Université de Paris

Année universitaire : 2020-2021

Projet de cryptographie

Mécanismes de certification en Python

Implémentation avec pyOpenSSL

Projet réalisé par :
- Ludivine BONNET, ludivine.bonnet@etu.u-paris.fr
- Quentin GUARDIA, quentin.guardia@etu.u-paris.fr
Sous la direction de M. Salem.

Table des matières

Introduction

Internet offre la possibilité de mettre en relation des clients avec des serveurs. Mais comment s'assurer de l'authenticité de ces entités ? Et comment être sûr que les données restent confidentielles lors des échanges ? Tout simplement, par l'usage de certificats. Ainsi, le certificat assure deux rôles principaux : il sert à certifier l'identité d'un tiers et chiffrer les échanges avec celui-ci.

Le certificat doit être signé par une Autorité de Certification (CA), qui atteste l'identité de la personne, de la machine ou du service associé. Ainsi une entité peut demander à être certifiée par une CA via une requête de signature (CSR). L'autorité d'enregistrement vérifie préalablement l'identité du demandeur. Ainsi, une sécurité maximale est offerte.
De plus, les CSR et signatures de certificats par des CA nécessitent un jeu de clés publique/privée pour plus de sécurité, comme nous l'expliquerons par la suite.

Nous proposons dans ce projet d'automatiser la création de :

- CSR
- CA
- signature de certificat

Et en prime la vérification de la correspondance clé privée/certificat et de la signature du certificat. Il sera possible de choisir l'algorithme de chiffrement pour les clés (RSA ou DSA), ainsi que la longueur en bits de la paire de clés.

Nous implémenterons cette automatisation en Python sous Linux. Pour cela, nous nous appuyons sur la bibliothèque pyOpenSSL qui se base sur le standard X.509 pour la création de certificats, soit le standard le plus utilisé. Il est défini par le RFC 5280.

État de l'art sur les certificats

Comme évoqué précédemment, un certificat numérique est une carte d'identité électronique dont le but est d'authentifier un utilisateur, un équipement informatique ou un serveur d'application sécurisé. Plusieurs éléments sont détenus dans un certificat. D'abord, faisons un petit rappel sur les mécanismes de sécurité.

La paire de clés

Une paire de clés est associée à chaque certificat : une clé privée protégée par le propriétaire du certificat et une clé publique incluse dans le certificat. Ces deux clés sont générées en même temps et sont obtenues à l'aide d'un algorithme de chiffrement asymétrique tel que RSA, ou encore DSA. Il faut donc que ces clés soient suffisamment longues pour que la clé privée ne soit pas retrouvée grâce à la clé publique. On préconise une longueur de 2048 bits.

Pour illustrer, l’expéditeur peut chiffrer ses messages avec sa clé privée, et les destinataires peuvent lire en déchiffrant à l’aide de la clé publique. Inversement, les destinataires peuvent répondre en chiffrant un message à l’aide de la clé publique, que seule la personne initiale peut déchiffrer à l’aide de sa clé privée. Cela assure l’intégrité et la confidentialité des messages.

La signature

La signature consiste à appliquer une fonction de hachage aux données pour les transformer en une empreinte. Ensuite, l’empreinte est chiffrée avec la clé privée de l’expéditeur. L’empreinte se calcule avec une fonction de hachage, comme SHA-256. Cette signature assure l’authentification, la non-répudiation et l’intégrité des données.

Composition du certificat

Ainsi, le certificat numérique est composé de :

- la clé publique du tiers
- un Serial Number pour que la CA puisse identifier le certificat
- informations identifiant le porteur de cette paire de clés (qui peut être une personne ou un équipement), telles que son nom, son adresse IP, son adresse de messagerie électronique, etc
- l'identité de la CA
- durée de vie du certificat
- la signature numérique de toutes les données ci-dessus par la CA
- l’algorithme utilisé pour la signature

Processus de certification

Création d’une CA

Pour que certification il y ait, il doit avant tout y avoir une CA. La CA existe à travers son certificat. Dans notre programme, il s’agira d’un certificat auto-signé. Il sera son propre émetteur. Il faudra avant tout créer une paire de clés. La clé publique sera associée au certificat. La clé privée signera le certificat avec la fonction de hachage SHA256. Il faut conserver précieusement la clé privée, d’autant plus qu’elle servira à signer d’autres certificats. De nombreux éléments seront contenus dans le certificat de la CA, dont son sujet (par exemple, l’adresse du site web), sa durée de vie ou son Serial Number.

Génération de la CSR

La Certificate Signing Request est une requête qui donne les informations du certificat qu'elle souhaite faire signer auprès de la CA. Ici aussi, il faut d'abord créer une paire de clés. Ça sera la paire de clés associée au futur certificat. La clé publique est à rattacher à la CSR, et la clé privée pour signer la CSR, toujours avec SHA256. On doit remplir de nombreuses informations comme le nom de l'organisme, son pays ou encore l'adresse mail du contact. Ici aussi, les clés sont à conserver précieusement. On envoie la requête auprès de la CA, pour qu'elle signe un certificat avec les informations apportées.

Création du certificat signé par la CA

La CA crée un nouveau certificat à partir de la CSR. Pour cela, elle remplit les champs avec les informations contenues dans la CSR. La CA rajoute au certificat :

- la clé publique de la CA
- l'identité de la CA pour l'émetteur
- une durée de vie
- le Serial Number, qui est aléatoire dans notre programme

Le certificat est signé avec la clé privée de la CA et toujours l'algorithme de SHA256. Le certificat est prêt !

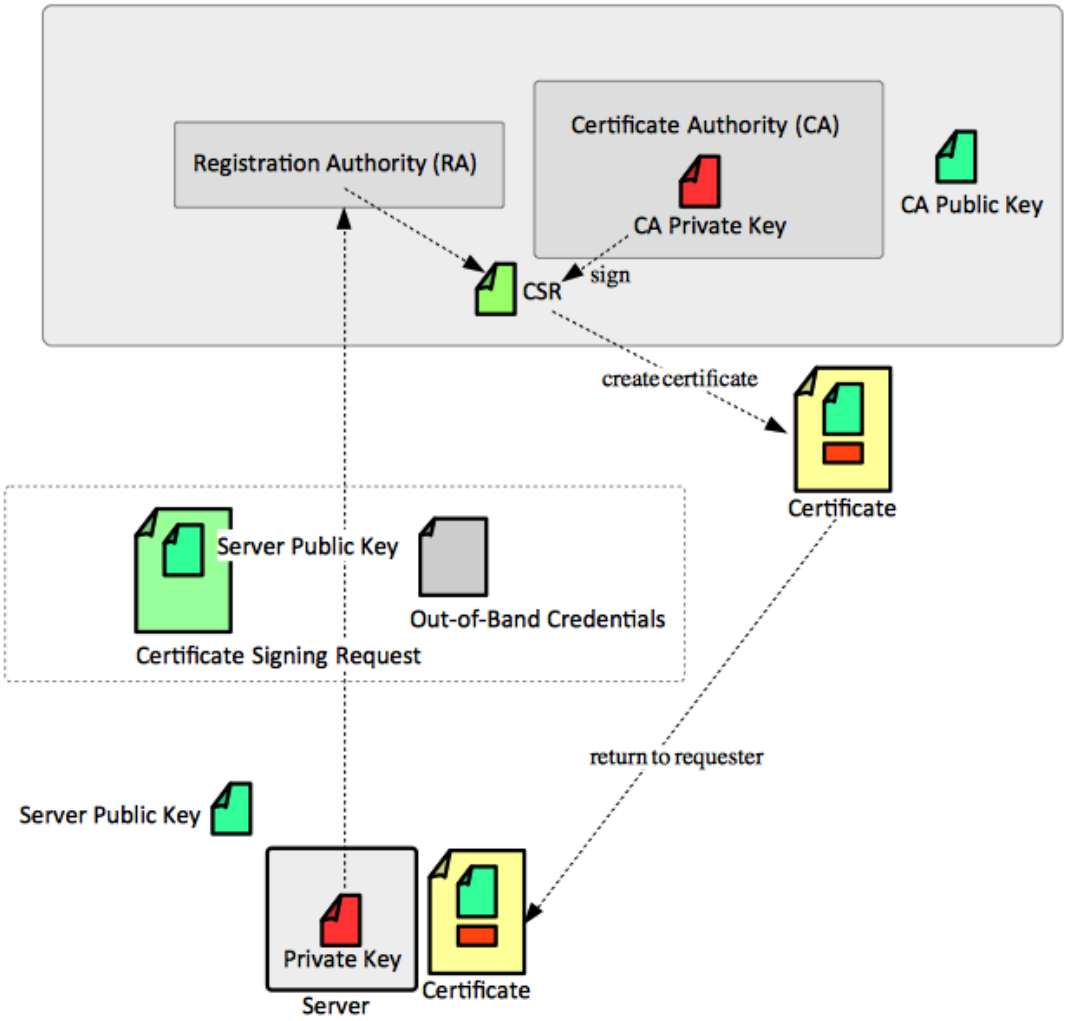
Extensions utilisées

Enfin, voici une dernière précision concernant les extensions des fichiers que nous utiliserons. Les clés privées et publiques seront enregistrées dans des fichiers .pem. Les CSR dans des fichiers .csr. Et les certificats dans des fichiers .crt.

Vérification

Nous avons en plus ajouter la possibilité de vérifier, d'une part la correspondance entre la clé privée et le certificat, d'autre part la validité de la signature de la CA.

Schéma récapitulatif



CSR passant par la RA avant de créer un certificat signé par la CA

Implémentation en python

Shebang

La première ligne permet au système d'exécuter le fichier comme un programme en lui indiquant l'interpréteur à utiliser, ici python3.

```
#!/usr/bin/env python3
```

Bibliothèques utilisées

Pour vérifier l'existence des fichiers à ouvrir ou vérifier qu'un fichier sur le point d'être enregistrer n'a pas un homonyme :

```
from pathlib import Path
import os.path
```

Pour le Serial Number aléatoire lors de la création d'un certificat par la CA :

```
import random as rd
```

Pour créer le parseur d'arguments

```
import argparse
```

Pour installer les bibliothèques manquantes, on fait un appel système si l'on rencontre une erreur lors de l'import. Pour la bibliothèque pyOpenSSL, on donne aussi en cas d'erreur la commande à exécuter pour éventuellement installer les paquets manquants. Pour PyCryptodome, importée sous le nom de Crypto.PublicKey, on vérifie de plus que la version soit supérieure ou égale à 3, pour éviter d'utiliser la version obsolète PyCrypto. PyCryptodrome nous servira à chiffrer les clés privées avec une passphrase, car bien que pyOpenSSL propose cette option, elle ne marche pas en pratique.

On s'est par ailleurs assurés de la présence de pip en installant PyOpenSSL, car c'est grâce à ce module que nous installons les bibliothèques non-standards manquantes.

```
import subprocess
import sys
import os
try:
        from OpenSSL import crypto, SSL
except ImportError:
        try:
                subprocess.check_call([sys.executable, "-m", "pip", "install",
"pyOpenSSL"])
        except:
                subprocess.check_call(["sudo","add-apt-repository",
"universe"])
                subprocess.check_call(["sudo","apt", "update"])
                subprocess.check_call(["sudo","apt", "install", "python3-pip"])
        subprocess.check_call([sys.executable, "-m", "pip", "install",
"pyOpenSSL"])
```

```
        print("Veuillez éxecuter le programme à nouveau")
        exit(0)

try:
        import Crypto.PublicKey
        from Crypto.PublicKey import RSA,DSA
        if int(Crypto.__version__[0])<3:
                subprocess.check_call([sys.executable, "-m", "pip", "install",
"-U", "pycryptodome"])
except ImportError:
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-U",
"pycryptodome"])
        from Crypto.PublicKey import RSA,DSA
```

Fonctions secondaires

entree_non_vide

C'est une fonction qui permet d'assurer qu'une entrée ne soit pas vide. Une entrée vide peut provoquer une erreur au moment de remplir les champs d'informations dans la CSR ou dans le sujet du CA. Pour cela, on fait itérer une boucle qui s'arrête lorsque la variable entree n'est plus nulle.

```
def entree_non_vide(texte):
        entree=""
        while entree=="":
                entree=input(texte)
        return entree
```

entree_entier

De la même manière, on vérifie que certaines entrées soient bien des chiffres, comme le champ concernant le nombre de domaines alternatifs dans la CSR ou encore la taille en bits de la clé. On a implémenté cela grâce à une structure try/except et une valeur par défaut, avec un or suivi d'une valeur après l'input.

```
def entree_entier(texte,defaut):
        entree=input(texte) or defaut
        try:
                return int(entree)
        except:
                print("Nombre incorrect")
                return entree_entier(texte,defaut)
```

enregistrer_fichier

Pour enregistrer un fichier, on appelle la fonction enregistrer_fichier qui prend en paramètres l'extension du fichier : pem, csr ou crt avec l'objet associé : key, certificate_request ou certificate. On demande un nom pour le fichier et on vérifie que le fichier n'existe pas déjà avant de l'enregistrer, grâce à Path() et is_file(). De même, le nom du fichier ne peut pas être vide.

```
def enregistrer_fichier(ftype, objet):
        nom=entree_non_vide("Sous quel nom enregistrer le fichier ."+ftype+" ?\
n")
        if Path(nom+"."+ftype).is_file():
                print("Fichier existant, veuillez choisir un nouveau nom")
                return enregistrer_fichier(ftype, objet)
        sortie = open(nom+"."+ftype, "w")
        sortie.write(objet.decode("utf-8") )
        sortie.close()
```

enregistrer_cle

La fonction pour enregistrer la paire de clés fait appel à la fonction ci-dessus. Plus exactement, la fonction prend une paire de clés en paramètre. Elle emploie dump pour convertir chacune des clés en chaîne de caractères. Puis pour chaque clé sous le format str, publique et privée, elle appelle la fonction enregistrer_fichier avec l'extension pem. On propose à l'utilisateur de rentrer une passphrase pour enregistrer la clé privée. Cela assure une sécurité complémentaire.

```
def enregistrer_cle(cle,type_c):
        print("On enregistre la clé privée")
        passphrase=input("Passphrase (optionnel):")
        if passphrase != "":
                enregistrer_passphrase(cle,passphrase,type_c)
        else:
                cle_privee = crypto.dump_privatekey(crypto.FILETYPE_PEM, cle)
                enregistrer_fichier("pem", cle_privee)

        print("On enregistre la clé publique")
        cle_publique = crypto.dump_publickey(crypto.FILETYPE_PEM, cle)
        enregistrer_fichier("pem", cle_publique)
```

ouvrir_fichier

Pour ouvrir un fichier, on appelle ouvrir_fichier qui prend en paramètre un type de fichier: crt pour un certificat, csr pour une CSR et clé privée pour un fichier pem. La fonction demande à l'utilisateur de donner le chemin du fichier. Si le chemin existe, alors le programme lit le fichier et convertir la chaîne récupérée vers le bon type. En cas d'erreur, on précise que le fichier entré n'est pas au bon format puis on rappelle la fonction pour réitérer. À noter qu'ici, si on ouvre une clé privée chiffrée à l'aide d'une passphrase, alors la passphrase sera automatiquement demandée.

```
def ouvrir_fichier(type_fichier):
        chemin=input("Chemin pour le fichier "+type_fichier+" :")

        if not Path(chemin).is_file():
                print("Fichier non trouvé, veuillez réessayer")
                return ouvrir_fichier(type_fichier)
        fichier=open(chemin, 'rt').read()
        try:
                if str(type_fichier)=="csr":

objet=crypto.load_certificate_request(crypto.FILETYPE_PEM, fichier)
                elif str(type_fichier)=="crt":
                        objet=crypto.load_certificate(crypto.FILETYPE_PEM,
fichier)
                elif str(type_fichier)=="clé privée":
```

```
                        try:

objet=crypto.load_privatekey(crypto.FILETYPE_PEM, fichier)
                        except:
                                print("Passphrase invalide. On recommence.")
                                return ouvrir_fichier(type_fichier)
        except:
                print("Le fichier entré n'est pas du bon type, veuillez
vérifier")
                return ouvrir_fichier(type_fichier)

        return objet
```

paire_cle

On commence enfin avec le module crypto de pyOpenSSL, avec paire_cle, la fonction qui permet de générer une paire de clés. On utilise pour cela la méthode Pkey(). On demande à l'utilisateur de choisir entre RSA et DSA pour l'algorithme de chiffrement. Il doit aussi choisir la longueur en bits des clés. Si une valeur incohérente est entrée, alors l'utilisateur doit entrer à nouveau la valeur concernée grâce à la fonction entree_entier. À la fin, une paire de clés est retournée avec l'algorithme de chiffrement, qui sera pratique de connaître par la suite.

```
def paire_cle():
        print("CRÉATION DE LA PAIRE DE CLÉS")
        cle=crypto.PKey()
        type_c=input("Chiffrement RSA ou DSA ? [R/D]:")

        bits=entree_entier("Clé de combien de bits ? (2048 par défaut):",2048)

        if type_c in {"r","R"}:
                cle.generate_key(crypto.TYPE_RSA, int(bits))
        elif type_c in {"d","D"}:
                cle.generate_key(crypto.TYPE_DSA, int(bits))
        else:
                print("Choix de chiffrement non valide, veuillez réessayer")
                return paire_cle()

        return cle, type_c
```

enregistrer_passphrase

Enfin, voici la fonction pour enregistrer une clé privée avec une passphrase. PyOpenSSL a normalement implémenté cette fonction, mais on s'est rendu compte qu'elle ne fonctionne pas pour le chiffrement au moment de l'enregistrement, bien que l'ouverture d'une clé chiffrée avec une passphrase demande automatiquement la passphrase. On a donc utilisé la libraire PyCryptodome. Dans un premier temps, on enregistre la clé sans chiffrement dans une fichier temporaire. Puis on la réouvre avec PyCryptodome pour la chiffrer et la réenregistrer sous le nom demandé, avant de supprimer le fichier temporaire.

```
def enregistrer_passphrase(cle,passphrase,type_c):
        sortie = open("tmp.txt", "w")
        sortie.write(crypto.dump_privatekey(crypto.FILETYPE_PEM,
cle).decode("utf-8"))
        sortie.close()

        f = open('tmp.txt','r')
```

```
        if type_c in {"r","R"}:
                key = RSA.importKey(f.read())
        else:
                key = DSA.importKey(f.read())
        f.close()

        nom=entree_non_vide("Sous quel nom enregistrer la clé privée chiffrée
(.pem) ?\n")
        if Path(nom).is_file():
                print("Fichier existant, veuillez choisir un nouveau nom")
                return enregistrer_passphrase(cle,passphrase)
        f = open(nom+".pem",'w')
        f.write(key.exportKey('PEM',passphrase=passphrase).decode('ascii'))
        f.close()
        os.remove("tmp.txt")
```

Fonctions primaires

requete_csr (côté demandeur de certificat)

Voici la fonction requete_csr. Elle permet de créer, comme son nom l'indique, une requête CSR. Pour cela, une paire de clés est générée avec la fonction précédemment expliquée. On prend soin de récupérer l'algorithme de chiffrement, RSA ou DSA, pour faciliter l'utilisation de la clé par PyCryptodome. Une création de CSR est demandée avec la méthode crypto.X509Req(), qui respecte la norme X.509. On demande ensuite à l'utilisateur de rentrer toutes les informations sur l'identité du certificat désiré, y compris les noms des domaines alternatifs si nécessaire, ce qui est un peu plus long à implémenter. On attache la clé publique avec la méthode set_pubkey(cle) et on signe avec la CSR privée de la sorte : csr.sign(cle, "sha256"). Puis on enregistre la paire de clés et la CSR avec les fonctions précédemment vues.

```
def requete_csr():
        print("CRÉATION DE LA REQUÊTE CSR\n");
        cle,type_c=paire_cle()
        pays=""
        liste=list()

        csr = crypto.X509Req()
        csr.get_subject().CN = entree_non_vide("Nom du certificat: ")

        while len(pays)!=2 : pays=input("Pays (2 car.):");
        csr.get_subject().countryName = pays
        csr.get_subject().stateOrProvinceName = entree_non_vide("Région: ")
        csr.get_subject().localityName = entree_non_vide("Ville: ")
        csr.get_subject().organizationName = entree_non_vide("Organisation: ")
        csr.get_subject().organizationalUnitName = entree_non_vide("Unité: ")
        csr.get_subject().emailAddress = entree_non_vide("Adresse mail: ")

        nb_san=entree_entier("Nombre de noms de domaine alternatifs (0 par
défaut):",0)

        csr.get_subject().subjectAltName = str(nb_san)

        for i in range(0, int(csr.get_subject().subjectAltName)):
                liste.append('DNS:'+input("Nom DNS "+str(i+1)+' :'))

        if int(csr.get_subject().subjectAltName) > 0:
```

```
                csr.add_extensions([crypto.X509Extension(b'subjectAltName',
False,','.join(liste).encode())])

        csr.set_pubkey(cle)
        csr.sign(cle, "sha256")

        enregistrer_cle(cle,type_c)

        csr_sortie = crypto.dump_certificate_request(crypto.FILETYPE_PEM, csr)
        print("On enregistre la requête")
        enregistrer_fichier("csr",csr_sortie)
```

creation_ca (côté certificateur)

On crée le certificat d'une CA avec creation_ca. D'abord, on crée une paire de clés avec paire_cle(). Puis on génère un certificat avec la méthode X509(). On ajoute la version, un Serial Number de 1, l'identité du certificateur (par exemple, l'adresse du site web), la durée de validité. Puis on ajoute les informations du certificat-même pour l'émetteur. C'est la ligne ca.set_issuer(ca.get_subject()). On attache la clé publique ici aussi, on ajoute les extensions nécessaires à la CA et on signe avec la clé privée. Une fois de plus, on enregistre la paire de clés sur le disque puis le certificat du certificateur.

```
def creation_ca():
        cle,type_c=paire_cle()
        ca = crypto.X509()
        ca.set_version(3)
        ca.set_serial_number(1)
        sujet=entree_non_vide("Sujet de la CA (ex: localhost): ")
        ca.get_subject().CN = sujet
        ca.gmtime_adj_notBefore(0)
        ca.gmtime_adj_notAfter(365*24*60*60)
        ca.set_issuer(ca.get_subject())
        ca.set_pubkey(cle)
        ca.add_extensions([crypto.X509Extension(b"basicConstraints", True,
b"CA:TRUE, pathlen:0"),crypto.X509Extension(b"keyUsage", True, b"keyCertSign,
cRLSign"), crypto.X509Extension(b"subjectKeyIdentifier", False, b"hash",
subject=ca),])
        ca.sign(cle, "sha256")

        ca_sortie = crypto.dump_certificate(crypto.FILETYPE_PEM, ca)

        print("On enregistre la CA")
        enregistrer_fichier("crt",ca_sortie)

        enregistrer_cle(cle,type_c)
```

signature_cert (côté certificateur)

On peut maintenant générer un certificat signé à partir d'une CSR et de la CA. Pour cela, on ouvre les fichiers suivants : la CSR, le certificat de la CA et la clé privée de la CA. Ici le certificateur crée un nouveau certificat. Il met le certificat de la CA en émetteur (issuer), il met les données contenues dans la csr dans le nouveau certificat et ajoute la clé publique contenue dans la csr. Ainsi l'entité qui fait la requête aura d'office la bonne clé privée. Le certificateur ajoute par la suite un Serial Number aléatoire et la durée de validité du futur certificat. Il signe le certificat avec la clé privée de la CA, précédemment ouverte. On enregistre le nouveau certificat !

```
def signature_cert():
        csr=ouvrir_fichier("csr")

        print("Certificat de la CA:")
        ca = ouvrir_fichier("crt")
        print("Il faut la clé privé de la CA")
        ca_cle = ouvrir_fichier("clé privée")

        serialnb=rd.getrandbits(64)

        cert = crypto.X509()
        cert.set_issuer(ca.get_subject())
        cert.set_subject(csr.get_subject())
        cert.set_pubkey(csr.get_pubkey())

        cert.gmtime_adj_notBefore(0)
        cert.gmtime_adj_notAfter(365*24*60*60)

        cert.set_serial_number(serialnb)

        cert.sign(ca_cle, "sha256")

        crt_sortie = crypto.dump_certificate(crypto.FILETYPE_PEM, cert)

        print("On enregistre le nouveau certificat créé")
        enregistrer_fichier("crt",crt_sortie)
```

verif

Pour terminer, voici la fonction qui a permis de vérifier la validité de notre travail. Elle offre deux choix. Le premier est la vérification de la correspondance entre la clé privée et le certificat, grâce à la méthode check_privatekey(). Le second est vérification de la CA qui a validé le certificat, grâce à la méthode verify_certificate(). Donc, dans notre cas de certificat d'une CA, on peut vérifier qu'il soit bien auto-signé. Toutes ces spécificités sont offertes par le module Crypto de pyOpenSSL.

```
def verif():
        choix=input("Vérifier:\n1: La clé privée du certificat\n2: L'émetteur
du certificat\nVotre choix:")

        print("On ouvre le certificat à vérifier")
        client_cert=ouvrir_fichier("crt")
        if str(choix) =="1":
                ctx = SSL.Context(SSL.TLSv1_METHOD)
                ctx.use_certificate(client_cert)
                ctx.use_privatekey(ouvrir_fichier("clé privée"))

                try:
                        ctx.check_privatekey()
                except SSL.Error:
                        print("\nIl n'y a pas de correspondance entre la clé
privée et le certificat\n")
                else:
                        print("\nLa clé privée correspond au certificat\n")

        elif str(choix)=="2":
                print("Certificat de la CA:")
                serveur_cert=ouvrir_fichier("crt")

                try:
                        store = crypto.X509Store()
```

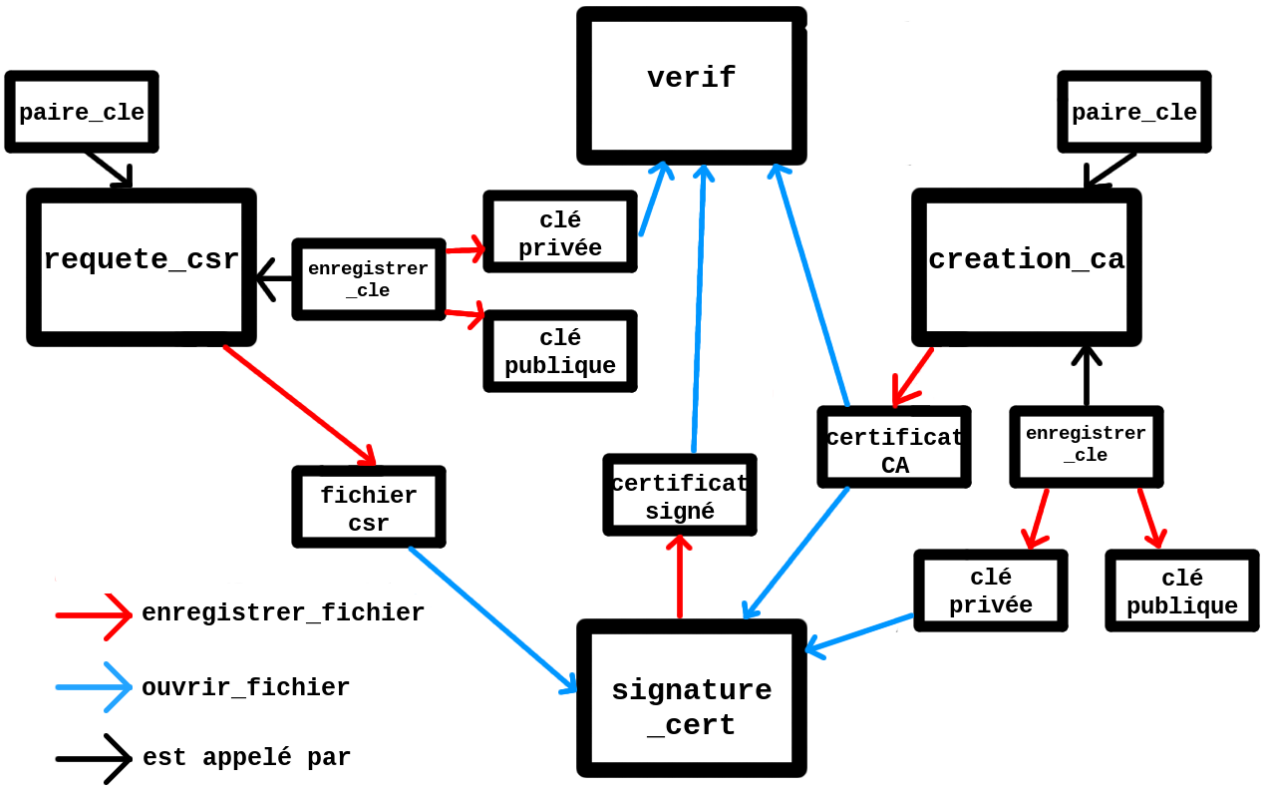
```
                        store.add_cert(serveur_cert)

                        store_ctx = crypto.X509StoreContext(store, client_cert)
                        store_ctx.verify_certificate()
                        print("\nCertificat bien signé par la CA\n")

                except Exception as e:
                        print("\nLes certificats ne correspondent pas\n")
                        print(e)
        else:
                print("Choix incorrect, veuillez recommencer")
                verif()
```

Schéma des fonctions

Juste après, un schéma récapitulant les relations entre les fonctions. Par soucis de clarté, on a volontairement amis les fonctions les moins utiles, telles que celles vérifiant que l'entrée soit correcte ou encore celle permettant de chiffrer la clé privée.



Mode d’emploi

Un bref manuel

D’abord, voyons ce que propose le programme si on l’exécute tel quel. Pour cela, on exécute dans un terminal, ouvert dans le dossier du programme code.py :

```
./code.py
```

```
quentin@quentin-hp:~/Bureau/M1 Descartes/Crypto/projet$ ./code.py
usage: code.py [-h] [--csr] [--ca] [--sign] [--verif]

Création automatique de CSR, CA, signature de certificat et vérification de la
signature et de la clé d'un certificat. Choix dans le chiffrement des clés.
Fonction de hashage: SHA256

optional arguments:
  -h, --help  show this help message and exit
  --csr       Génération d'une CSR
  --ca        Création d'une CA
  --sign      Création d'un certificat à partir de la CA
  --verif     Vérifications de la clé et signature d'un certificat
```

S’il y a bien droit d’exécution, on apprend les commandes possibles. Il existe d’abord l’argument --help, qui ne permet en réalité que d’afficher la présente sortie.

Du côté du demandeur :

- L’argument --csr sert à un utilisateur souhaitant faire une demande de certificat signé auprès d’une CA.

Du côté du certificateur :

- L’argument --ca sert à créer une CA.
- L’argument --sign sert à générer une certificat à partir d’une CSR et d’une CA.

L’argument --verif permet de tester la validité des sorties fichier.

Mettons cela en pratique !

Création de la CSR

On commence par créer une CSR. Il faut pour cela générer les clés et choisir leurs caractéristiques. Puis on complète les informations du certificat que l'on veut créer. Enfin, on enregistre. Pour cela, on exécute dans le terminal :

```
./code.py --csr
```

Puis on complète. Voici un exemple :

```
quentin@quentin-hp:~/Bureau/M1 Descartes/Crypto/projet$ ./code.py --csr
CRÉATION DE LA REQUÊTE CSR

CRÉATION DE LA PAIRE DE CLÉS
Chiffrement RSA ou DSA ? [R/D]:R
Clé de combien de bits ? (2048 par défaut):
Nom du certificat: premier_cert
Pays (2 car.):FR
Région: IDF
Ville: Paris
Organisation: Univ
Unité: Univ
Adresse mail: requeteur@mail.com
Nombre de noms de domaine alternatifs (0 par défaut):0
On enregistre la clé privée
Passphrase (optionnel):
Sous quel nom enregistrer le fichier .pem ?
cert_priv
On enregistre la clé publique
Sous quel nom enregistrer le fichier .pem ?
cert_pub
On enregistre la requête
Sous quel nom enregistrer le fichier .csr ?
requete
```

On peut vérifier l'existence des fichiers créés et leur cohérence avec cat :

```
quentin@quentin-hp:~/Bureau/M1 Descartes/Crypto/projet$ cat requete.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIC1TCCAb0CAQAwgY8xFTATBgNVBAMMDHByZW1pZXJfY2VydDELMAkGA1UEBhMC
RlIxDDAKBgNVBAgMA0lERjEOMAwGA1UEBwwFUGFyaXMxDTALBgNVBAoMBFVuaXYx
DTALBgNVBAsMBFVuaXYxITAfBgkqhkiG9w0BCQEWEnJlcXVldGV1ckBtYWlsLmNv
bTEKMAgGA1UdEQwBMDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAJ/n
rklW27o/EJk8ETHa1HzyXsoemnIMH5d07cW51CQjFCziqi6jgNZT9ycjgEgd+QCx
FzUavfyi9DCxjaz9t0Q/SaOgEYPXZ8xxXnOaUaEbFtnuZfTUEnNVbBSxrmXIm96f
X3xU9SIALjyTDEyyIcB6bGJ0GYuRn5I1V2vMpDPHKdoPFkcYwt4ogSbowdWD6rsz
dPRzDo5urZzBCGfycTUmSH1bcTZW6kzzKfMNgrhFEwv44HJFLg4t2Rx+vLelOivj
YFPZLmscpCg8UJJqiipq8hg9b+0uNcUlsLBzkhIe/AV6c5KVBIAyN9PJRQiDHV0Q
5Qs42R2tb91iEmWUcCUCAwEAAaAAMA0GCSqGSIb3DQEBCwUAA4IBAQB5KTnm5wmX
8UE2qDkTd0ye7jCY+WvcoBqincuv9NcPLdCTkaq0z4pmlsNshsxIw0rVJ3Z5jNIh
WXneAewkPwCI8u2X513/CC+s7POGQn+bsvZ9DSSttuJBBDIE2GRyNSrRcXq0kd5y
EFeyb0Ls5YM97QXp34Lo7mK6xfK4LUZU5OOXob35Vxs9S4/gupQYgxcA8aC6FeLa
auDb5bjHsZJl5OkbpX/7drbt8kqfFXkUfWwou6X7FNwy4fDLKtROYlitWEnhJTh4
hCLly54zmBtWsj9FZ1VEFtpMpCPve6iDq9Dapvl26DIF0CPwdu7fYmQtVKl03E8C
tu8ffmmJP22x
-----END CERTIFICATE REQUEST-----
```

On peut faire de même avec les fichiers cert_priv.pem et cert_pub.pem, qui sont également corrects d'après nos vérifications. On va maintenant essayer de faire valider cette CSR auprès d'un certificateur.

Création de la CA

Nous sommes à présent du côté du certificateur. On génère des clés avant de donner notre identificateur. Puis, on enregistre les clés et le nouveau certificat auto-signé :

```
./code.py --ca
```

```
quentin@quentin-hp:~/Bureau/M1 Descartes/Crypto/projet$ ./code.py --ca
CRÉATION DE LA PAIRE DE CLÉS
Chiffrement RSA ou DSA ? [R/D]:D
Clé de combien de bits ? (2048 par défaut):1024
Sujet de la CA (ex: localhost): www.monsite.fr
On enregistre la CA
Sous quel nom enregistrer le fichier .crt ?
ca
On enregistre la clé privée
Passphrase (optionnel):
Sous quel nom enregistrer le fichier .pem ?
ca_priv
On enregistre la clé publique
Sous quel nom enregistrer le fichier .pem ?
ca_pub
```

Signature d'un certificat par la CA

Le certificateur peut maintenant créer le certificat signé. Pour cela, il sélectionne le certificat de la CA, sa clé privée et le fichier de la CSR.

```
./code.py --sign
```

Pour poursuivre l'exemple, on obtient :

```
quentin@quentin-hp:~/Bureau/M1 Descartes/Crypto/projet$ ./code.py --sign
Chemin pour le fichier csr :requete.csr
Certificat de la CA:
Chemin pour le fichier crt :ca.crt
Il faut la clé privé de la CA
Chemin pour le fichier clé privée :ca_priv.pem
S'il y a, passphrase:
On enregistre le nouveau certificat créé
Sous quel nom enregistrer le fichier .crt ?
certif_valide
```

Vérification du certificat signé

On peut à présent vérifier la clé privée et la signature de notre nouveau certificat, en deux étapes. Pour cela, sélectionne d'abord le premier choix. On renseigne alors, le nouveau certificat et la clé privée créée au début. Puis on relance la commande, avec cette fois le second choix. Il faut pour celui-ci renseigner le nouveau certificat et enfin le certificat certificateur.

La commande correspondante est la suivante :

```
./code.py --verif
```

Puis il faut faire son choix : 1 ou 2.

Ce qui nous donne le résultat ci-dessous.

```
quentin@quentin-hp:~/Bureau/M1 Descartes/Crypto/projet$ ./code.py --verif
Vérifier:
1: La clé privée du certificat
2: L'émetteur du certificat
Votre choix:1
On ouvre le certificat à vérifier
Chemin pour le fichier crt :certif_valide.crt
Chemin pour le fichier clé privée :cert_priv.pem

La clé privée correspond au certificat

quentin@quentin-hp:~/Bureau/M1 Descartes/Crypto/projet$ ./code.py --verif
Vérifier:
1: La clé privée du certificat
2: L'émetteur du certificat
Votre choix:2
On ouvre le certificat à vérifier
Chemin pour le fichier crt :certif_valide.crt
Certificat de la CA:
Chemin pour le fichier crt :ca.crt

Certificat bien signé par la CA
```

Tout est valide !

Gestion de mauvaises entrées

Voici un exemple de gestion de mauvaises entrées par le programme. Ici, nous avons entré des lettres à la place d'un entier pour la longueur des clés de la CSR. Puis, nous avons sélectionner un nom de clé publique déjà existant, et un nom de clé privée vide. Le résultat est le suivant :

```
quentin@quentin-hp:~/Bureau/M1 Descartes/Crypto/projet$ ./code.py --csr
CRÉATION DE LA REQUÊTE CSR

CRÉATION DE LA PAIRE DE CLÉS
Chiffrement RSA ou DSA ? [R/D]:r
Clé de combien de bits ? (2048 par défaut):aaaaaaaaaaaaaaaaaa
Veuillez rentrer un entier
CRÉATION DE LA PAIRE DE CLÉS
Chiffrement RSA ou DSA ? [R/D]:R
Clé de combien de bits ? (2048 par défaut):
Nom du certificat: cert2
Pays (2 car.):FR
Région: IDF
Ville: Paris
Organisation: Univ
Unité: Univ
Adresse mail: qguardia66@gmail.com
Nombre de noms de domaine alternatifs (0 par défaut):0
On enregistre la clé privée
Passphrase (optionnel):
Sous quel nom enregistrer le fichier .pem ?
cert_pub
Fichier existant, veuillez choisir un nouveau nom
Sous quel nom enregistrer le fichier .pem ?

Sous quel nom enregistrer le fichier .pem ?
cert_priv2
On enregistre la requête
Sous quel nom enregistrer le fichier .csr ?
requete2
```

Conclusion

Nous avons pu voir le long de ce projet, comment il était possible d'automatiser la création de CSR pour un organisme demandeur, et de CA et certificats signés pour un organisme certificateur. Cela en python avec des bibliothèques pyOpenSSL et PyCryptodome. Il a pour cela fallu implémenta de nombreuses fonctionnalités, comme la génération d'une paire de clés, nécessaires pour la CSR et la CA. Nous avons bien réussi à créer un certificat à partir d'une CSR par le biais de la CA. Certificat valide comme nous avons pu le vérifier grâce à une fonction prévue à cet effet. Il faut seulement être sous Linux pour bénéficier de l'installation automatique, du paquet pip notamment.

En plus de cela, le programme vérifie que chaque entrée soit correcte pour éviter les erreurs. Si par exemple on rentre un nom de fichier déjà pris lors de l'enregistrement, alors on demande d'entrer une nouvelle valeur.

Par soucis de sécurité, sous souhaitions ajouter la possibilité de chiffrer les clés privées à l'aide d'une passphrase. Nous avons rencontré un problème en implémentant cela. En effet, le paramètre à ajouter au moment de chiffrer et déchiffrer la clé n'avait aucun effet. On a donc, grâce à la bibliothèque PyCryptodome, développé nous-même les fonctions de chiffrement et déchiffrement de clé privée par une passphrase.

Avec un peu plus de temps et de liberté, il aurait également été intéressant d'envoyer une CSR à un serveur distant à l'aide de la bibliothèque python Certsrv. On aurait également pu assurer la vérification des données contenues dans la CSR par une RA, puis faire en sorte de renvoyer le nouveau certificat signé vers le client qui a fait la requête.

Sources

Images

- Logo de l'Université de Paris : https://reacte.lem.univ-paris-diderot.fr/wp-content/uploads/2019/06/logo_UParis.png
- Schéma sur le processus de certification: https://kb.novaordis.com/images/c/c7/PublicKeyInfrastructure.png
- Schéma récapitulant le rôle des fonctions : œuvre personnelle
- Captures d'écran du mode d'emploi : œuvres personnelles

Références

- crypto - Generic cryptographic module, https://pypi.org/project/pyOpenSSL/
- Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, https://tools.ietf.org/html/rfc5280
- Internet X.509 Public Key Infrastructure Certificate and CRL Profile, https://tools.ietf.org/html/rfc2459
- openssl(1) – Linux manpage, https://linux.die.net/man/1/openssl
- Welcome to PyCryptodome's documentation, https://pycryptodome.readthedocs.io/en/latest/

Annexe

Aussi disponible sous « code.py » en annexe.

```
#!/usr/bin/env python3

from pathlib import Path
import os.path
import random as rd
import argparse
import subprocess
import sys
import os


#Import des bibliothèques non-standards
#OpenSSL et pip
try:
        from OpenSSL import crypto, SSL
except ImportError:
        try:
                subprocess.check_call([sys.executable, "-m", "pip", "install",
"pyOpenSSL"])
        except:
                subprocess.check_call(["sudo","add-apt-repository",
"universe"])
                subprocess.check_call(["sudo","apt", "update"])
                subprocess.check_call(["sudo","apt", "install", "python3-pip"])
        subprocess.check_call([sys.executable, "-m", "pip", "install",
"pyOpenSSL"])
        print("Veuillez éxecuter le programme à nouveau")
        exit(0)
```

```
#Cryptodome pour RSA et DSA
try:
        import Crypto.PublicKey
        from Crypto.PublicKey import RSA,DSA
        if int(Crypto.__version__[0])<3:
                subprocess.check_call([sys.executable, "-m", "pip", "install",
"-U", "pycryptodome"])
except ImportError:
        subprocess.check_call([sys.executable, "-m", "pip", "install", "-U",
"pycryptodome"])
        from Crypto.PublicKey import RSA,DSA



#Pour éviter des erreurs, on s'assure que certaines entrées soient non-vides
def entree_non_vide(texte):
        entree=""
        while entree=="":
                entree=input(texte)
        return entree


#On vérifie que les valeurs soient bien des entiers
def entree_entier(texte,defaut):
        entree=input(texte) or defaut
        try:
                return int(entree)
        except:
                print("Nombre incorrect")
                return entree_entier(texte,defaut)


#On enregistre une clé privée avec une passphrase
def enregistrer_passphrase(cle,passphrase,type_c):
        sortie = open("tmp.txt", "w")
        sortie.write(crypto.dump_privatekey(crypto.FILETYPE_PEM,
cle).decode("utf-8"))
        sortie.close()

        f = open('tmp.txt','r')
        if type_c in {"r","R"}:
                key = RSA.importKey(f.read())
        else:
                key = DSA.importKey(f.read())
        f.close()

        nom=entree_non_vide("Sous quel nom enregistrer la clé privée chiffrée
(.pem) ?\n")
        if Path(nom).is_file():
                print("Fichier existant, veuillez choisir un nouveau nom")
                return enregistrer_passphrase(cle,passphrase)

        f = open(nom+".pem",'w')
        f.write(key.exportKey('PEM',passphrase=passphrase).decode('ascii'))
        f.close()
        os.remove("tmp.txt")


#Enregistrer paire clés en appelant enregistrer_fichier()
def enregistrer_cle(cle,type_c):
        print("On enregistre la clé privée")
        passphrase=input("Passphrase (optionnel):")
        if passphrase != "":
```

```
                enregistrer_passphrase(cle,passphrase,type_c)
        else:
                cle_privee = crypto.dump_privatekey(crypto.FILETYPE_PEM, cle)
                enregistrer_fichier("pem", cle_privee)

        print("On enregistre la clé publique")
        cle_publique = crypto.dump_publickey(crypto.FILETYPE_PEM, cle)
        enregistrer_fichier("pem", cle_publique)


#Enregistrer un fichier sur le disque
def enregistrer_fichier(ftype, objet):
        nom=entree_non_vide("Sous quel nom enregistrer le fichier ."+ftype+" ?\
n")
        if Path(nom+"."+ftype).is_file():
                print("Fichier existant, veuillez choisir un nouveau nom")
                return enregistrer_fichier(ftype, objet)
        sortie = open(nom+"."+ftype, "w")
        sortie.write(objet.decode("utf-8") )
        sortie.close()


#Ouvrir un fichier
def ouvrir_fichier(type_fichier):

        chemin=input("Chemin pour le fichier "+type_fichier+" :")

        if not Path(chemin).is_file():
                print("Fichier non trouvé, veuillez réessayer")
                return ouvrir_fichier(type_fichier)

        fichier=open(chemin, 'rt').read()
        try:
                if str(type_fichier)=="csr":

objet=crypto.load_certificate_request(crypto.FILETYPE_PEM, fichier)
                elif str(type_fichier)=="crt":
                        objet=crypto.load_certificate(crypto.FILETYPE_PEM,
fichier)
                elif str(type_fichier)=="clé privée":
                        try:

objet=crypto.load_privatekey(crypto.FILETYPE_PEM, fichier)
                        except:
                                print("Passphrase invalide. On recommence.")
                                return ouvrir_fichier(type_fichier)
        except:
                print("Le fichier entré n'est pas du bon type, veuillez
vérifier")
                return ouvrir_fichier(type_fichier)

        return objet


#Générer une paire de clés
def paire_cle():
        print("CRÉATION DE LA PAIRE DE CLÉS")
        cle=crypto.PKey()
        type_c=input("Chiffrement RSA ou DSA ? [R/D]:")
        bits=entree_entier("Clé de combien de bits ? (2048 par défaut):",2048)

        if type_c in {"r","R"}:
                cle.generate_key(crypto.TYPE_RSA, int(bits))
```

```
        elif type_c in {"d","D"}:
                cle.generate_key(crypto.TYPE_DSA, int(bits))
        else:
                print("Choix de chiffrement non valide, veuillez réessayer")
                return paire_cle()

        return cle,type_c


#Signature d'une CSR pour créer un certificat
def signature_cert():
        csr=ouvrir_fichier("csr")

        print("Certificat de la CA:")
        ca = ouvrir_fichier("crt")
        print("Il faut la clé privé de la CA")
        ca_cle = ouvrir_fichier("clé privée")

        serialnb=rd.getrandbits(64)

        cert = crypto.X509()
        cert.set_issuer(ca.get_subject())
        cert.set_subject(csr.get_subject())
        cert.set_pubkey(csr.get_pubkey())

        cert.gmtime_adj_notBefore(0)
        cert.gmtime_adj_notAfter(365*24*60*60)

        cert.set_serial_number(serialnb)

        cert.sign(ca_cle, "sha256")

        crt_sortie = crypto.dump_certificate(crypto.FILETYPE_PEM, cert)

        print("On enregistre le nouveau certificat créé")
        enregistrer_fichier("crt",crt_sortie)


#Création de la CA
def creation_ca():
        cle,type_c=paire_cle()
        ca = crypto.X509()
        ca.set_version(3)
        ca.set_serial_number(1)
        sujet=entree_non_vide("Sujet de la CA (ex: localhost): ")
        ca.get_subject().CN = sujet
        ca.gmtime_adj_notBefore(0)
        ca.gmtime_adj_notAfter(365*24*60*60)
        ca.set_issuer(ca.get_subject())
        ca.set_pubkey(cle)
        ca.add_extensions([crypto.X509Extension(b"basicConstraints", True,
b"CA:TRUE, pathlen:0"),crypto.X509Extension(b"keyUsage", True, b"keyCertSign,
cRLSign"), crypto.X509Extension(b"subjectKeyIdentifier", False, b"hash",
subject=ca),])
        ca.sign(cle, "sha256")

        ca_sortie = crypto.dump_certificate(crypto.FILETYPE_PEM, ca)

        print("On enregistre la CA")
        enregistrer_fichier("crt",ca_sortie)

        enregistrer_cle(cle,type_c)
```

```
#Création d'une CSR
def requete_csr():
        print("CRÉATION DE LA REQUÊTE CSR\n");
        cle,type_c=paire_cle()
        pays=""
        liste=list()

        csr = crypto.X509Req()
        csr.get_subject().CN = entree_non_vide("Nom du certificat: ")

        while len(pays)!=2 : pays=input("Pays (2 car.):");
        csr.get_subject().countryName = pays
        csr.get_subject().stateOrProvinceName = entree_non_vide("Région: ")
        csr.get_subject().localityName = entree_non_vide("Ville: ")
        csr.get_subject().organizationName = entree_non_vide("Organisation: ")
        csr.get_subject().organizationalUnitName = entree_non_vide("Unité: ")
        csr.get_subject().emailAddress = entree_non_vide("Adresse mail: ")

        nb_san=entree_entier("Nombre de noms de domaine alternatifs (0 par
défaut):",0)

        csr.get_subject().subjectAltName = str(nb_san)

        for i in range(0, int(csr.get_subject().subjectAltName)):
                liste.append('DNS:'+input("Nom DNS "+str(i+1)+' :'))

        if int(csr.get_subject().subjectAltName) > 0:
                csr.add_extensions([crypto.X509Extension(b'subjectAltName',
False,','.join(liste).encode())])

        csr.set_pubkey(cle)
        csr.sign(cle, "sha256")

        enregistrer_cle(cle,type_c)

        csr_sortie = crypto.dump_certificate_request(crypto.FILETYPE_PEM, csr)
        print("On enregistre la requête")
        enregistrer_fichier("csr",csr_sortie)


#Vérification de la signature d'un certificat à partir de la CA + de
l'association clé privée/certificat
def verif():
        choix=input("Vérifier:\n1: La clé privée du certificat\n2: L'émetteur
du certificat\nVotre choix:")

        print("On ouvre le certificat à vérifier")
        client_cert=ouvrir_fichier("crt")
        if str(choix) =="1":
                ctx = SSL.Context(SSL.TLSv1_METHOD)
                ctx.use_certificate(client_cert)
                ctx.use_privatekey(ouvrir_fichier("clé privée"))

                try:
                        ctx.check_privatekey()
                except SSL.Error:
                        print("\nIl n'y a pas de correspondance entre la clé
privée et le certificat\n")
                else:
                        print("\nLa clé privée correspond au certificat\n")

        elif str(choix)=="2":
```

```
                print("Certificat de la CA:")
                serveur_cert=ouvrir_fichier("crt")

                try:
                        store = crypto.X509Store()
                        store.add_cert(serveur_cert)

                        store_ctx = crypto.X509StoreContext(store, client_cert)
                        store_ctx.verify_certificate()
                        print("\nCertificat bien signé par la CA\n")

                except Exception as e:
                        print("\nLes certificats ne correspondent pas\n")
                        print(e)
        else:
                print("Choix incorrect, veuillez recommencer")
                verif()


#Parseur d'arguments
parser = argparse.ArgumentParser(description='Création automatique de CSR, CA,
signature de certificat et vérification de la signature et de la clé d\'un
certificat. Choix dans le chiffrement des clés. Fonction de hashage: SHA256')
parser.add_argument('--csr', help='Génération d\'une CSR',action='store_true')
parser.add_argument('--ca', help='Création d\'une CA',action='store_true')
parser.add_argument('--sign', help='Création d\'un certificat à partir de la
CA',action='store_true')
parser.add_argument('--verif', help='Vérifications de la clé et signature d\'un
certificat',action='store_true')

args = parser.parse_args()
if args.csr:
        requete_csr()
elif args.ca:
        creation_ca()
elif args.sign:
        signature_cert()
elif args.verif:
        verif()
else:
        parser.print_help()
```