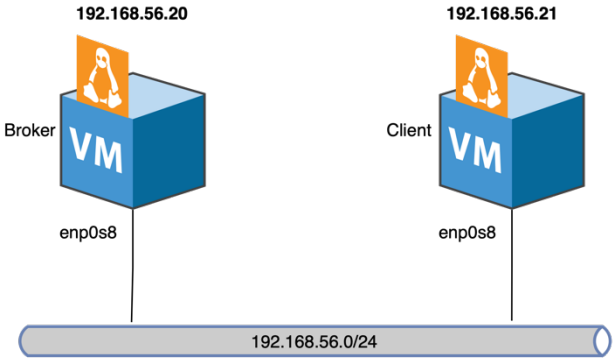
TP
SÉCURISATION D'UN CONNEXION MQTT

Objectif terminal :
Sécuriser une connexion mqtt pour le client et le serveur.

Objectifs intermédiaires :

- Créer des machines virtuelles en utilisant virtualbox et vagrant.
- Créer un certificat SSL/TLS.
- Configurer un broker MQTT :
  - Configurer le MQTTS.
  - Implémenter les certificats.
  - Implémenter l'authentification par mot de passe.
- Utiliser un certificat pour authentifier le client.

TOPOLOGIE DU TP :



Nous allons utiliser Vagrant et virtualbox pour déployer deux machines.

CRÉATION DES MACHINES VIRTUELLES :

1. Lancer un terminal.

2. Télécharger le fichier Vagrantfile :

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

*Nous allons utiliser **VirtualBox** et **Vagrant** pour déployer deux machines virtuelles (VMs). **VirtualBox** est un hyperviseur qui permet d'exécuter des VMs sur un hôte, tandis que **Vagrant** est un outil d'automatisation qui facilite la gestion, la configuration et le provisionnement des VMs via des fichiers de configuration. Cette approche permet de déployer rapidement des environnements reproductibles et cohérents.*

```
cat  Vagrantfile
```

```
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/focal64"

  config.vm.define "broker" do |broker|
    broker.vm.hostname = "broker"
    broker.vm.network "private_network", ip: "192.168.56.20"

    broker.vm.provider "virtualbox" do |vb|
      vb.memory = "1024"
      vb.cpus = 1
    end

    broker.vm.provision "shell", inline: <<-SHELL
      # Création de l'utilisateur client
      id -u client &>/dev/null || useradd -m -s /bin/bash client
      echo "client:password123" | chpasswd

      # Génération de la clé SSH pour l'utilisateur vagrant
      sudo -u vagrant mkdir -p /home/vagrant/.ssh
      sudo -u vagrant ssh-keygen -t rsa -N "" -f /home/vagrant/.ssh/id_rsa || true

      sudo apt-get update

       # Installer openssl et mosquitto
      sudo apt-get install -y openssl mosquitto-clients mosquitto
      # Configuration du pare-feu UFW
      sudo ufw default deny incoming
      sudo ufw default allow outgoing
      sudo ufw allow 22/tcp   # SSH
      sudo ufw allow 1883/tcp  # MQTT sans TLS
      sudo ufw allow 8883/tcp  # MQTT avec TLS
      echo "y" | sudo ufw enable  # Force l'activation sans confirmation

      # Vérification du statut
      sudo ufw status verbose

      # Copie de la clé publique
      cp /home/vagrant/.ssh/id_rsa.pub /vagrant/broker_key.pub
    SHELL
  end

  config.vm.define "client" do |client|
    client.vm.hostname = "client"
    client.vm.network "private_network", ip: "192.168.56.21"

    client.vm.provider "virtualbox" do |vb|
      vb.memory = "1024"
      vb.cpus = 1
    end

    client.vm.provision "shell", inline: <<-SHELL
      # Création de l'utilisateur client
      id -u client &>/dev/null || useradd -m -s /bin/bash client
      echo "client:password123" | chpasswd

      # Configuration du répertoire .ssh pour l'utilisateur client
      mkdir -p /home/client/.ssh
      chmod 700 /home/client/.ssh
      touch /home/client/.ssh/authorized_keys
      chmod 600 /home/client/.ssh/authorized_keys

      # Ajout de la clé publique du broker aux clés autorisées
```

```
      if [ -f /vagrant/broker_key.pub ]; then
        cat /vagrant/broker_key.pub >> /home/client/.ssh/authorized_keys
      fi

      sudo apt-get update

       # Installer openssl, mosquito et x11 pour utilizer des apps graphiques
      sudo apt-get install -y openssl mosquitto-clients xauth x11-apps

      # Ajouter la variable DISPLAY dans le fichier .bashrc pour qu'elle soit
disponible à chaque session
      echo "export DISPLAY=localhost:10.0" >> /home/vagrant/.bashrc

      # Correction des propriétés
      chown -R client:client /home/client/.ssh

      # Configuration explicite de sshd
      cat > /etc/ssh/sshd_config.d/custom.conf << EOL
PasswordAuthentication yes
PubkeyAuthentication yes
PermitRootLogin no
EOL

      # Redémarrage du service SSH
      systemctl restart sshd
    SHELL
  end
end
```

3. Créer les machines virtuelles

```
vagrant up
```

4. Vérifier la création des machines virtuelles

```
vagrant status
```

```
Current machine states:

broker                    running (virtualbox)
client                    running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

5. Se connecter à la VM broker

```
vagrant ssh broker
```

6. Lancer un second terminal.

7. Se connecter à la VM client

```
vagrant ssh client -- -X
```

CRÉATION DES CERTIFICATS SERVEUR :

Nous allons utiliser openssl. C'est une boîte à outils de cryptographie robuste pour le SSL/TLS, utilisée pour sécuriser les communications sur les réseaux informatiques. Elle offre des fonctionnalités pour la création de clés cryptographiques, de certificats, la gestion de l'authentification SSL/TLS, et la réalisation de diverses opérations de chiffrement.

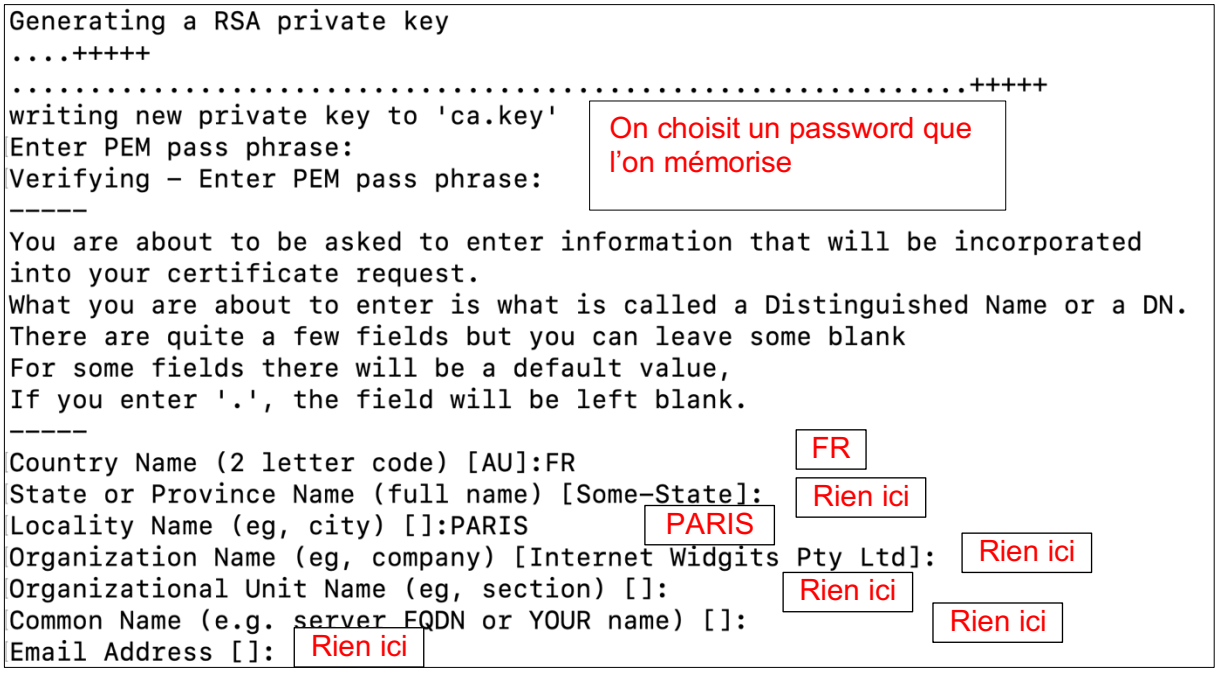
1. Créer un certificat CA :

Une Autorité de Certification (CA) est une entité de confiance qui émet des certificats numériques pour valider l'identité des parties dans une communication sécurisée par SSL/TLS.

Le certificat va nous permettre d'auto-signer notre certificat serveur sans passer par un véritable CA. On utilise cette méthode en local ou en mode développement. En production, il faudra passer par un CA de confiance.

Depuis la VM broker, saisissez la commande openssl suivante :

```
openssl req -new -x509 -days 1826 -extensions v3_ca -keyout ca.key -out ca.crt
```



- openssl req : Utilise le module req d'OpenSSL pour gérer les demandes de signature de certificat (CSR) et générer des certificats auto-signés.
- -new : Crée une nouvelle demande de signature de certificat (CSR).
- -x509 : Indique que l'on souhaite générer un certificat auto-signé plutôt qu'une simple CSR. Les certificats X.509 sont le standard pour les certificats SSL/TLS.
- -days 1826 : Définit la durée de validité du certificat à 1826 jours, soit environ 5 ans.
- -extensions v3_ca : Spécifie l'utilisation des extensions X.509 version 3 pour une autorité de certification, permettant d'ajouter des informations supplémentaires telles que les usages de clé, les politiques de certificat, etc.
- -keyout ca.key : Spécifie le fichier de sortie pour la clé privée générée (ca.key).
- -out ca.crt : Spécifie le fichier de sortie pour le certificat auto-signé généré (ca.crt).

2. Créer la clé privée pour le serveur :

```
openssl genrsa -out server.key 2048
```

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

- genrsa : Indique à OpenSSL de générer une nouvelle paire de clés RSA. RSA (Rivest-Shamir-Adleman) est l'un des premiers algorithmes de cryptographie à clé publique et est largement utilisé pour sécuriser les transmissions de données.
- -out server.key : Spécifie le fichier de sortie (server.key) où la clé privée RSA générée sera sauvegardée. Ce fichier contiendra la clé privée en format PEM (Privacy Enhanced Mail), un format de fichier utilisé pour stocker et échanger des données cryptographiques.
- 2048 : Définit la taille de la clé en bits. Dans cet exemple, une clé de 2048 bits est générée. La taille de la clé est un facteur important pour la sécurité du chiffrement ; 2048 bits sont considérés comme sécurisés pour une utilisation actuelle, offrant un bon équilibre entre sécurité et performance.

3. Créer une demande de signature de certificat (CSR) :

```
openssl req -out server.csr -key server.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]:
Locality Name (eg, city) []:PARIS
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Newton
Organizational Unit Name (eg, section) []:CIEL
Common Name (e.g. server FQDN or YOUR name) []:172.27.0.2
Email Address []:ciel@newton.fr

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

FR

Rien ici

PARIS

Votre organisation

Votre section

IP du broker

Un mail bidon

Rien ici

- -out server.csr : Spécifie le fichier de sortie (server.csr) où la demande de signature de certificat générée sera sauvegardée. La CSR contient des informations d'identification du serveur (telles que le nom commun, l'organisation, le pays, etc.) ainsi que la clé publique du serveur.
- -key server.key : Indique le fichier contenant la clé privée RSA (server.key) qui sera utilisée pour générer la CSR. Cette clé privée ne sera pas incluse dans la CSR elle-même, mais la clé publique correspondante et les informations d'identification seront

4. Signer la Demande de Signature de Certificat (CSR) et générer un certificat SSL/TLS signé:

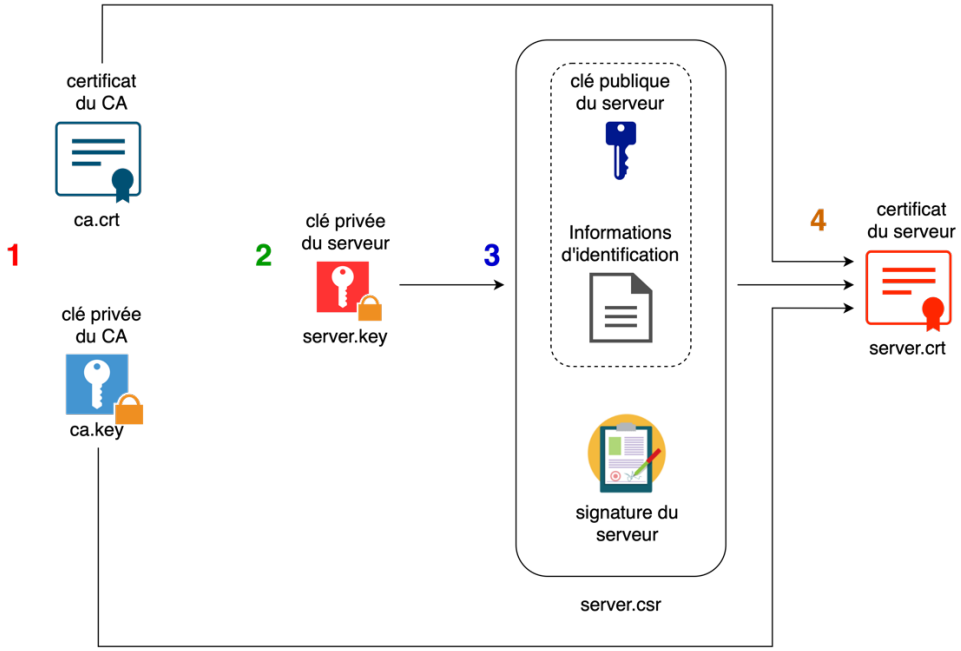
```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -
CAcreateserial -out server.crt -days 360
```

```
Signature ok
subject=C = FR, ST = Some-State, L = PARIS, O = Newton, OU = CIEL, CN = 172.27.0.2,
emailAddress = ciel@newton.fr
Getting CA Private Key
Enter pass phrase for ca.key:
```

Entrer ici le mot de passe défini lors de la création du certificat CA

- in server.csr : Spécifie le fichier d'entrée contenant la CSR. server.csr est le fichier qui contient la demande de certificat générée par ou pour le serveur, qui souhaite obtenir un certificat signé.
- CA ca.crt : Indique le certificat de l'Autorité de Certification (CA) utilisé pour signer la CSR. ca.crt contient le certificat public de la CA.
- CAkey ca.key : Spécifie la clé privée de l'Autorité de Certification (ca.key) qui correspond au certificat public spécifié par -CA. Cette clé privée est utilisée pour signer effectivement la CSR et générer le certificat signé.
- out server.crt : Définit le nom du fichier de sortie pour le certificat signé. Dans cet exemple, le certificat signé est sauvegardé dans server.crt.

EN RÉSUMÉ :



1 : `openssl req -new -x509 -days 1826 -extensions v3_ca -keyout ca.key -out ca.crt`

2 : `openssl genrsa -out server.key 2048`

3 : `openssl req -out server.csr -key server.key -new`

4 : `openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 360`

CONFIGURATION DU BROKER :

1. Modifier le fichier de configuration mosquitto.conf :

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

2. Ajouter les lignes suivantes en fin de fichier :

```
listener 8883
cafile /home/vagrant/ca.crt
certfile /home/vagrant/server.crt
keyfile /home/vagrant/server.key
```

3. Relancer le service mosquitto.

```
sudo systemctl restart mosquitto
```

4. Vérifier l'état running du service mosquitto

```
sudo systemctl status mosquitto
```

RECOPIE DU CERTIFICAT D'AUTORITÉ SUR LE CLIENT :

1. Editer le contenu du fichier ca.crt

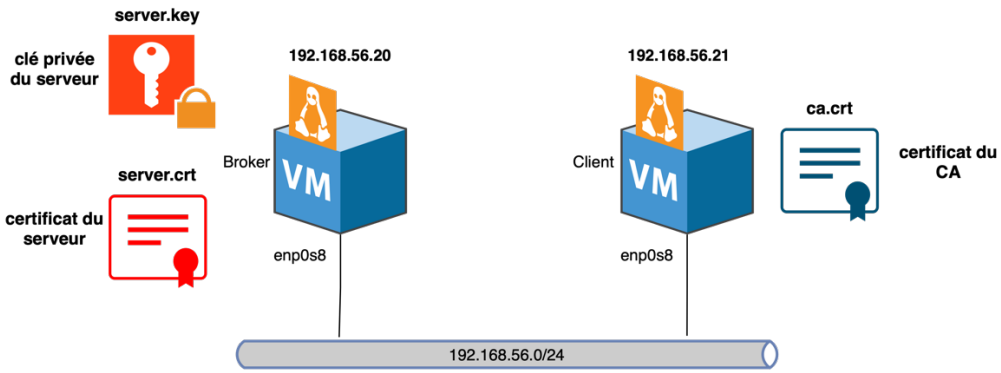
```
cat /home/vagrant/ca.crt
```

2. Copier son contenu dans le presse-papier et créer un fichier ca.crt sur le client

```
nano /home/vagrant/ca.crt
```

3. Coller le contenu du presse-papier dans fichier ca.crt édité précédemment sur le client.

TEST DU MQTTS:



1. S'abonner à un topic depuis la VM broker :

```
mosquitto_sub -h 192.168.56.20 -p 8883 --cafile /ca.crt -t
your/topic
```

2. Publier un message depuis la VM client :

```
mosquitto_pub -h 192.168.56.20 -p 8883 --cafile /ca.crt -t
your/topic -m "Hello world"
```

AUTHENTIFICATION DU CLIENT PAR MOT DE PASSE :

1. Modifier le fichier de configuration mosquitto.conf sur la VM broker :

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

2. Ajouter les lignes suivantes en fin de fichier :

```
allow_anonymous false
password_file /mosquitto_passwd
```

3. Créer des utilisateurs clients :

```
sudo mosquitto_passwd -c /mosquitto_passwd userclient
```

```
Password:
Reenter password:
```

```
sudo mosquitto_passwd  /mosquitto_passwd userbroker
```

4. Relancer le service mosquitto

```
sudo systemctl restart mosquitto
```

5. Vérifier les utilisateurs/mot de passe créé :

Tout se trouve dans le fichier mosquitto_passwd :

```
cat /mosquitto_passwd
```

```
userclient:$6$OvNJCOKGrIdI0xFf$He6Y7CTq6VBSmH9QW2OgogaUHKsuzH9LJPuvUCQKuivqgK3OOw75F4sbY/
1d5G/wmp7rAwhG61PJpm5098LxOA==
userbroker:$6$FIcGeh9rw19Zr6o+$ajleHiJlGAjlTlD279mQKsygnBUb55SpegREVE9m8KiWOMdg/DsnO+nYUS
Og9ayLcvFUszBeAgjnFjx6BAwtRw==
userclient2:$6$gtuGskWMC0SermQ9$SWK/SXSr4+Thoh1PKttV5tHK7+vJn96m6E0JEMC25EFHuq4wdtlf5zrdR
uKtMzlQttpxnXdiK4lFMIc57PLF/Q==
```

On voit les utilisateurs en clair et les mots de passe hachés.

userclient:6OvNJCOKGrIdI0xFf$He6Y7CTq6VBSmH9QW2OgogaUHKsuzH9LJPuvUCQKuivqgK3OOw75F4sbY/1d5G/wmp7rAwhG61PJpm5098LxOA==

userclient : C'est le nom d'utilisateur. Dans ce cas, l'utilisateur s'appelle userclient.

6OvNJCOKGrIdI0xFf$He6Y7CTq6VBSmH9QW2OgogaUHKsuzH9LJPuvUCQKuivqgK3OOw75F4sbY/1d5G/wmp7rAwhG61PJpm5098LxOA== : Ceci est le mot de passe hashé pour userclient. Le format du hash suit la convention utilisée par mosquitto_passwd, où :

- 6 indique le type de l'algorithme de hachage utilisé. 6 représente SHA-512, un choix courant pour le hachage des mots de passe en raison de sa sécurité.

- Le texte entre le premier et le second $ (OvNJCOKGrIdI0xFf dans cet exemple) est le "salt", une chaîne aléatoire ajoutée au mot de passe avant le hachage pour assurer que les hashages de deux mots de passe identiques soient différents.
- Le reste (He6Y7CTq6VBSmH9QW2OgogaUHKsuzH9LJPuvUCQKuivqgK3OOw75F4sbY/1d5G/wmp7rAwhG61PJpm5098LxOA==) est le mot de passe réellement hashé avec le salt.

TEST DE L'AUTHENTIFICATION PAR MOT DE PASSE

1. S'abonner à un topic depuis le broker :

```
mosquitto_sub -h 192.168.56.20 -p 8883 --cafile /ca.crt -u userbroker -P xxxxxxxxxx -t your/topic
```

2. Publier un message depuis le client :

```
mosquitto_pub -h 192.168.56.20 -p 8883 --cafile /ca.crt -u userclient -P xxxxxxxxxx -t your/topic -m "Hello world"
```

CRÉATION DES CERTIFICATS CLIENT :

À la place ou en plus de l'authentification du client par mot de passe, on peut créer une authentification client par certificat. Le processus est similaire à la génération de certificat serveur. Le CA reste le même, il n'est donc pas nécessaire de le créer.

1. Créer une clé privée pour le client sur la VM client :

```
openssl genrsa -out client.key 2048
```

2. Créer une demande de signature de certificat (CSR) :

```
openssl req -out client.csr -key client.key -new
```

Pour la dernière étape on va avoir besoin de la clé privée du CA sur le conteneur client (on a déjà le certificat CA).

3. Editer le fichier ca.key (clé privée du CA) sur le broker:

```
cat /home/vagrant/ca.key
```

4. Copier le contenu du fichier dans le presse-papier.

5. Créer un fichier ca.key sur la VM client.

```
nano /home/vagrant/ca.key
```

6. Coller le contenu du presse-papier dans ce fichier.

Retour sur le conteneur client :

7. Signer la Demande de Signature de Certificat (CSR) et générer un certificat SSL/TLS signé :

```
openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -
CAcreateserial -out client.crt -days 360
```

CONFIGURATION DU BROKER POUR AUTHENTIFICATION CLIENT :

1. Modifier du fichier de configuration mosquitto.conf sur le broker :

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

2. Ajouter la ligne suivante en fin de fichier :

```
require_certificate true
```

3. Relancer le service mosquitto

```
sudo systemctl restart mosquitto
```

TEST DE L'AUTHENTIFICATION DU CLIENT PAR CERTFICAT ET MOT DE PASSE

1. S'abonner à un topic depuis le broker :

```
mosquitto_sub -h 192.168.56.20 -p 8883 --cafile /ca.crt --cert
/server.crt --key /server.key -u userbroker -P xxxxxxxx -t
"your/topic"
```

2. Publier un message depuis le client :

```
mosquitto_pub -h 192.168.56.20 -p 8883 --cafile /ca.crt --cert
/client.crt --key /client.key -u userclient -P xxxxxxxx -t
"your/topic" -m "Hello world"
```