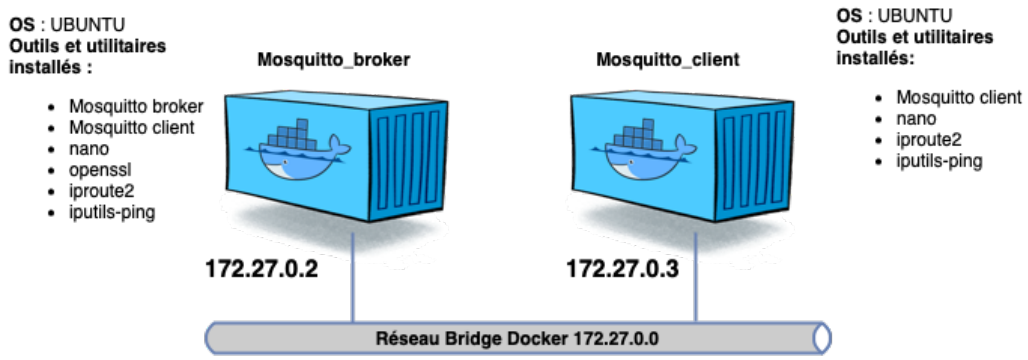
TP
SÉCURISATION D'UN CONNEXION MQTT

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

Objectifs intermédiaires :

- Créer des conteneurs docker en utilisant docker-compose.
- Créer un certificat SSL.
- 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 docker pour déployer deux machines. On aurait pu créer des machines virtuelles, mais docker est nettement plus rapide et facile à mettre en œuvre.

CRÉATION DES CONTENEURS :

1. Téléchargement du fichier docker-compose.yaml :

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

Docker Compose est un outil pour définir et gérer des applications multi-conteneurs Docker avec des configurations simples et déclaratives. Il permet aux utilisateurs de configurer les services, réseaux et volumes de leur application dans un fichier YAML unique, facilitant le déploiement et la mise à l'échelle cohérente de l'environnement complet avec une seule commande. Docker Compose simplifie ainsi le développement, les tests et la production en automatisant et en regroupant la gestion des conteneurs.

```
cat docker-compose.yaml
```

```
version: '3.8'

services:
  mosquitto_broker:
    image: ubuntu:latest
    container_name: mosquitto_broker
    command: >
      /bin/sh -c "apt-get update && apt-get install -y mosquitto mosquitto-clients
nano openssl iproute2 iputils-ping && mosquitto -c /etc/mosquitto/mosquitto.conf"
    ports:
      - "1883:1883"
      - "8883:8883"
    volumes:
      - ./broker-config:/etc/mosquitto
    networks:
      app_network:
        ipv4_address: 172.27.0.2
    restart: unless-stopped

  mosquitto_client:
    image: ubuntu:latest
    container_name: mosquitto_client
    command: >
      /bin/sh -c "apt-get update && apt-get install -y mosquitto-clients nano
iproute2 iputils-ping && tail -f /dev/null"
    networks:
      app_network:
        ipv4_address: 172.27.0.3
    restart: unless-stopped

networks:
  app_network:
    ipam:
      driver: default
      config:
        - subnet: 172.27.0.0/24
```

2. Instanciation des conteneurs et ressources associées définies dans le fichier docker-compose.yaml

```
docker-compose  up -d
```

```
Creating network "mqtt_app_network" with the default driver
Creating mosquitto_broker ... done
Creating mosquitto_client ... done
```

3. Vérification de le création des conteneurs

```
docker ps
```

```
CONTAINER ID   IMAGE           COMMAND                  CREATED         STATUS         PORTS
                     NAMES
d58b478862bd   ubuntu:latest   "/bin/sh -c 'apt-get…"   4 minutes ago   Up 4 minutes
                     mosquitto_client
e66e6bcc2c43   ubuntu:latest   "/bin/sh -c 'apt-get…"   4 minutes ago   Up 4 minutes   0.0.0.0:1883->1883/tcp,
 0.0.0.0:8883->8883/tcp   mosquitto_broker
```

4. Exécution d'un processus à l'intérieur du conteneur broker

```
docker exec -it mosquitto_broker /bin/bash
```

```
root@e66e6bcc2c43:/#
```

Nous sommes à présent à l'intérieur de notre conteneur.

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. Création 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.

```
openssl req -new -x509 -days 1826 -extensions v3_ca -keyout ca.key -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]:
Locality Name (eg, city) []:PARIS
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

On choisit un password que l'on mémorise

FR

Rien ici

PARIS

Rien ici

Rien ici

Rien ici

- 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. Création d'une 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. Création d'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. Signature de la Demande de Signature de Certificat (CSR) et génération d'un certificat SSL/TLS signé:

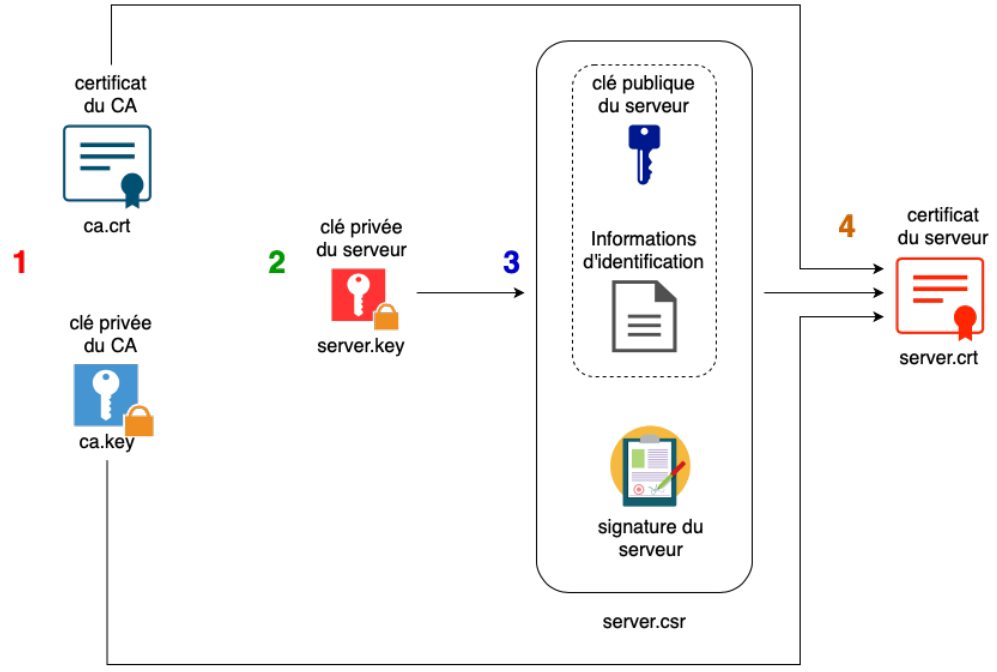
```
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -
CAcreateserial -out server.crt -days 360
```

```
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 :

5. Modification du fichier de configuration mosquitto.conf :

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

6. Ajout des lignes suivantes en fin de fichier :

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

7. Relancement du service mosquitto :

```
service mosquitto restart
```

RECOPIE DU CERTIFICAT D'AUTORITÉ SUR LE CLIENT :

1. Lancement d'une nouvelle fenêtre de terminal.

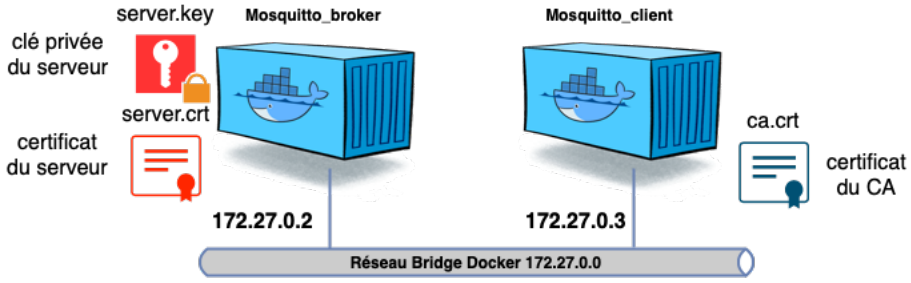
2. Copier du fichier ca.crt du conteneur mosquitto_broker vers le système hôte :

```
docker cp mosquitto_broker:/ca.crt .
```

3. Copie du fichier ca.crt du système hôte vers le conteneur mosquitto_client :

```
docker cp ca.crt mosquitto_client:/
```

TEST DU MQTTS:




```
mosquitto_sub -h 172.27.0.2 -p 8883 --cafile /ca.crt -t your/topic
```

1. Abonnement à un topic depuis le broker :

```
mosquitto_sub -h 172.27.0.2 -p 8883 --cafile /ca.crt -t your/topic
```

2. Connexion dans le conteneur du client depuis le système hôte :

```
docker exec -it mosquitto_client /bin/bash
```

3. Publication d'un message depuis le client :

```
mosquitto_pub -h 172.27.0.2 -p 8883 --cafile /ca.crt -t your/topic -
m "Hello world"
```

AUTHENTIFICATION DU CLIENT PAR MOT DE PASSE :

1. Modification du fichier de configuration mosquitto.conf :

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

2. Ajout des lignes suivantes en fin de fichier :

```
allow_anonymous false
password_file /mosquitto_passwd
```

3. Création des utilisateurs clients :

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

```
Password:
Reenter password:
```

```
mosquitto_passwd -c /mosquitto_passwd userbroker
```

4. Relancement du service mosquitto

```
service mosquitto restart
```

5. Vérification des utilisateurs/mot de passe :

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. Abonnement à un topic depuis le broker :

```
mosquitto_sub –h 172.27.0.2 –p 8883 ––cafile /ca.crt –u userbroker –P xxxxxxxxxx –t your/topic
```

2. Publication d'un message sur le client :

```
mosquitto_pub –h 172.27.0.2 –p 8883 ––cafile /ca.crt –u userclient2 –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. Installation de l'utilitaire openssl sur le conteneur mosquitto_client.

```
apt install openssl
```

2. Création d'une clé privée pour le client :

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

3. Création d'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 certficat CA).

4. Lancement d'une nouvelle fenêtre de terminale du système host.

5. Copie de la clé privée du CA du conteneur broker vers le système host :

```
docker cp mosquitto_broker:/ca.key .
```

6. Copie de la clé privée du CA du système host vers le conteneur client :

```
docker cp ca.key mosquitto_client:/
```

7. Signature de la Demande de Signature de Certificat (CSR) et génération d'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. Modification du fichier de configuration mosquitto.conf sur le broker :

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

2. Ajout de la ligne suivante en fin de fichier :

```
require_certificate true
```

3. Relancement du service mosquitto

```
service mosquitto restart
```

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

1. Abonnement à un topic depuis le broker :

```
mosquitto_sub –h 172.27.0.2 –p 8883 ––cafile /ca.crt ––cert
/server.crt ––key /server.key –u userbroker –P xxxxxxxx –t
"your/topic"
```

2. Publication d'un message sur le client :

```
mosquitto_pub –h 172.27.0.2 –p 8883 ––cafile /ca.crt ––cert
/client.crt ––key /client.key –u userclient2 –P xxxxxxxx –t
"your/topic" –m "Hello world"
```