Julien Voeung 4A Alternance Groupe 46

Rapport Projet de Cryptographie appliquée

Projet n°2 : PKI et Python

Le but de ce projet est de recréer le cheminement de signature des certificats en partant de la génération de paire de clé privé et publique, jusqu'à la signature du certificat requête par l'autorité d'enregistrement.

Ce projet se décompose en plusieurs étapes qui sont :

- Coder en python avec le package 'cryptography' [^2] un générateur de certificats (X509 avec clé RSA): en ayant la possibilité de choisir ses algorithmes de signature et de chiffrement
- Créer son autorité racine
- Créer son autorité d'enregistrement
- Signer les certificats générés par l'autorité d'enregistrement
- Parser un certificat et vérifier sa validité cryptographique et sa durée de vie

J'ai commencé par utiliser les fonctionnalités de génération de clé privée et par la suite la génération de sa clé publique suivant l'algorithme asymétrique RSA, disponible dans la librairie Cryptography sur Python.

```
20  #génération d'une clé publique et privée RSA
21  private_key = rsa.generate_private_key(
22  public_exponent=65537,
23  key_size=2048,)
24
25  public_key = private_key.public_key()
```

L'étape suivant consiste à exporter la clé privée sur notre disque locale pour la suite du projet. Pour cela des méthodes sont également disponible dans la librairie.

```
27  # Write our key to disk for safe keeping
28  with open("private_key_request.pem", "wb") as f:
29      f.write(private_key.private_bytes(
30          encoding=serialization.Encoding.PEM,
31          format=serialization.PrivateFormat.TraditionalOpenSSL,
32          encryption_algorithm=serialization.BestAvailableEncryption(b"passphrase"),
33      ))
```

Vient l'étape de la génération du certificat, ici l'utilisateur aura un choix à faire.

Nous donnons la possibilité à l'utilisateur une méthode de hash au choix parmi une liste proposée, lors de la méthode sign qui survient avec la clé privée générée.

```
print("Fonction de hashage : ")

print("1. SHA 256")
print("2. SHA3 256")
print("3. SHA1")
print("4. MD5")
print("5. BLAKE2b")
```

Selon le choix de l'utilisateur, on applique la méthode de hachage correspondant dans le code qui suit.

```
])
# Sign the CSR with our private key.
).sign(private_key, hashes.SHA256())
```

Méthode qui correspond à la fin de la génération du certificat.

```
# Generate a CSR
csr = x509.CertificateSigningRequestBuilder().subject_name(x509.Name([
    # Provide various details about who we are.
    x509.NameAttribute(NameOID.COUNTRY_NAME, u"FR"),
    x509.NameAttribute(NameOID.STATE_OR_PROVINCE_NAME, u"Ile de France"),
    x509.NameAttribute(NameOID.LOCALITY_NAME, u"Paris"),
    x509.NameAttribute(NameOID.ORGANIZATION_NAME, u"ESIEA"),
    x509.NameAttribute(NameOID.COMMON_NAME, u"RequestJulienEsiea"),
])
# Sign the CSR with our private key.
).sign(private_key, hashes.SHA256())
```

Enfin nous exportons simplement ce certificat sur notre disque local pour la suite du projet.

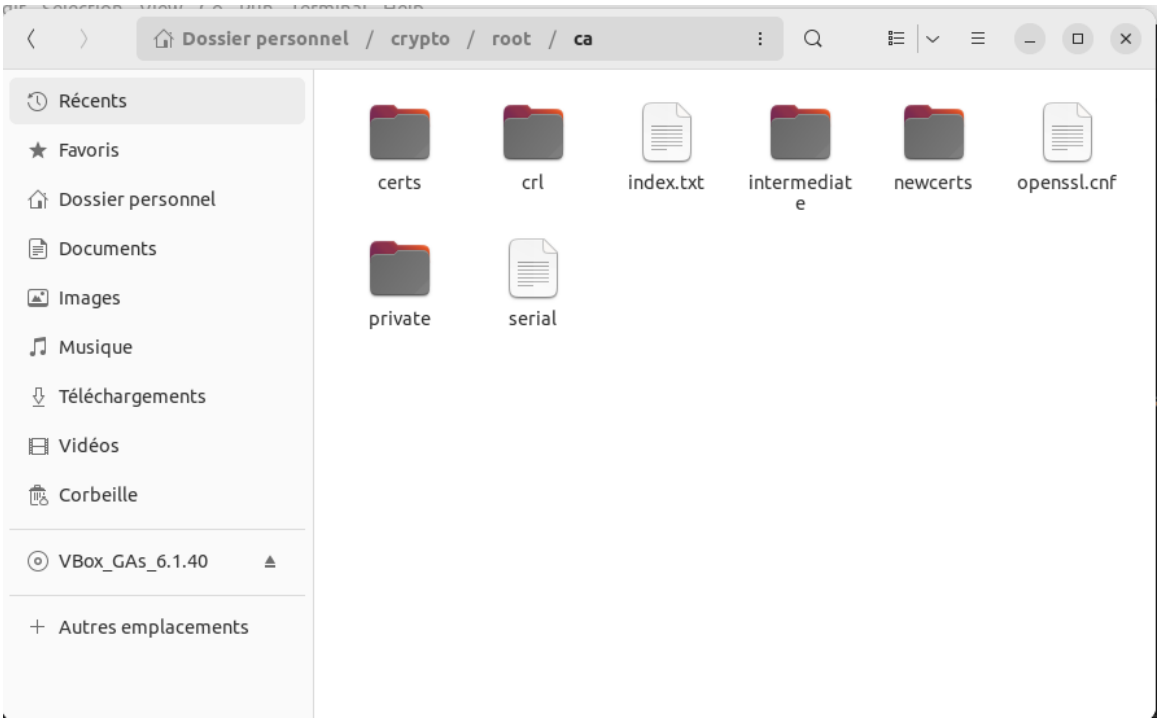
```
with open("request.pem", "wb") as f:
    f.write(csr.public_bytes(serialization.Encoding.PEM))
```

A cette étape du projet, nous avons terminé la partie en Python du projet qui concerne les éléments de notre request. Dans un cas pratique, cela représente un client qui possède une paire de clé et qui souhaite signer son certificat à une autorité de certification.

Pour ce faire, il sera signé par une autorité d'enregistrement qui est lui-même certifié par une autorité racine. Ce qui veut dire que le certificat de l'autorité d'enregistrement aura été signé par l'autorité racine.

Ainsi l'autorité d'enregistrement joue un rôle d'intermédiaire entre notre client muni de sa request et entre l'autorité racine, elle fera le rôle de signer la request et de ce fait sera reconnu par l'autorité racine.

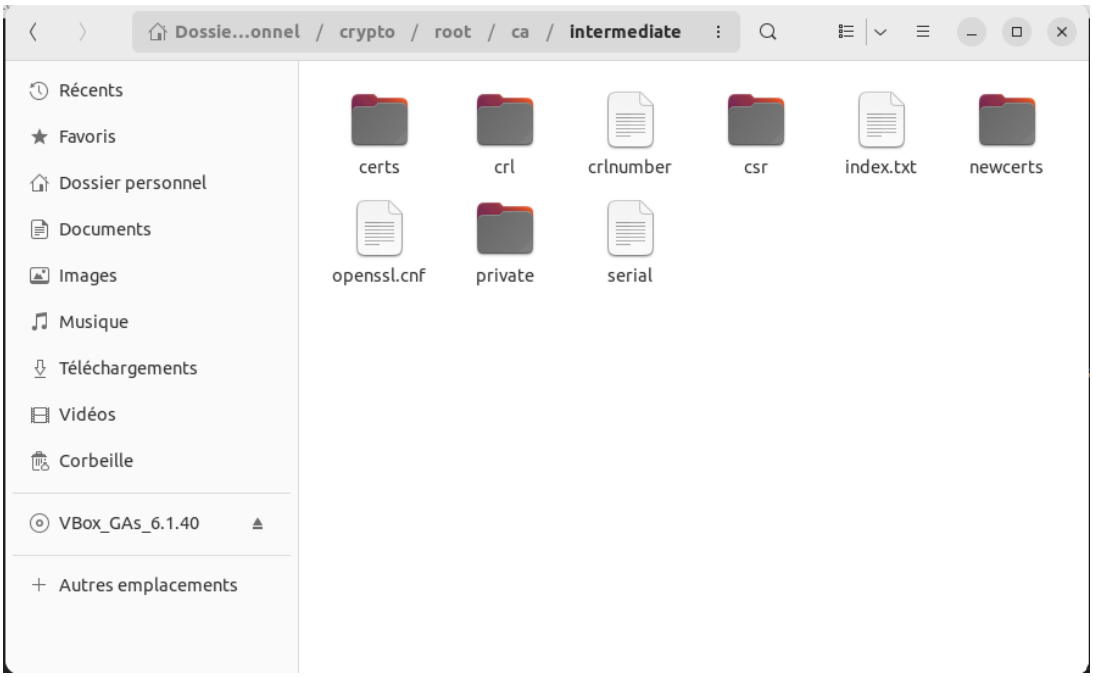
Pour la suite du projet, nous utiliserons OpenSSL. Nous devons créer une arborescence schématisant l'autorité root, l'intermédiaire et les request.



On trouvera dans le dossier private la clé privée de l'autorité root et on trouvera dans le dossier certs le certificat de l'autorité root.

On trouvera dans le dossier newcerts les request donc les certificats à signer.

On trouvera dans le dossier intermediate la même arborescence que dans ca, mais qui concerneront cette fois ci l'autorité d'enregistrement.



Les fichiers openssl.cnf sont des fichiers de configurations qui seront important notamment dans la signature des certificats, ces fichiers contiennent des informations qui nous permettent de stipuler les informations de validité pour la signature d'un certificat.

Nous commencerons par crée une clé privée pour l'autorité root avec OpenSSL.

```
openssl genrsa -aes256 -out private/ca.key.pem 4096
```

Puis on crée le certificat pour l'autorité racine.

```
openssl req -config openssl.cnf -key private/ca.key.pem -new -x509 \
  -days 7300 -sha256 -extensions v3_ca -out certs/ca.cert.pem
```

Maintenant nous passons à l'autorité d'enregistrement, on crée une clé privée.

```
julien@julien-VirtualBox:~/crypto/root/ca$ sudo openssl genrsa -aes256 -out intermediate/private/intermediateKey.pem 4096
```

On crée une request afin de la faire signer par notre autorité racine afin que notre autorité d'enregistrement soit bien reconnue et certifiée.

```
julien@julien-VirtualBox:~/crypto/root/ca$ sudo openssl req -config intermediate/openssl.cnf -new -sha256 -key intermediate/private/intermediateKey.pe
m -out intermediate/csr/intermediateCsr.pem
```

Puis on crée notre certificat en le signant par notre autorité racine.

```
julien@julien-VirtualBox:~/crypto/root/ca$ sudo openssl x509 -req -extfile intermediate/openssl.cnf -extensions v3_intermediate_ca -CA certs/ca.cert.p
em -CAkey private/ca.key.pem -CAcreaterserial -days 1825 -in intermediate/csr/intermediateCsr.pem -out intermediate/certs/intermediateCert.crt
Certificate request self-signature ok
```

Lorsqu'une application (par exemple, un navigateur Web) tente de vérifier un certificat signé par l'autorité de certification intermédiaire, elle doit également vérifier le certificat intermédiaire par rapport au certificat racine. Pour compléter la chaîne de confiance, créez une chaîne de certificats CA à présenter à l'application.

Pour créer la chaîne de certificats CA, concaténez les certificats intermédiaire et racine ensemble. Nous utiliserons ce fichier ultérieurement pour vérifier les certificats signés par l'autorité de certification intermédiaire.

```
sudo cat intermediate/certs/intermediateCert.crt certs/ca.cert.pem > intermediate/certs/root-chain-cert.crt
```