# TP5 - Signatures et Certicats

## 1 Signatures

### Question 1 :

Pour envoyer un message privé et la signature RSA associée à votre binôme, nous allons suivre les étapes suivantes :

1. **Générer la signature du message avec la clé privée.**
2. **Envoyer le message et la signature.**
3. **Vérifier la signature reçue avec la clé publique.**

#### Clés fournies

- Clé privée : d=23d = 23d=23
- Clé publique : (e,n)=(7,55)(e, n) = (7, 55)(e,n)=(7,55)

#### Exemple avec un message

Prenons un message simple, par exemple, le nombre m=42m = 42m=42.

#### Génération de la signature

La signature sss est générée en chiffrant le message mmm avec la clé privée ddd: s=mdmod ns = m^d \mod ns=mdmodn

#### Chiffrement du message pour l'envoyer

Le message mmm est chiffré avec la clé publique eee: c=memod nc = m^e \mod nc=memodn

#### Vérification de la signature

Pour vérifier la signature, le récepteur déchiffre la signature sss avec la clé publique eee et compare le résultat avec le message reçu mmm.

#### Implémentation en Python

**Fonction pour l'exponentiation modulaire**

```
def mod_exp(base, exp, mod):
    result = 1
    base = base % mod
    while exp > 0:
```

```
        if exp % 2 == 1:
            result = (result * base) % mod
        exp = exp >> 1
        base = (base * base) % mod
    return result
```

**Générer la signature et chiffrer le message**

```
def generate_signature(message, d, n):
    return mod_exp(message, d, n)

def encrypt_message(message, e, n):
    return mod_exp(message, e, n)
```

**Vérifier la signature**

```
def verify_signature(signature, message, e, n):
    decrypted_signature = mod_exp(signature, e, n)
    return decrypted_signature == message
```

## Exemple complet

**1. Générer la signature et chiffrer le message**

```
d = 23
e = 7
n = 55
message = 42

# Générer la signature
signature = generate_signature(message, d, n)
print(f"Signature: {signature}")

# Chiffrer le message
cipher_text = encrypt_message(message, e, n)
print(f"Chiffré: {cipher_text}")
```

**2. Vérifier la signature**

```
received_message = 42
received_signature = signature

# Vérifier la signature
is_valid = verify_signature(received_signature, received_message, e, n)
print(f"Signature valide: {is_valid}")
```

## Résultat attendu

L'exécution de ce code devrait produire :

Signature: 9
Chiffré: 3
Signature valide: True

## Explications

1. **Génération de la signature** : s=4223mod 55=9s = 42^{23} \mod 55 = 9s=4223mod55=9
2. **Chiffrement du message** : c=427mod 55=3c = 42^7 \mod 55 = 3c=427mod55=3
3. **Vérification de la signature** : 97mod 55=429^7 \mod 55 = 4297mod55=42

La signature est vérifiée correctement, ce qui montre que le message est authentique.

## En résumé

Ce protocole assure que le message et la signature sont envoyés et vérifiés correctement en utilisant le chiffrement RSA. Vous pouvez tester cette implémentation avec votre binôme pour envoyer et vérifier un message.

# Question 2 :

Pour générer un couple de clés (clé privée et clé publique) avec OpenSSL et utiliser ces clés pour signer un message, voici les étapes à suivre :

### Étapes

1. **Générer un couple de clés privé/public avec OpenSSL.**
2. **Générer une signature du message avec la clé privée.**
3. **Vérifier la signature avec la clé publique.**

### Génération des Clés

#### Générer une clé privée RSA

```sh
openssl genpkey -algorithm RSA -out private_key.pem
```

#### Générer la clé publique correspondante

```sh
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

```

### Génération de la Signature

#### Créer un fichier de message

Créez un fichier texte nommé `message.txt` et écrivez votre message dedans.

#### Générer la signature avec la clé privée

```sh
openssl dgst -sha256 -sign private_key.pem -out message.sig message.txt
```

### Vérification de la Signature

#### Vérifier la signature avec la clé publique

```sh
openssl dgst -sha256 -verify public_key.pem -signature message.sig message.txt
```

### Exemple Complet

#### Génération des Clés

1. **Générer une clé privée** :
   ```sh
   openssl genpkey -algorithm RSA -out private_key.pem
   ```

2. **Générer la clé publique** :
   ```sh
   openssl rsa -pubout -in private_key.pem -out public_key.pem
   ```

#### Génération de la Signature

1. **Créer un fichier de message** :
   Créez un fichier `message.txt` avec le contenu :
   ```
   This is a secret message.
   ```

2. **Générer la signature** :
   ```sh
   openssl dgst -sha256 -sign private_key.pem -out message.sig message.txt
   ```
   ```

#### Vérification de la Signature

1. **Vérifier la signature** :
   ```sh
   openssl dgst -sha256 -verify public_key.pem -signature message.sig message.txt
   ```

### Envoi et Vérification par le Binôme

1. **Envoyer le couple (message, signature)** :
   Envoyez les fichiers `message.txt` et `message.sig` à votre binôme, ainsi que la clé publique `public_key.pem`.

2. **Votre binôme vérifie l'authenticité** :
   Votre binôme peut utiliser la commande suivante pour vérifier l'authenticité :
   ```sh
   openssl dgst -sha256 -verify public_key.pem -signature message.sig message.txt
   ```

Si la signature est valide, OpenSSL affichera `Verified OK`, sinon il affichera une erreur indiquant que la vérification a échoué.

### Exemple d'Exécution

#### Génération des clés
```sh
openssl genpkey -algorithm RSA -out private_key.pem
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

#### Création du message
```sh
echo "This is a secret message." > message.txt
```

#### Signature du message
```sh
openssl dgst -sha256 -sign private_key.pem -out message.sig message.txt
```

#### Vérification de la signature
```sh
openssl dgst -sha256 -verify public_key.pem -signature message.sig message.txt
```

### Résultats
- Si la signature est correcte, vous verrez `Verified OK`.

- Si la signature est incorrecte, OpenSSL affichera une erreur de vérification.

En suivant ces étapes, vous pouvez utiliser OpenSSL pour générer des clés, signer un message, envoyer le couple (message, signature) à votre binôme et vérifier l'authenticité du message reçu.

# 2 Certicats

## Question 3 :

Pour créer un certificat au format X.509 auto-signé en utilisant OpenSSL, suivez les étapes ci-dessous :

### Étape 1 : Générer une paire de clés RSA

Générer une paire de clés RSA de 1024 bits et protéger la clé privée par un mot de passe.

openssl genpkey -algorithm RSA -out maCle.pem -aes256 -pass pass:motdepasse -pkeyopt rsa_keygen_bits:1024

### Étape 2 : Extraire la partie publique de la clé RSA

Créer un fichier ne contenant que la partie publique de votre clé RSA.

openssl rsa -in maCle.pem -pubout -out maClePublique.pem -passin pass:motdepasse

### Étape 3 : Créer un certificat auto-signé

Utiliser la commande `req` d'OpenSSL pour créer un certificat auto-signé au format X.509, valable 365 jours.

1. Générer une demande de signature de certificat (CSR) avec votre clé privée :

openssl req -new -key maCle.pem -out maCle.csr -passin pass:motdepasse

2. Créer le certificat auto-signé en utilisant la CSR :

openssl x509 -req -days 365 -in maCle.csr -signkey maCle.pem -out maCertificat.crt -passin pass:motdepasse

### Étape 4 : Visualiser les informations du certificat

Utiliser la commande `x509` d'OpenSSL pour visualiser en clair les informations du certificat créé.

openssl x509 -in maCertificat.crt -text -noout

## Exemple Complet

**Génération des clés RSA**
openssl genpkey -algorithm RSA -out maCle.pem -aes256 -pass pass:motdepasse -pkeyopt rsa_keygen_bits:1024

**Extraction de la clé publique**
openssl rsa -in maCle.pem -pubout -out maClePublique.pem -passin pass:motdepasse

**Création de la CSR**
openssl req -new -key maCle.pem -out maCle.csr -passin pass:motdepasse

**Création du certificat auto-signé**
openssl x509 -req -days 365 -in maCle.csr -signkey maCle.pem -out maCertificat.crt -passin pass:motdepasse

**Visualisation des informations du certificat**
openssl x509 -in maCertificat.crt -text -noout

## Résultats

L'exécution de ces commandes affichera des informations détaillées sur le certificat, y compris :

- Version du certificat
- Numéro de série
- Algorithme de signature
- Émetteur
- Validité (période de validité)
- Sujet (informations fournies lors de la création de la CSR)
- Clé publique
- Extensions du certificat (si présentes)

En suivant ces étapes, vous pouvez générer un certificat X.509 auto-signé en utilisant OpenSSL et examiner son contenu pour vérifier les informations qu'il contient.

## Question 4 :

Pour endosser le rôle d'une autorité de certification (CA) et répondre à une demande de certificat d'un utilisateur, suivez les étapes ci-dessous. Nous allons générer une requête de certificat (CSR) en tant qu'utilisateur, puis signer cette requête en tant que CA.

### Étapes pour l'Utilisateur

1. **Générer une paire de clés RSA**
2. **Créer une demande de signature de certificat (CSR)**
3. **Envoyer la CSR au binôme (CA)**

### Étapes pour le CA

1. **Générer un certificat auto-signé pour le CA**
2. **Signer la CSR de l'utilisateur pour générer un certificat signé**
3. **Envoyer le certificat signé à l'utilisateur**

### Étapes pour l'Utilisateur

#### 1. Générer une paire de clés RSA

openssl genpkey -algorithm RSA -out userKey.pem -aes256 -pass pass:userpassword -pkeyopt rsa_keygen_bits:2048

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

openssl req -new -key userKey.pem -out userRequest.csr -passin pass:userpassword

#### 3. Envoyer la CSR au binôme (CA)

Envoyez le fichier `userRequest.csr` à votre binôme.

### Étapes pour le CA

#### 1. Générer un certificat auto-signé pour le CA

openssl genpkey -algorithm RSA -out caKey.pem -aes256 -pass pass:capassword -pkeyopt rsa_keygen_bits:2048
openssl req -new -x509 -key caKey.pem -out caCert.pem -days 365 -passin pass:capassword

#### 2. Signer la CSR de l'utilisateur pour générer un certificat signé

Pour utiliser la commande `ca`, il est nécessaire de configurer quelques fichiers et répertoires pour OpenSSL. Pour simplifier, nous utiliserons une configuration minimale.

Créez un fichier de configuration minimal `openssl.cnf` :

```
mkdir -p demoCA/newcerts
touch demoCA/index.txt
echo 1000 > demoCA/serial

cat > openssl.cnf <<EOL
[ ca ]
default_ca = CA_default

[ CA_default ]
dir = ./demoCA
certs = \$dir/certs
new_certs_dir = \$dir/newcerts
database = \$dir/index.txt
serial = \$dir/serial
RANDFILE = \$dir/private/.rand
private_key = caKey.pem
certificate = caCert.pem
default_days = 365
default_md = sha256
policy = policy_any

[ policy_any ]
countryName = optional
stateOrProvinceName = optional
localityName = optional
organizationName = optional
organizationalUnitName = optional
commonName = supplied
emailAddress = optional
EOL
```

Signer la CSR :

```
openssl ca -config openssl.cnf -in userRequest.csr -out userCert.pem -keyfile caKey.pem
-cert caCert.pem -passin pass:capassword
```

## Étapes pour vérifier le certificat signé

### 1. Visualiser le certificat signé

```
openssl x509 -in userCert.pem -text -noout
```

### 2. Vérifier le certificat signé

```
openssl verify -CAfile caCert.pem userCert.pem
```