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.

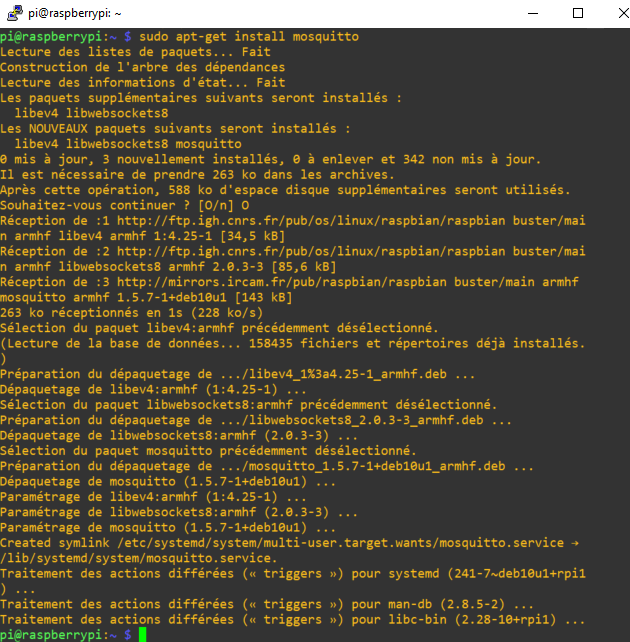
MOSQUITTO ( <https://mosquitto.org/> ) propose une installation gratuite d’un broker open source.  
MOSQUITTO est porté sur Windows, MAC et la plupart des distributions Linux dont Raspbian.

# Installation d'un broker MQTT Mosquitto sur RPi

Ouvrir une session SSH sur RPi :

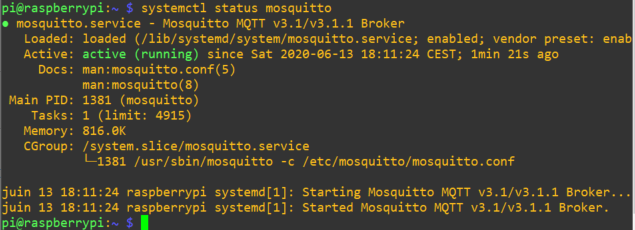
~$ sudo apt-get update

~$ sudo apt-get install mosquitto



Vérifier que le service mosquitto est actif :

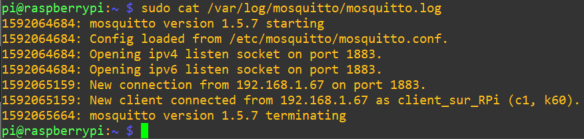
~$ systemctl status mosquitto

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.*

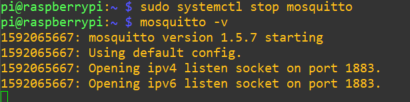
Un fichier .log enregistre toutes les activités du broker, pour le visualiser :  
sudo cat /var/log/mosquitto/mosquitto.log



### Démarrage et arrêt du service mosquitto

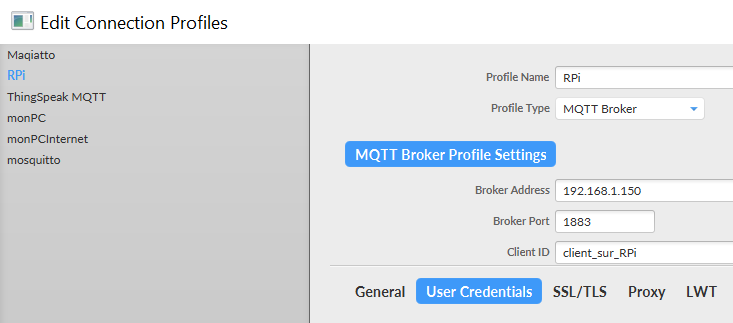
sudo systemctl start mosquitto  
sudo systemctl stop mosquitto  
sudo systemctl restart mosquitto

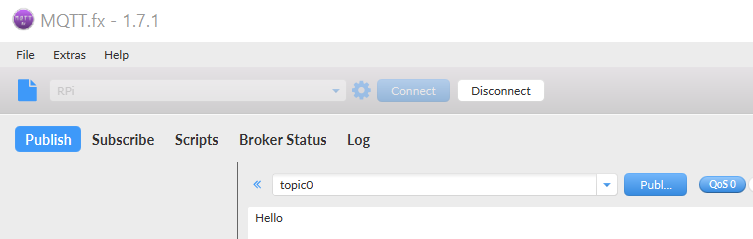
Après arrêt du service il est possible de lancer mosquitto :



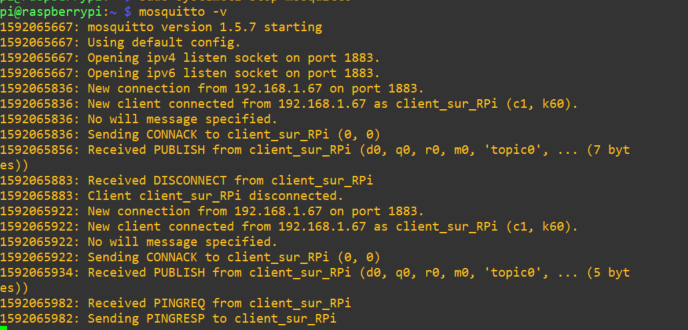
mosquitto  
mosquitto -v   
(affiche tous les échanges)

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).





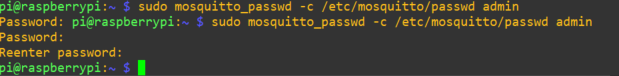
Le broker affiche les échanges. Il interroge régulièrement le client afin de vérifier la connexion.  
En effet si le client est abonné à un topic le broker lui transmettra tout message sur ce topic.



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

### Configuration des mots de passe MQTT

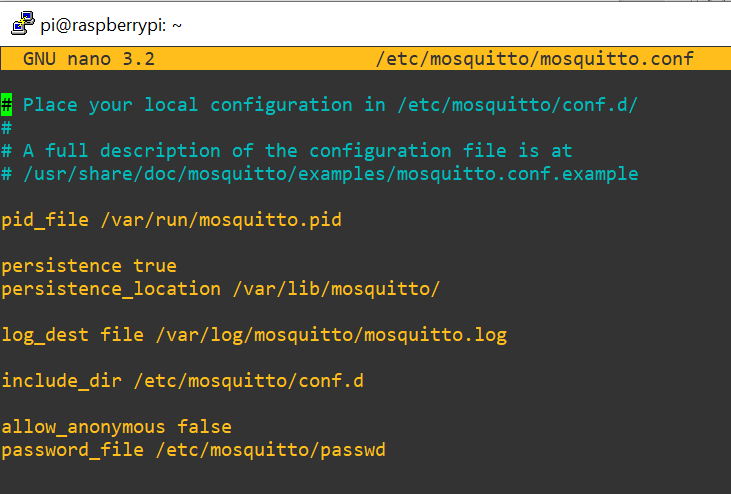
mosquitto\_passwd ( <https://mosquitto.org/man/mosquitto_passwd-1.html> ) est un utilitaire permettant de générer un fichier de mot de passe.  
Les mots de passe sont stockés dans /etc/mosquitto/passwd :  
sudo mosquitto\_passwd c /etc/mosquitto/passwd admin (-c pour create)

( 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

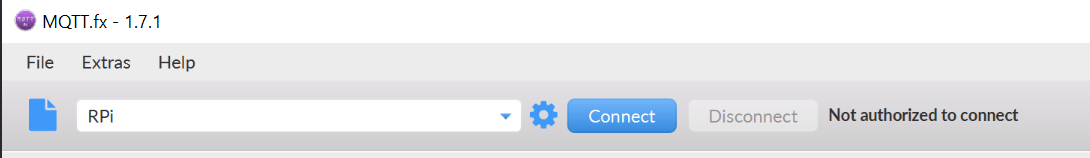
Compléter le fichier comme suit



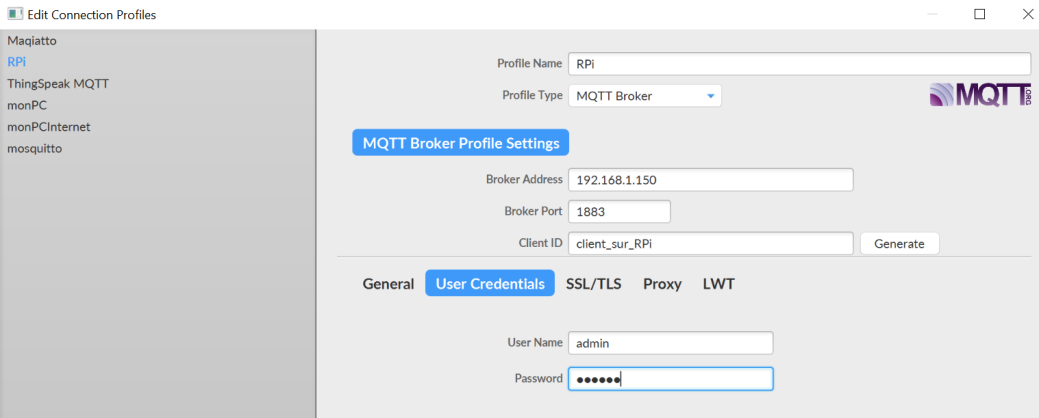
allow\_anonymous false : désactive toutes les connexions non authentifiées ;  
password\_file : indique à Mosquitto où rechercher les informations sur l’utilisateur et le mot de passe.

redémarrer Mosquitto avec :

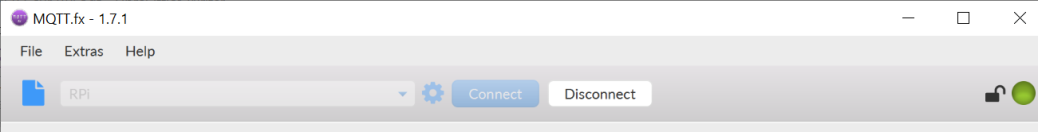
sudo systemctl restart mosquitto

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

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.

# Cryptage SSL/TLS

SSL est normalisé TLS : <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>

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.

J’ai intercepté la clé, je peux tout décrypter





J’ai un message pour toi

Ok voici ma clé privée de cryptage

Voici mon message crypté

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 chiffrement RSA. <https://fr.wikipedia.org/wiki/Chiffrement_RSA>

Dans ce type de chiffrement 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.

Je comprends plus rien, la clé de cryptage ne permet pas le décryptage





J’ai un message pour toi

Ok voici ma clé publique de cryptage

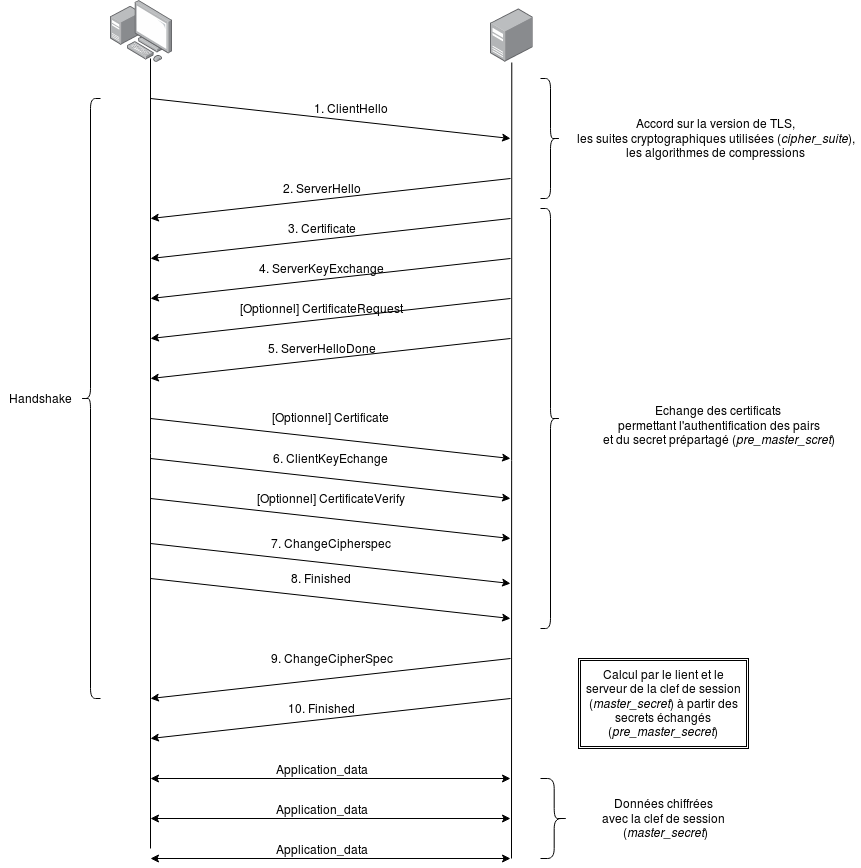
Voici mon message crypté avec ta clé publique



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

Le chiffrement 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](https://fr.wikipedia.org/wiki/Certificat_numérique) [X.509](https://fr.wikipedia.org/wiki/X.509) délivré par une [autorité de certification](https://fr.wikipedia.org/wiki/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) ;

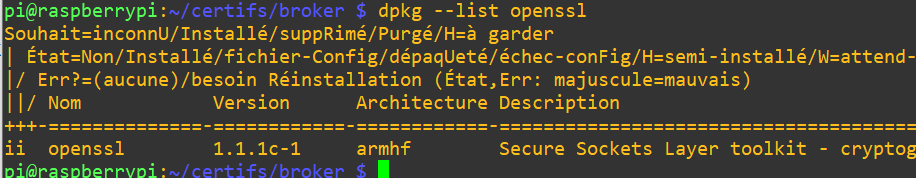
Principe d’un échange TLS (cf Wikipédia) :

### Configuration de MQTT SSL sur Raspberry Pi

***OpenSSL*** est une boîte à outils de [chiffrement](https://fr.wikipedia.org/wiki/Chiffrement) comportant deux [bibliothèques](https://fr.wikipedia.org/wiki/Bibliothèque_logicielle), libcrypto et libssl, fournissant respectivement une implémentation des algorithmes [cryptographiques](https://fr.wikipedia.org/wiki/Cryptographie) et du [protocole de communication](https://fr.wikipedia.org/wiki/Protocole_de_communication) [SSL/TLS](https://fr.wikipedia.org/wiki/Transport_Layer_Security), ainsi qu'une [interface en ligne de commande](https://fr.wikipedia.org/wiki/Interface_en_ligne_de_commande), openssl.

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

OpenSSL est normalement préinstallé sur les distributions Rpi.  
Pour vérifier : $dpkg –list openssl :

q pour quitter.

### Création des clés et certificats

$ mkdir certs

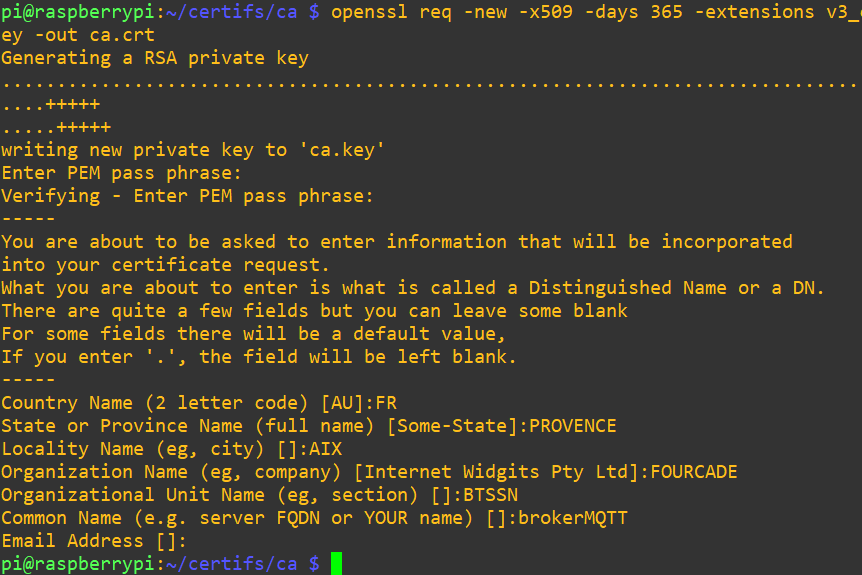
$ cd certifs

$ mkdir ca

$ cd ca/

$ openssl req -new -x509 -days 365 -extensions v3\_ca -keyout ca.key -out ca.crt

*Remarque : le mot de passe sera 123456.*

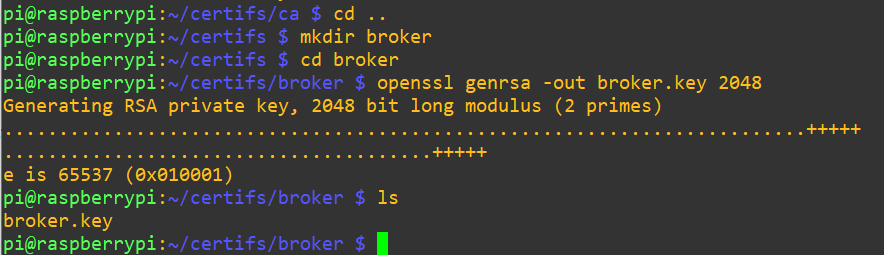


À présent les certificats et clés sont créés dans deux fichiers.

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

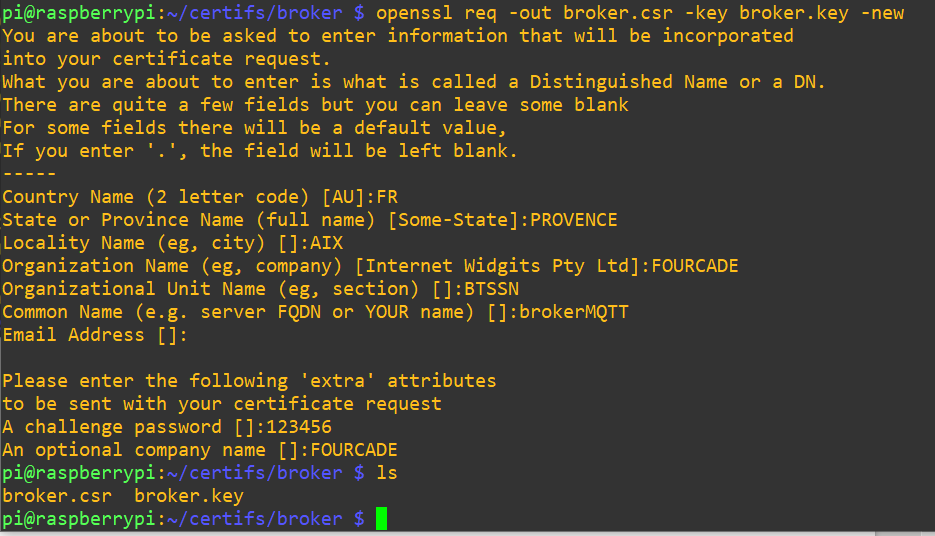
Le certificat de base étant créée dans le répertoire ca, il est possible de créer les clés et certificats pour le broker.

Génération d’une clé privée !

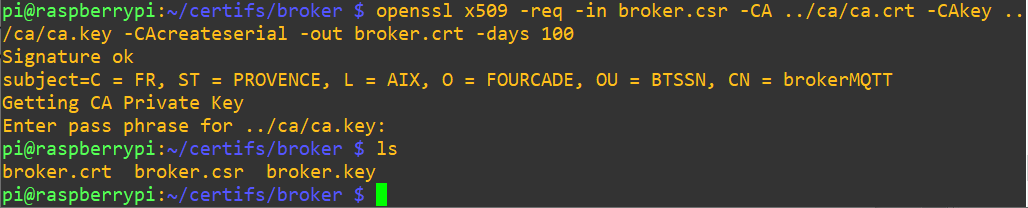


### Création du fichier de requêtes de signatures

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



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

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

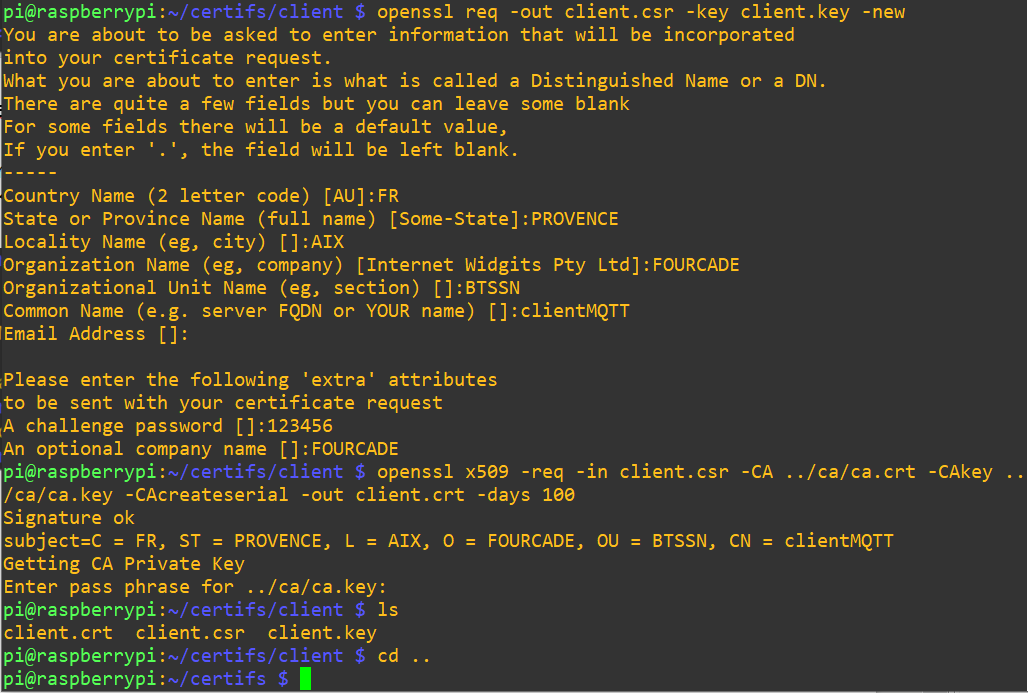
### Création du certificat du client

Il suffit de recommencer pour le client :

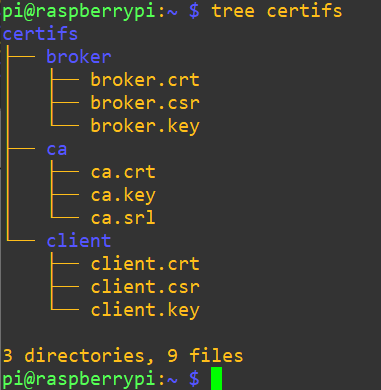
openssl genrsa -out client.key 2048

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



Vérifier la structure de la configuration avec tree :



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

Exemple pour le fichier broker.key :



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 :

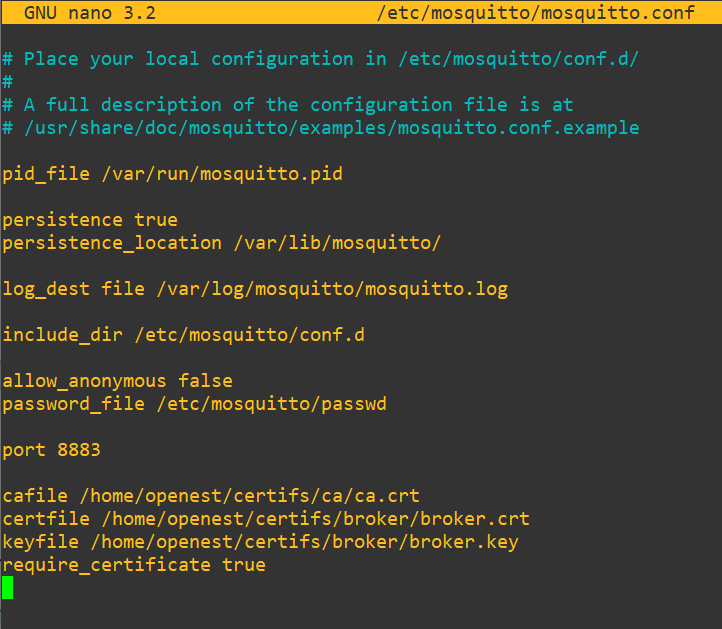
port 8883

cafile /home/openest/certs/ca/ca.crt

certfile /home/openest/certs/broker/broker.crt

keyfile /home/openest/certs/broker/broker.key

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

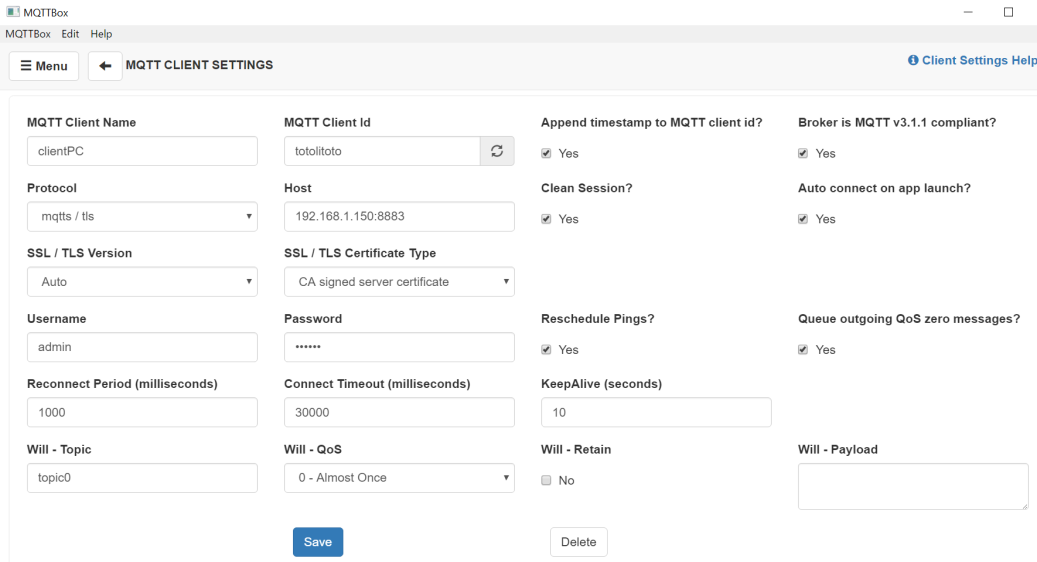
Au préalable installer sur le PC équipé de MQTT.fx le client MQTT: <http://workswithweb.com/mqttbox.html>

Sur Raspberry arrêter le service mosquitto :   
$ systemctl stop mosquitto

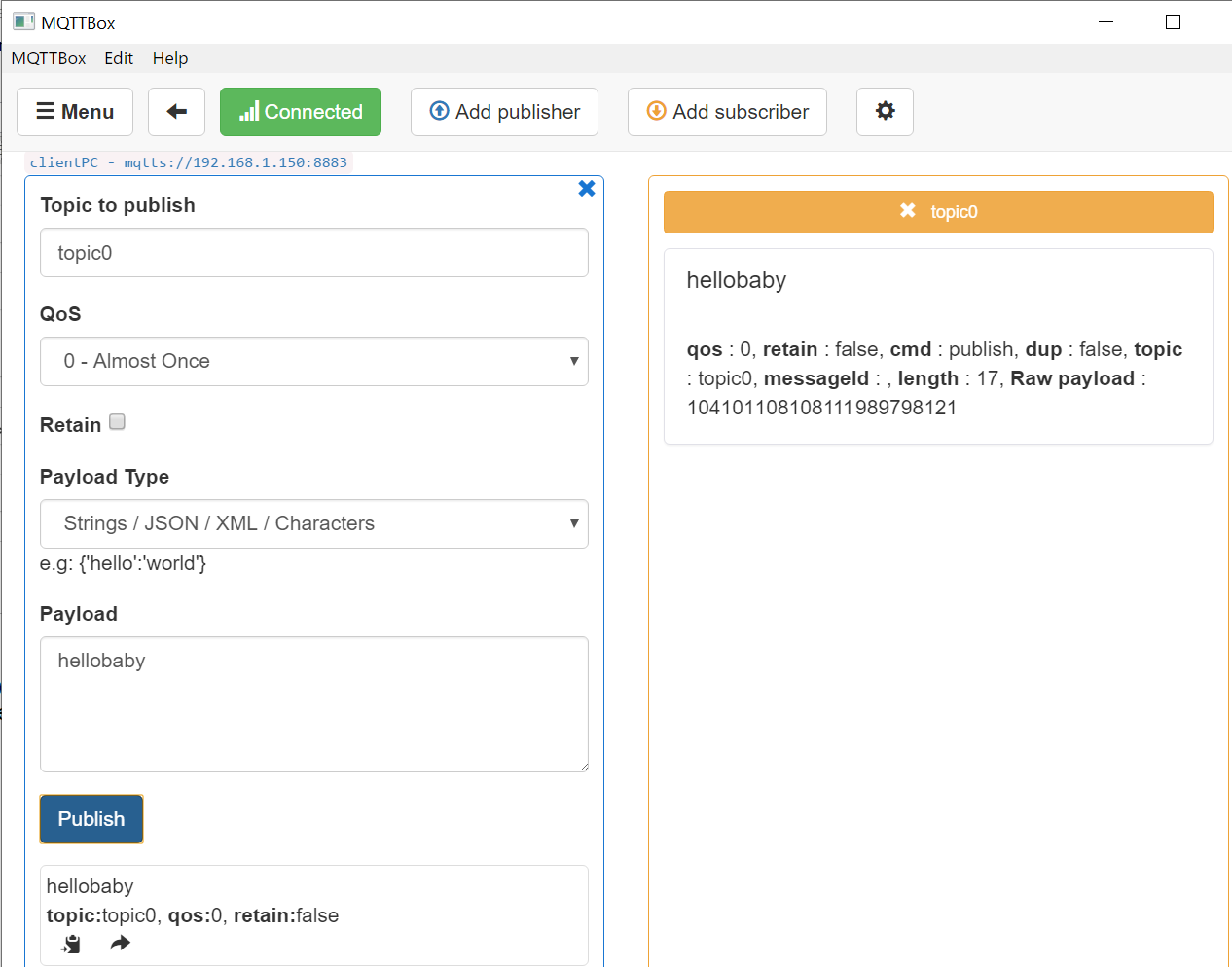
Entrer le compte et le passe raspberry demandés.  
  
Lancer mosquitto en demandant de charger le fichier de configuration mosquitto.conf :  
$ sudo mosquitto -c /etc/mosquitto/mosquitto.conf

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

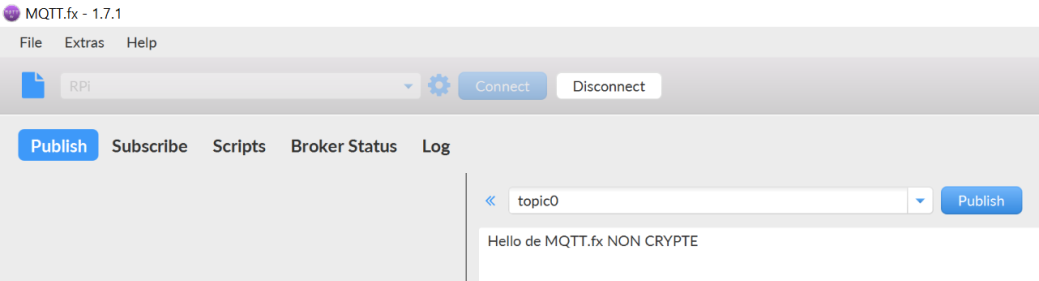
Refaire un essai avec MQTT.fx, connexion (port 1883) , souscription à topic0 et publication, comme précédemment.  
Lancer en plus MQTTbox, que l’on configure pour crypter les messages en SSH (cliquer sur la petite roue).



Revenir sur l’écran de communication, se connecter sur le Raspberry (bouton Connected).  
Souscrire au topic0.  
Publier un message…



WireShark permet l’analyse fine des messages réseaux :

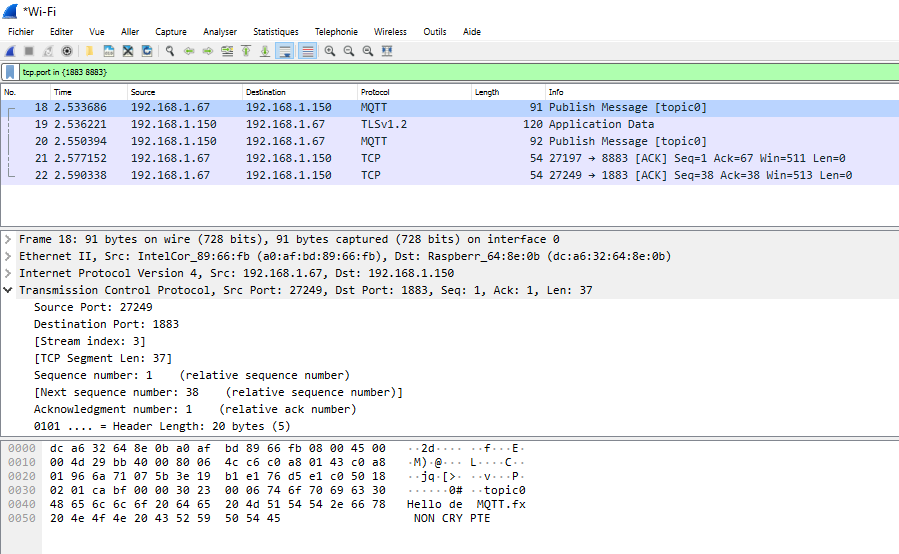


Interception du message non crypté :

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

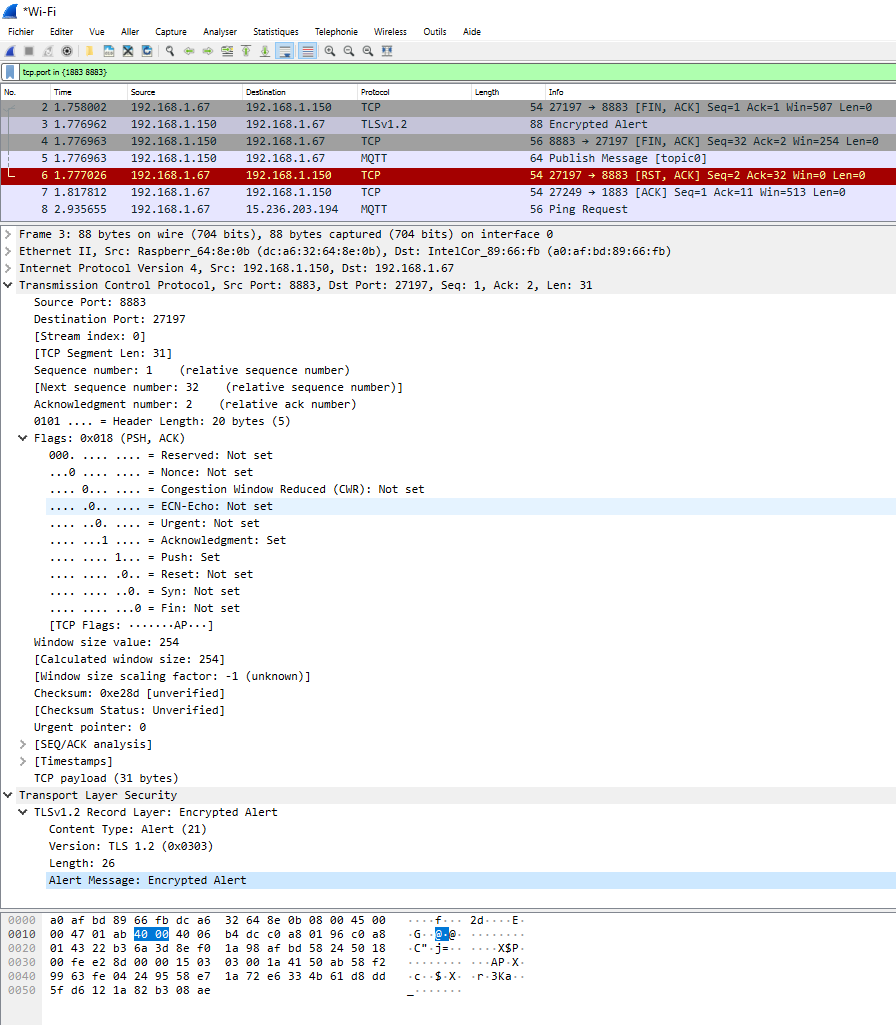
Envoyer un message non crypté avec MQTT.fxMQTTbox.

On peut très bien lire le message.



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

Le message est maintenant crypté .  
On remarque les échanges de clés.



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