

TP3

Gilles Menez - UNS - UFR Sciences - Dépt. Informatique

4 février 2022

Objectifs pédagogiques

- ✓ Protocole application : MQTT

Table des matières

1	Installation Mosquitto	3
2	Test fonctionnel du client et du serveur	5
2.1	Dump du serveur	5
2.2	Un serveur public ?	6
3	API MQTT pour ESP32	7
3.1	Installation de l'API	7
4	Expérimentation	8
4.1	Coté ESP32	9
4.2	Coté "console" / Client Mosquitto	13
4.3	TO TRY	14
4.4	Client en Python	15
4.4.1	Le callback : "on_message"	16
4.4.2	Les boucles	16
4.4.3	Client en Java	16
4.4.4	TO TRY	16
5	Node-Red et MQTT	17
6	TODO	19
7	Sécurité	20
7.1	Autorisation des utilisateurs	20
7.1.1	Impact au niveau du client	21
7.1.2	Validation des topics/utilisateurs	21
7.2	Sécurisation des flux	22
7.3	La fausse bonne idée!	22
7.4	Chiffrement	22
7.4.1	Chiffrement symétrique	23

7.4.2	Chiffrement asymétrique	23
7.5	Authentification et intégrité	24
7.5.1	Cas normal - Échange classique :	24
7.5.2	Cas d'Attaque : MITM	25
7.6	La signature électronique	26
7.6.1	Signature électronique pour transmettre	27
7.7	Certificats	28
7.7.1	Certificat électronique	28
7.7.2	Autorité de certification (CA)	28
7.7.3	Verification du certificat d'un serveur	30
8	Mise en oeuvre : Mosquitto Security	31
8.1	Les éléments du "dialogue"	31
8.2	TODO	36
8.2.1	Certificat Authority (CA)	36
8.2.2	Serveur/Broker	37
8.3	Un client ESP	39
8.4	Le client Python	40
9	Consommation	41
10	Limitations MQTT : Kafka by Apache ?!	41

1 Installation Mosquitto

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

Pour réduire votre dépendance à la disponibilité d'un réseau Internet et comme c'est assez simple à faire, installez client **et** serveur/broker sur votre machine.

<https://mosquitto.org/download/>

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

➤ Le broker :

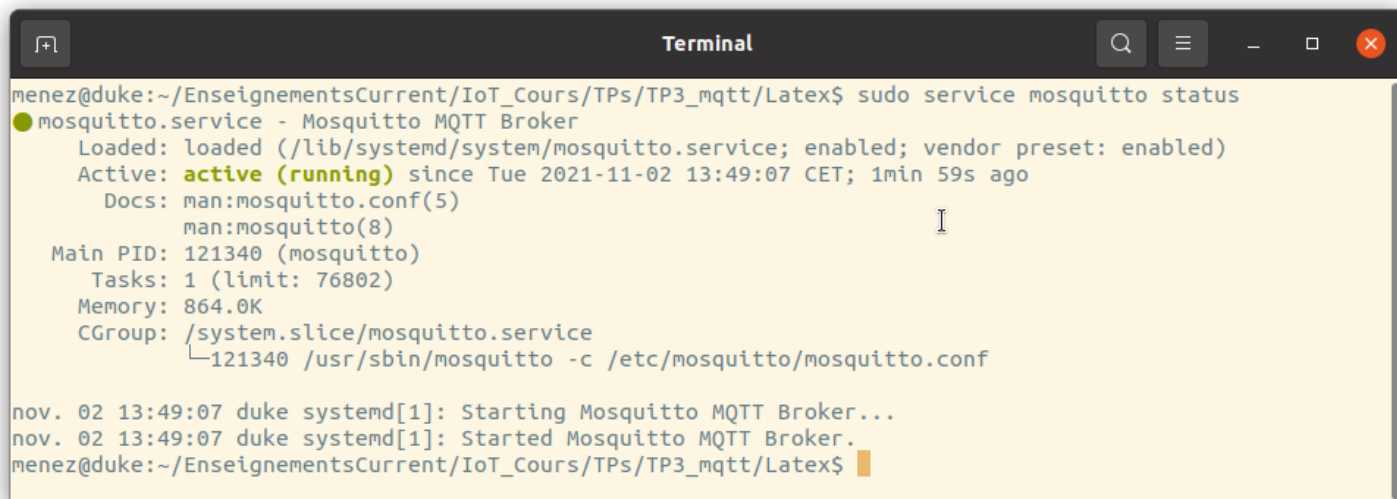
```
sudo apt-get install mosquitto
```

➤ Le client (pub, sub et passw) :

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

Sous Linux, vous pouvez vérifier que le service "tourne" :

```
sudo service mosquitto status
```



```
menez@duke:~/EnseignementsCurrent/IoT_Cours/TPs/TP3_mqtt/Latex$ sudo service mosquitto status
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2021-11-02 13:49:07 CET; 1min 59s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
   Main PID: 121340 (mosquitto)
    Tasks: 1 (limit: 76802)
   Memory: 864.0K
   CGroup: /system.slice/mosquitto.service
           └─121340 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

nov. 02 13:49:07 duke systemd[1]: Starting Mosquitto MQTT Broker...
nov. 02 13:49:07 duke systemd[1]: Started Mosquitto MQTT Broker.
menez@duke:~/EnseignementsCurrent/IoT_Cours/TPs/TP3_mqtt/Latex$
```

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 que le serveur MQTT s'exécute avec le fichier de configuration :

/etc/mosquitto/mosquitto.conf

Ce fichier

- active la persistance, 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 :

```

1  #File : /etc/mosquitto/conf.d/gm.conf
2  #  Changement  => sudo service mosquitto restart
3  #the default config will only bind to localhost as a move to
4  #a more secure default posture compared to previous MQTT versions.
5
6  #Section Extra Listener  :
7  #Running the broker with a listener defined will bind by default to
8  #0.0.0.0 / :: and so will be accessible from any interface.
9  #It is still possible to bind to a specific address/interface !.
10 listener 1883
11 protocol mqtt
12
13 #Section Connection :
14 #By default it will also only allow anonymous connections (without
15 #username/password) from localhost, to allow anonymous from REMOTE
16 #add:
17 allow_anonymous true

```

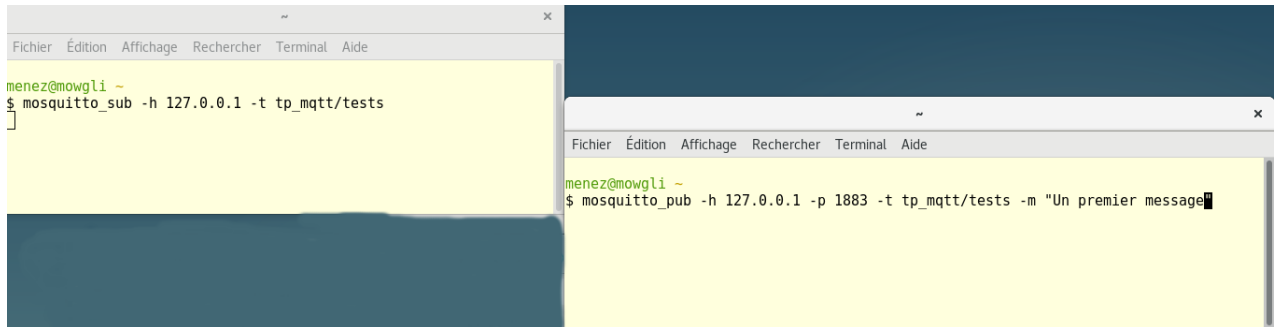
Quelques remarques :

- 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.

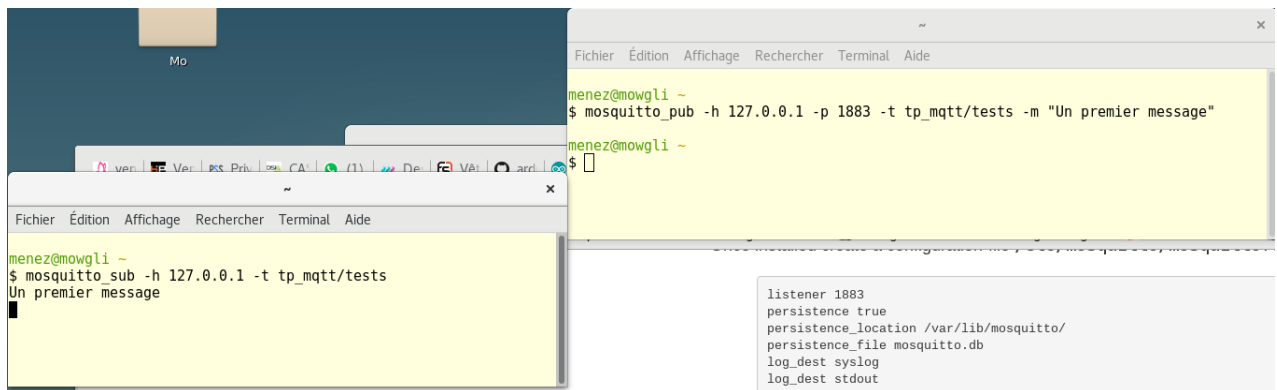
2 Test fonctionnel du client et du serveur

On utilise le serveur que vous venez d'installer :

- 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 ... ça va venir !

2.1 Dump du serveur

Avec cette commande, 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.

2.2 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) :

➤ <https://www.hivemq.com/mqtt-demo/>

"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."

broker.hivemq.com

➤ <https://iot.eclipse.org/getting-started/>

iot.eclipse.org

➤ <https://docs.shiftr.io/interfaces/mqtt/>

broker.shiftr.io

3 API MQTT pour ESP32

Pour utiliser MQTT au niveau de l'ESP et de sa programmation il y a plusieurs possibilités :

- Il y a l'API de Espressif les concepteurs de l'ESP.

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

- Dans le cadre de l'IDE Arduino, nous utiliserons plutôt la bibliothèque : `"pubsubclient"` (by O'Leary) :

`https://github.com/knolleary/pubsubclient`

- **Bien lire les limitations !** ...notamment sur la taille d'un message.

3.1 Installation de l'API

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`

Pour les MacOS ...merci Robin :

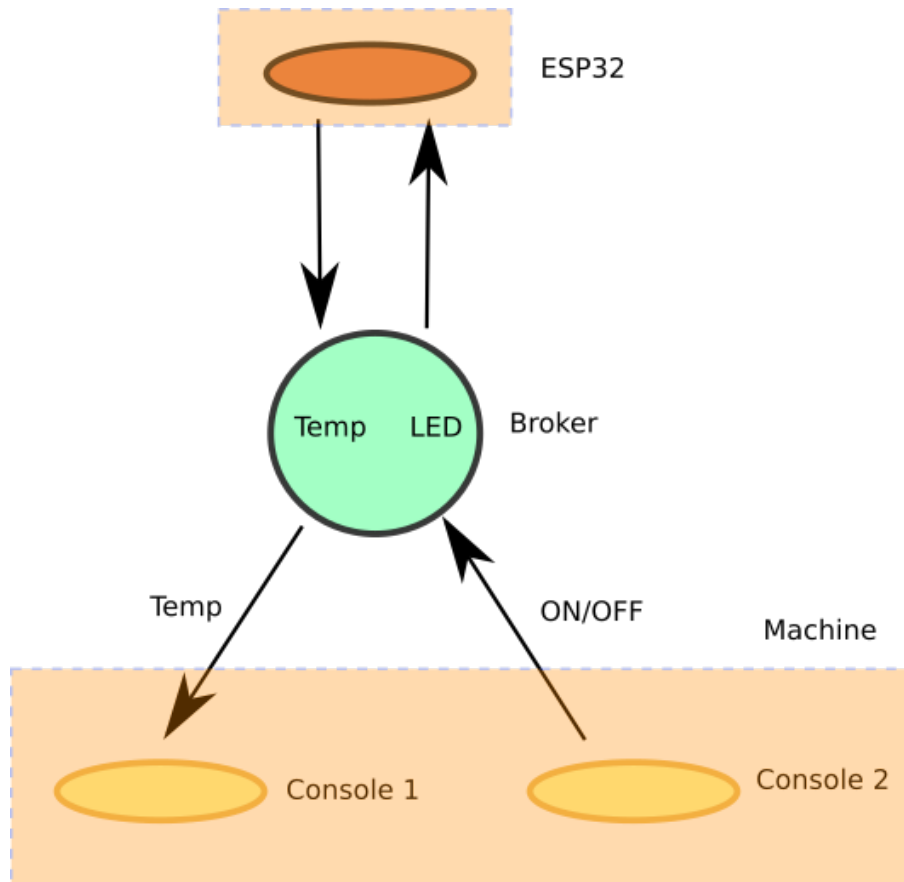
Pour le problème sur MacOS du PubSubClient.h introuvable il faut l'installer manuellement :

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`

4 Expérimentation

Pour mieux appréhender MQTT, vous allez expérimenter une utilisation dans le contexte suivant :

- Un ESP32 et une machine classique (de supervision) utilise un MQTT broker pour "échanger" des informations.



Pour l'ESP, cet échange consiste en :

- ① Publier régulièrement les valeurs de "son" capteur de température.
- ② Recevoir des positionnements ("ON"/"OFF"/...) de sa LED interne (la bleue).

Pour la machine :

- ① S'abonner aux publications de l'ESP et donc pouvoir connaître les évolutions de température du capteur distant.
- ② Pouvoir allumer/éteindre la LED de l'ESP en lui envoyant des ordres/commandes ("ON"/"OFF") à partir des commandes dans une console ou d'une logique programmée.

Pour commencer et notamment simplifier la mise au point du dialogue, au niveau de la machine cliente on va utiliser les commandes "mosquitto" dans un terminal/console pour souscrire au topic de la température et pour publier les commandes de la LED.

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

Mais vous pourriez aussi le faire au niveau d'un programme écrit dans un langage quelconque, le choix est large :

```

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

```

4.1 Coté ESP32

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

➤ <https://github.com/knolleary/pubsubclient>
 ➤ <http://www.steves-internet-guide.com/using-arduino-pubsub-mqtt-client/>

Il faut noter que d'autres clients MQTT existent :

➤ <https://github.com/256dpi/arduino-mqtt>
 ➤ <https://www.esp8266.com/viewtopic.php?t=8172>
 ➤ ...

Au niveau de l'ESP, le code qui suit montre comment :

- ① Associer (mais **pas encore connecter**) à notre ESP (qui est client) un broker : Ligne 52
- ② Ligne 54 : on associe aussi un "callback" au client qui sera invoqué à la réception d'une publication.
- ③ Dans ce callback (Ligne 67), on analyse le message MQTT reçu pour en déduire le topic concerné.
On peut avoir souscrit à plusieurs topics!
- ④ La connexion au broker est réalisée par la fonction Ligne 104.

Cette fonction permet de plus de souscrire à un topic passé en paramètre : Ligne 118

Dans cet exemple, l'utilisation de la fonction "connect" (Ligne 112) est **particulièrement simpliste** :

- ✓ un identifiant "générique" :
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 sera utilisé pour contrôler les connexions au broker.
Il est néanmoins souhaitable que cet identificateur soit unique au niveau du broker ...
- ✓ pas de sécurisation de la connexion :
Le User Id et son mot de passe ne sont pas fournis : NULL, NULL

Cela ne va pas durer !;-)

⑤ Dans la boucle (Ligne 130), périodiquement on publie (Ligne 147) la nouvelle température.

ET on invoque la fonction "loop" du client MQTT : Ligne 151

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

➤ les fonctions `set_LED()` et `get_Temperature()` sont creuses.

```

1  /*****
2   Based on Rui Santos work :
3   https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/
4   Modified by GM
5   *****/
6  #include <WiFi.h>
7  #include <PubSubClient.h>
8  #include <Wire.h>
9  #include "classic_setup.h"
10 #include "OneWire.h"
11 #include "DallasTemperature.h"
12
13
14 /*===== MQTT broker/server =====*/
15 const char* mqtt_server = "192.168.1.101";
16 //const char* mqtt_server = "public.cloud.shiftr.io"; // Failed in 2021
17 // need login and passwd (public,public) mqtt://public:public@public.cloud.shiftr.io
18 //const char* mqtt_server = "broker.hivemq.com"; // anonymous Ok in 2021
19 //const char* mqtt_server = "test.mosquitto.org"; // anonymous Ok in 2021
20 //const char* mqtt_server = "mqtt.eclipseprojects.io"; // anonymous Ok in 2021
21
22 /*===== MQTT TOPICS =====*/
23 #define TOPIC_TEMP "my/sensors/temp"
24 #define TOPIC_LED "my/sensors/led"
25
26 /*===== ESP is MQTT Client =====*/
27 WiFiClient espClient; // Wifi
28 PubSubClient client(espClient); // MQTT client
29
30 /*===== GPIO =====*/
31 const int ledPin = 19; // LED Pin
32
33 /* ----- TEMP ----- */
34 OneWire oneWire(23); // Pour utiliser une entite oneWire sur le port 23
35 DallasTemperature tempSensor(&oneWire) ; // Cette entite est utilisee par le capteur de
    temperature
36
37 float temperature = 0;
38 float light = 0;
39
40 /*===== Arduino IDE paradigm : setup+loop =====*/
41 void setup() {
42     Serial.begin(9600);
43     while (!Serial); // wait for a serial connection. Needed for native USB port only
44
45     connect_wifi(); // Connexion Wifi
46     print_network_status();
47
48     // Initialize the output variables as outputs
49     pinMode(ledPin, OUTPUT);
50     digitalWrite(ledPin, LOW); // Set outputs to LOW
51
52     // Init temperature sensor

```

```

53 | tempSensor.begin();
54 |
55 | // set server of our client
56 | client.setServer(mqtt_server, 1883);
57 | // set callback when publishes arrive for the subscribed topic
58 | client.setCallback(mqtt_pubcallback);
59 | }
60 |
61 | /*===== TO COMPLETE =====*/
62 | void set_LED(int v){
63 |
64 | }
65 |
66 | float get_Temperature(){
67 |     return 22.5;
68 | }
69 |
70 | /*===== CALLBACK =====*/
71 | void mqtt_pubcallback(char* topic,
72 |                       byte* message,
73 |                       unsigned int length) {
74 |     /*
75 |      * Callback if a message is published on this topic.
76 |      */
77 |     Serial.print("Message arrived on topic: ");
78 |     Serial.println(topic);
79 |     Serial.print("=>");
80 |
81 |     // Byte list to String and print to Serial
82 |     String messageTemp;
83 |     for (int i = 0; i < length; i++) {
84 |         Serial.print((char)message[i]);
85 |         messageTemp += (char)message[i];
86 |     }
87 |     Serial.println();
88 |
89 |     // Feel free to add more if statements to control more GPIOs with MQTT
90 |
91 |     // If a message is received on the topic,
92 |     // you check if the message is either "on" or "off".
93 |     // Changes the output state according to the message
94 |     if (String(topic) == TOPIC_LED) {
95 |         Serial.print("so... changing output to ");
96 |         if (messageTemp == "on") {
97 |             Serial.println("on");
98 |             set_LED(HIGH);
99 |         }
100 |         else if (messageTemp == "off") {
101 |             Serial.println("off");
102 |             set_LED(LOW);
103 |         }
104 |     }
105 | }
106 |
107 | /*===== SUBSCRIBE =====*/
108 | void mqtt_mysubscribe(char *topic) {
109 |     /*
110 |      * Subscribe to a MQTT topic
111 |      */
112 |     while (!client.connected()) { // Loop until we're reconnected
113 |
114 |         Serial.print("Attempting MQTT connection ... ");

```

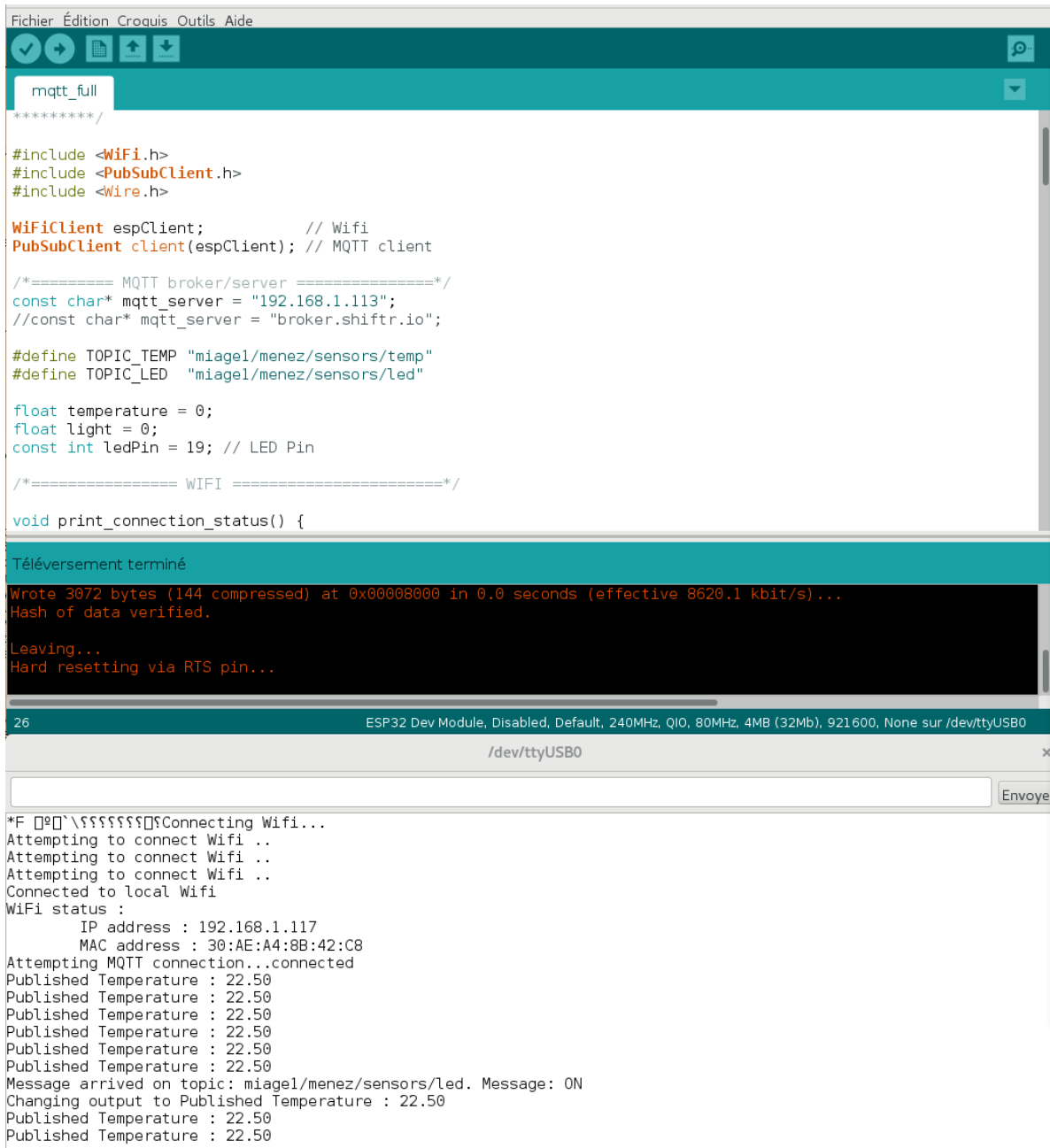
```

115 // Attempt to connect => https://pubsubclient.knolleary.net/api
116 if (client.connect("esp32", /* Client Id when connecting to the server */
117                  NULL, /* No credential */
118                  NULL)) {
119     Serial.println("connected");
120     // then Subscribe topic
121     client.subscribe(topic);
122 } else {
123     Serial.print("failed ,rc=");
124     Serial.print(client.state());
125
126     Serial.println("try again in 5 seconds");
127     delay(5000); // Wait 5 seconds before retrying
128 }
129 }
130 }
131
132 /*===== LOOP =====*/
133 void loop() {
134     int32_t period = 5000; // 5 sec
135
136     /*— subscribe to TOPIC_LED if not yet ! */
137     if (!client.connected()) {
138         mqtt_mysubscribe((char *) (TOPIC_LED));
139     }
140
141     /*— Publish Temperature periodically */
142     delay(period);
143     temperature = get_Temperature();
144     // Convert the value to a char array
145     char tempString[8];
146     dtostrf(temperature, 1, 2, tempString);
147     // Serial info
148     Serial.print("Published Temperature:"); Serial.println(tempString);
149     // MQTT Publish
150     client.publish(TOPIC_TEMP, tempString);
151
152
153     /* Process MQTT ... une fois par loop() ! */
154     client.loop();
155 }

```

4.2 Coté "console" / Client Mosquitto

Cette partie montre l'IDE :



The screenshot shows the Arduino IDE with a file named 'mqtt_full'. The code is as follows:

```

***** /

#include <WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>

WiFiClient espClient;          // Wifi
PubSubClient client(espClient); // MQTT client

/*===== MQTT broker/server =====*/
const char* mqtt_server = "192.168.1.113";
//const char* mqtt_server = "broker.shiftr.io";

#define TOPIC_TEMP "miagel/menez/sensors/temp"
#define TOPIC_LED  "miagel/menez/sensors/led"

float temperature = 0;
float light = 0;
const int ledPin = 19; // LED Pin

/*===== WIFI =====*/

void print_connection_status() {

```

Below the code, a status bar indicates 'Téléversement terminé' (Upload finished). The serial monitor shows the following output:

```

Wrote 3072 bytes (144 compressed) at 0x00008000 in 0.0 seconds (effective 8620.1 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...

26 ESP32 Dev Module, Disabled, Default, 240MHz, QIO, 80MHz, 4MB (32Mb), 921600, None sur /dev/ttyUSB0
/dev/ttyUSB0

*F [0] \\\??????Connecting Wifi...
Attempting to connect Wifi ..
Attempting to connect Wifi ..
Attempting to connect Wifi ..
Connected to local Wifi
WiFi status :
  IP address : 192.168.1.117
  MAC address : 30:AE:A4:8B:42:C8
Attempting MQTT connection...connected
Published Temperature : 22.50
Published Temperature : 22.50
Published Temperature : 22.50
Published Temperature : 22.50
Published Temperature : 22.50
Published Temperature : 22.50
Message arrived on topic: miagel/menez/sensors/led. Message: ON
Changing output to Published Temperature : 22.50
Published Temperature : 22.50
Published Temperature : 22.50

```

- ① Sur le code, on voit le broker et les topics que l'ESP va utiliser.
- ② Sur la console de l'IDE, on voit :
 - ✓ les traces des "publish" émis par l'ESP.
 - ✓ et au milieu, on voit que l'ESP reçoit une publication sur le topic LED auquel il s'est abonné.

The image shows two terminal windows. The top window displays the output of the 'ifconfig' command for a Raspberry Pi, showing details for the loopback interface 'lo' and the Ethernet interface 'enp0s31f6'. It also shows the configuration for the wireless interface 'wlan0'. The bottom window shows the execution of the 'mosquitto_sub' command, which is subscribed to the 'temp' topic on the 'miage1/menez/sensors' MQTT broker. The output of the subscription shows a series of '22.50' values, indicating temperature readings being received by the subscriber.

```

Fichier  Édition  Affichage  Rechercher  Terminal  Aide
inet6 fe80::c509:1a53:2ad7:9834/64 scope link
      valid_lft forever preferred_lft forever
menez ~$ : mosquitto_pub -h 192.168.1.113 -t miage1/menez/sensors/led -m "ON"
menez ~$ : clear
menez ~$ : ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s31f6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 54:bf:64:6b:58:76 brd ff:ff:ff:ff:ff:ff
    inet 134.59.131.45/24 brd 134.59.131.255 scope global enp0s31f6
        valid_lft forever preferred_lft forever
    inet6 fe80::56bf:64ff:fe6b:5876/64 scope link
        valid_lft forever preferred_lft forever
3: enp2s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN group default qlen 1000
    link/ether 54:bf:64:6b:3f:a1 brd ff:ff:ff:ff:ff:ff
5: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:1e:2a:cc:e1:ab brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.116/24 brd 192.168.1.255 scope global dynamic wlan0
        valid_lft 83764sec preferred_lft 83764sec
    inet6 fe80::c509:1a53:2ad7:9834/64 scope link
        valid_lft forever preferred_lft forever
menez ~$ : mosquitto_pub -h 192.168.1.113 -t miage1/menez/sensors/led -m "ON"
menez ~$ :

Fichier  Édition  Affichage  Rechercher  Terminal  Aide
menez ~/EnseignementsCurrent/Cours_IoT/TPs/TP3$ : mosquitto_sub -h 192.168.1.113 -t miage1/menez/sensors/temp
x22.50
22.50
ye 22.50
22.50
22.50
22.50
22.50
22.50
22.50

```

Cet écran montre deux clients/console :

- ① Sur la console du haut, on utilise un client mosquitto pour publier un message sur le broker.
- ② Sur la console du bas, un autre client qui s'est abonné au topic de l'ESP.

4.3 TO TRY

- ① Vous devez vous approprier ces exemples et faire marcher.
Il n'y a pas de "rendu" sur cette partie.

4.4 Client en Python

Je vous propose 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.

```

1  # Let's talk MQTT in Python
2  # first install : https://pypi.org/project/paho-mqtt/
3  # Author : G.MENEZ
4
5  import paho.mqtt.client as mqtt
6  import time
7  import json
8
9  #-----
10
11 def on_message(client, userdata, message):
12     print("\nmessage_received_{}".format(str(message.payload.decode("utf-8"))))
13     print("on_topic_{}".format(message.topic))
14     print("with_qos_{}".format(message.qos))
15     print("with_retain_flag_{}".format(message.retain))
16
17 #=====
18
19 if __name__=="__main__":
20
21     #-----
22     clientname = "P1"
23     print("Creating_new_Client_{}".format(clientname))
24     client = mqtt.Client(clientname) # create new instance
25     client.on_message=on_message # attach function to callback
26
27     #-----
28     broker_address="192.168.1.101"
29     """ broker_address="iot.eclipse.org" """
30     print("Connecting_to_broker_{}".format(broker_address))
31     client.connect(broker_address) # connect to broker
32
33     client.loop_start() #----- start the loop
34
35     #-----
36     topicname = "miagel/menez/sensors/led"
37
38     print("\nSubscribing_to_topic_{}".format(topicname))
39     client.subscribe(topicname)
40
41     """
42     payload = "{
43     payload+="\ "Temperature \ ":10 "
44     payload+=" , "
45     payload+="\ "Humidity \ ":50 "
46     payload+="}"
47
48     payload = {
49         "Temperature":10 ,
50         "Humidity":50 }
51     msg = json.dumps(payload)
52     """
53
54     msg = "off"
55     for i in range(3) : # On publie 3 messages "off"
56         print("\nPublishing_message_{}_to_topic_{}".format(msg, topicname))
57         client.publish(topicname, payload=msg, qos=2, retain=False)

```

```

58         time.sleep(5)
59
60     time.sleep(100)           # wait in seconds before
61     client.loop_stop()       #----- stop the loop and the script

```

4.4.1 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.

4.4.2 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.

4.4.3 Client en Java

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/>

4.4.4 TO TRY

A ce stade, l'essentiel est d'essayer de faire marcher ...

Tous les clients du broker (ESP32, console, programme Python) fonctionnent simultanément !

5 Node-Red et MQTT

Vous vous souvenez de Node red ? ;-)

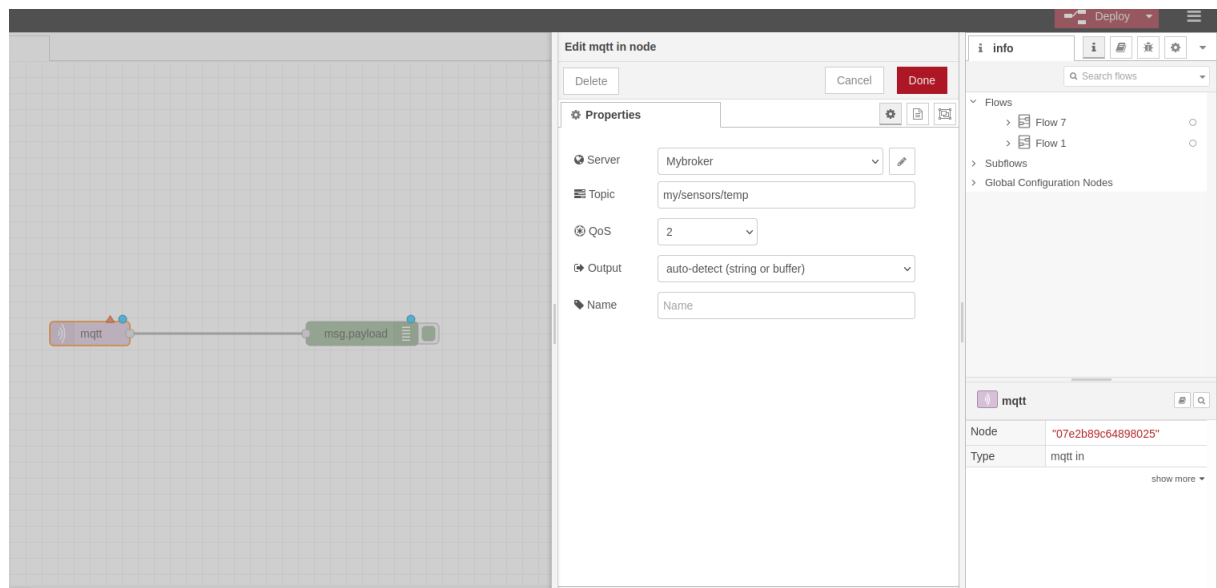
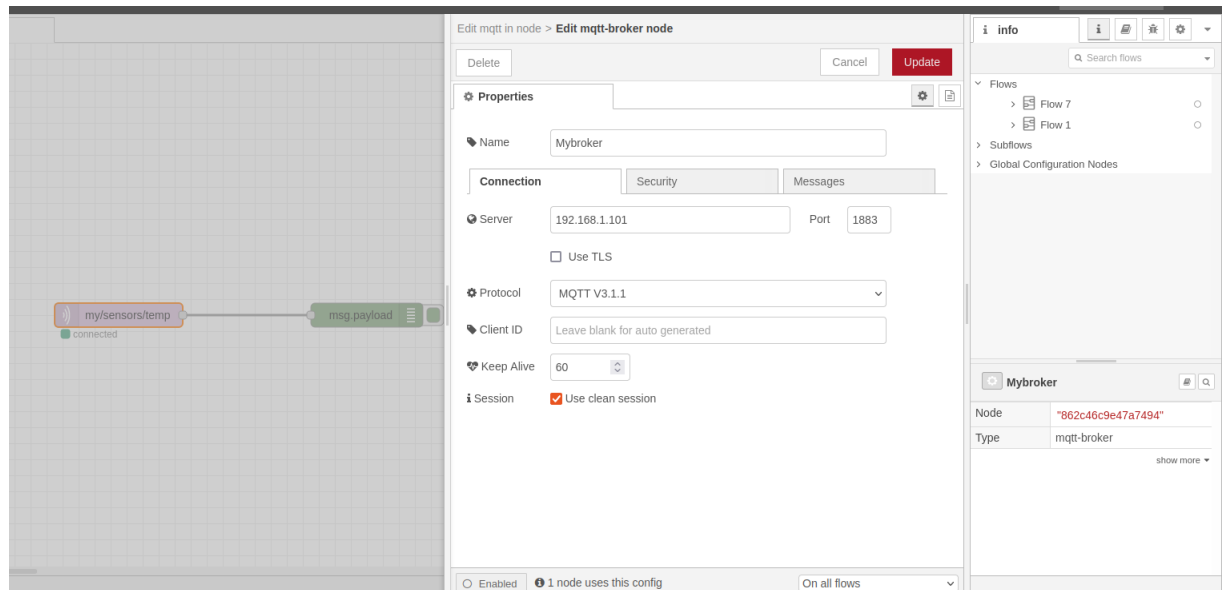
On peut facilement ajouter des noeuds pour participer (PUB/SUB) à des échanges MQTT.

Je redémarre mon container mynodered que j'avais éteint :

```
sudo docker restart mynodered
```

et je rappelle mon onglet dans le navigateur : "http ://localhost :1880"

J'ajoute un noeud "mqtt_in" que je configure, un noeud de debug pour voir que je reçois bien les PUBLISH de l'ESP sur le broker et je déploie !



The screenshot displays the Node-RED web interface. On the left, a flow is visible with a node labeled 'my/sensors/temp' (with a green 'connected' indicator) connected to a 'msg.payload' node. The central panel is titled 'Edit mqtt in node > Edit mqtt-broker node' and contains configuration fields: Name (Mybroker), Connection (selected), Security, and Messages. The Connection tab is active, showing Server (192.168.1.101), Port (1883), Use TLS (unchecked), Protocol (MQTT V3.1.1), Client ID (Leave blank for auto generated), Keep Alive (60), and Session (Use clean session checked). The right panel shows a 'debug' console with a list of messages. The first message is an error: 'RequestError: connect ECONNREFUSED 192.168.1.1:443'. Subsequent messages are temperature readings: '22.50'.

my/sensors/temp connected → msg.payload

Edit mqtt in node > Edit mqtt-broker node

Delete Cancel Update

Properties

Name: Mybroker

Connection Security Messages

Server: 192.168.1.101 Port: 1883

☐ Use TLS

Protocol: MQTT V3.1.1

Client ID: Leave blank for auto generated

Keep Alive: 60

Session: ☒ Use clean session

debug

10/11/2021, 17:08:19 node: Recent Quakes
msg : error
"RequestError: connect ECONNREFUSED 192.168.1.1:443"

10/11/2021, 17:10:23 node: b13e6779d545a573
my/sensors/temp : msg.payload : string[5]
"22.50"

10/11/2021, 17:10:31 node: b13e6779d545a573
my/sensors/temp : msg.payload : string[5]
"22.50"

10/11/2021, 17:10:42 node: b13e6779d545a573
my/sensors/temp : msg.payload : string[5]
"22.50"

10/11/2021, 17:10:42 node: b13e6779d545a573
my/sensors/temp : msg.payload : string[5]
"22.50"

10/11/2021, 17:10:42 node: b13e6779d545a573
my/sensors/temp : msg.payload : string[5]
"22.50"

10/11/2021, 17:10:46 node: b13e6779d545a573
my/sensors/temp : msg.payload : string[5]
"22.50"

10/11/2021, 17:10:53 node: b13e6779d545a573
my/sensors/temp : msg.payload : string[5]
"22.50"

10/11/2021, 17:10:56 node: b13e6779d545a573

6 TODO

- ① Vous devez vous approprier ces exemples et pratiquez MQTT.
- ② Bien sûr vous continuez d'utiliser "Json" au niveau des payloads/messages.
- ③ Sur la base de la maquette de régulation de température, vous réalisez un service d'urgence permettant d'alerter en cas d'incendie !
 - Vous devez détecter l'incendie au niveau de l'objet.
Évidemment la méthode de détection sera appréciée.
 - Vous devez "informer" et "gérer" ... en MQTT !
L'information et sa gestion sera aussi appréciée.
Au niveau de l'ESP, la manifestation de la présence "reconnue" par les "pompiers" d'un feu se manifestera par le clignotement de la LED "builtin" ... on pourrait imaginer que cela équivaut au déclenchement du système d'arrosage (sprinkler).
 - Lorsque le feu est maîtrisé, "on" (les pompiers, une autorité, ...) doit pouvoir remettre l'ESP en fonctionnement normal via MQTT.

N'oubliez PAS qu'il y a plusieurs ESP dans le bâtiment !

7 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 :-)

Pour l'instant je bug un peu sur la mise en oeuvre TLS/SSL dans le contexte Arduino IDE MAIS des yeux nouveaux pourraient peut être résoudre le problème ?

Cela serait apprécié ... et s'intéresser à la sécurité ce n'est vraiment pas du temps perdu !

- Attention, on est bien dans une configuration où l'on dispose PAS d'un nom de domaine !
- Donc Let's encrypt ne me semble pas être une solution ?

7.1 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 !

Pour répondre à ce premier problème, on peut configurer le broker Mosquitto 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 :

```

1 listener 1883
2 protocol mqtt
3
4 allow_anonymous false
5 password_file /etc/mosquitto/passwd
6
7 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.

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 client ESP ou Python :-)

7.1.1 Impact au niveau du client

Est ce que vous vous rappelez la ligne 112 dans le code du client ".ino" (page 7) ?

```
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";

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

7.1.2 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)

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

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

7.2 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 :- (

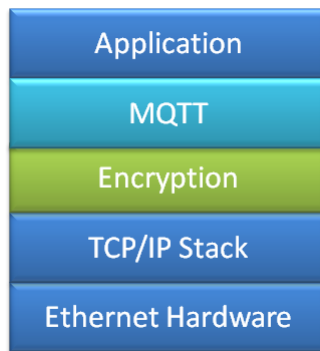
Tout cela est pour l'instant transmis en clair !

7.3 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.

7.4 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**.

7.4.1 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://hpbnc.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.

7.4.2 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.

7.5 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 !

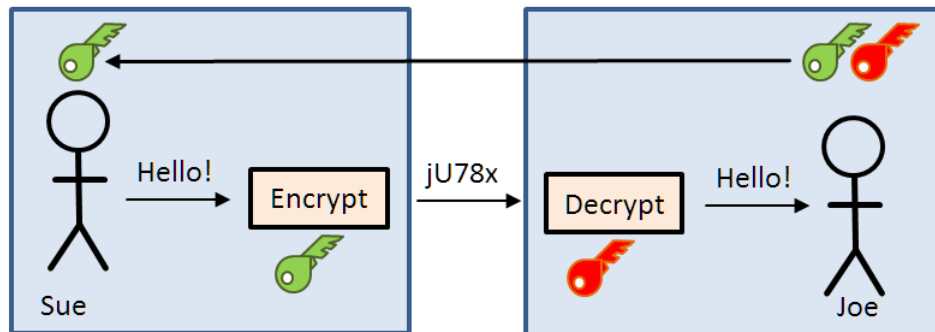
- C'est l'attaque MITM : "**Man In The Middle**"

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).

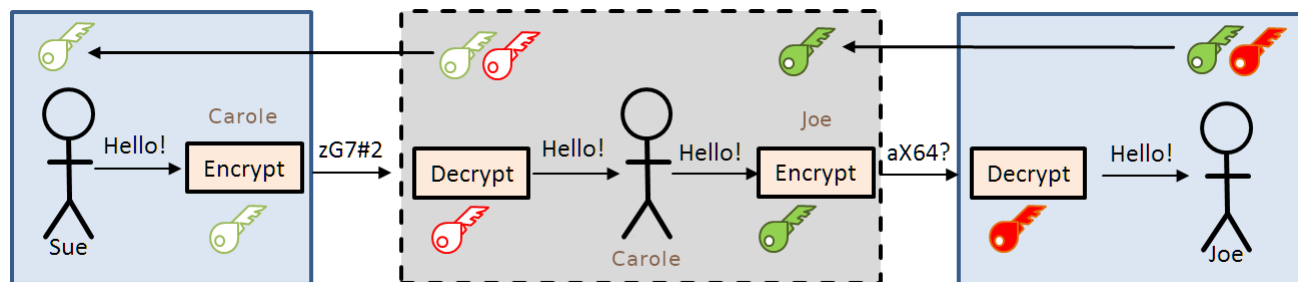
7.5.1 Cas normal - Échange classique :



- ✓ 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.

7.5.2 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").

7.6 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** ?

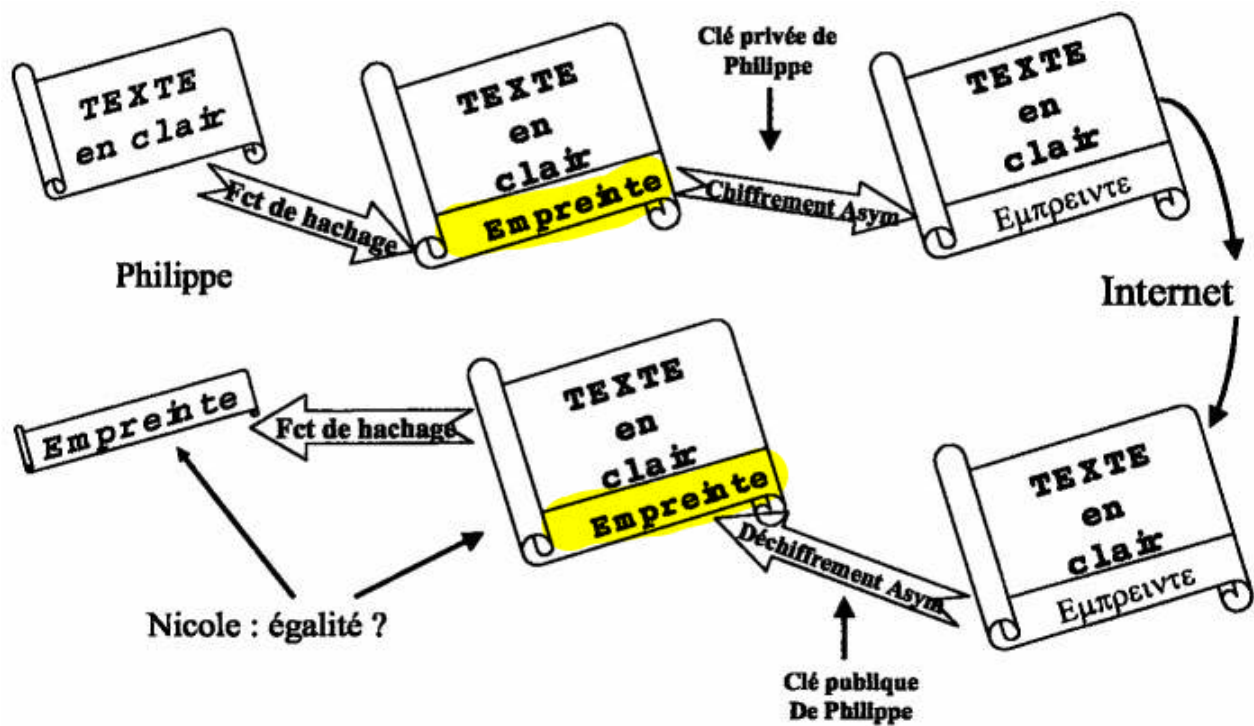
Cette question ne veut pas résoudre MITM puisque les messages émis par le pirate sont INTEGRES.

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

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) !

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.

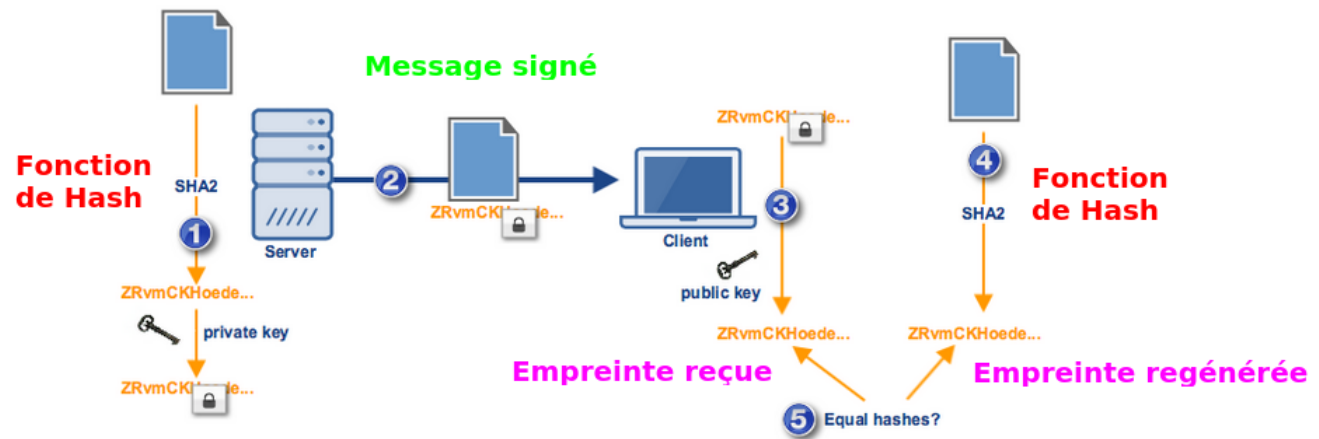
7.6.1 Signature électronique pour transmettre

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 (empreinte chiffrée) 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 "authentifé".

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.

7.7 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") que ce message vient de la bonne personne.

7.7.1 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 utilisateur 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.

7.7.2 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 certificates (CA)" contenus dans "mon" firefox.

- Tous sont émis par des autorités reconnues.

The screenshot displays the Firefox Certificate Manager interface. The 'Autorités' tab is active, showing a list of installed certificates. The 'certSIGN ROOT CA G2' is highlighted. To the right, a detailed view of this certificate is provided.

certSIGN ROOT CA G2	
Nom du sujet	RO CERTSIGN SA
Nom de l'émetteur	RO CERTSIGN SA
Validité	Pas avant: Mon, 06 Feb 2017 09:27:35 GMT Pas après: Thu, 06 Feb 2042 09:27:35 GMT
Informations sur la clé publique	Algorithme: RSA Taille de la clé: 4096 Exposant: 65537 Module: C0:C5:75:19:91:7D:44:74:74:87:FE:0E:3B:96:DC:D8:01:1...
Divers	Numéro de série: 11:00:34:B6:4E:C6:36:2D:36 Algorithme de signature: SHA-256 with RSA Encryption Version: 3 Télécharger
Empreintes numériques	SHA-256: 65:7C:FE:2F:A7:3F:AA:38:46:25:71:F3:32:A2:36:3A:46:F... SHA-1: 26:F9:93:B4:ED:3D:28:27:B0:B9:4B:A7:E9:15:1D:A3:8D:...

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

7.7.3 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 et de vérifier son authenticité (histoire d'être certain que c'est celle 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 son certificat (qui contient sa clé publique)

Point important, ce certificat n'est pas 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.

CONCLUSION :

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

8 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).

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 éléments à sécuriser.

8.1 Les éléments 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.

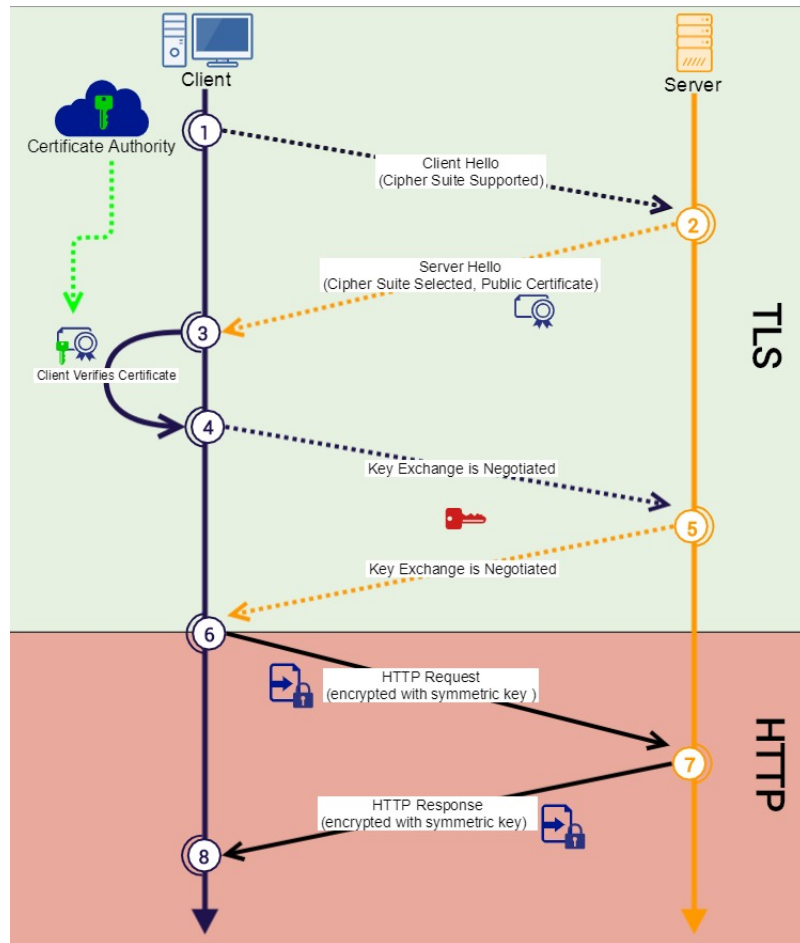
L'élément principal d'une PKI est donc l'**autorité de certification (CA)**.

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;-) on va s'"auto certifier"! (ou du moins on va essayer :-)

- 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, ...

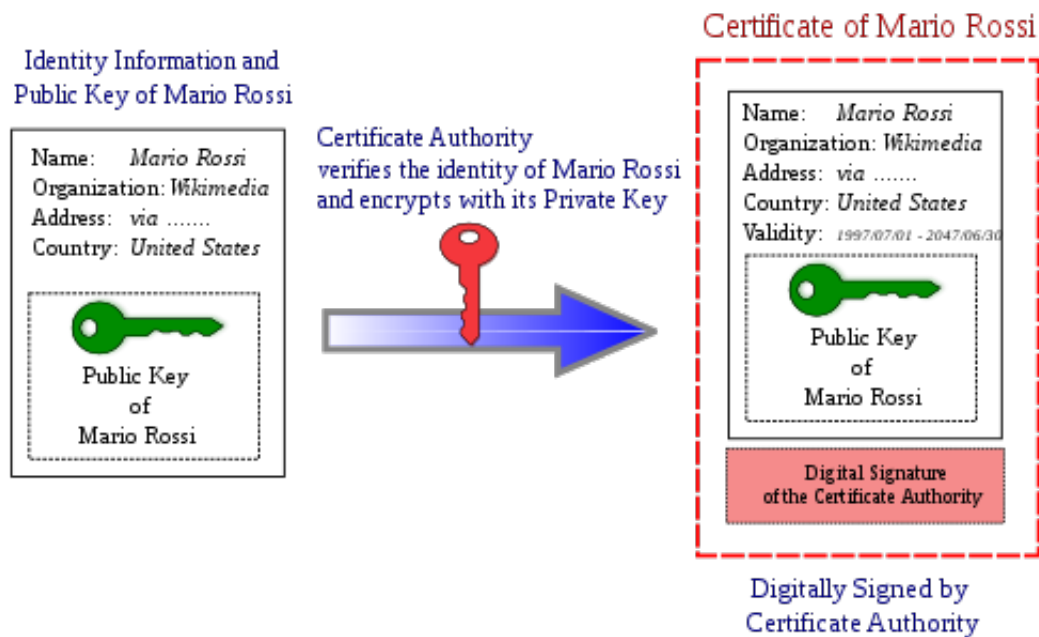
Cette autorité fait des vérifications éventuellement manuelles ... forcément ça coûte de l'argent (périodiquement car les certificats ont une durée de vie) !

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 après 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/>

8.2 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-18-04-1>
- <https://dzone.com/articles/mqtt-security-securing-a-mosquitto-server>
- <https://abhatikar.medium.com/secure-iot-for-successful-iot-41e029ac79d2>
- ...

je vous demande d'essayer de sécuriser votre flux MQTT.

Je dis bien "essayer" parce que pour être honnête je n'y suis pas totalement arrivé ... notamment du côté de l'ESP.

Je n'arrive pas à générer un certificat pour mon serveur MQTT qui satisfasse l'échange TLS!

8.2.1 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.

On crée une "pair key" pour la CA.

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

Cette "pair key" :

- ✓ utilise RSA (asymétrique)
- ✓ est de taille 2048,
- ✓ est matérialisée par le fichier "ca.key"
- ✓ contient à la fois la clef publique et la clef privée.

On va éviter de la laisser traîner !

Ensuite cette clé permet générer le certificat de l'autorité de certification d'une durée de validité de 10 ans (3650 days) :

```
menez@duke:Mosquitto_Conf_TLS$ openssl req -new -x509 -days 3650 -key ca.key -out ca.crt
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]:CoteAzur
Locality Name (eg, city) []:Sophia
Organization Name (eg, company) [Internet Widgits Pty Ltd]:M1
Organizational Unit Name (eg, section) []:CA
Common Name (e.g. server FQDN or YOUR name) []:192.168.1.101
Email Address []:menez@unice.fr
```

Ce certificat est matérialisée par le fichier "ca.crt"

Sur Moodle, je vous donne un script qui "contient" le sujet de ce certificat et vous n'auriez rien à taper!

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

Cette "pair key" est matérialisée par le fichier "server.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.
- Comme nous n'avons pas (encore) de domaine, j'utilise l'adresse IP de mon broker.

```
menez@duke:Mosquitto_Conf_TLS$ openssl req -new -out server.csr -key server.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]:CoteAzur
Locality Name (eg, city) []:Sophia
Organization Name (eg, company) [Internet Widgits Pty Ltd]:M1
Organizational Unit Name (eg, section) []:Server
Common Name (e.g. server FQDN or YOUR name) []:192.168.1.101
```

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 []:

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 server.csr -CA ca.crt -CAkey ca.key -CAcreateserial  
Signature ok  
subject=C = FR, ST = CoteAzur, L = Sophia, O = M1, OU = Server, CN = 192.168.1.101, emailAd=
```

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

Ces fichiers doivent entrer dans la configuration du broker (cf répertoire /etc/mosquitto/conf.d) :

Il faut donc les placer dans mosquitto :

```
menez@duke:Mosquitto_Conf_TLS$ ls  
ca.crt ca.key ca.srl gm.conf server.crt server.csr server.key  
  
menez@duke:Mosquitto_Conf_TLS$ sudo cp ca.crt /etc/mosquitto/ca_certificates/  
menez@duke:Mosquitto_Conf_TLS$ sudo cp server.crt /etc/mosquitto/certs/  
menez@duke:Mosquitto_Conf_TLS$ sudo cp server.key /etc/mosquitto/certs/
```

Il faut aussi modifier le fichier de configuration du broker en conséquence :

```
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.

```

8.3 Un client ESP

Il faut aussi placer le "ca.crt" dans le client !

C'est là que j'ai du mal avec la bibliothèque WiFiClientSecure ... elle n'arrive à établir une connexion vers le broker.

Alors que j'y arrive avec les clients mosquitto utilisant les mêmes certificats :-)

8.4 Le client Python

```

1  #https://github.com/eclipse/paho.mqtt.python#subscribe-unsubscribe
2
3  import time
4  import paho.mqtt.client as mqtt
5  import ssl
6
7  #define callbacks
8  def on_message(client, userdata, message):
9      print("received message =",str(message.payload.decode("utf-8")))
10
11 def on_log(client, userdata, level, buf):
12     print("log: ",buf)
13
14 def on_connect(client, userdata, flags, rc):
15     print("publishing ")
16
17     client.publish("muthu","muthupavithran",)
18
19 client=mqtt.Client()
20 client.on_message=on_message
21 client.on_log=on_log
22 client.on_connect=on_connect
23 print("connecting to broker")
24
25 ==> Port 1883, mdp et pas de TLS
26 #client.username_pw_set("darkvador", password="6poD2R2")
27 #client.connect("192.168.1.101", 1883, 60)
28
29 ==> TLS
30 client.tls_set("./ca.crt",certfile="./client.crt", keyfile="./client.key")
31 client.tls_insecure_set(True)
32 client.username_pw_set("darkvador", password="6poD2R2")
33 client.connect("192.168.1.101", 8883, 60)
34
35
36
37 ##start loop to process received messages
38 client.loop_start()
39 #wait to allow publish and logging and exit
40 time.sleep(1)

```

9 Consommation

La gestion de la consommation est un autre point tout à fait caractéristique de l'IOT.

- On va donc en profiter pour jeter un oeil sur les slides abordant les problèmes de consommation :

Fichier : `consommation_slides.pdf`

On doit désormais retrouver dans vos scripts ce type de préoccupation !

10 Limitations MQTT : Kafka by Apache ? !

<https://www.esp32.com/viewtopic.php?t=6654>