

TP

MQTTS ENTRE UN ESP32 ET UN BROKER

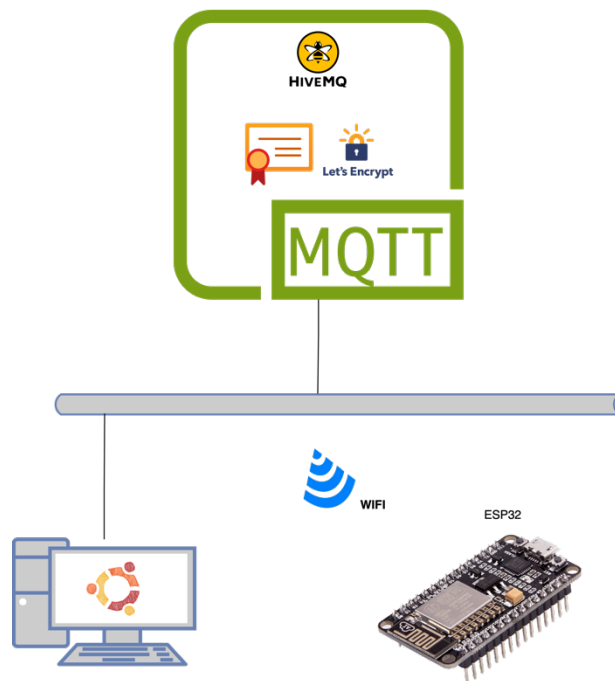
Objectif terminal :

Sécuriser une connexion mqtt entre un ESP32 et un broker.

Objectifs intermédiaires :

- Créer un broker mqtt sur Hivemq.
- Intégrer un certificat de CA dans un code C++ ESP32

TOPOLOGIE DU TP :



CRÉATION D'UN BROKER MQTT HIVEMQ :


1. Créer un broker sécuriser sur le cloud IOT Hivemq :

La société **HiveMQ** propose un broker dans le cloud gratuit limité à 100 sessions et 10GB par mois, ce qui est largement suffisant dans le domaine pédagogique.

Il faut créer un compte gratuit sur <https://console.hivemq.cloud> (identification par adresse email, compte gmail ou github)

Un fois connecté, il faut créer un **cluster** et choisir le type de serveur (AWS ou AZURE). Le choix n'entraîne aucune conséquence. Avec le compte gratuit, on peut créer 2 clusters.

Serverless
 FREE

 **Running**

URL

.s1.eu.hivemq.cloud

Port (TLS)
 8883

Started
 Wed, Feb 16, 2022, 11:52 AM

MANAGE CLUSTER

Quand on clique sur **MANAGE CLUSTER** on accède à la gestion du broker.
 Dans l'onglet **OVERVIEW**, on a le rappel de l'URL et des ports par lesquels on peut accéder au broker.

Connection Settings

Cluster URL
 EDIT

.s1.eu.hivemq.cloud

Port

8883

Websocket Port

8884

TLS URI

.s1.eu.hivemq.cloud:8883/mqtt

Websocket URI

.s1.eu.hivemq.cloud:8884/mqtt

Current Usage

MQTT Client Sessions		Data Traffic	
<div>0 / 100</div>		<div>0 B / 10 GB</div>	
Last update	Price per session	Last update	Price per GB
19 seconds ago	FREE	19 seconds ago	FREE

Dans l'onglet **ACCESS MANAGEMENT**, on peut gérer les informations d'identifications pour plusieurs utilisateurs.

Access Management

Credentials

Define one or more sets of credentials that allow MQTT clients to connect to your HiveMQ Cloud cluster. To learn more [check out our Security Fundamentals guide](#).

Username *

At least 5 characters

Password *

At least 8 characters, 1 digit, 1 uppercase character

Confirm Password *

Passwords must match

Permission *

Add permissions to limit access

> CREATE CREDENTIAL

Username	Permission type	Actions
bouhenic	<div><div></div> Publish and Subscribe</div>	<div>DELETE</div>

Il est à noter que les mots de passe **ne peuvent être modifiés ni revisualiser ultérieurement**, il faut donc prendre la précaution de les noter dans un lieu sûr.

TEST DU BROKER HIVEMQ:

1. Depuis le terminal, saisir le commande mosquitto suivante permettant l'abonnement au topic « topic/test »:

```
mosquitto_sub -h 5exxxxxxxxxxxxxxxxxxxd80fd.s1.eu.hivemq.cloud -p 8883 -u xxxxxxxx -P xxxxxxxxxx -t topic/test
```

Hivemq propose un client mqtt. Nous allons l'utiliser pour publier un message sur le topic « topic/test ».

2. Se connecter sur Hivemq et sélectionner l'onglet WEB CLIENT :



3. Saisir vos identifiants/mot de passe de façon à connecter le client sur le broker.

Web Client

Connect to your HiveMQ Cloud Cluster to debug and test out your use-case with our MQTT client in real time. It allows you to publish, subscribe, and receive messages across various topics within your cluster.

Connection Settings

Connect to your HiveMQ Cloud Cluster with your credentials. Do not worry you can quickly connect with autogenerated credentials.

Username *

sbouhenic

Password *



Disconnect


The WebClient is connected

On peut aussi s'abonner au topic depuis la fenêtre suivante :

Topic Subscriptions 1

Subscribe to topics to receive messages from the HiveMQ cluster. You can also set the Quality of Service (QoS) for each topic. The higher the QoS, the more reliable the message delivery is. You can always subscribe to the (#) wildcard to receive all messages.

TOPIC	QoS	ACTIONS
 topic/test	QoS: 0	

Messages 1  

0 Topic: topic/test QoS: 0

hello newton

4. Publier un message à l'aide de la fenêtre suivante :

Send Message 1

If you cannot see any messages, make sure you are subscribed to the correct topics. You can always subscribe to the (#) wildcard to receive all messages.

Message *

Topic * **QoS ***

Vous testez sur le terminal :

Hello newton

Nous n'avons pas eu besoin de spécifier le certificat du CA dans la commande mosquitto avec l'option `-cafile`. En effet, hivemq utilise un certificat connu let's encrypt. On peut le vérifier en saisissant la commande openssl suivante :

```
openssl s_client -connect 5e7xxxxxxxxxxxx.s1.eu.hivemq.cloud:8883
-servername 5e7xxxxxxxxxxxx.s1.eu.hivemq.cloud -showcerts
```

Il y a beaucoup de chose dans la réponse. On voit les certificats du serveur et du CA intermédiaire. On voit également la chaine de certification qui indique la profondeur (depth).

```
depth=2 C = US, 0 = Internet Security Research Group, CN = ISRG Root X1
verify return:1
depth=1 C = US, 0 = Let's Encrypt, CN = R11
verify return:1
depth=0 CN = *.s1.eu.hivemq.cloud
verify return:1
```

- **depth=0** → Le certificat du serveur (*s1.eu.hivemq.cloud*).
- **depth=1** → Le certificat intermédiaire (*Let's Encrypt R1 1*).
- **depth=2** → Le certificat racine (*ISRG Root X1*).

Le certificat racine n'est pas transmis, il est présent dans la banque de certificat. On peut la visualiser en saisissant :

PROGRAMMATION DE L'ESP32 :

- ```
cd arduino
./arduino-ide 2.3.4 Linux 64bit.AppImage
```

- ```
git clone https://github.com/bouhenic/mqtts
cd mqtts/ESP32
```

- ```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>

const char* ssid = "xxxxxxxxxx"; //Compléter ici
const char* password = "xxxxxxxxxx"; //Compléter ici
const char* mqtt_server = "5xxxxxxxxxxxxxxxxxxxxxx75efd80fd.s1.eu.hivemq.cloud";
const int mqtt_port = 8883;
const char* mqtt_user = "xxxxxxxxxx"; //Compléter ici
const char* mqtt_pass = "xxxxxxxxxx"; //Compléter ici

// Certificat racine ISRG Root X1 (Let's Encrypt)
const char* ca_cert = \
"-----BEGIN CERTIFICATE-----\n" \
"MIIFazCCA10gAwIBAgIRAIQz7DSQONZRGPGu20CiwAwDQYJKoZIhvcNAQELBQAw\n" \
"\"TzELMAkGA1UEBhMCVVMwKTAhBgNVBAoTIEludGVybmV0IFNlY3VyaXR5IFJlc2Vh\n" \
"\"cmNoNEEdyb3V3M0RwNDMwYDQwEwYUJHIFJvb3QgWDEwHhcNMTUwNjA0MTUwNDM4\n" \
"\"wHcNMZUwNjA0MTUwNDMwYDQwEwYUJHIFJvb3QgWDEwEwYUJHIFJvb3QgWDEw\n" \
"\"ZXQgU2VjZjXJpdHkgUmVzZWYyZG9jR3JvdXAxFTATBgNVBAMTDElTUKcgUm9vdCBY\n" \
"\"MTCCAIwDQYJKoZIhvcNAQEBBQADggIPADCCAgoCggIBAK3oJHP0FDfzm54rVyg\n" \
"\"h77ct984K1XuPOZXoHj3dcK1/vVqbvYATyjb3miGbESTtrFj/RQSa78f0uoxmyF+\n" \
"\"0TM8ukj13Xnfs7j/EvEhmkvBioZxaUpmZmyPfjxwv60pIgbz5MDmgK7iS4+3mX6U\n" \
"\"A5/TR5d8mUgju+g4rk8Kb4Mu0ULXjIB0ttov0DiNewNwIRt18jA8+o+u3dpjq+sW\n" \
"\"T8K0EUt+zwvo/7V3LvSye0rgTBiLDHCNAymg4VMk7BPZ7hm/ELNKjd+Jo2FR3qyH\n" \
"\"B5T0Y3HsLuJwV5iB4YLcNHLSdu87kGJ55tukmi8mxdAQ4Q7e2RC0Fvu396j3x+UC\n" \
"\"B5iNPgivi5+I3lq02DZ77DnKxHZu8A/LJBdiB3QW0KtZB6awBdpUKD9jf1b0SHzUv\n" \
"\"KBds0pjBqAlkd25HN7r0rFleaJ1/ctaJxQZBKTSZPt0m9STJEdaao0xAH0ahmbWn\n" \
"\"0LFuhjuefXKneUg4We0+UxgCW0PjdvAvBbI+e0oc3MFEVzG6uBQE3xdk3SzyntN\n" \
"\"jh8BCNAw1FtxNrQHusEwMfxIt4I7mkZ9YIqiomyCzLq9gwQbooMDQaHwBfEbwrwb\n" \
"\"qHyG00aoSCqI3Haadr8faqU9GY/rOPNk3sgrDQoo/fb4hVC1CLQJ13hef4Y53CI\n" \
"\"rU7m2Ys6xt0nUw7/vGT1M0NPagMBAAGjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNV\n" \
"\"HRMBAf8EBTADAQH/MB0GA1UdDgQWBBS5tFnme7bl5AfDdwAiIyBpY9umbbjANBgk\n" \
"\"hkiG9w0BAQsFAAOCAQEAVR9YqbyyqFDQDLHYGmkGJykIrGF1XIPu+ILLas/V91ZL\n" \
"\"ubhzEFntIzd+50xx+7LSYK05qAvqFyFWhfFQDlnrzuBZ6brJFe+GnY+EGPbk6ZGQ\n" \
"\"3BebYhtF8GaV0nxvwu077x/Py9au3/GpsMiu/X1+mvoiB0v/2X/qkSSisRc0j/KK\n" \
"\"NFTy2PwBjVS5uCbMioqziUwhtDyC3+6WVwW6LLv3xLfHTjCuVjHjInZktHCgK05\n" \
"\"0RAZ74JMP1+GslWYHh4h0h0w571jaztX0oJwTdwJ4LntCadNh0dijnsnzvzHu7Jr\n" \
"-----\n\";
```

```

"TkXWStAmzOVyyghqpZXjFaH3p03JLF+l+/+sKAIuvtd7u+Nxe5AW0wdeRLN8NwdC\n" \
"jNPElpzVmbUq4JUagEiuTDkHszxHpFKVK7q4+63SM1N95R1NbdWhscdCb+ZAJzVc\n" \
"oyi3B43njTQ05yOf+1CceWxG1bQVs5ZufpsMLjq4Ui0/1lvh+wjChP4kqK0J2qxq\n" \
"4RgqsahDYVvTH9w7jXbyLeiNdd8XM2w9U/t7y0Ff/9yi0GE44Za4rF2LN9d11TPA\n" \
"mRGunUHBcnWEvgJBQl9nJEiU0Zsnvgc/ubhPgXRR4Xq37Z0j4r7g1SgEEzwxA57d\n" \
"emyPxcYxn/eR44/KJ4EBs+lVDR3veyJm+kXQ99b21/+jh5Xos1AnX5iIitreGCc=\n" \
"-----END CERTIFICATE-----\n";

WiFiClientSecure wifiClient;
PubSubClient client(wifiClient);

void connectToWiFi() {
 Serial.println("Connexion au WiFi...");
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
 }
 Serial.println("\nConnecté au WiFi");
}

void connectToMQTT() {
 while (!client.connected()) {
 Serial.println("Connexion au broker MQTT...");
 if (client.connect("ESP32Client", mqtt_user, mqtt_pass)) {
 Serial.println("Connecté au broker MQTT");
 client.subscribe("xxxxx/xxxxxx"); //Compléter ici le topic
 } else {
 Serial.print("Échec de connexion, rc=");
 Serial.println(client.state());
 delay(5000);
 }
 }
}

void callback(char* topic, byte* payload, unsigned int length) {
 Serial.print("Message reçu sur le sujet : ");
 Serial.println(topic);

 Serial.print("Message : ");
 for (unsigned int i = 0; i < length; i++) {
 Serial.print((char)payload[i]);
 }
 Serial.println();
}

void setup() {
 Serial.begin(115200);

 connectToWiFi();

 wifiClient.setCACert(ca_cert); // Configure le certificat pour la sécurité
 client.setServer(mqtt_server, mqtt_port);
 client.setCallback(callback);

 connectToMQTT();
}

void loop() {
 if (!client.connected()) {
 connectToMQTT();
 }
 client.loop();
}

```

6. Modifier les paramètres suivants du programme fourni (ssid, password, mqtt\_server, mqtt\_user, mqtt\_pass et topic).
7. Sélectionner la carte « ESP32 Dev Module » dans l'onglet Tools/Boards Manager

8. Téléverser le programme dans l'ESP32.

#### TEST DU PROGRAMME TÉLÉVERSÉ :

5. Saisir la commande mosquitto suivante pour publier un message en remplaçant l'url de votre broker, le topic, le message, votre username et votre password :

```
Mosquitto_pub -h 5e7xxxxxxxxxxxx.s1.eu.hivemq.cloud -p 8883 -u
xxxxxx -P xxxxxx -t formation/IOTsecurity -m "Hello IOT security"
```

6. Sur le serial monitor d'Arduino, vérifier la réception du message :

```
Output Serial Monitor x
Message (Enter to send message to 'ESP32 Dev Module' on '/dev/cu.usbserial-0001')

ets 00 0p0v00 00I0000080404,len:3504
entry 0x400805cc
Connexion au WiFi...
.....
Connecté au WiFi
Connexion au broker MQTT...
Connecté au broker MQTT
Message reçu sur le sujet : formation/IOTsecurity
Message : Hello IOT security
```

# ANNEXE : Explication détaillée

## 1. Inclusion des bibliothèques

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
```

## 2. Déclaration des informations de connexion

```
const char* ssid = "xxxxxxxxx"; //Compléter ici
const char* password = "xxxxxxxxx"; //Compléter ici
const char* mqtt_server = "5xxxxxxxxxxxxxxxxxxxxxx75efd80fd.s1.eu.hivemq.cloud";
const int mqtt_port = 8883;
const char* mqtt_user = "xxxxxxxxx"; //Compléter ici
const char* mqtt_pass = "xxxxxxxxx"; //Compléter ici
```

- **ssid et password** : Informations d'identification du réseau Wi-Fi.
- **mqtt\_server** : Adresse du **broker MQTT**, ici hébergé chez **HiveMQ Cloud**.
- **mqtt\_port = 8883** : Port standard pour MQTT sécurisé (MQTTs).
- **mqtt\_user et mqtt\_pass** : Identifiants pour s'authentifier auprès du broker.

## 3. Certificat racine Let's Encrypt

```
const char* ca_cert = \
"-----BEGIN CERTIFICATE-----\n" \
"MIIFazCCA10gAwIBAgIRAIIQz7DSQ0NZRGPgu20CiwAwDQYJKoZIhvcNAQELBQAw\n" \
"... (contenu du certificat) ...\n" \
"-----END CERTIFICATE-----\n";
```

- Ce certificat **ISRG Root X1** provient de **Let's Encrypt**.
- Il permet à l'ESP32 de **vérifier l'identité du serveur MQTT** pour éviter les attaques MITM (Man-In-The-Middle).

## 4. Création des objets WiFi et MQTT

```
WiFiClientSecure wifiClient;
PubSubClient client(wifiClient);
```

- **WiFiClientSecure wifiClient;**
  - Initialise un client Wi-Fi sécurisé (SSL/TLS) pour la connexion MQTT.
- **PubSubClient client(wifiClient);**
  - Crée un client MQTT sécurisé en utilisant wifiClient.

## 5. Fonction de connexion au Wi-Fi

```
void connectToWiFi() {
 Serial.println("Connexion au WiFi...");
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 delay(500);
 Serial.print(".");
 }
 Serial.println("\nConnecté au WiFi");
}
```

- **WiFi.begin(ssid, password);** → Démarre la connexion au réseau Wi-Fi.
- **Boucle while (WiFi.status() != WL\_CONNECTED)**



- Attend que l'ESP32 soit connecté.
- Affiche des "." toutes les 500 ms pour montrer l'attente.

## 6. Connexion au broker MQTT

```
void connectToMQTT() {
 while (!client.connected()) {
 Serial.println("Connexion au broker MQTT...");
 if (client.connect("ESP32Client", mqtt_user, mqtt_pass)) {
 Serial.println("Connecté au broker MQTT");
 client.subscribe("xxxxx/xxxxxx"); // Compléter ici le topic
 } else {
 Serial.print("Échec de connexion, rc=");
 Serial.println(client.state());
 delay(5000);
 }
 }
}
```

- **client.connected()**
  - Vérifie si l'ESP32 est déjà connecté au broker MQTT.
- **Tentative de connexion avec client.connect("ESP32Client", mqtt\_user, mqtt\_pass)**
  - ESP32Client : Nom de l'ESP32 pour le broker MQTT.
  - mqtt\_user et mqtt\_pass : Identifiants pour l'authentification.
- **Si la connexion réussit :**
  - Affiche "Connecté au broker MQTT".
  - Souscrit au **topic** "xxxxx/xxxxxx" (doit être remplacé par un vrai topic).
- **Si la connexion échoue :**
  - Affiche l'erreur MQTT (client.state()).
  - Attends **5 secondes** avant de retenter.

## 7. Fonction de réception des messages MQTT

```
void callback(char* topic, byte* payload, unsigned int length) {
 Serial.print("Message reçu sur le sujet : ");
 Serial.println(topic);

 Serial.print("Message : ");
 for (unsigned int i = 0; i < length; i++) {
 Serial.print((char)payload[i]);
 }
 Serial.println();
}
```

- Cette fonction est appelée **automatiquement** lorsqu'un message MQTT est reçu.
- **Affiche le topic et le message** reçu en console série.

## 8. Initialisation dans setup()

```
void setup() {
 Serial.begin(115200);

 connectToWiFi();

 wifiClient.setCACert(ca_cert); // Configure le certificat pour la sécurité
 client.setServer(mqtt_server, mqtt_port);
 client.setCallback(callback);

 connectToMQTT();
}
```

- **Serial.begin(115200);** → Initialise la console série pour le débogage.
- **connectToWiFi();** → Connecte l'ESP32 au Wi-Fi.
- **wifiClient.setCACert(ca\_cert);**

- Configure le **certificat racine Let's Encrypt** pour la connexion sécurisée.
- **client.setServer(mqtt\_server, mqtt\_port);**
  - Définit l'adresse et le port du **broker MQTT**.
- **client.setCallback(callback);**
  - Associe la fonction `callback()` pour recevoir les messages MQTT.
- **connectToMQTT();** → Connecte l'ESP32 au **broker MQTT sécurisé**.

## 9. Boucle principale `loop()`

```
void loop() {
 if (!client.connected()) {
 connectToMQTT();
 }
 client.loop();
}
```

- **Si la connexion MQTT est perdue** → La fonction `connectToMQTT();` tente de la rétablir.
- **`client.loop();`** → Maintient la connexion MQTT active et gère la réception des messages.