fournis : NULL, NULL
  C'est aussi possible parce que le serveur/broker le permet. Il pourrait aussi être configuré pour imposer un vrai login.

Cela ne va pas durer ! ;-)

⑥ Dans la boucle (Ligne 150), périodiquement on publie (Ligne 167) la payload.

⑦ **ET on invoque la fonction "loop" du client MQTT** : Ligne 170

Cette fonction traite les évènements MQTT.

Il manque quelques petites choses pour faire le lien avec vos capteurs/actionneurs :

➢ les fonctions `set_LED()` et `get_Temperature()` ne sont pas forcément achevées ou conforme aux exercices passés.

### 3.1.2 Just for information : An example of asynchronous MQTT

L'idée de ce cours est d'éclairer quelques notions et on a déjà fait de l'asynchrone dans le cadre du serveur HTTP.

Pour information, il existe des bibliothèques de clients asynchrones MQTT :

https://randomnerdtutorials.com/esp32-mqtt-publish-ds18b20-temperature-arduino/

Mais on ne peut pas tout faire !

## 3.2 Others MQTT client APIs

### 3.2.1 With mosquitto CLI

Pour commencer et notamment simplifier la mise au point du dialogue, vous pouvez utiliser les commandes "mosquitto" dans un terminal/console pour tester : souscrire/publier sur le topic.

➢ Cela permet de voir facilement la réaction du broker que vous venez d'installer et de configurer.

### 3.2.2 With a client in a(ny) programming language (optionnal)

On peut aussi réaliser un client MQTT par la programmation dans un langage quelconque, le choix est large :

https://cumulocity.com/guides/device-sdk/mqtt-examples/

| | |
|---|---|
| Arduino : | https://github.com/256dpi/arduino-mqtt |
| Processing : | https://github.com/256dpi/processing-mqtt |
| JavaScript : | https://github.com/mqttjs/MQTT.js |
| Go : | https://github.com/256dpi/gomqtt |
| Ruby : | https://github.com/njh/ruby-mqtt |
| C : | http://www.eclipse.org/paho/clients/c/ |
| C++ : | http://www.eclipse.org/paho/clients/cpp |
| Java : | http://www.eclipse.org/paho/clients/java |
| Python : | http://www.eclipse.org/paho/clients/python |

---

### 3.2.3 With a client in Python

Je vous propose dans ce qui suit un client MQTT écrit en Python (package paho-mqtt anciennement python-mosquitto) et destiné à tourner sur un host/PC permettant ainsi de dialoguer avec l'ESP à la place de la console.

```python
# Let's talk MQTT in Python
# first install : https://pypi.org/project/paho-mqtt/
# sudo apt install python3-paho-mqtt sous Ubuntu
# Author : G.MENEZ
import paho.mqtt.client as mqtt
import time
import json
#-----------------------------------------------
def on_message(client, userdata, message) : # Callback on arriving messages
    print("\nmessage received = {}".format(str(message.payload.decode("utf-8"))))
    print("on topic = {}".format(message.topic))
    print("with qos = {}".format(message.qos))
    print("with retain flag = {}".format(message.retain))

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if __name__=="__main__":
    #------------MQTT client instanciation -------------------
    clientname = "P1"
    print("Creating new Client : {}".format(clientname))
    client = mqtt.Client(clientname)   # create new instance
    client.on_message=on_message  # attach function to callback

    #----------connect to broker ---------------
    broker_address="test.mosquitto.org" # "192.168.1.101"
    """ broker_address="iot.eclipse.org" """
    print("Connecting to broker {}".format(broker_address))
    client.connect(broker_address) # connect to broker

    client.loop_start()     #------------ start the loop

    #----------Subscribe ----------------------------
    topicname = "IoT/menez/sensors"
    print("\nSubscribing to topic {}".format(topicname))
    client.subscribe(topicname)

    #Make payload .. and Serialize it !
    payload = {
        "Temperature":10,
        "Humidity":50 }
    msg = json.dumps(payload)

    # Publish it 3 times every 5 sec
    for i in range(3) :
        print("\nPublishing message {} to topic {}".format(msg, topicname))
        client.publish(topicname, payload=msg, qos=2, retain=False)
        time.sleep(5)

    time.sleep(100)         # wait in seconds before
    client.loop_stop()      #------------ stop the loop and the script
```

### 3.2.4 Few comments on this code

#### Le callback : "on_message"

Vous remarquez la présence de la fonction "on_message" qui est appelée à chaque fois qu'un des topics auxquels on s'est inscrit publie un message.

➢ Ce callback est "attaché" au client MQTT à la ligne 25.

#### Les boucles ...

From : https://pypi.org/project/paho-mqtt/#network-loop

```
"Network loop"

These functions are the driving force behind the client. If they are
not called, incoming network data will not be processed and outgoing
network data may not be sent in a timely fashion. There are four
options for managing the network loop. Three are described here, the
fourth in "External event loop support" below. Do not mix the
different loop functions.
```

### 3.2.5 With a client in Java (optional work !)

Vous pouvez aussi faire ce client en Java (ou autre) si vous êtes plus confortable dans ce langage :

http://www.bytesofgigabytes.com/mqtt/java-as-mqtt-publisher-and-subscriber-client/

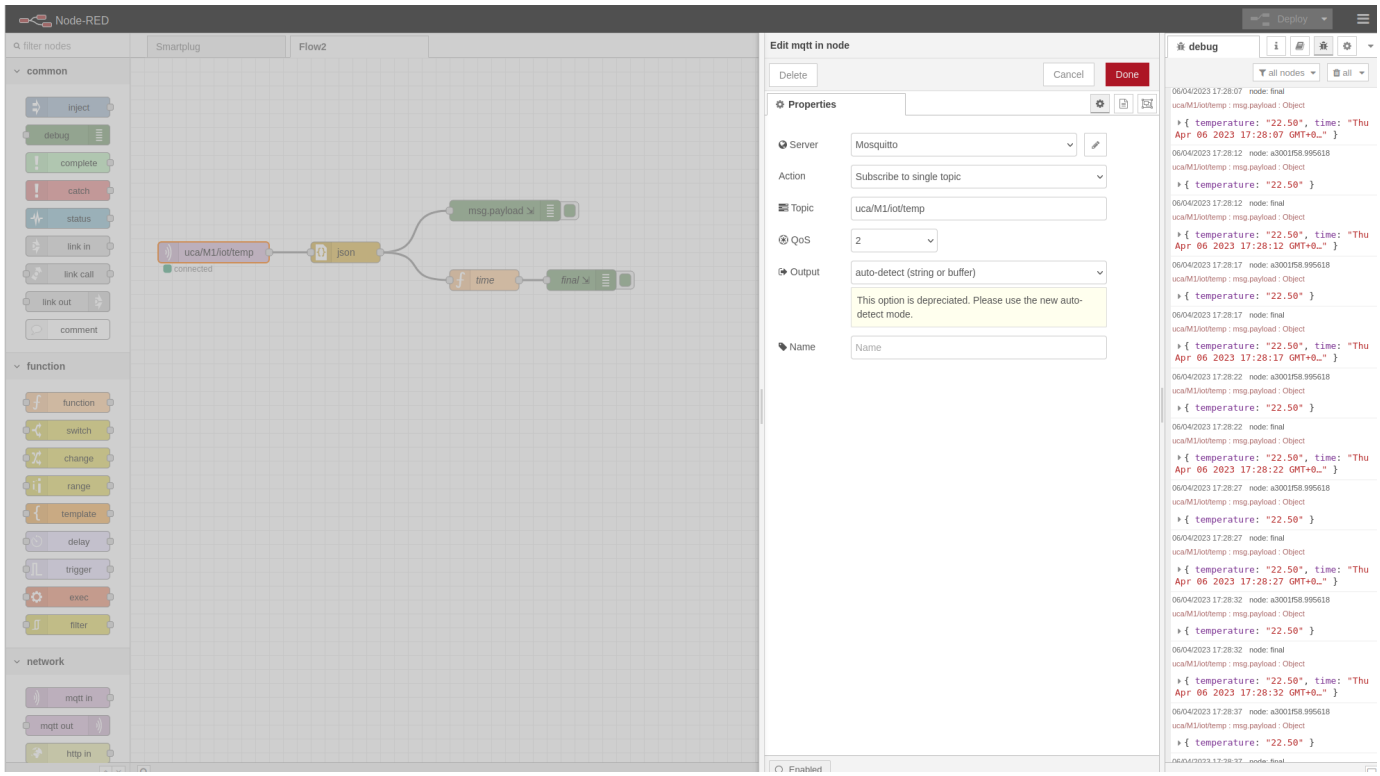
# 4 La connexion avec Node-Red

On peut facilement ajouter des noeuds à un flow Node-Red pour participer (PUB/SUB) à des échanges MQTT.

- Le noeud du flow Node-Red fait de ce dernier un client MQTT.

  **Attention**, vous allez recevoir les publications de TOUS les objets . . . il va falloir gérer !

Je rappelle Node-Red dans le navigateur sur le port 1880 : `http://localhost:1880`




On ajoute :

1. un noeud "mqtt_in" (cf palette) que je configure au niveau du choix du server/broker et du topic
2. un noeud JSON pour convertir la chaîne en "objet" JSON
3. un noeud de debug pour voir que je reçois bien les PUBLISH de l'ESP sur le broker

et on déploie !

On voit bien sur la colonne de droite (le debug) l'arrivée des messages en provenance de l'ESP qui est en train de publier ses températures.

## 4.1 JSON via MQTT

Vous devez ainsi, avec un moindre effort, pouvoir ré-utiliser le dashboard développé dans le cadre de HTTP/JSON.

- Il "suffit" de passer par MQTT pour recevoir la payload du status.

# 5 TODO : Swimming Pool !

Jusqu'à présent vos objets/ESP ont échangé :

➢ avec une machine de configuration qui, parce qu'elle avait envoyé une requête HTTP (GET) sur l'ESP, avait obtenu en réponse sa page HTML de configuration.

➢ avec une machine de supervision qui, parce que la configuration l'avait choisi, recevait périodiquement une requête HTTP (POST) contenant le status de l'objet.

Ces échanges d'information sont orientés vers le "centre du réseau" et à apriori l'accès aux machines de configuration et/ou supervisions sont restreints par et à la société qui met en place ce réseau d'objets.

Dans l'exemple qui suit, cette société souhaite mettre en place un service "hotspot" (at the edge) qui nécessitera aux objets d'échanger leur localisation et leur température.
Utiliser HTTP, pour mettre en relation les différents objets nécessiterait de "rebondir" sur le centre du réseau et d'implémenter une ré-émission.

- On va voir tout l'intérêt de MQTT pour faire communiquer des objets qui ignorent, a priori, l'existence des "autres".

## 5.1 Un topic : "piscine"

Pour cet exercice, vous voilà chacun propriétaire d'une immense piscine équipée d'un beau thermomètre connecté !

Pour les besoins de ce nouveau service la société met en place un topic dédié sur un broker MQTT :

- **Broker** : test.mosquitto.org
- **Topic** : uca/iot/piscine
- **Période** des "publishs" : réglable (def : 10 minutes en exploitation)

Chaque ESP publie au moins

- sa "température" (supposée être la température de l'eau de la piscine),
- son identifiant (par exemple son adresse MAC),
- et sa "localisation/location" (latitude, longitude).

Pour faire simple, ces informations s'intègrent dans le schéma JSON utilisé par l'ESP pour le reporting du status du TP précédent.

- On aurait pu/du le réduire mais cela ferait un schéma Json différent et cela compliquerait le code de l'ESP pour pas grand chose.
- Température, identifiant et localisation font donc déjà partie du status.

Vous pouvez choisir pour votre objet/piscine une coordonnée dans les Alpes Maritimes :

https://www.coordonneesgps.net/coordonnees-gps/alpes-maritimes-115-region

On ajoute au schéma JSON du status une section "piscine" avec deux champs booléens :

```
"piscine": {
  "hotspot" : False,
  "occuped" : False
}
```

### 5.1.1 "hotspot" and "occuped"

"hotspot" est un indicateur calculé "on the edge" par l'ESP :

- **Un objet/piscine est un hotspot si la valeur de sa température est supérieure à celles de tous les autres objets (piscines) dans un rayon de 10 Kms.**
- Si c'est un hotspot, il allume sa led interne (Pin 2) sinon cette dernière reste éteinte.
- Les valeurs retenues par "votre calcul" sont les dernières publiées sur le topic température du broker.

Dans la partie "piscine" du json, un indicateur de présence (calculé aussi "on the edge" à partir d'un capteur local) : "occuped"

- Si le capteur de lumière détecte une intensité lumineuse **inférieure à un seuil**, on peut considérer qu'il y a une présence ("occuped" = true) autour de la piscine.
- Sinon la piscine n'est pas utilisée.

### 5.1.2 Calcul de la proximité

Pour ce qui est du calcul de la "proximité", vous pouvez chercher à calculer la distance :

- en la codant directement dans l'ESP : https://en.wikipedia.org/wiki/Haversine_formula

  L'URL suivante évoque cette solution en JS :

  https://cloud.google.com/blog/products/maps-platform/how-calculate-distances-map-maps-javascript-api?hl=en

- vous pouvez trouver une API plus ou moins gratuite sur le réseau.

  Par exemple : https://distancematrix.ai/distance-matrix-api

  Certaines intègrent même le temps de trajet ... pour aller dans la piscine du voisin !

  Il y aura certainement des limitations donc "calmos" sur la période des requêtes.

- vous pouvez aussi définir une API "perso".

  Ce travail peut être collectif l'API serait alors utilisée par plusieurs !!

## 5.2 Consommation

En exploitant les modes présentés dans le document sur la consommation, essayez de restreindre la consommation de l'ESP !

Malheureusement, ce qui suit est optionnel au niveau de l'UE.

C'est un scandale d'écrire ceci car la sécurité est un élément crucial de l'IoT mais compte tenu du temps imparti à cette UE ... il en sera ainsi :-(

J'ai quand même laissé dans ce document quelques éléments pour démarrer si vous êtes un peu curieux ou si vous en aviez un jour besoin ?

# 6 Sécurité

Pour l'instant, on l'a peu évoqué, MAIS la sécurité est un élément fondamental de l'IoT.

https://www.paloaltonetworks.com/cyberpedia/what-is-iot-security

➢ Il y a pas mal de points à sécuriser ... on s'y met ?

Ce que l'on va faire sur MQTT aurait pu être fait sur HTTP. Mais le TP était déjà bien riche et long :-(

## 6.1 Autorisation des utilisateurs

Dans le cas présent, à l'évidence, la connexion au broker est une faille majeure.

➢ Dans la configuration actuelle, pratiquement n'importe qui peut se connecter au broker, "souscrire à", "publier sur" n'importe quel topic.

Ceci permettrait d'accéder (en lecture/écriture) aux flux de votre application IoT et par exemple déclencher une fausse alarme incendie !

### 6.1.1 Configuration du broker

Pour répondre à ce premier problème, on peut configurer le broker (Mosquitto par exemple) pour **exiger l'authentification du client** avant qu'une connexion ne soit autorisée.

La validité de l'authentification repose sur

➢ un nom d'utilisateur,

➢ et un mot de passe

Vous pouvez trouver de l'aide dans :

➢ https://mosquitto.org/documentation/authentication-methods/

➢ http://www.steves-internet-guide.com/mqtt-username-password-example/

➢ https://domopi.eu/la-securisation-du-protocole-mqtt/

et ce qui suit exploite ces sites.

La première chose est de modifier la configuration du broker pour activer l'authentification des clients :

```
listener 1883
protocol mqtt

allow_anonymous false
password_file /etc/mosquitto/passwd

per_listener_settings false
```

**Quelques remarques sur cette configuration :**

- "listener" : ce champ spécifie le port concerné par la configuration. 1883 correspond au service "Plain MQTT protocol" donc **NON encrypté**.
- "allow_anonymous false" désactivera toutes les connexions non authentifiées pour le listener.
- on précise avec le paramètre "password_file" le chemin du fichier qui stockera les identifiants / mots de passe étant autorisés à se connecter et à publier / souscrire sur notre broker.

  Même vide, le fichier **doit** exister !
- Enfin, le paramètre "per_listener_settings" vous permet de préciser que les paramètres d'authentification définis précédemment sont globaux ou à appliquer par listener.

### 6.1.2 Login/Mots de passe du broker

Passons maintenant à la génération/remplissage de ce fichier stockant nos identifiants.

```
mosquitto_passwd -c -b /etc/mosquitto/passwd darkvador 6poD2R2
```

Vous pouvez vérifier que le mot de passe stocké est hashé dans le fichier !

- Mais il sera "en clair" au niveau du code du client ESP ou Python ... pas top :-(

### 6.1.3 Impact au niveau du client

Est ce que vous vous rappelez la ligne 116 dans le code ".ino" du client (cf section "MQTT coté ESP32") ?

```
    if (client.connect("esp32",
          NULL, /* No credential */
          NULL)) { ...
```

Puisque le broker demande une authentification, il va falloir faire évoluer cela :

```
          String mqttId = "DeathStar-";
          const int mqttPort = 1883;
          const char* mqttUser = "darkvador";
          const char* mqttPassword = "6poD2R2";
```

```
5
6           mqttId += String(random(0xffff), HEX);
7           // Attempt to connect
8           if (client.connect(mqttId.c_str(),
9                              mqttUser,
10                             mqttPassword)) {
```

### 6.1.4 Validation des topics/utilisateurs

Dans la configuration du broker, on peut aussi restreindre l'accès aux topics sur la base de l'identification des clients : **Access Control Lists** (ACL)

➢ Ce n'est pas parce qu'un utilisateur à accès à un broker qu'il peut forcément avoir accès à tous les topics !

http://www.steves-internet-guide.com/topic-restriction-mosquitto-configuration/

Intéressant . . . mais je n'ai pas essayé !

## 6.2 Sécurisation des flux

La restriction d'accès bien qu'indispensable pose quelques problèmes . . . et ne résout pas tout !

➢ La présence des identifiants dans le code de l'objet !?

➢ Les flux réseaux restent lisibles et notamment les identifiants :-(

**Tout cela est pour l'instant transmis en clair !**

## 6.3 La fausse bonne idée !

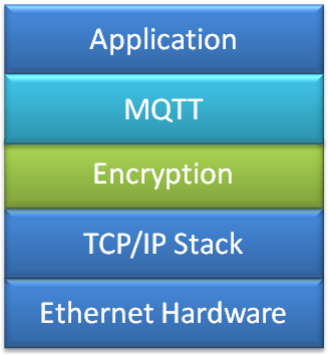
On pourrait imaginer que l'application se charge de chiffrer les payloads/contenus et réalise ainsi une "end-to-end encryption" ?

➢ Mais au niveau de la couche application, on ne pourrait pas masquer les informations de services (adresse IP, port, . . . ) utilisées par les couches inférieures.

Ces informations sont aussi très sensibles et donnent beaucoup de pistes à d'éventuels pirates !

➢ De même les topics pourraient difficilement être chiffrés par le client puisque c'est le broker qui les gère.

Une meilleure approche consiste donc à mettre en place une couche de cryptage au-dessus de la pile de communication et d'intégrer le broker dans le périmètre de sécurité :




De cette façon, l'application (ou la partie du langage MQTT) n'a pas besoin de mettre en oeuvre le protocole de cryptage lui-même, il suffit de parler à la couche de cryptage et celle-ci fera tout le travail.

## 6.4 Chiffrement

Références :

- https://www.cairn.info/revue-les-cahiers-du-numerique-2003-3-page-101.htm
- https://en.wikipedia.org/wiki/Public-key_cryptography

Pour assurer la confidentialité d'un document électronique, on "chiffre" le texte du document.

- Il s'agit d'éviter que quelqu'un (non souhaité) puisse interpréter l'information !

On lui applique un ensemble de fonctions mathématiques (un algorithme de chiffrement) avec des caractéristiques très particulières en utilisant une variable : **la clé de chiffrement**.

Une fois le texte chiffré, il est "incompréhensible". Pour obtenir la version lisible, il faut le déchiffrer, c'est-à-dire appliquer une autre fonction mathématique compatible avec la première, avec une autre variable : **la clé de déchiffrement**.

Seul le possesseur de la clé de déchiffrement peut déchiffrer le texte. La valeur de la clé de déchiffrement dépend évidemment de la valeur de la clé de chiffrement.

Il faut noter que les algorithmes de chiffrement sont généralement publics et ont fait l'objet de standardisation.

- Dans ce contexte, **c'est donc le secret de certaines clés qui permet d'assurer la confidentialité**.

Il y a deux grandes familles d'algorithmes de chiffrement : **symétriques et asymétriques**.

### 6.4.1 Chiffrement symétrique

Dans les **algorithmes symétriques**, aussi appelés "algorithmes à clé secrète", **la clé de chiffrement est la même que la clé de déchiffrement**.

De ce fait, pour que le texte chiffré ne soit lisible que par le destinataire, **la valeur de cette clé doit être un secret partagé entre l'émetteur et le destinataire uniquement**.

➢ Ceci explique le qualificatif de "clé secrète".

DES (Data Encryption Standard) et AES (Advanced Encryption Standard) sont les algorithmes symétriques les plus connus.

L'approche nécessite peu de puissance de calcul (ce qui est plutôt bien) :

https://hpbn.co/transport-layer-security-tls/

**MAIS** elle ne supporte pas le passage à l'échelle car :

**Comment garder un secret alors qu'il est partagé ?!**

Il suffit d'un interlocuteur piraté pour remettre en cause vos échanges avec lui (ou plus si cette clé couvre plusieurs interlocuteurs) et dans tous les cas servir de point de départ d'une intrusion plus profonde.

### 6.4.2 Chiffrement asymétrique

L'autre ensemble d'algorithmes est celui des **algorithmes asymétriques** ou à **clé publique**.

Ils ont été conçus pour utiliser des clés qui possèdent plusieurs propriétés :

- ✔ **La clé de chiffrement est différente de la clé de déchiffrement** (d'où le terme asymétrique) ;
- ✔ Les deux clés (une pour chiffrer, l'autre pour déchiffrer) sont créées ensemble avec une fonction mathématique.

  Elles forment un couple ("key pair"), l'une ne va pas sans l'autre, mais il est **impossible avec une des clés de découvrir l'autre**.
- ✔ **Tout texte chiffré avec une des clés (de chiffrement ou de déchiffrement) peut être déchiffré avec l'autre clé (de déchiffrement ou de chiffrement) et uniquement avec celle-ci** !

En pratique, pour utiliser ces algorithmes, il faut générer un couple de clés pour chaque utilisateur :

- ✔ clé privée secrète "classiquement" pour déchiffrer,
- ✔ et d'une clé publique "classiquement" pour chiffrer.

  Contrairement à la première, cette clé peut être diffusée sans modération.

RSA (du nom des trois inventeurs Rivest, Shamir, Adleman) est un "cryptosystem" qui implémente cette approche.

## 6.5 Authentification et intégrité

Ce ne n'est pas parce que je peux décoder une information, que je connais avec exactitude l'auteur de cette information !

Par conséquent, je **peux/dois douter de la véracité** de cette information car un pirate peut utiliser la clé publique de quelqu'un pour se faire passer pour lui !

➢ C'est l'attaque MITM : **"Man In The Middle"**

---

**Le scénario :**

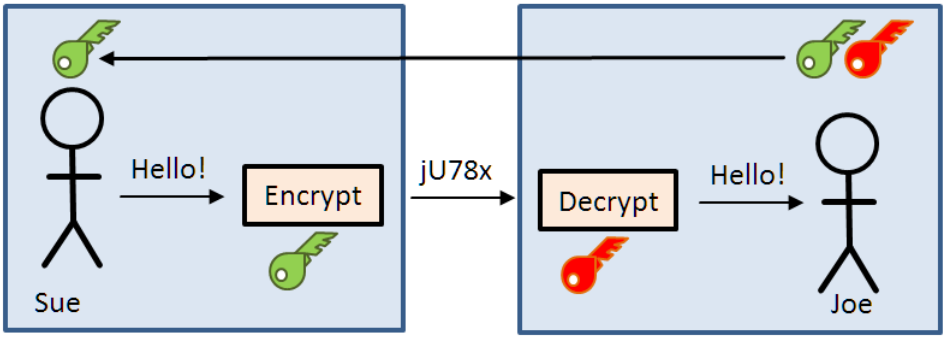
Joe et Sue veulent échanger des données confidentielles. Carole veut les intercepter !

➢ Ils possèdent chacun une clé privée (respectivement Js, Ss et Cs)

➢ et une clé publique (respectivement Jp, Sp et Cp).

---

### 6.5.1 Cas normal - Échange classique :




✔ Joe et Sue échangent leur clé publique . . . sinon ils ne pourront pas se lire !

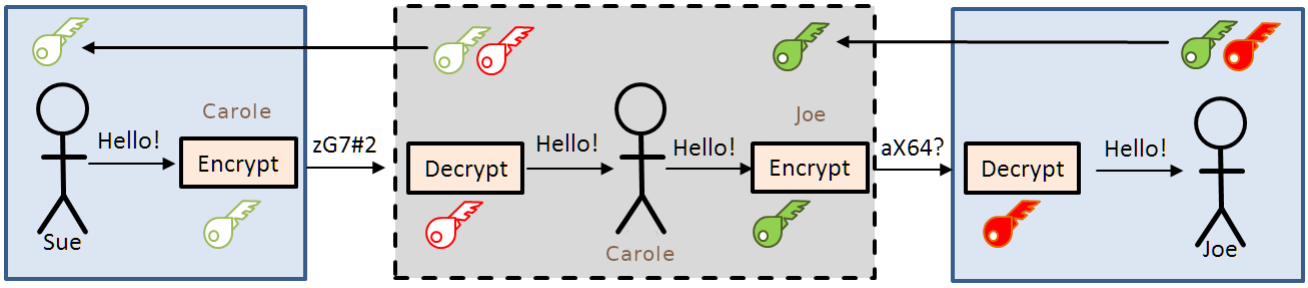
✔ A partir de ce moment Sue envoie des messages à Joe en utilisant Jp.

Joe répond en utilisant la Sp de Sue.

✔ Carole voit tout passer : Jp, Sp et les messages cryptés ("jU78x").

MAIS **dans cette configuration**, les clefs publiques ne lui servent à rien et les messages ne sont pas déchiffrables sans les clefs privées : Ss et Js.

### 6.5.2 Cas d'Attaque : MITM




Joe confiant dans la procédure de chiffrement asymétrique envoie de façon non sécurisée sa clef publique à Sue pour que cette dernière puisse lui envoyer des messages codés que lui seul pourra lire.

Sue, qui sait que Joe veut discuter avec elle, s'attend à recevoir la clef publique de Joe.

**MAIS ils oublient un "Use Case" :**

Carole peut se placer sur le lien (Sue <-> Joe), intercepter ces clés **et surtout** répondre à la place de Joe et Sue!

- ✔ Conséquences sur Joe :

  Carole qui a la clé publique de Joe peut désormais lui envoyer des informations qu'il pourra décoder. Joe n'a aucune raison de ne pas penser que ces informations viennent de Sue.

  A ce stade, **Carole maîtrise le flux d'entrée de Joe** MAIS elle ne peut pas encore interpréter le flux de sortie de Joe.

  Puisque Joe veut discuter avec Sue, il s'attend à recevoir une clef publique à son tour . . . théoriquement en provenance de Sue.

  **MAIS** Carole lui renvoie donc sa propre clé publique (Cp) **en se faisant passer ainsi pour Sue**.

  Lorsque Joe enverra un message à Sue, il utilisera donc, sans le savoir, la clé publique de Carole pour chiffrer le message. Et Carole pourra **le déchiffrer avec sa clé privée (Cs)**.

  **A ce stade, Carole maîtrise désormais aussi le flux de sortie de Joe**.

- ✔ Conséquences sur Sue :

  Sue qui s'attendait à recevoir la clé de Joe, reçoit en réalité celle de Carole. **Carole maîtrise désormais le flux de sortie de Sue** puisqu'elle peut le déchiffrer.

  Mais Sue voulant permettre à Joe de lui envoyer des messages chiffrés va lui envoyer sa clé publique.

  Carole en possession de cette clé **maîtrise désormais le flux d'entrée de Sue** puisqu'elle peut lui donner des informations.

---

Ainsi, Joe et Sue sont chacun persuadés d'utiliser la clé de l'autre, alors qu'ils utilisent en réalité tous les deux la clé de Carole.

➢ **Cette dernière a accès en lecture et en écriture à TOUS les flux !**

Une (cf wikipedia pour les autres) des réponses à cette problématique repose sur **la "signature" électronique et le "certificat numérique signé"** qui vont permettre d'assurer les fonctions d'authentification (de l'interlocuteur => "on est sûr de parler à la bonne personne") et d'intégrité (du message => "le message reçu est identique à celui qui a été envoyé").

## 6.6 La signature électronique

Le première chose qu'il faut essayer d'assurer est la suivante :

➢ Est ce que l'information que je recois est celle qui a été émise **DANS SON INTEGRALITE**?

Cette question ne veut pas résoudre MITM puisque les messages émis par le pirate sont INTEGRES.

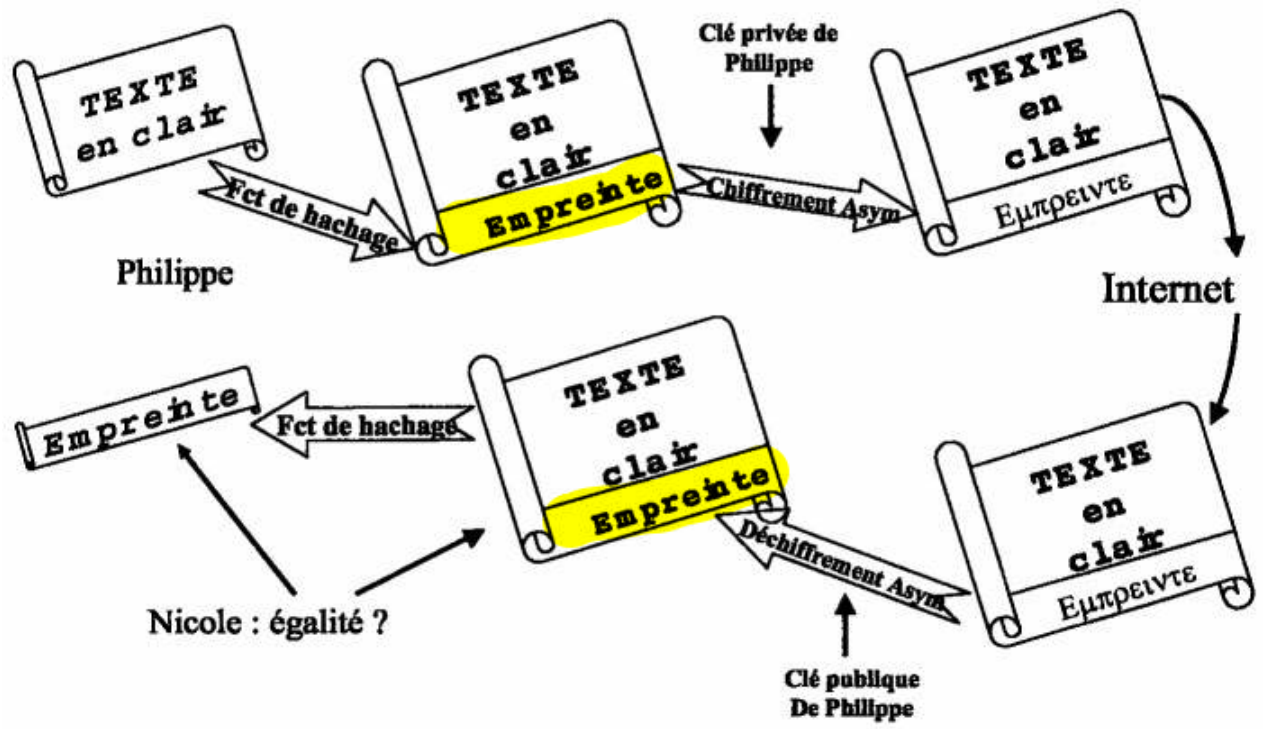
➢ Elle veut juste permettre d'éviter qu'un message soit **PARTIELLEMENT** modifié sans qu'on s'en apercoive.

L'idée est "signer électroniquement" l'information pour créer un lien bijectif entre l'information et cette signature.

➢ Si on "touche" l'information alors la signature n'est plus valide (pour ce contenu) !

➢ La **véracité** de l'information est "assurée".

Pour générer une signature électronique, il faut dans un premier temps utiliser une fonction de hachage sur le texte, dont le résultat est une suite de bits de taille fixe, bien inférieure à la taille du texte initial.



* (https://www.cairn.info/revue-les-cahiers-du-numerique-2003-3-page-101.htm)

Cette suite de bits est aussi appelée "condensé" ou "empreinte", car la fonction de hachage est telle que **si un bit du texte d'origine est modifié, le résultat de la fonction sera, avec de très fortes probabilités, différent**.

➢ MD5 (Message Digest) et SHA (Secure Hash Algorithm) sont parmi les fonctions les plus connues.

### 6.6.1 Signature électronique pour transmettre

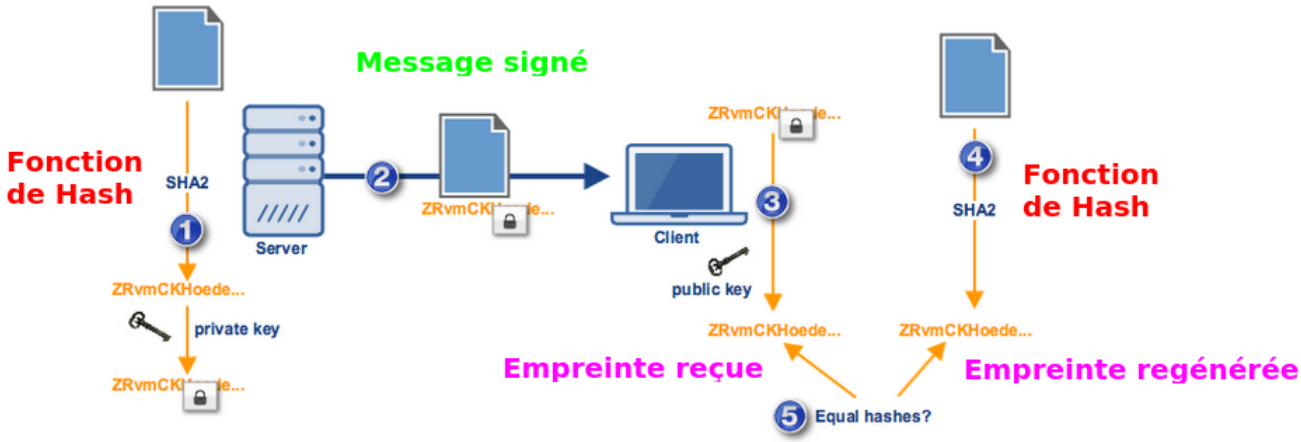
Pour réaliser une émission avec signature électronique,

① Avant d'envoyer un message, l'émetteur calcule d'abord l'empreinte du message.

② Il chiffre ensuite cette empreinte par un algorithme asymétrique avec la clé privée de l'utilisateur/émetteur.

Ce résultat est appelé **signature électronique**.

③ Avant l'envoi, cette signature est ajoutée au message, qui devient un **message signé**.




https://www.jscape.com/blog/what-is-a-digital-signature

A la réception,

① Le serveur qui reçoit l'ensemble, déchiffre cette empreinte chiffrée avec la clé publique de l'émetteur.

② Puis il recalcule la fonction de hachage sur le message reçu et compare le résultat avec l'empreinte déchiffrée.

③ **Si les deux sont égaux, cela veut dire**

- ✔ que le message n'a pas été modifié durant le transfert
- ✔ ET que l'émetteur est "authentifié".

En effet, si le message a été modifié durant le transfert, les 2 empreintes seront différentes.

De plus, être capable de déchiffrer, avec la clé publique d'une personne, une empreinte chiffrée, **prouve que cette empreinte a obligatoirement été chiffrée avec la clé privée de la personne, clé que seul possède l'émetteur**.

➢ **Cela authentifie donc l'émetteur !** ... du moins celui dont on a la clé publique. Mais est-ce le bon ?

Les logiciels courants cumulent les deux fonctions (chiffrement et signature) : Ils chiffrent le message signé !

➢ Pour lire le message, il faudra être capable de casser le chiffrement

➢ Pour le modifier, il faudra casser la signature.

## 6.7 Certificats

La situation est la suivante :

a) Vous recevez un message dont vous êtes certain qu'il n'a pas été modifié.

b) Mais vous n'êtes pas sûr de l'identité de la personne qui vous l'envoie.

c) Vous aimeriez que ce message soit "accompagné" d'une pièce d'identité qui prouve (parce qu'elle a été émise par une autorité "sûre"/"indiscutable") que ce message vient de la bonne personne.

### 6.7.1 Certificat électronique

Un passeport contient des informations concernant son propriétaire (nom, prénom, adresse ... la signature manuscrite), la date de validité, ainsi qu'un tampon et une présentation (forme, couleur, papier) qui permettent de reconnaître que ce passeport n'est pas un faux, qu'il a été **délivré par une autorité bien connue**.

Un "certificat électronique" de personne ou de serveur est l'équivalent électronique d'une carte d'identité ou d'un passeport.

**Ce certificat va permettre d'identifier un interlocuteur et de lui associer sa clé publique (sans que l'on puisse douter de cette association !)** :

C'est un fichier qui contient des informations :

✔ le nom de l'autorité (de certification) qui a créé le certificat,

✔ l'identifiant de la personne ou du serveur à qui appartiennent la clé publique et le certificat.

Pour une personne, cela peut être son adresse email. Pour un serveur web, c'est en général son nom de domaine,

✔ la clé publique de la personne ou du serveur,

✔ les dates de début et de fin de validité de la clé publique et du certificat,

✔ et enfin, la signature électronique (/empreinte) de ce certificat.

**Cette signature "certifie" que la clé publique est bien celle de l'identifiant puisque qu'elle a été établie sur toutes les informations contenues dans le certificat.**

Cette signature est ensuite chiffrée avec la clé privée de l'autorité de certification qui a délivré ce certificat.

Le format standard le plus courant pour les certificats est le format X.509; utilisé notamment par HTTPS, IPsec, PGP et SSH.

En plus des informations de base, un certificat X.509 contient des informations complémentaires :

- ✔ le nom de l'algorithme de chiffrement et de signature avec lesquels la clé publique du certificat est compatible ;
- ✔ le rôle du certificat.

### 6.7.2 Autorité de certification (CA)

Un certificat n'a de sens que si il peut être vérifié . . . comme un passeport !

L'autorité de certification est une entité qui délivre des certificats pour une communauté d'utilisateurs "au sommet" d'une infrastructure de gestion de clés (IGC).

- ➢ C'est l'équivalent de la préfecture pour un passeport.

En quoi cela règle le problème ?

L'autorité de certification est un "interlocuteur" bien connu . . . on place rarement les préfectures dans des caves d'immeubles ?!

- ➢ On sait donc qu'on s'adresse à un interlocuteur "bien connu" donc de confiance . . . sauf si le DNS a été piraté !

Cette autorité, préalablement à toute action, a généré un couple de clés publique-privée pour elle-même.

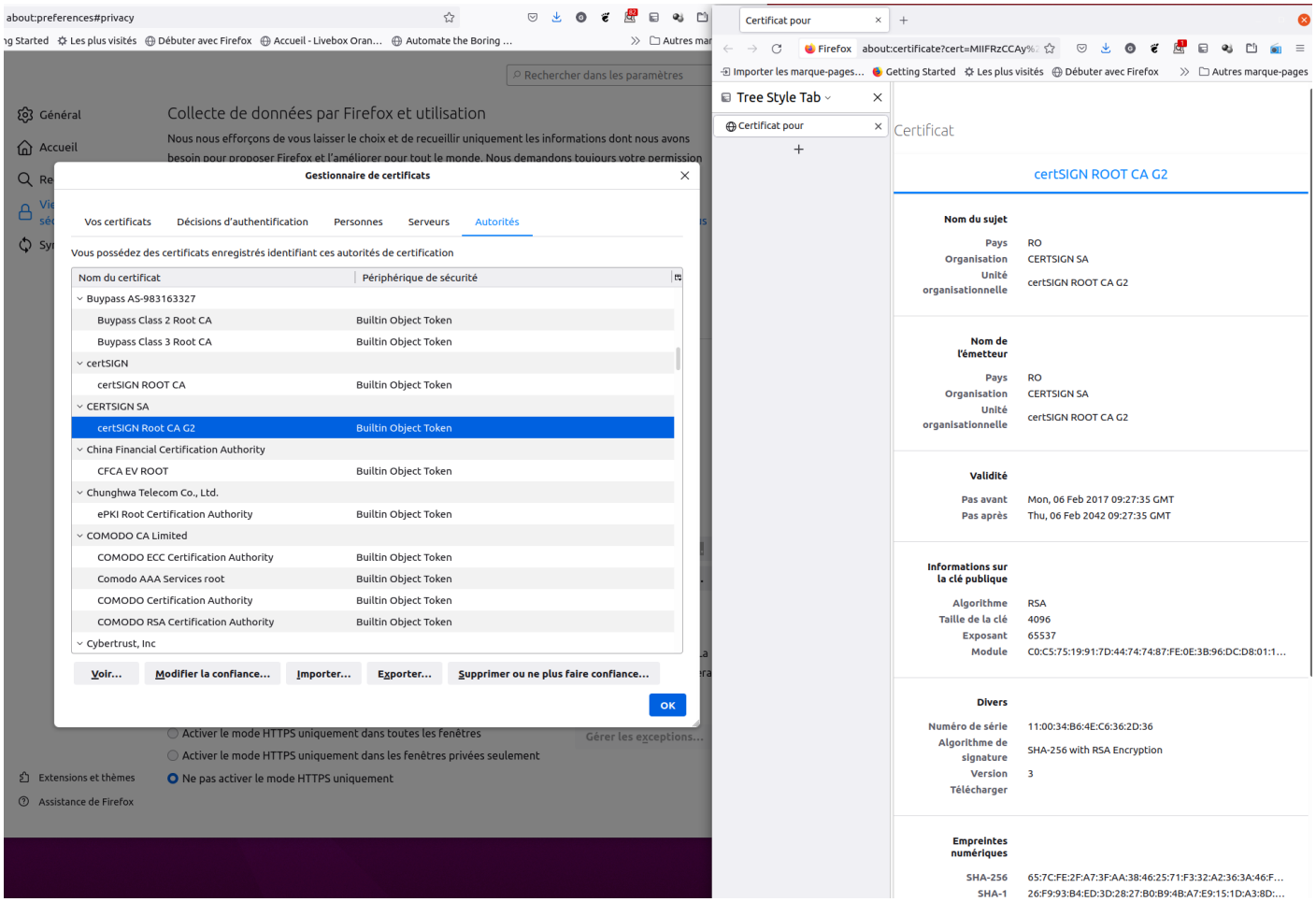
Ensuite elle a très largement diffusé la valeur de sa clé publique, sous la forme d'un **certificat d'autorité de certification** :

"certificat CA" : certificat Certification Authority

Les utilisateurs qui veulent utiliser et faire confiance aux certificats émis par cette autorité, insèrent ce certificat dans leurs outils : navigateur, client de messagerie, . . . .

Ci dessous, les "trusted certificats (CA)" contenus dans "mon" firefox.

- ➢ Tous sont émis par des autorités reconnues.




Ceci permettra à aux clients qui utilisent ces autorités de valider les certificats des serveurs avec lesquels ils souhaitent échanger.

### 6.7.3 Verification du certificat d'un serveur

* (https://www.cairn.info/revue-les-cahiers-du-numerique-2003-3-page-101.htm)

Quand Nicole veut envoyer un message chiffré à Philippe qui travaille dans un laboratoire CNRS, le logiciel de messagerie de Nicole (le client) a besoin de connaître la clé publique de Philippe.

➢ Si ce logiciel ne connaît pas cette clé, il peut interroger l'annuaire électronique du CNRS pour récupérer le certificat de Philippe.

On peut aussi imaginer que Philippe donne directement à Nicole son certificat (qui contient sa clé publique)

**Point important, ce certificat n'est par signé par Philippe MAIS par une "autorité de certification (CA)" . . . celle du CNRS par exemple !**

Le poste de Nicole configuré pour faire confiance à cette autorité a stocké la clé publique de cette autorité de certification.

Le logiciel de messagerie de Nicole peut alors vérifier la signature du certificat de Philippe :

- ➢ Car si il peut décrypter la signature (/empreinte) du certificat de Philippe c'est qu'elle a été encryptée avec la clé de l'autorité CNRS.
- ➢ Donc ce document a bien été créé par l'autorité de certification CNRS et n'a pas été falsifié. Il est valide !
- ➢ La clé publique est bien celle de l'identifiant figurant dans le certificat : Philippe !

Avec cette assurance, le logiciel de messagerie peut récupérer la clé publique contenue dans ce certificat et l'utiliser avec confiance en étant certain que c'est celle de Philippe.

Evidemment, les dates de validité contenues dans le certificat sont aussi vérifiées avant de le déclarer valide.

---

### 6.7.4 CONCLUSION :

**Le serveur doit fournir un certificat signé par une autorité de certification pour justifier de son identité auprès du client** !

# 7 Mise en oeuvre : Mosquitto Security

L'idée est d'utiliser ces concepts dans le cadre d'une mise en oeuvre de la sécurisation des flux entre des clients et un broker(/serveur) MQTT (Mosquitto . . . le vôtre comme cela on peut facilement le configurer).

La solution ici présentée s'appuie sur TLS/SSH qui permet un transport sécurisé des informations sur l'Internet avec les fonctions d'authentification du serveur et du client, d'intégrité et de confidentialité des échanges :

- ➢ La connexion est **privée** car les données sont cryptées entre le client et le serveur.
- ➢ Les parties qui communiquent sont **authentifiées** afin de s'assurer que chaque partie parle avec l'hôte auquel elle est destinée.
- ➢ La connexion est **fiable** dans la mesure où aucune modification de la communication ne peut se produire sans être détectée.

La mise en oeuvre repose essentiellement sur les générations des différentes clés et certificats pour les élements à sécuriser.

## 7.1 Les élements du "dialogue"

Dans un contexte Web, il y a le client (le navigateur), le serveur (le serveur Web) et l'autorité de certification (CA : Certification Authority).

Dans le contexte MQTT, il y a le client, le broker et l'autorité :

① L'infrastructure de clé publique (Public Key Infrastructure, ou PKI) désigne l'ensemble des serveurs servant à signer, distribuer et valider les certificats.

Une PKI est composée d'une autorité de certification (CA), d'une autorité de dépôt (Repository) et d'une liste de révocation de certificats.

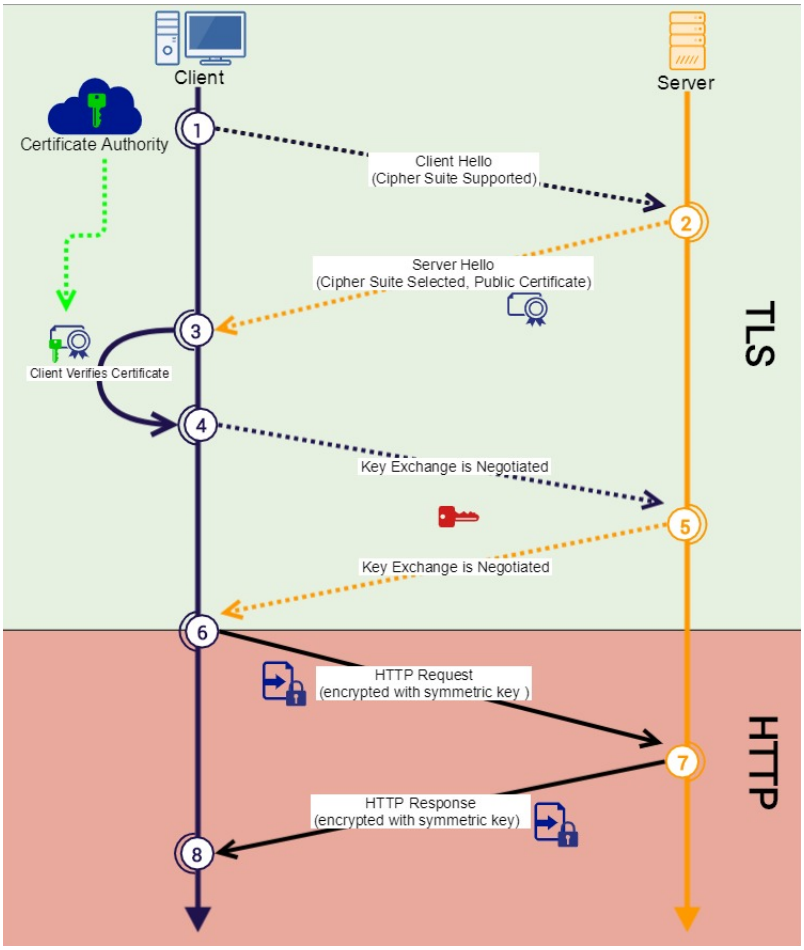
L'élément principal d'une PKI est donc l'**autorité de certification (CA)**.

- ➢ C'est elle qui signe les certificats numériques faisant référence à son autorité.

Dans une pratique "professionnelle", le broker (ou plutôt l'administrateur du broker) paye une autorité de certification de confiance internationale (par exemple, VeriSign, DigiCert) pour signer un certificat pour le domaine où le broker est hébergé (par exemple, "AWS IoT" utilise l'autorité de certification Verisign).

Mais nous on est povvvvvvvre ;-) et donc on va s'"auto certifier" !

② Le **client MQTT** (idem un navigateur) entame un dialogue en spécifiant au serveur le type de cryptage supporté.




by https://medium.com/iocscan/transport-layer-security-tls-ssl-8e02b6d1d648

This cipher suite is composed of multiple parts with various algorithms for each part.

- ✔ Authentication Algorithm - Determines how authentication of both parties is performed (relates to certificates) - Provides Authentication
- ✔ Key Exchange Algorithm - Determines how encryption keys (keys used to encrypt the data) are exchanged
- ✔ Bulk Encryption Algorithm - Determines which algorithm to use to encrypt the data between client and server - Makes the data Private
- ✔ Message Authentication Code Algorithm - Determines which algorithm to verify the integrity of the data - Makes the data Reliable

Before a client application and a server can exchange data over a SSL/TLS connection, these two parties need to agree first on a common set of algorithms to secure the connection. If the two parties fail to reach an agreement, then a connection won't be established.

Pour que la négociation SSL/TLS ait lieu, l'administrateur système du serveur doit avoir préparé au minimum 2 fichiers :

- ✔ sa clé privée,
- ✔ et le certificat du serveur (qui contient sa clé publique).

Pour obtenir ce certificat, il s'est adressé à une autorité de certification (i.e. "CA") à laquelle il a du fournir plusieurs informations :

- ✔ Le nom du serveur web/broker,
- ✔ La compagnie,
- ✔ Sa localisation,
- ✔ Sa clé publique, ...

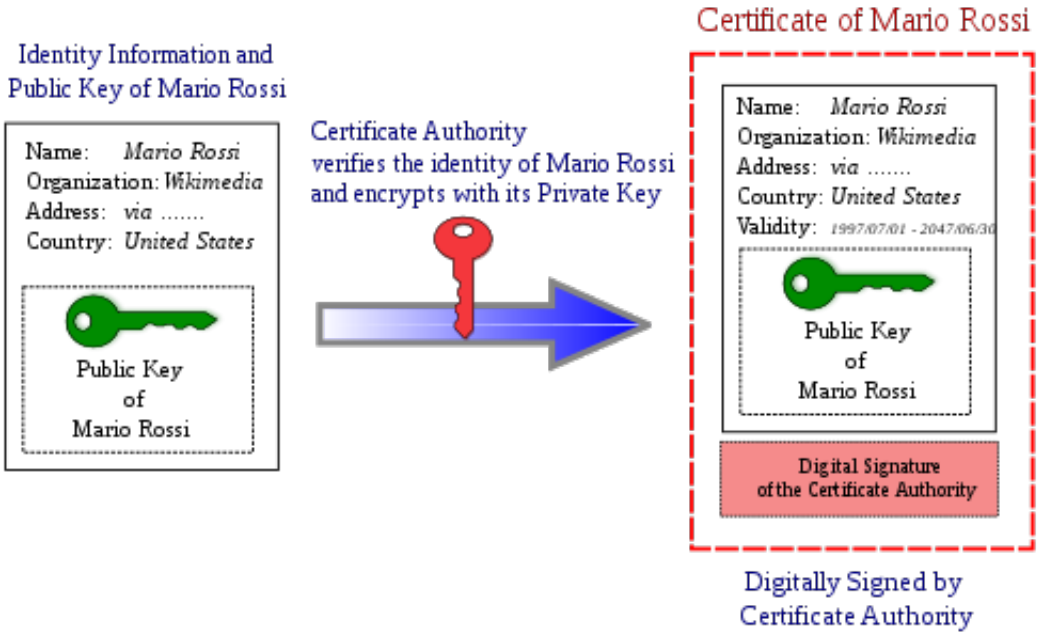
Cette autorité fait des vérifications éventuellement manuelles ... forcément ça coûte de l'argent (périodiquement car les certificats ont une durée de vie) !

En cas de demande de ce certificat auprès d'une autorité de certification telle que DigiCert Trust Services, un fichier supplémentaire doit être créé :

- ➢ Certificate Signing Request (CSR), généré à partir de la clé privée.

Si la CA est satisfaite de son enquête, elle émet un **certificat signé** (dont la signature est cryptée).

Ce certificat devient "celui du serveur" :




by https://commons.wikimedia.org/wiki/File:PublicKeyCertificateDiagram_It.svg

**Le serveur présentera ce certificat aux différents clients qui souhaitent être certains de son identité.**

Toute personne avec la clé publique de la CA peut récupérer la signature du certificat du serveur ... et ainsi pouvoir valider cette signature !

③ Le serveur répond au client **en renvoyant "son" certificat**.

Dans le certificat du serveur, il y a donc "la" clef publique du serveur ... pour que le client puisse lui envoyer des informations cryptées.

Ceci aprés que le client ait vérifié que le certificat que le serveur lui envoie à bien été signé par la CA !

④ Le navigateur/client reçoit puis authentifie le certificat du serveur grâce au certificat de l'Autorité de Certification (CA certificat) intégré nativement dans le navigateur.

➢ On a vu que le navigateur contient un "trusted store" de certificats CA.

Il vérifie que la clé de chiffrement du certificat du serveur reçu est la même que celle du certificat de la CA qu'il détient :

✔ L'identité du serveur est ainsi confirmée (ou pas).

⑤ Ensuite client et serveur vont mettre en place une clé symétrique ("session key") pour poursuivre le dialogue :

Le client (en fonction du chiffrement) crée le secret pré-maître pour la session, le chiffre avec la clé publique du serveur et envoie le secret pré-maître chiffré au serveur.


by https://www.ssl.com/article/ssl-tls-handshake-overview/

## 7.2 TODO

Sur la base de quelques expériences,

- ➢ https://mosquitto.org/man/mosquitto-tls-7.html
- ➢ http://www.steves-internet-guide.com/mosquitto-tls/
- ➢ https://medium.com/himinds/mqtt-broker-with-secure-tls-communication-on-ubuntu-18-04-lts-and-an-esp
- ➢ https://dzone.com/articles/mqtt-security-securing-a-mosquitto-server
- ➢ https://abhatikar.medium.com/secure-iot-for-successful-iot-41e029ac79d2
- ➢ ...

je vous demande de sécuriser votre flux MQTT.
J'ai longtemps galéré sur cette partie ! Merci à Sébastien pour avoir trouvé le bon pointeur :
https://github.com/espressif/arduino-esp32/issues/5021
et surtout
https://github.com/knolleary/pubsubclientpull/851/files

### 7.2.1 Certificat Authority (CA)

Le serveur qui héberge le broker doit posséder un certificat (X.509) correctement signé par une CA (Autorité de Certification)

Dans notre cas, afin d'éviter d'"acheter" un certificat et de le faire signer par une vraie CA, nous allons générer un certificat racine "auto-signé".

- ➢ Et on s'en servira ensuite pour signer le certificat du serveur/broker.

La commande qui suit crée deux choses :

① une "pair key" pour la CA : matérialisée par le fichier "ca.key"

Cette "pair key" :

- ✔ utilise RSA (asymétrique)
- ✔ est de taille 2048,
- ✔ contient à la fois la clef publique et la clef privée.

On va éviter de la laisser traîner !

② Ensuite cette clé permet générer le certificat de l'autorité de certification d'une durée de validité de 10 ans (3650 days) : matérialisé par le fichier "ca.crt"

```
menez@duke:Mosquitto_Conf_TLS$ openssl req -new -x509 -days 365 \
-extensions v3_ca -keyout ca.key -out ca.crt -passout pass:1234

Generating a RSA private key
.....................................................................+++++
..................+++++
writing new private key to 'ca.key'
-----
```

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:Cote d'Azur
Locality Name (eg, city) []:Sophia
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UCA
Organizational Unit Name (eg, section) []:Master Info
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:menez@unice.fr
```

**Petite remarque** : le CN du certificat de la CA est vide !

### 7.2.2 Serveur/Broker

On crée une "pair key" pour le broker (serveur qui va utiliser la CA pour justifier son identification).

```
menez@duke:Mosquitto_Conf_TLS$ openssl genrsa -out mosquitto.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.........................................+++++
.............+++++
e is 65537 (0x010001)
```

Cette "pair key" est matérialisée par le fichier "mosquitto.key"

Maintenant, nous créons une demande de certificat auprès de la CA

- ➢ Lorsque vous remplissez le formulaire, le nom commun (Common Name : CN) est important et est généralement le nom de domaine du serveur/broker. **Ici, j'ai mis l'adresse du serveur abritant le broker.**
- ➢ Comme nous n'avons pas (encore) de domaine, j'utilise l'adresse IP de mon broker.

```
menez@duke:Mosquitto_Conf_TLS$ openssl req -out mosquitto.csr \
-key mosquitto.key -new -subj '/CN=192.168.1.24'
```

Cette demande de "certificat signé" est matérialisée par le fichier "server.csr".

RMQ : Organizational Unit Name
cf 04 de https://medium.com/jungletronics/bulletproof-tls-ssl-mosquitto-e662c62a269b

En temps "normal" on devrait envoyer cette demande à la CA . . . mais dans le cas présent, c'est nous la CA !

- ➢ "On" vérifie donc puis signe le certificat du serveur/broker.

```
menez@duke:Mosquitto_Conf_TLS$ openssl x509 -req -in mosquitto.csr \
-CA ca.crt -CAkey ca.key -CAcreateserial -out mosquitto.crt \
-days 365 -passin pass:1234
```

```
Signature ok
subject=CN = 192.168.1.24
Getting CA Private Key
```

Le certificat du serveur signé est matérialisé par le fichier : "mosquitto.crt"

Ces fichiers doivent entrer dans la configuration du broker. Il faut donc les placer dans mosquitto selon le contenu du fichier de configuration (/etc/mosquitto/conf.d):

```
menez@duke:Mosquitto_Conf_TLS$ ls
ca.crt        ca.key        ca.srl        mosquitto.crt  mosquitto.csr  mosquitto.key

menez@duke:Mosquitto_Conf_TLS$ sudo cp ca.crt      /etc/mosquitto/ca_certificates/
menez@duke:Mosquitto_Conf_TLS$ sudo cp mosquitto.crt  /etc/mosquitto/certs/server.crt
menez@duke:Mosquitto_Conf_TLS$ sudo cp mosquitto.key  /etc/mosquitto/certs/server.key
```

Il faut aussi modifier le fichier de configuration du broker pour signifier l'utilisation de TLS:

```
1  #https://mosquitto.org/man/mosquitto-conf-5.html
2  # Global
3  #log_dest file /var/log/mosquitto/mosquitto.log
4  per_listener_settings false
5
6  # Default listener
7  listener 1883 0.0.0.0
8
9  # Security
10 allow_anonymous false
11 password_file /etc/mosquitto/passwd
12
13 # Certificate listener
14 listener 8883
15 protocol mqtt
16 cafile   /etc/mosquitto/ca_certificates/ca.crt
17 certfile /etc/mosquitto/certs/server.crt
18 keyfile  /etc/mosquitto/certs/server.key
19 #tls_version tlsv1.1
20 require_certificate true
21 use_identity_as_username false
22
23 #When using certificate based encryption there are three options that
24 #affect authentication.
25 # a) The first is require_certificate, which may be set to true or false.
26
27 #If false, the SSL/TLS component of the client will verify the server
28 #but there is no requirement for the client to provide anything for
```

```
29 #the server: authentication is limited to the MQTT built in
30 #username/password.
31
32 #If require_certificate is true, the client must provide a valid
33 #certificate in order to connect successfully.
34
35 #In this case, the second and third options, use_identity_as_username
36 #and use_subject_as_username, become relevant.
37
38 #If set to true, use_identity_as_username causes the Common Name (CN)
39 #from the client certificate to be used instead of the MQTT username
40 #for access control purposes. The password is not used because it is
41 #assumed that only authenticated clients have valid certificates. This
42 #means that any CA certificates you include in cafile or capath will
43 #be able to issue client certificates that are valid for connecting to
44 #your broker.
45
46 #If use_identity_as_username is false, the client must authenticate as
47 #normal (if required by password_file) through the MQTT options. The
48 #same principle applies for the use_subject_as_username option, but
49 #the entire certificate subject is used as the username instead of
50 #just the CN.
51
52
```

...et ne pas oubliez de redémarrer le broker :

```
sudo service mosquitto restart
```

## 7.3 Un client ESP

Pour l'ESP, on procède à l'identique :

```
menez@duke:Mosquitto_Conf_TLS$ openssl genrsa -out esp.key 2048

Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................................+++++
..............+++++
e is 65537 (0x010001)

menez@duke:Mosquitto_Conf_TLS$ openssl req -out esp.csr -key esp.key -new

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
```

```
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:PACA
Locality Name (eg, city) []:Sophia
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UCA
Organizational Unit Name (eg, section) []:Master
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:menez@unice.fr

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

menez@duke:Mosquitto_Conf_TLS$ openssl x509 -req -in esp.csr -CA ca.crt \
-CAkey ca.key -CAcreateserial -out esp.crt -days 365 -passin pass:1234

Signature ok
subject=C = FR, ST = PACA, L = Sophia, O = UCA, OU = Master, emailAddress = menez@unice.fr
Getting CA Private Key
```

On récupère ainsi les clés, le certificat et le certificat request de l'ESP dans les fichiers :

- ➢ esp.key
- ➢ esp.crt
- ➢ esp.csr

Il reste à intégrer cela dans le code de l'ESP. Vous aurez aussi besoin du certificat de la CA authority.

```
/*********
 Based on Rui Santos work :
 https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/
 Modified by GM
*********/
#include <WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>
#include "ArduinoJson.h"
#include "classic_setup.h"
#include "OneWire.h"
#include "DallasTemperature.h"
#include "sensors.h"
#include "WiFiClientSecure.h"

const char* CA_cert = \
"-----BEGIN CERTIFICATE-----\n" \
"MIIDyzCCArOgAwIBAgIURjL+WGuyCOowJSBCDh/ATao+1mwwDQYJKoZIhvcNAQEL\n" \
"BQAwdTELMAkGA1UEBhMCRlIxEzARBgNVBAgMClNvbWUtU3RhdGUxDzANBgNVBAcM\n" \
"BlNvcGhpYTEhMB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMRowGwYJ\n" \
"KoZIhvcNAQkBFg5tZW5lekB1bmljZS5mcjAeFw0yMjEwMTgxODQxMTdaFw0yMzEw\n" \
"MTgxODQxMTdaMHUxCzAJBgNVBAYTAkZSMRMwEQYDVQQIDApTb211LVN0YXRlMQ8w\n" \
```

```
"DQYDVQQHDAZTb3BoaWExITAfBgNVBAoMGEludGVybmV0IFdpZGdpdHMgUHR5IEx0\n" \
"ZDEdMBsGCSqGSIb3DQEJARYObWVuZXpAdW5pY2UuZnIwggEiMA0GCSqGSIb3DQEB\n" \
"AQUAA4IBDwAwggEKAoIBAQC/HKkV28JFE9iug7CtFtd2tRNJkutvUvrky78EfIvQ\n" \
"Atd8VYUS5fWIp1SwWPnlDBV3RwDYOXm+FgtSebvLovw48IZ1bw/TXdvErtYwSmme\n" \
"k23CcqswCTmddcP0XCTwQcLVqrujajqhjUhaDfuWksuQHmqiBXdb9yfr8os7YndI\n" \
"AKIhezBK+jVf9YsReRg4RutAcn+GQwBa9acIikvEee+NbRiK+sU3OjLykMWbak1a\n" \
"YY81N6d4MHJsqvDffyx13l+/DiICdjQ6hkkQ2yRPHt3K1w7+T+6ejwvtdIInB3tH\n" \
"aak66jXfOjyqOwpTRICqXlddqhISatJeLmDMU6u1aJxLAgMBAAGjUzBRMB0GA1Ud\n" \
"DgQWBBRzIOHC/2RH1e1OuRLO3PJkUMdaDDAfBgNVHSMEGDAWgBRzIOHC/2RH1e10\n" \
"uRLO3PJkUMdaDDAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3DQEBCwUAA4IBAQCQ\n" \
"QIEoEng++s2JoxxQO2GBV9b9GHYcLO9MGvfEjTp4Fh8eD1RePSwd+mML5KcuqTSo\n" \
"tgUg+DEn9GD1FJXJgGyJ+K62D1OmCv5xLl0HCSVPxaP3+9VsERow+bOq8RO5l7M/\n" \
"WiDGPD9TufFQOOu+W9BoYi2TFJ29rn35pB3V9hmiRi6dlQqVepJvmbwO6bQPbjO1\n" \
"/ExFuVuL6DRxJi9pbBuAPoI7NkqVLAYAHi+Q68q8fxfIZOK7tZOYKyK2m2wKdVXT\n" \
"DyhT8gHcWpVtCD1a/Hh2a3itoNO9smyHByb9VXlhd3Qdiqwsr VK1bME7ANNToaFk\n" \
"lMxTipDVOgju5ODNLI3t\n" \
"-----END CERTIFICATE-----";


const char* ESP_CA_cert = \
"-----BEGIN CERTIFICATE-----\n" \
"MIIDYDCCAkgCFBsEb2dUp6CmFvzAF42K5m5sgChLMA0GCSqGSIb3DQEBCwUAMHUx\n" \
"CzAJBgNVBAYTAkZSMRMwEQYDVQQIDApTb211LVNOYXRlMQ8wDQYDVQQHDAZTb3Bo\n" \
"aWExITAfBgNVBAoMGEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDEdMBsGCSqGSIb3\n" \
"DQEJARYObWVuZXpAdW5pY2UuZnIwHhcNMjIxMDE4MTg0MjQ4WhcNMjMxMDE4MTg0\n" \
"MjQ4WjBkMQswCQYDVQQGEwJGUjETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UE\n" \
"CgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMR0wGwYJKoZIhvcNAQkBFg5tZW5l\n" \
"ekB1bmljZS5mcjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAN0zHhLd\n" \
"agE/MV8gHzHbDpLUmQG8gGCqXrtlnXIzIWuHdjSrVPHzDL7H6U5XmfA+1O+Vlq00\n" \
"gZayrlhQbM8ImppUN8aXBG27bMe+2HCbD8j6mdzadnrlk3PMEiwTm+F1jmZJ2Xa+\n" \
"91Y/Kukrw178xzqFJWvauS9iHa3M7jgxMKCuTbWFXrNiyvQUV93RZ24bem4urdWX\n" \
"pa21CNpXAdGgnY+10VX4qgPt1tq1AR58sBPB3CF4gCanSxdReCFbc4/ApyznDB74\n" \
"njx4IQl3ddtnrZjjKiK818rfkRrMvFfOMzO9p7kQOr4f3Rg2LOHMLHnAN3l0tbTS\n" \
"HNpwdqc5w38AVJECAwEAATANBgkqhkiG9w0BAQsFAAOCAQEAdbWORTJAUXi7SOL4\n" \
"ydn+ZU6hk1EUpe6AokO3NJE2JJtPE9/t5gPwhERGPVufPF0hMDL1B5HEPawm9GHm\n" \
"zSb/A9pEx5KcLiZbGPlMrrwSFl0xq01lC8QpIAhom7E6aXmWYwyouIPz9jJ6lr54\n" \
"B9mAnSmETIweVMZFK7k+Uy1a2TDX3Y8Ynifj4AHtJVmCE0xvrb8R/sLqeFPobOZ1\n" \
"kSCCrTzRZIYvpo5rdbhUNrdSMbz/iZHKTTslWOjAPU1QNYLuMsYbok4RNqy9PBdI\n" \
"qV1/Ee7McwCKm2bAOvhSXfmj3P38/dP/tIoJ/vc6UeBgtRJV1N/H3nOALOMN9Ae6\n" \
"c7Ea3g==\n" \
"-----END CERTIFICATE-----";


const char* ESP_RSA_key= \
"-----BEGIN RSA PRIVATE KEY-----\n" \
"MIIEpAIBAAKCAQEA3TMeEt1qAT8xXyAfMdsOktSZAbyAYKpeu2WdcjMha4d2NKtU\n" \
"8fMMvsfpTleZ8D7U75WWrTSBlrKuWFBszwiamlQ3xpcEbbtsx77YcJsPyPqZ3Np2\n" \
"euWTc8wSLBOb4XWOZknZdr73Vj8q6SvDXvzHOoUla9q5L2IdrczuODEwoK5NtYVe\n" \
"s2LK9BRX3dFnbht6bi6t1ZelrbUI2lcB0aCdj7XRVfiqA+3W2rUBHnywE8HcIXiA\n" \
"JqdLF1F4IVtzj8CnLOcMHviePHghCXd122etmOMqIrzXyt+RGsy8V84zM72nuRA6\n" \
"vh/dGDYs4cwsecA3eXS1tNIc2nB2pznDfwBUkQIDAQABAoIBAQC93ijMTJ23IEUC\n" \
"wBHGu59QzYfgg6s2TkcuV9TDt7vSDt6Z6w7o95VTFUnf4zXRkDO6wY4qck1hy1H3\n" \
"oGElfN8n5W5S2kIE5sJC2j/yuEbeXqO/c44Lg5AD74ERsGs3Ja7rgixVzymDMIlm\n" \
"yBfVKujgBHXQlFgsoe2Tx7obJa313M9ndgluozejODd6y1NB1ceT9aZR49sOZrql\n" \
"wwX6Usx7hMvBCeIbv/KlRTgIxxgTMiz/D/KlmJHE1kIZVsTsCIVfkHOZPUcDSMQ5\n" \
"N5DKloQfOp/UqeIWz6mUfBETJwHdeDYD8P1M8EUtkTScWhni+azILbsClakSxLVZ\n"
```

```
"7HAw3qgBAoGBAPHjw02RPU9qY+PNAwJJqdRjzXvV4Wkb7SasmSxJL7ROzy3OG9Le\n" \
"VPEhdIoMYkANVp+A28VrIEIFm2nvmqODC+gIFqtLek0/kPcyWHuF0lBVs1RfK7QL\n" \
"Pc1mLR1fpujmsNYQvTZviKYp42C8TGCEThIsI3ir5p6NSWi0/WclVy3BAoGBAOoa\n" \
"YdBPE/Og4Xf03q11TDBuYdcagY+9DPgC2OmYh3OVh5zQg1k4Gbp1U88jN9vumVOK\n" \
"HjEJcMafDKTBIqvwpf7mYwwOKRQ2OTsbJ/9bkPu0LfSMd6Iz4y9QFHy5Qt9a9Pmm\n" \
"LkuglGIqByvxTGYnGydqDHYlUqb0xvpLBcmEhnrRAoGAQQpzO2owSChdT6WSh4zz\n" \
"I27qk7/A6phdviZu0V+keitmgfSHur2ZrXTHd085I+vyPNVTQtQ5SkdjsKqCtDDK\n" \
"hMuX/NtAuYC8JBvfeudSO9MXczpb7/+NOKsGun07zigpqIW7uuVmovdkcmBvUQyC\n" \
"SZSgvRPL4RowgP0ZniCGPoECgYBGt2N2vnyDasTRLKobccRrGNz7Umly1yPUw19s\n" \
"6QN5ueC2URJsYBL9jTWc68GS8Tdl0eIe5AAqL59tIV5s3Av4vPWMjbaBAbIgjap+\n" \
"a9WXAy2ios/8snndL+6QYpI90EcSJs3zUHWiVZnG/2QZs5RzSoirI8kc3/I/Z2w/\n" \
"HIZXEQKBgQDQkyOVMSjflcw+45ehYzDxz8GCa0oQ3FivKnTBE2G6Ge0W062BtPNv\n" \
"MKG9sig8CHVnAs7ltbk93g2ab7BbI1RG7rHg4QvzWjtZ0kUlXMNf013LsggHI17u\n" \
"9M2ld1rsAWOfQL1jeyMDtCxMFleezKtyjvqmJJiryei6gR5F7VGbhQ==\n" \
"-----END RSA PRIVATE KEY-----";

/*===== MQTT broker/server ========*/
const char* mqtt_server = "192.168.1.24";
//const char* mqtt_server = "public.cloud.shiftr.io"; // Failed in 2021
// need login and passwd (public,public) mqtt://public:public@public.cloud.shiftr.io
//const char* mqtt_server = "broker.hivemq.com"; // anynomous Ok in 2021
//const char* mqtt_server = "test.mosquitto.org"; // anynomous Ok in 2021
//const char* mqtt_server = "mqtt.eclipseprojects.io"; // anynomous Ok in 2021

/*===== MQTT TOPICS ================*/
#define TOPIC_SENSORS "esp/sensors" //
#define TOPIC_FIRE_EMERGENCY  "emergency/fire"  // Topic pour communication entre ESP et pompiers
#define TOPIC_FIREFIGHTERS_EMERGENCY "emergency/firefighters"
#define TOPIC_COMMAND_CENTER "command/center" // Topic pour communication entre ESP et centre de commande hors sensors


/*===== ESP is MQTT Client ========*/
WiFiClientSecure espClient;           // Wifi
PubSubClient client(espClient); // MQTT client
const char* mqtt_user = "darkvador";
const char* mqtt_pass = "6poD2R2";

/*============== GPIO ==============*/
const int ledPin = 19; // LED Pin

/* ---- TEMP ---- */
OneWire oneWire(23); // Pour utiliser une entite oneWire sur le port 23
DallasTemperature tempSensor(&oneWire) ; // Cette entite est utilisee par le capteur de temperature
const int LightPin = A5;


String temperature = "";
String light = "";
String state = "NORMAL";
String location = "5eme etage";
boolean HelpOnTheWay = false;
const char* emergencyStatus = "";
StaticJsonDocument<256> jdoc;
char jpayload[256];

/*===== Arduino IDE paradigm : setup+loop =====*/
```

```
135 void setup() {
136   Serial.begin(9600);
137   while (!Serial); // wait for a serial connection. Needed for native USB port only
138 
139   connect_wifi(); // Connexion Wifi
140   print_network_status();
141 
142   // Initialize the output variables as outputs
143   pinMode(ledPin, OUTPUT);
144   digitalWrite(ledPin, LOW);// Set outputs to LOW
145 
146   // Init temperature sensor
147   tempSensor.begin();
148 
149   // set server of our client
150   espClient.setCACert(CA_cert);
151   espClient.setCertificate(ESP_CA_cert);
152   espClient.setPrivateKey(ESP_RSA_key);
153   client.setServer(mqtt_server, 8883);
154   // set callback when publishes arrive for the subscribed topic
155   client.setCallback(mqtt_pubcallback);
156 }
157 
158 /*============= TO COMPLETE ==================*/
159 void set_LED(int v){
160   digitalWrite(ledPin, v);
161 }
162 
163 void blinding_LED(){
164   if(digitalRead(ledPin)){
165      set_LED(LOW);
166   } else {
167      set_LED(HIGH);
168   }
169 }
170 
171 /*============== CALLBACK ==================*/
172 void mqtt_pubcallback(char* topic,
173                       byte* message,
174                       unsigned int length) {
175   /*
176    * Callback if a message is published on this topic.
177    */
178   Serial.print("Message arrived on topic : ");
179   Serial.println(topic);
180 
181   StaticJsonDocument<256> doc;
182   deserializeJson(doc, message, length);
183 
184   // Feel free to add more if statements to control more GPIOs with MQTT
185 
186   // If a message is received on the topic,
187   // you check if the message is either "normal" or "onFire".
188   // Changes the output state according to the message
189 
190    if (String(topic) == TOPIC_FIREFIGHTERS_EMERGENCY) {
```

```
191       Serial.print("Message received from firefighters ");
192       if (doc["status"] == "en route" && doc["state"] == "fire") {
193         HelpOnTheWay = true;
194         emergencyStatus = doc["status"];
195         createJsonCommandCenter();
196         client.publish(TOPIC_COMMAND_CENTER,jpayload);
197       } else if(doc["status"] == "sur place" && doc["state"] == "fire"){
198         emergencyStatus = doc["status"];
199         createJsonCommandCenter();
200         client.publish(TOPIC_COMMAND_CENTER,jpayload);
201       } else if(doc["status"] == "retour en caserne" && doc["state"] == "normal"){
202         emergencyStatus = doc["status"];
203         state = "NORMAL";
204         createJsonCommandCenter();
205         client.publish(TOPIC_COMMAND_CENTER,jpayload);
206         set_LED(LOW);
207         HelpOnTheWay = false;
208         emergencyStatus = "";
209       }
210     }
211   }
212
213   void createJsonSensors(){
214     jdoc.clear();
215     jdoc["temperature"] = temperature;
216     jdoc["light"] = light;
217     jdoc["location"] = location;
218     jdoc["ip"] = WiFi.localIP().toString();
219     serializeJson(jdoc, jpayload);
220   }
221
222   void createJsonEmergency(){
223     jdoc.clear();
224     jdoc["emergency"] = "FIRE EMERGENCY";
225     jdoc["location"] = location;
226     jdoc["ip"] = WiFi.localIP().toString();
227     serializeJson(jdoc, jpayload);
228   }
229
230   void createJsonCommandCenter(){
231     jdoc.clear();
232     jdoc["state"] = state;
233     jdoc["emergencyStatus"] = emergencyStatus;
234     jdoc["location"] = location;
235     jdoc["ip"] = WiFi.localIP().toString();
236     serializeJson(jdoc, jpayload);
237   }
238
239   // On utilise la lumière pour détecter un feu, car plus complexe de changer la température
240   void detectFire(){
241     if(light.toInt()>2200){
242       state = "ONFIRE";
243     }
244   }
245
246   // Si un feu est détecté ou si le centre de commande prévient d'un incendie, l'esp envoit ses coordonnées au pompiers
```

```
247 void callForHelp(){
248     createJsonEmergency();
249     client.publish(TOPIC_FIRE_EMERGENCY, jpayload);
250 }
251 
252 /*============= SUBSCRIBE ======================*/
253 void mqtt_mysubscribe(char *topic) {
254   /*
255    * Subscribe to a MQTT topic
256    */
257   while (!espClient.connected()) { // Loop until we're reconnected
258 
259     Serial.print("Attempting MQTT connection...");
260     // Attempt to connect => https://pubsubclient.knolleary.net/api
261     if (client.connect("esp32", /* Client Id when connecting to the server */
262                                     mqtt_user,    /* No credential */
263                                     mqtt_pass)) {
264       Serial.println("connected");
265       // then Subscribe topic
266       client.subscribe(topic);
267     } else {
268       Serial.print("failed, rc=");
269       Serial.print(client.state());
270 
271       Serial.println(" try again in 5 seconds");
272       delay(5000); // Wait 5 seconds before retrying
273     }
274   }
275 }
276 
277 /*================= LOOP ======================*/
278 void loop() {
279   int32_t period = 5000; // 5 sec
280   unsigned long startTime = 0;
281   startTime = millis();
282 
283   while(millis() < startTime + period){
284     delay(1000);
285     if(state == "ONFIRE"){
286     blinding_LED();
287   }
288   }
289   /*--- subscribe to TOPIC_LED if not yet ! */
290   if (!client.connected()) {
291     mqtt_mysubscribe((char *)(TOPIC_FIREFIGHTERS_EMERGENCY));
292   }
293   /*--- Publish Temperature periodically   */
294   light = get_light(LightPin);
295   temperature = get_temperature(tempSensor);
296   // Serial info
297  // Serial.print("Published Temperature : "); Serial.println(temperature);
298   //Serial.print("Published Light : "); Serial.println(light);
299   // MQTT Publish
300   createJsonSensors();
301   client.publish(TOPIC_SENSORS,jpayload);
302   detectFire();
```

```
303     if(state == "ONFIRE"){
304       if(!HelpOnTheWay){
305         callForHelp();
306       }
307       blinding_LED();
308     }
309
310
311     /* Process MQTT ... une fois par loop() ! */
312     client.loop();
313   }
```

## 7.4 Le client Python TLS

```python
#https://github.com/eclipse/paho.mqtt.python#subscribe-unsubscribe

import time
import paho.mqtt.client as mqtt
import ssl

#define callbacks
def on_message(client, userdata, message):
  print("received message =",str(message.payload.decode("utf-8")))

def on_log(client, userdata, level, buf):
  print("log: ",buf)

def on_connect(client, userdata, flags, rc):
  print("publishing ")

  client.publish("muthu","muthupavithran",)

client=mqtt.Client()
client.on_message=on_message
client.on_log=on_log
client.on_connect=on_connect
print("connecting to broker")

#==>  Port 1883, mdp et pas de TLS
#client.username_pw_set("darkvador", password="6poD2R2")
#client.connect("192.168.1.101", 1883, 60)

#===> TlS
client.tls_set("./ca.crt",certfile="./client.crt", keyfile="./client.key")
client.tls_insecure_set(True)
client.username_pw_set("darkvador", password="6poD2R2")
client.connect("192.168.1.101", 8883, 60)



##start loop to process received messages
client.loop_start()
#wait to allow publish and logging and exit
time.sleep(1)
```