



Compte Rendu : Système de Chat Sécurisé utilisant des Certificats OpenSSL

Réalisé par : El Abadi Mohamed

Année universitaire : 2023/2024

Table des matières

Introduction.....	3
1. Préparation des Certificats OpenSSL.....	3
1.1. Génération de la Clé Privée.....	3
1.2. Création d'un Fichier de Configuration OpenSSL.....	3
1.3. Génération de la Demande de Signature de Certificat (CSR).....	3
1.4. Génération du Certificat Auto-signé.....	4
2. Développement du Serveur.....	4
2.1. Code du Serveur.....	4
2.2. Explications.....	4
3. Développement du Client.....	5
3.1. Code du Client.....	5
3.2. Explications.....	5
4. Tests et Résultats.....	5
Conclusion.....	6
Références.....	6

```
elabadi@elabadi-virtual-machine:~/TP Digital certificate$ openssl req -new -key
server.key -out server.csr -config openssl.cnf
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) []:ma
State or Province Name (full name) []:rabat
Locality Name (eg, city) []:rabat
Organization Name (eg, company) []:ensias
Organizational Unit Name (eg, section) []:ensias
Common Name (eg, fully qualified host name) []:certificat
```

1.4. Génération du Certificat Auto-signé

Création du certificat auto-signé avec la CSR et la clé privée :

```
elabadi@elabadi-virtual-machine:~/TP Digital certificate$ openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt -extensions req_ext -extfile openssl.cnf
Certificate request self-signature ok
subject=C = ma, ST = rabat, L = rabat, O = ensias, OU = ensias, CN = certificat
```

2. Développement du Serveur

2.1. Code du Serveur

Le code du serveur initialise un socket sécurisé et écoute les connexions entrantes :

```
home > elabadi > TP Digital certificate > server.py
1  import socket
2  import ssl
3  import threading
4
5  def handle_client(connection, address):
6      print(f"Connexion de {address}")
7      try:
8          while True:
9              data = connection.recv(1024)
10             if not data:
11                 break
12             print(f"Reçu de {address}: {data.decode()}")
13             connection.sendall(data)
14         finally:
15             connection.shutdown(socket.SHUT_RDWR)
16             connection.close()
17
18  def main():
19      context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
20      context.load_cert_chain(certfile="server.crt", keyfile="server.key")
21
22      server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
23      server_socket.bind(('0.0.0.0', 12345))
24      server_socket.listen(5)
25
26      print("Serveur en écoute sur le port 12345")
27
28      try:
29          while True:
30              client_socket, addr = server_socket.accept()
31              connection = context.wrap_socket(client_socket, server_side=True)
32              threading.Thread(target=handle_client, args=(connection, addr)).start()
33      except Exception as e:
34          print(f"Erreur du serveur : {e}")
35      finally:
36          server_socket.close()
37
38  if __name__ == "__main__":
39      main()
40
```

2.2. Explications

- **Contexte SSL** : Chargement du certificat et de la clé privée.
- **Socket Serveur** : Écoute sur le port 12345 et accepte les connexions.
- **Gestion des Clients** : Chaque client est géré dans un thread séparé.

3. Développement du Client

3.1. Code du Client

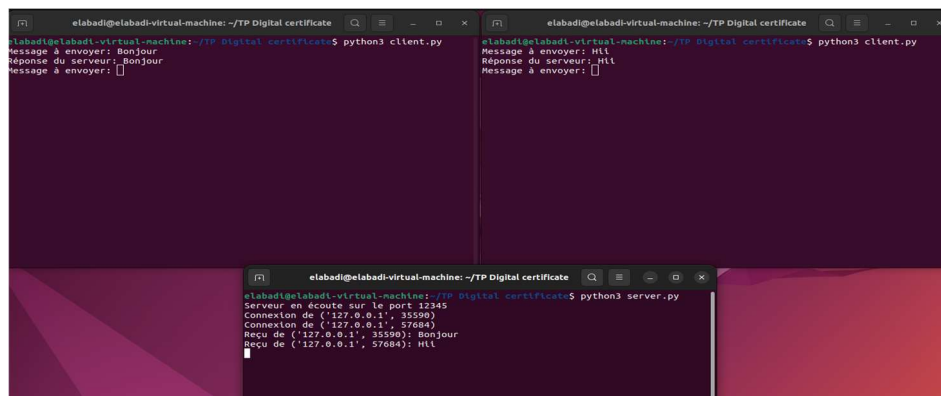
Le code du client initialise une connexion sécurisée au serveur :

```
home > elabadi > TP Digital certificate > client.py
1  import socket
2  import ssl
3
4  def main():
5      context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH)
6      context.load_verify_locations("server.crt")
7
8      raw_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9      connection = context.wrap_socket(raw_socket, server_hostname="localhost")
10
11     connection.connect(('localhost', 12345))
12
13     try:
14         while True:
15             message = input("Message à envoyer: ")
16             if message.lower() == 'quit':
17                 break
18             connection.sendall(message.encode())
19             response = connection.recv(1024)
20             print(f"Réponse du serveur: {response.decode()}")
21     except Exception as e:
22         print(f"Erreur du client : {e}")
23     finally:
24         connection.close()
25
26 if __name__ == "__main__":
27     main()
28
```

3.2. Explications

- **Contexte SSL** : Configuration pour vérifier le certificat du serveur.
- **Connexion** : Établissement de la connexion sécurisée au serveur.
- **Communication** : Envoi et réception de messages.

4. Tests et Résultats



```
elabadi@elabadi-virtual-machine: ~/TP Digital certificate
elabadi@elabadi-virtual-machine: ~/TP Digital certificate$ python3 client.py
Message à envoyer: Bonjour
Réponse du serveur: Bonjour
Message à envoyer:

elabadi@elabadi-virtual-machine: ~/TP Digital certificate
elabadi@elabadi-virtual-machine: ~/TP Digital certificate$ python3 client.py
Message à envoyer: Hll
Réponse du serveur: Hll
Message à envoyer:

elabadi@elabadi-virtual-machine: ~/TP Digital certificate
elabadi@elabadi-virtual-machine: ~/TP Digital certificate$ python3 server.py
Serveur en écoute sur le port 12345
Connexion de ('127.0.0.1', 35590)
Reçu de ('127.0.0.1', 35590): Bonjour
Reçu de ('127.0.0.1', 57684): Hll
```

Le système de chat fonctionne correctement avec des connexions sécurisées. Les certificats sont validés et les communications sont chiffrées, assurant la confidentialité et l'intégrité des données échangées.

Conclusion

Ce projet démontre la mise en place d'un système de chat sécurisé utilisant SSL/TLS avec des certificats OpenSSL. L'utilisation de certificats auto-signés permet de sécuriser les communications, bien que pour un déploiement en production, il serait préférable d'utiliser des certificats délivrés par une autorité de certification reconnue.

Références

- Documentation OpenSSL : <https://www.openssl.org/docs/>
- Documentation Python **ssl** : <https://docs.python.org/3/library/ssl.html>