

L'internet des objets utilise fréquemment le protocole MQTT pour échanger des messages à faible charge. Le broker peut être un service en ligne payant ou gratuit. Il est également possible de construire un broker adapté aux besoins de l'application. Le nano-ordinateur Raspberry Pi est particulièrement adapté à cet emploi en raison de son très faible coût et de sa faible consommation.

Ce TP montre la réalisation du broker MQTT sur Raspberry Pi puis la sécurisation des données avec SSL/TLS.

MOSQUITTO ( <https://mosquitto.org/> ) propose une installation gratuite d'un broker open source.

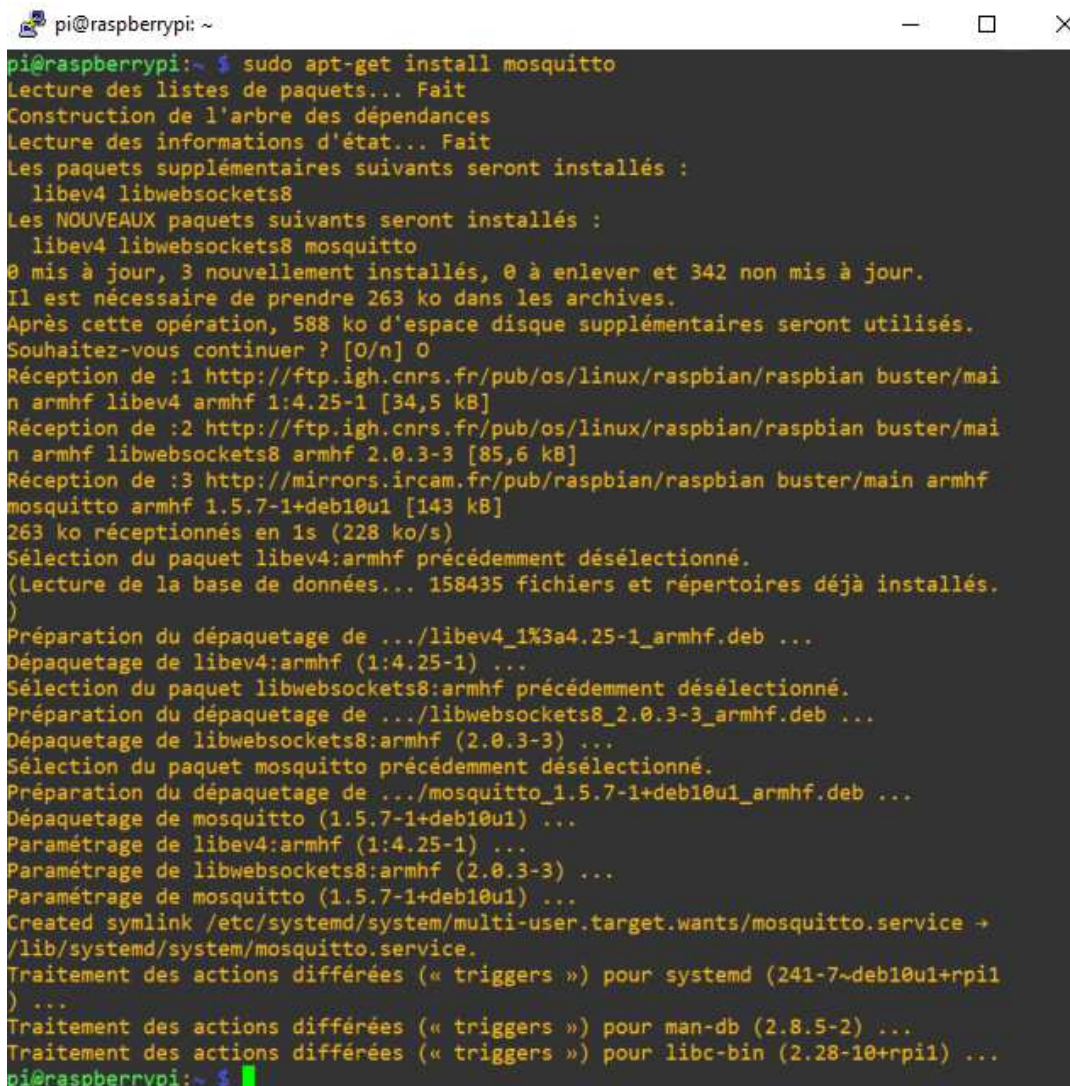
MOSQUITTO est porté sur Windows, MAC et la plupart des distributions Linux dont Raspbian.

# 1 Installation d'un broker MQTT Mosquitto sur RPi

Ouvrir une session SSH sur RPi :

```
~$ sudo apt-get update
```

```
~$ sudo apt-get install mosquitto
```



```
pi@raspberrypi:~$ sudo apt-get install mosquitto
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les paquets supplémentaires suivants seront installés :
  libev4 libwebsockets8
Les NOUVEAUX paquets suivants seront installés :
  libev4 libwebsockets8 mosquitto
0 mis à jour, 3 nouvellement installés, 0 à enlever et 342 non mis à jour.
Il est nécessaire de prendre 263 ko dans les archives.
Après cette opération, 588 ko d'espace disque supplémentaires seront utilisés.
Souhaitez-vous continuer ? [O/n] O
Réception de :1 http://ftp.igh.cnrs.fr/pub/os/linux/raspbian/raspbian buster/main armhf libev4 armhf 1:4.25-1 [34,5 kB]
Réception de :2 http://ftp.igh.cnrs.fr/pub/os/linux/raspbian/raspbian buster/main armhf libwebsockets8 armhf 2.0.3-3 [85,6 kB]
Réception de :3 http://mirrors.ircam.fr/pub/raspbian/raspbian buster/main armhf mosquitto armhf 1.5.7-1+deb10u1 [143 kB]
263 ko réceptionnés en 1s (228 ko/s)
Sélection du paquet libev4:armhf précédemment désélectionné.
(Lecture de la base de données... 158435 fichiers et répertoires déjà installés.)
Préparation du dépaquetage de .../libev4_1%3a4.25-1_armhf.deb ...
Dépaquetage de libev4:armhf (1:4.25-1) ...
Sélection du paquet libwebsockets8:armhf précédemment désélectionné.
Préparation du dépaquetage de .../libwebsockets8_2.0.3-3_armhf.deb ...
Dépaquetage de libwebsockets8:armhf (2.0.3-3) ...
Sélection du paquet mosquitto précédemment désélectionné.
Préparation du dépaquetage de .../mosquitto_1.5.7-1+deb10u1_armhf.deb ...
Dépaquetage de mosquitto (1.5.7-1+deb10u1) ...
Paramétrage de libev4:armhf (1:4.25-1) ...
Paramétrage de libwebsockets8:armhf (2.0.3-3) ...
Paramétrage de mosquitto (1.5.7-1+deb10u1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/mosquitto.service → /lib/systemd/system/mosquitto.service.
Traitement des actions différées (« triggers ») pour systemd (241-7~deb10u1+rp1) ...
Traitement des actions différées (« triggers ») pour man-db (2.8.5-2) ...
Traitement des actions différées (« triggers ») pour libc-bin (2.28-10+rp1) ...
pi@raspberrypi:~$
```

Vérifier que le service mosquitto est actif :

```
~$ systemctl status mosquitto
```

```
pi@raspberrypi:~$ systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enab
   Active: active (running) since Sat 2020-06-13 18:11:24 CEST; 1min 21s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
   Main PID: 1381 (mosquitto)
      Tasks: 1 (limit: 4915)
     Memory: 816.0K
    CGroup: /system.slice/mosquitto.service
            └─1381 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

juin 13 18:11:24 raspberrypi systemd[1]: Starting Mosquitto MQTT v3.1/v3.1.1 Broker...
juin 13 18:11:24 raspberrypi systemd[1]: Started Mosquitto MQTT v3.1/v3.1.1 Broker.
pi@raspberrypi:~$
```

En cas de problème, relance le service : `~$ sudo systemctl enable mosquitto.service.`

Le broker est maintenant installé.

*Attention, le Raspberry Pi est généralement à l'intérieur d'un réseau local, pour disposer d'un accès au broker depuis Internet il faudra configurer le routeur Internet de manière à rediriger les requêtes sur le port 1883 (en claire) ou 8883 (pour SSL) vers le Raspberry Pi.*

Un fichier .log enregistre toutes les activités du broker, pour le visualiser :

`sudo cat /var/log/mosquitto/mosquitto.log`

```
pi@raspberrypi:~$ sudo cat /var/log/mosquitto/mosquitto.log
1592064684: mosquitto version 1.5.7 starting
1592064684: Config loaded from /etc/mosquitto/mosquitto.conf.
1592064684: Opening ipv4 listen socket on port 1883.
1592064684: Opening ipv6 listen socket on port 1883.
1592065159: New connection from 192.168.1.67 on port 1883.
1592065159: New client connected from 192.168.1.67 as client_sur_RPi (c1, k60).
1592065664: mosquitto version 1.5.7 terminating
pi@raspberrypi:~$
```

### Démarrage et arrêt du service mosquitto

```
sudo systemctl start mosquitto
sudo systemctl stop mosquitto
sudo systemctl restart mosquitto
```

Après arrêt du service il est possible de lancer mosquitto :

```
mosquitto
mosquitto -v
(affiche tous les échanges)
```

```
pi@raspberrypi:~$ sudo systemctl stop mosquitto
pi@raspberrypi:~$ mosquitto -v
1592065667: mosquitto version 1.5.7 starting
1592065667: Using default config.
1592065667: Opening ipv4 listen socket on port 1883.
1592065667: Opening ipv6 listen socket on port 1883.
```

À partir d'un PC sur le même réseau que le Raspberry Pi il est possible de tester le broker avec MQTT.fx (voir TP 2).

## IOT et PROTOCOLE MQTT – serveur MQTT sur Raspberry Pi

Edit Connection Profiles

Maqiatto  
RPI  
ThingSpeak MQTT  
monPC  
monPCInternet  
mosquitto

Profile Name

RPI

Profile Type

MQTT Broker

MQTT Broker Profile Settings

Broker Address

192.168.1.150

Broker Port

1883

Client ID

client\_sur\_RPi

General

User Credentials

SSL/TLS

Proxy

LWT

MQTT.fx - 1.7.1

File Extras Help

RPI

Connect Disconnect

Publish Subscribe Scripts Broker Status Log

topic0

Publ... QoS 0

Hello

Le broker affiche les échanges. Il interroge régulièrement le client afin de vérifier la connexion. En effet si le client est abonné à un topic le broker lui transmettra tout message sur ce topic.

```
pi@raspberrypi:~$ mosquitto -v
1592065667: mosquitto version 1.5.7 starting
1592065667: Using default config.
1592065667: Opening ipv4 listen socket on port 1883.
1592065667: Opening ipv6 listen socket on port 1883.
1592065836: New connection from 192.168.1.67 on port 1883.
1592065836: New client connected from 192.168.1.67 as client_sur_RPi (c1, k60).
1592065836: No will message specified.
1592065836: Sending CONNACK to client_sur_RPi (0, 0)
1592065856: Received PUBLISH from client_sur_RPi (d0, q0, r0, m0, 'topic0', ... (7 bytes))
1592065883: Received DISCONNECT from client_sur_RPi
1592065883: Client client_sur_RPi disconnected.
1592065922: New connection from 192.168.1.67 on port 1883.
1592065922: New client connected from 192.168.1.67 as client_sur_RPi (c1, k60).
1592065922: No will message specified.
1592065922: Sending CONNACK to client_sur_RPi (0, 0)
1592065934: Received PUBLISH from client_sur_RPi (d0, q0, r0, m0, 'topic0', ... (5 bytes))
1592065982: Received PINGREQ from client_sur_RPi
1592065982: Sending PINGRESP to client_sur_RPi
```

L'avantage d'installer un broker "privé" est de pouvoir sécuriser et crypter les données.

## Configuration des mots de passe MQTT

mosquitto\_passwd ( [https://mosquitto.org/man/mosquitto\\_passwd-1.html](https://mosquitto.org/man/mosquitto_passwd-1.html) ) est un utilitaire permettant de générer un fichier de mot de passe.

Les mots de passe sont stockés dans /etc/mosquitto/passwd :

`sudo mosquitto_passwd -c /etc/mosquitto/passwd admin (-c pour create)`

```
pi@raspberrypi:~$ sudo mosquitto_passwd -c /etc/mosquitto/passwd admin
Password: pi@raspberrypi:~$ sudo mosquitto_passwd -c /etc/mosquitto/passwd admin
Password:
Reenter password:
pi@raspberrypi:~$
```

( ici le mot de passe est « 123456 » )

Création d'un fichier de configuration Mosquitto pour activer l'utilisation d'un mot de passe :

`sudo nano /etc/mosquitto/mosquitto.conf`

Compléter le fichier comme suit

```
pi@raspberrypi: ~
GNU nano 3.2 /etc/mosquitto/mosquitto.conf
# 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 /var/run/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

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

include_dir /etc/mosquitto/conf.d

allow_anonymous false
password_file /etc/mosquitto/passwd
```

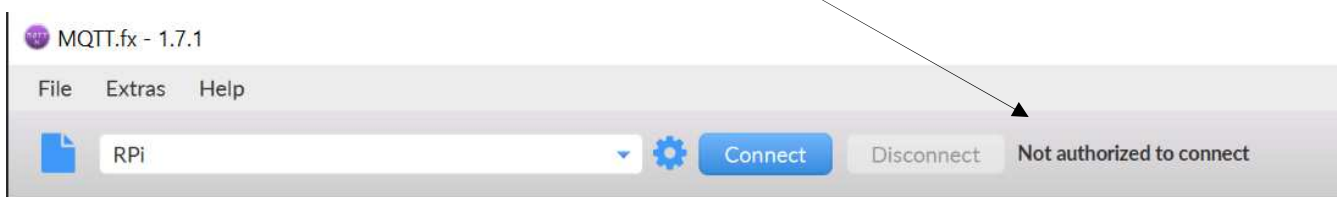
`allow_anonymous false` : désactive toutes les connexions non authentifiées ;

`password_file` : indique à Mosquitto où rechercher les informations sur l'utilisateur et le mot de passe.

redémarrer Mosquitto avec :

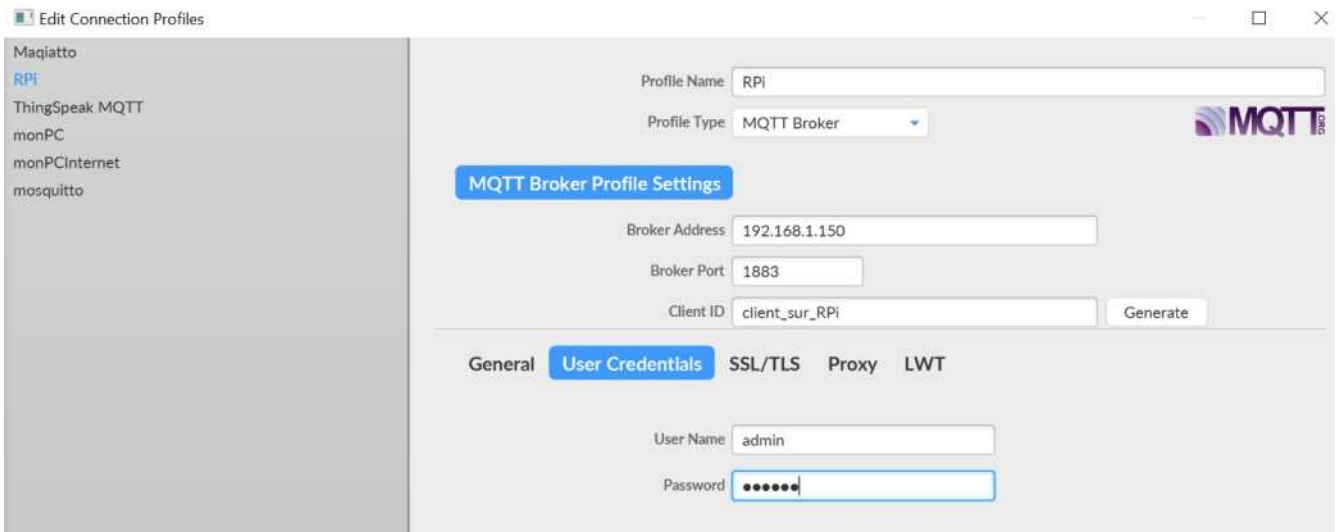
`sudo systemctl restart mosquitto`

Essayez de vous connecter sans mot de passe avec MQTT.fx.

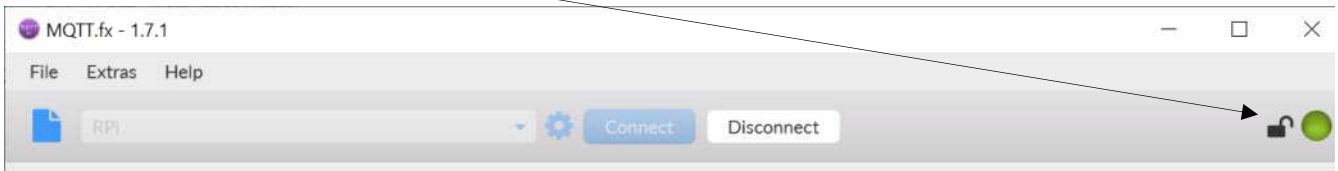


## IOT et PROTOCOLE MQTT – serveur MQTT sur Raspberry Pi

Entrer maintenant l'utilisateur et le mot de passe dans la configuration de la connexion MQTT.



La connexion s'établit maintenant.



Il y a maintenant une protection par mot de passe pour l'accès au broker Mosquitto. Cependant les mots de passe sont transmis non chiffrés sur Internet, ce qui est une faille de sécurité. Un cryptage SSL y remédiera.

## 2 Cryptage SSL/TLS

SSL est normalisé TLS : [https://fr.wikipedia.org/wiki/Transport\\_Layer\\_Security](https://fr.wikipedia.org/wiki/Transport_Layer_Security).

### Principe :

On peut sécuriser les échanges de données grâce à un chiffrement AES :

[https://fr.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard](https://fr.wikipedia.org/wiki/Advanced_Encryption_Standard)

L'algorithme AES est un des plus utilisés sur Internet. C'est un cryptage SYMETRIQUE, la même clé privée est utilisé pour le cryptage et le décryptage.



Le problème consiste donc à échanger la clé privée entre les deux correspondants sans qu'elle ne puisse être interceptée.

La clé privée sera échangée grâce à un algorithme de cryptage ASYMETRIQUE à clé publique.

Le principe repose sur un chiffement RSA. [https://fr.wikipedia.org/wiki/Chiffrement\\_RSA](https://fr.wikipedia.org/wiki/Chiffrement_RSA)

Dans ce type de chiffement il existe une clé de cryptage et une clé de décryptage. Le destinataire donne sa clé de cryptage (clé publique) à l'émetteur. La clé publique peut être interceptée mais c'est sans conséquence, car on ne peut que crypter avec, pas décrypter.

L'émetteur crypte le message avec la clé publique, l'envoie au destinataire qui peut seul le décrypter avec sa clé privée. La clé privée ne voyageant pas, elle ne peut être interceptée.



Il est maintenant possible d'échanger comme n'importe quel message une clé privée pour un futur échange par cryptage symétrique.

Le chiffement SSL normalisé par TLS repose sur les deux principes.

Dans la phase de négociation, l'émetteur et le récepteur échange une clé privée avec un cryptage asymétrique. Ensuite le cryptage est symétrique et utilise la clé privée

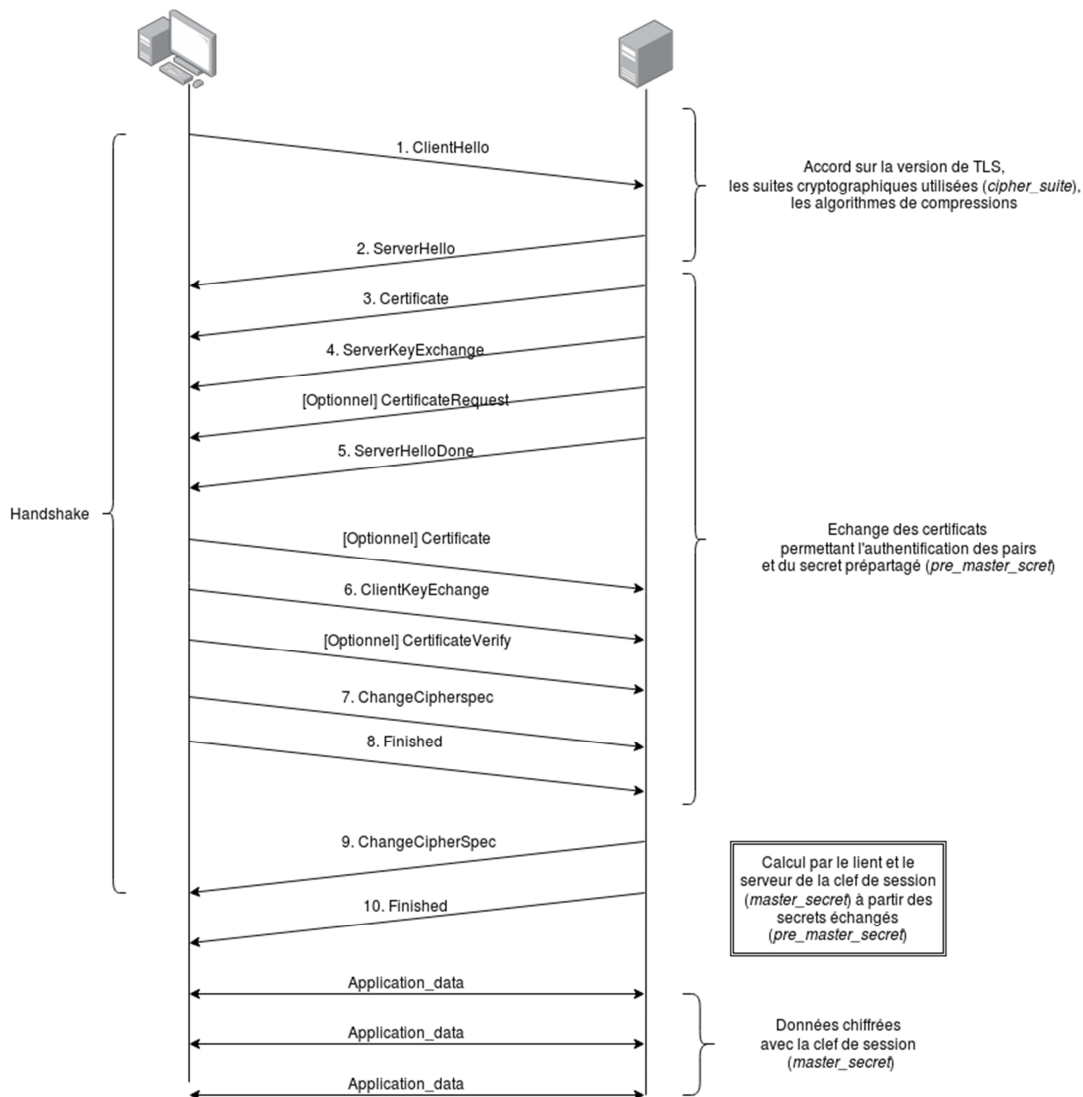
Afin d'être certain de communiquer avec bonne machine distante et non un pirate, l'utilisateur authentifie le serveur TLS sur lequel il se connecte. Cette authentification est réalisée par l'utilisation d'un [certificat numérique X.509](#) délivré par une [autorité de certification](#) généralement payante.

Dans le cas de MQTT la plupart du temps le serveur s'auto certifie.

- Le client doit disposer d'un logiciel de cryptage/décryptage type AES
- Le serveur doit disposer :
  - du même logiciel ;
  - d'un certificat ;
  - d'une clé de cryptage publique (qui sera donnée à l'émetteur) ;
  - d'une clé de décryptage privée ;
  - d'une clé de cryptage/décryptage symétrique privée (qui sera donnée à l'émetteur) ;

## IOT et PROTOCOLE MQTT – serveur MQTT sur Raspberry Pi

Principe d'un échange TLS (cf Wikipédia) :



## Configuration de MQTT SSL sur Raspberry Pi

**OpenSSL** est une boîte à outils de [chiffrement](#) comportant deux [bibliothèques](#), libcrypto et libssl, fournissant respectivement une implémentation des algorithmes [cryptographiques](#) et du [protocole de communication SSL/TLS](#), ainsi qu'une [interface en ligne de commande](#), openssl.

(Wikipédia) <https://fr.wikipedia.org/wiki/OpenSSL>

OpenSSL est normalement préinstallé sur les distributions Rpi.

Pour vérifier : `$dpkg -get-architecture openssl` :

```
pi@raspberrypi:~/certifs/broker $ dpkg --get-architecture openssl
Souhait=inconnU/Installé/suppRimé/Purgé/H=à garder
| État=Non/Installé/fichier-Config/dépaqUeté/échec-conFig/H=semi-installé/W=attend-
|/ Err?=(aucune)/besoin Réinstallation (État,Err: majuscule=mauvais)
||/ Nom                Version             Architecture Description
+++-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
ii  openssl             1.1.1c-1           armhf          Secure Sockets Layer toolkit - cryptog
pi@raspberrypi:~/certifs/broker $
```

q pour quitter.

## Création des clés et certificats

```
$ mkdir certs
$ cd certifs
$ mkdir ca
$ cd ca/
$ openssl req -new -x509 -days 365 -extensions v3_ca -keyout ca.key -out ca.crt
```

```
pi@raspberrypi:~/certifs/ca $ openssl req -new -x509 -days 365 -extensions v3_
ey -out ca.crt
Generating a RSA private key
.....+++++
.....+++++
writing new private key to 'ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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]:PROVENCE
Locality Name (eg, city) []:AIX
Organization Name (eg, company) [Internet Widgits Pty Ltd]:FOURCADE
Organizational Unit Name (eg, section) []:BTSSN
Common Name (e.g. server FQDN or YOUR name) []:brokerMQTT
Email Address []:
pi@raspberrypi:~/certifs/ca $
```

Remarque : le mot de passe sera 123456.

À présent les certificats et clés sont créés dans deux fichiers.

```
pi@raspberrypi:~/certifs/ca $ ls
ca.crt  ca.key
cd .. pour remonter d'un niveau
```

Le certificat de base étant créée dans le répertoire ca, il est possible de créer les clés et certificats pour le broker.

Génération d'une clé privée !

```
pi@raspberrypi:~/certifs/ca $ cd ..
pi@raspberrypi:~/certifs $ mkdir broker
pi@raspberrypi:~/certifs $ cd broker
pi@raspberrypi:~/certifs/broker $ openssl genrsa -out broker.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
pi@raspberrypi:~/certifs/broker $ ls
broker.key
pi@raspberrypi:~/certifs/broker $
```

### Création du fichier de requêtes de signatures

```
$ openssl req -out broker.csr -key broker.key -new
```

```
pi@raspberrypi:~/certifs/broker $ openssl req -out broker.csr -key broker.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]:PROVENCE
Locality Name (eg, city) []:AIX
Organization Name (eg, company) [Internet Widgits Pty Ltd]:FOURCADE
Organizational Unit Name (eg, section) []:BTSSN
Common Name (e.g. server FQDN or YOUR name) []:brokerMQTT
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456
An optional company name []:FOURCADE
pi@raspberrypi:~/certifs/broker $ ls
broker.csr  broker.key
pi@raspberrypi:~/certifs/broker $
```

Envoie du fichier. csr (Certificate Signing Request) de requête de signature du certificat vers l'autorité de validation :

```
openssl x509 -req -in broker.csr -CA ../ca/ca.crt -CAkey ../ca/ca.key -
CAcreateserial -out broker.crt -days 100
```

```
pi@raspberrypi:~/certifs/broker $ openssl x509 -req -in broker.csr -CA ../ca/ca.crt -CAkey ..
/ca/ca.key -CAcreateserial -out broker.crt -days 100
Signature ok
subject=C = FR, ST = PROVENCE, L = AIX, O = FOURCADE, OU = BTSSN, CN = brokerMQTT
Getting CA Private Key
Enter pass phrase for ../ca/ca.key:
pi@raspberrypi:~/certifs/broker $ ls
broker.crt  broker.csr  broker.key
pi@raspberrypi:~/certifs/broker $
```

## Création du certificat du client

Il suffit de recommencer pour le client :

```
openssl genrsa -out client.key 2048
openssl req -out client.csr -key client.key -new
openssl x509 -req -in client.csr -CA ../ca/ca.crt -CAkey ../ca/ca.key -
CAcreateserial -out client.crt -days 100
```

```
pi@raspberrypi:~/certifs/client$ openssl req -out client.csr -key client.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]:PROVENCE
Locality Name (eg, city) []:AIX
Organization Name (eg, company) [Internet Widgits Pty Ltd]:FOURCADE
Organizational Unit Name (eg, section) []:BTSSN
Common Name (e.g. server FQDN or YOUR name) []:clientMQTT
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456
An optional company name []:FOURCADE
pi@raspberrypi:~/certifs/client$ openssl x509 -req -in client.csr -CA ../ca/ca.crt -CAkey ..
/ca/ca.key -CAcreateserial -out client.crt -days 100
Signature ok
subject=C = FR, ST = PROVENCE, L = AIX, O = FOURCADE, OU = BTSSN, CN = clientMQTT
Getting CA Private Key
Enter pass phrase for ../ca/ca.key:
pi@raspberrypi:~/certifs/client$ ls
client.crt  client.csr  client.key
pi@raspberrypi:~/certifs/client$ cd ..
pi@raspberrypi:~/certifs$
```

Vérifier la structure de la configuration avec tree :

```
pi@raspberrypi:~$ tree certifs
certifs
├── broker
│   ├── broker.crt
│   ├── broker.csr
│   └── broker.key
├── ca
│   ├── ca.crt
│   ├── ca.key
│   └── ca.srl
└── client
    ├── client.crt
    ├── client.csr
    └── client.key

3 directories, 9 files
pi@raspberrypi:~$
```

## IOT et PROTOCOLE MQTT – serveur MQTT sur Raspberry Pi

Les fichiers clés sont codés en ASCII et peuvent donc être visualisés :

Exemple pour le fichier broker.key :

```
pi@raspberrypi: ~/certifs/broker
pi@raspberrypi:~/certifs/broker $ cat broker.key
-----BEGIN RSA PRIVATE KEY-----
MIIEEwIBAAKCAQEAA4fYCVRI9tqU3zrODxmBgb7CPC4/04DyRbdk6p8A1DjGWkj3E
yB1MODjHg1rzVmxmHdSIXhIHpOTqZqMBSB4of5Asy/mHJaoTnqx87e+sSZY4x/LS
oci3GumRi6TJEY0WEJMUbx2Cdq9+ou0EI9jkHIayeyHS5j4IEtJP/q7nTcgrXExU
Ye2RjyMJXcoT7v0E8SBBcGhuvP1ZqZHDjrJ/H6Z60WHNNprxIare0n9ERqNfwocS
Ww0PAzAUMgPeUBesrZjWUDy+kw9L3J05bsVZTHf8Ko9n2bMSdZA57jg3iDC3djb
20AifQcdFeeQc8p18RhLWdw0iSXE103wWZV1QIDAQABAoIBAQCybBP5YrbyUtGX
W06TpSRg7RR2t650J+VZz07g6PsZkxQU9MM+1MvL1SINWbLNqjiUE28+gJQio1Nn
ig0ICjBGUezdf4I0vnBRmbfoFZmAXasmT1n8v8W97XKsxoIyYCptGgZGUpXC695d
/jOHrsqor7+dNN8YSSEM456g5zDjm9A9q4i+Lq2WxRCyHxL0dYk18dD0E8tyD7dg
JvwuulmiaxDocTvZmVwOpYGW+Dx+0xIac0CpzsB5DgU4sbeoZ6Rm3I6t0vTd00c
NxDMUDYwGLSGXNc2Eq5rT7dB0s5qoAR1+aXYpC/SVdAxx6rdUsD6BfdocCBFVom4
675Jxw59AoGBAPy44y+iIbwrldFRftB7mBurvZa0o1Ef0itp9KBfCo2BrPjTaOvt
5d7WBRzH6egJYSIxK5vu4rs7uMZDZ+hHApYoQewWFnUnGxRNQJRcELrBn0tPbpG6
0tS/TKkm/83wAKudf6bqpl7YqM8Sv/HLGxNP1PGiP7BA1fs2EqpB/wBXAoGBAOTk
RDPdEfpBvYL010wKVVCqO+G5na6M5w01Xa+mxGIbfk4JoxUzgCPMt+g3nvnGnJNZ
7mFPxgjMLKZUBOLFGR+TT+amUnDONvXYMjfnCjxXx+LK18BSVPCuBXTw/XZgbrj1
6uaE71gh2srRQ/G0GXA9Epsg+plwdoDSyV8xDA+zAoGAQ7rZSD7Pg7qyLwEEV6OI
FIJmgSlQS8FuuYvf/8r47Qu/67/c8QAPBzWckSYShkVazB6W4QeR8etsZrU5I+D
Z7DAWiFhS2x7mWlK0XgLE8Ioiupy26NpePL7/K1zDSIxS80eFQH2LJ0sQLLptkeU
rgxLKacuInJsv7KyNecSDGkCgYAYG0P0lgZBR/I6n2ua3ggeR6Qjm4Sv15dYbPE8
aLvC3VGfsuxGz6bULxJxePsKoDXU6nIKSnOFqWp8XRdUqt780j1Cu1SA7CGLXW5g
Xf9/uJa/RKhCJ8L/ipcXKUfQIPqy7V+kZKQeHgrNKVA9rrrWCo1tXKLiwrLTDLV91
vj5tkQKBgCDq1+kDg6uQk+6cUrzWuXQDimepxx/7r1nNXvpcUH0+Y0dedPCF4Gb/
MvAMFvKIz6TOGs8DQF3qAJFAhT17Y91x72YN63ZfeFwbxH2EySQ05CNm2WJhsqgH
okQK72wKmRxHq107U1Jb2QSnier1YTVzW/sgdf/meiH1kS0kwqiB
-----END RSA PRIVATE KEY-----
pi@raspberrypi:~/certifs/broker $
```

Ce n'est pas très intéressant mais montre la complexité d'une clé RSA 2048 bits.

## Configuration du serveur MQTT avec SSH

Modifier le fichier de configuration de mosquitto comme suit :

Configuration de mosquitto pour utiliser le cryptage SSL :

```
$ sudo nano /etc/mosquitto/mosquitto.conf
```

Ajouter :

```
port 8883
cafile /home/openest/certs/ca/ca.crt
certfile /home/openest/certs/broker/broker.crt
keyfile /home/openest/certs/broker/broker.key
```

```
GNU nano 3.2 /etc/mosquitto/mosquitto.conf

# 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 /var/run/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

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

include_dir /etc/mosquitto/conf.d

allow_anonymous false
password_file /etc/mosquitto/passwd

port 8883

cafile /home/openest/certifs/ca/ca.crt
certfile /home/openest/certifs/broker/broker.crt
keyfile /home/openest/certifs/broker/broker.key
require_certificate true
```

```
require_certificate true
```

port 8883 indique que les messages arrivant sur le port 8883 (port utilisé par MQTT en SSH) utilise les fichiers de certification et la clé donnés ensuite.

cafile : fichier de certification SSH (auto certification) ;

certfile : fichier de clé publique du broker ;

keyfile : fichier de clé privée du broker.

## Essais MQTT avec TLS

Au préalable installer sur le PC équipé de MQTT.fx le client MQTT: <http://workswithweb.com/mqttbox.html>

Sur Raspberry arrêter le service mosquitto :

```
$ systemctl stop mosquitto
```

Entrer le compte et le passe raspberry demandés.

Lancer mosquitto en demandant de charger le fichier de configuration mosquitto.conf :

```
$ sudo mosquitto -c /etc/mosquitto/mosquitto.conf
```

Le broker attend maintenant une requête sur les ports 1883 (non crypté) et 8883 (crypté SSH).

Refaire un essai avec MQTT.fx, connexion (port 1883) , souscription à topic0 et publication, comme précédemment.

Lancer en plus MQTTbox, que l'on configure pour crypter les messages en SSH (cliquer sur la petite roue).

The screenshot shows the MQTTBox application window with the title bar 'MQTTBox Edit Help'. The main menu bar includes 'Menu', 'MQTT CLIENT SETTINGS', and 'Client Settings Help'. The settings are organized into a grid of fields and checkboxes:

- MQTT Client Name:** clientPC
- MQTT Client Id:** totolito (with a refresh icon)
- Append timestamp to MQTT client id?:** ☒ Yes
- Broker is MQTT v3.1.1 compliant?:** ☒ Yes
- Protocol:** mqtt / tls (dropdown)
- Host:** 192.168.1.150:8883
- Clean Session?:** ☒ Yes
- Auto connect on app launch?:** ☒ Yes
- SSL / TLS Version:** Auto (dropdown)
- SSL / TLS Certificate Type:** CA signed server certificate (dropdown)
- Username:** admin
- Password:** \*\*\*\*\*
- Reschedule Pings?:** ☒ Yes
- Queue outgoing QoS zero messages?:** ☒ Yes
- Reconnect Period (milliseconds):** 1000
- Connect Timeout (milliseconds):** 30000
- KeepAlive (seconds):** 10
- Will - Topic:** topic0
- Will - QoS:** 0 - Almost Once (dropdown)
- Will - Retain:** ☐ No
- Will - Payload:** (empty text area)

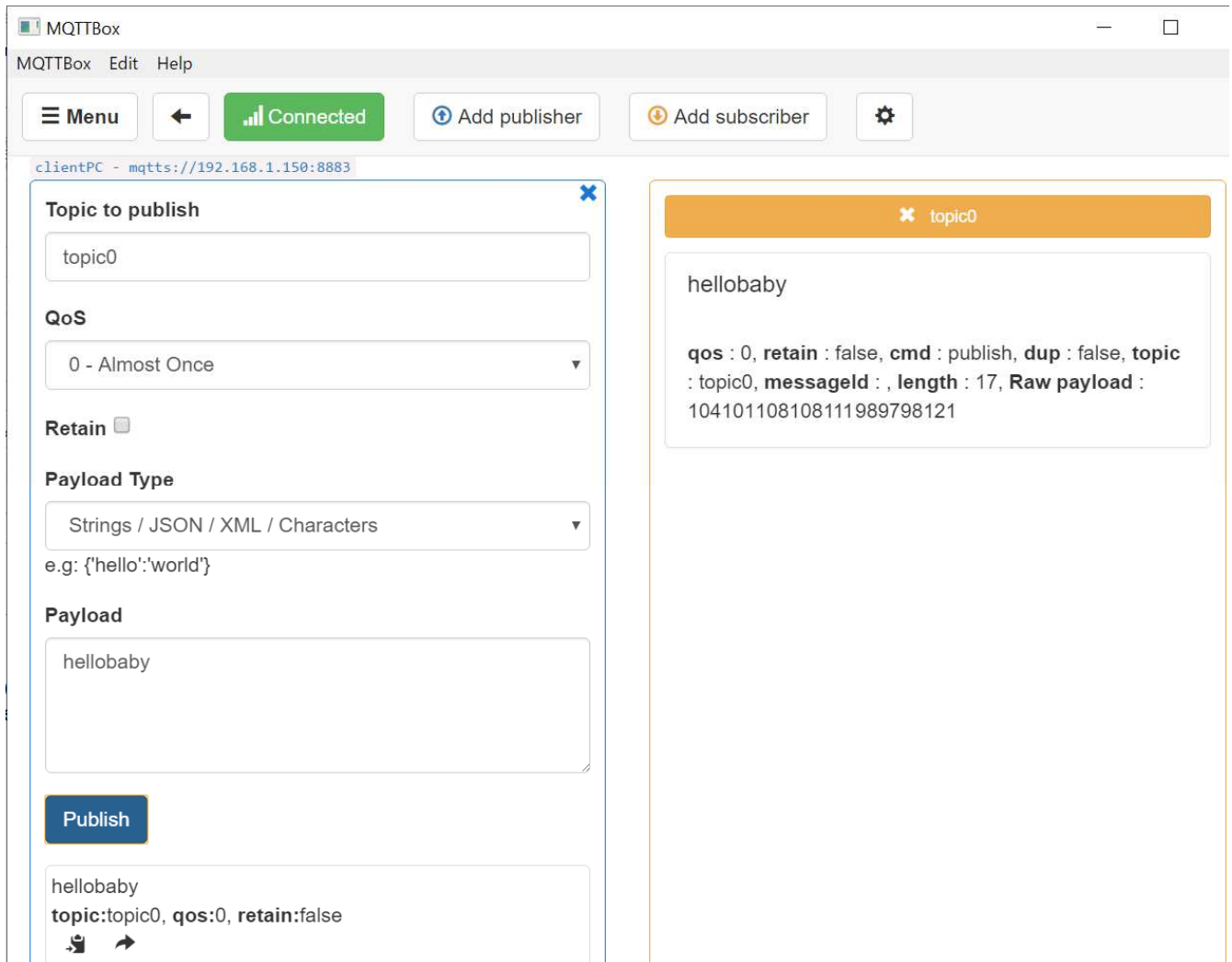
At the bottom, there are 'Save' and 'Delete' buttons.

Revenir sur l'écran de communication, se connecter sur le Raspberry (bouton Connected).

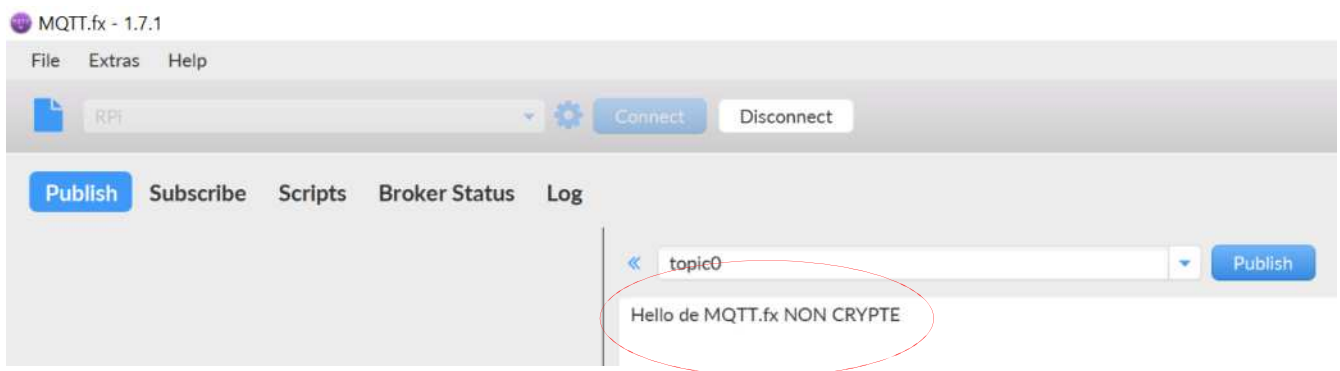
Souscrire au topic0.

Publier un message...

## IOT et PROTOCOLE MQTT – serveur MQTT sur Raspberry Pi



WireShark permet l'analyse fine des messages réseaux :



Interception du message non crypté :

entrer le filtre : `tcp.port in {1883 8883}`

Envoyer un message non crypté avec MQTT.fxMQTTbox.

On peut très bien lire le message.

## IOT et PROTOCOLE MQTT – serveur MQTT sur Raspberry Pi

\*Wi-Fi

Fichier Editer Vue Aller Capture Analyser Statistiques Telephonie Wireless Outils Aide

top port in (1883 8883)

No.	Time	Source	Destination	Protocol	Length	Info
18	2.533686	192.168.1.67	192.168.1.150	MQTT	91	Publish Message [topic0]
19	2.536221	192.168.1.150	192.168.1.67	TLSv1.2	120	Application Data
20	2.550394	192.168.1.150	192.168.1.67	MQTT	92	Publish Message [topic0]
21	2.577152	192.168.1.67	192.168.1.150	TCP	54	27197 → 8883 [ACK] Seq=1 Ack=67 Win=511 Len=0
22	2.590338	192.168.1.67	192.168.1.150	TCP	54	27249 → 1883 [ACK] Seq=38 Ack=38 Win=513 Len=0

> Frame 18: 91 bytes on wire (728 bits), 91 bytes captured (728 bits) on interface 0

> Ethernet II, Src: IntelCor\_89:66:fb (a0:af:bd:89:66:fb), Dst: Raspberr\_64:8e:0b (dc:a6:32:64:8e:0b)

> Internet Protocol Version 4, Src: 192.168.1.67, Dst: 192.168.1.150

▼ Transmission Control Protocol, Src Port: 27249, Dst Port: 1883, Seq: 1, Ack: 1, Len: 37

Source Port: 27249

Destination Port: 1883

[Stream index: 3]

[TCP Segment Len: 37]

Sequence number: 1 (relative sequence number)

[Next sequence number: 38 (relative sequence number)]

Acknowledgment number: 1 (relative ack number)

0101 .... = Header Length: 20 bytes (5)

```

0000  dc a6 32 64 8e 0b a0 af  bd 89 66 fb 08 00 45 00  ..2d....f...E.
0010  00 4d 29 bb 40 00 80 06  4c c6 c0 a8 01 43 c0 a8  ..M)@L...C.
0020  01 96 6a 71 07 5b 3e 19  b1 e1 76 d5 e1 c0 50 18  ..jq[>...v...P.
0030  02 01 ca bf 00 00 30 23  00 06 74 6f 70 69 63 30  ....0#...topic0
0040  48 65 6c 6c 6f 20 64 65  20 4d 51 54 54 2e 66 78  Hello de MQTT.fx
0050  20 4e 4f 4e 20 43 52 59  50 54 45                NON CRY PTE
  
```

## IOT et PROTOCOLE MQTT – serveur MQTT sur Raspberry Pi

Refaire la manipulation avec MQTTBox (qui crypte les messages).

Le message est maintenant crypté .

On remarque les échanges de clés.

The image shows a Wireshark capture of network traffic on a Raspberry Pi. The top pane displays a list of packets. Packet 6 is selected, showing a TLSv1.2 Encrypted Alert message. The bottom pane provides a detailed view of this message, including the TLSv1.2 Record Layer and the Encrypted Alert content.

No.	Time	Source	Destination	Protocol	Length	Info
2	1.758002	192.168.1.67	192.168.1.150	TCP	54	27197 → 8883 [FIN, ACK] Seq=1 Ack=1 Win=507 Len=0
3	1.776962	192.168.1.150	192.168.1.67	TLSv1.2	88	Encrypted Alert
4	1.776963	192.168.1.150	192.168.1.67	TCP	56	8883 → 27197 [FIN, ACK] Seq=32 Ack=2 Win=254 Len=0
5	1.776963	192.168.1.150	192.168.1.67	MQTT	64	Publish Message [topic0]
6	1.777026	192.168.1.67	192.168.1.150	TCP	54	27197 → 8883 [RST, ACK] Seq=2 Ack=32 Win=0 Len=0
7	1.817812	192.168.1.67	192.168.1.150	TCP	54	27249 → 1883 [ACK] Seq=1 Ack=11 Win=513 Len=0
8	2.935655	192.168.1.67	15.236.203.194	MQTT	56	Ping Request

Frame 3: 88 bytes on wire (704 bits), 88 bytes captured (704 bits) on interface 0  
> Ethernet II, Src: Raspberr\_64:8e:0b (dc:a6:32:64:8e:0b), Dst: IntelCor\_89:66:fb (a0:af:bd:89:66:fb)  
> Internet Protocol Version 4, Src: 192.168.1.150, Dst: 192.168.1.67  
Transmission Control Protocol, Src Port: 8883, Dst Port: 27197, Seq: 1, Ack: 2, Len: 31  
Source Port: 8883  
Destination Port: 27197  
[Stream index: 0]  
[TCP Segment Len: 31]  
Sequence number: 1 (relative sequence number)  
[Next sequence number: 32 (relative sequence number)]  
Acknowledgment number: 2 (relative ack number)  
0101 .... = Header Length: 20 bytes (5)  
Flags: 0x018 (PSH, ACK)  
000. .... = Reserved: Not set  
...0 .... = Nonce: Not set  
... 0... = Congestion Window Reduced (CWR): Not set  
... .0.. = ECN-Echo: Not set  
... ..0. = Urgent: Not set  
... ...1 = Acknowledgment: Set  
... ....1... = Push: Set  
... ..0.. = Reset: Not set  
... ....0. = Syn: Not set  
... ....0 = Fin: Not set  
[TCP Flags: .....AP...]  
Window size value: 254  
[Calculated window size: 254]  
[Window size scaling factor: -1 (unknown)]  
Checksum: 0xe28d [unverified]  
[Checksum Status: Unverified]  
Urgent pointer: 0  
> [SEQ/ACK analysis]  
> [Timestamps]  
TCP payload (31 bytes)  
Transport Layer Security  
TLSv1.2 Record Layer: Encrypted Alert  
Content Type: Alert (21)  
Version: TLS 1.2 (0x0303)  
Length: 26  
Alert Message: Encrypted Alert

0000 a0 af bd 89 66 fb dc a6 32 64 8e 0b 08 00 45 00 .....f... 2d....E.  
0010 00 47 01 ab 40 00 40 06 b4 dc c0 a8 01 96 c0 a8 .G..@.....  
0020 01 43 22 b3 6a 3d 8e f0 1a 98 af bd 58 24 50 18 .C".j=...X\$P.  
0030 00 fe e2 8d 00 00 15 03 03 00 1a 41 50 ab 58 f2 .....APX.  
0040 99 63 fe 04 24 95 58 e7 1a 72 e6 33 4b 61 d8 dd .c..\$.X..r.3Ka..  
0050 5f d6 12 1a 82 b3 08 ae .....  
\_.....

- Ligne 6 : envoi de la clé publique du client ;
- Ligne 5 : annonce du type de message (MQTT) ;
- Ligne 4 : envoi de la clé publique du serveur ;
- Ligne 3 : Envoi du message crypté ;
- Ligne 2 : acquittement.