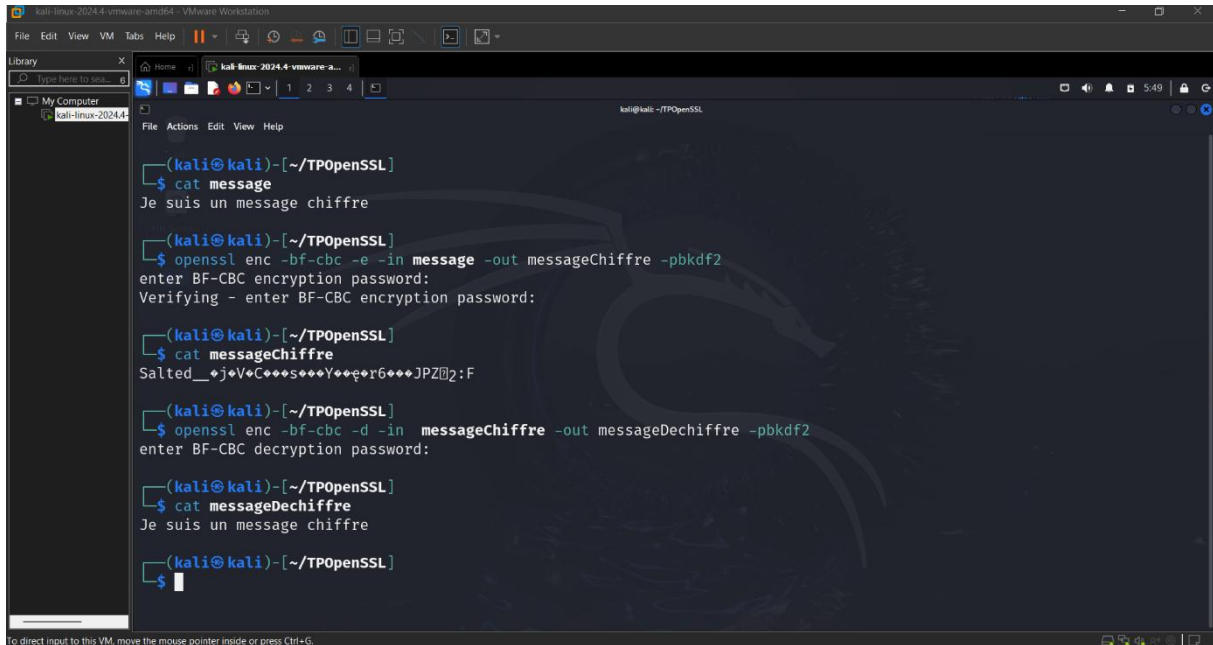


TRAITE DU TP D'INFOSEC

Question 1 : Créer un fichier nommé message. Puis chiffrer le avec le système Blowfish en mode CBC, avec une clé générée par mot de passe, le chiffré étant stocké dans le fichier messageChiffre.



```
(kali@kali)-[~/TPOpenSSL]
$ cat message
Je suis un message chiffré

(kali@kali)-[~/TPOpenSSL]
$ openssl enc -bf-cbc -e -in message -out messageChiffre -pbkdf2
enter BF-CBC encryption password:
Verifying - enter BF-CBC encryption password:

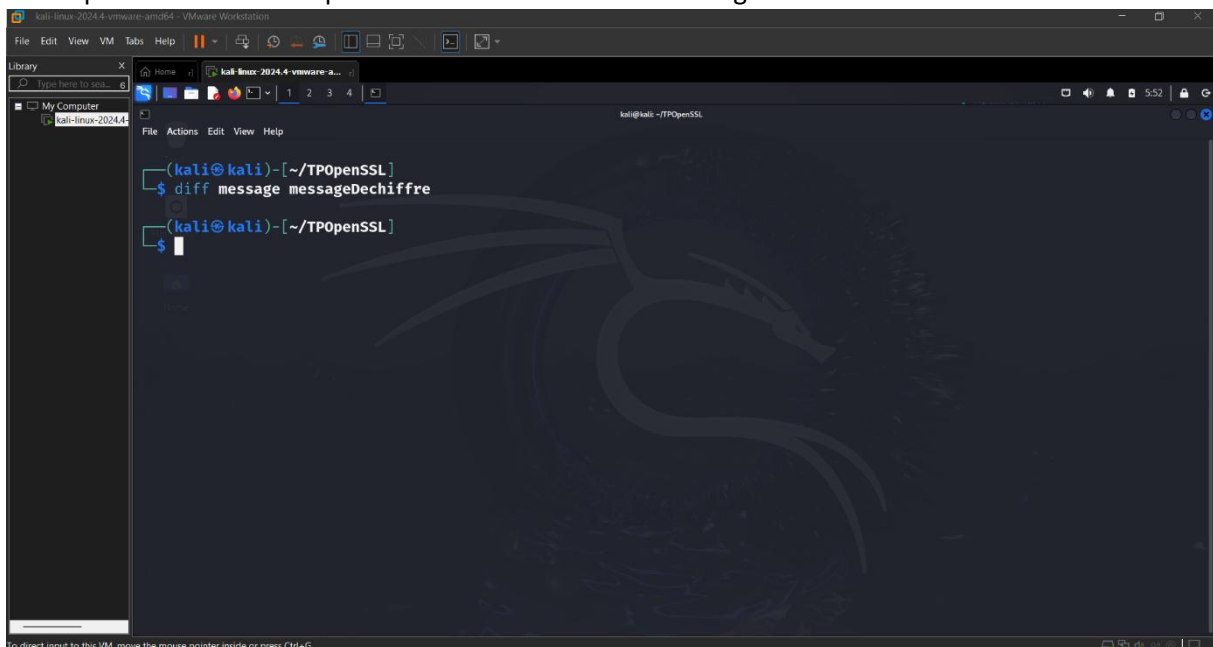
(kali@kali)-[~/TPOpenSSL]
$ cat messageChiffre
Salted__jV+C+++++Y++++r6+++JPZ02:F

(kali@kali)-[~/TPOpenSSL]
$ openssl enc -bf-cbc -d -in messageChiffre -out messageDechiffre -pbkdf2
enter BF-CBC decryption password:

(kali@kali)-[~/TPOpenSSL]
$ cat messageDechiffre
Je suis un message chiffré

(kali@kali)-[~/TPOpenSSL]
$
```

Question 2: Vérifier que le fichier initial (message) et celui déchiffré (messageDechiffre) sont identiques. Pour cela vous pouvez afficher leur contenu ou regarder leur différence

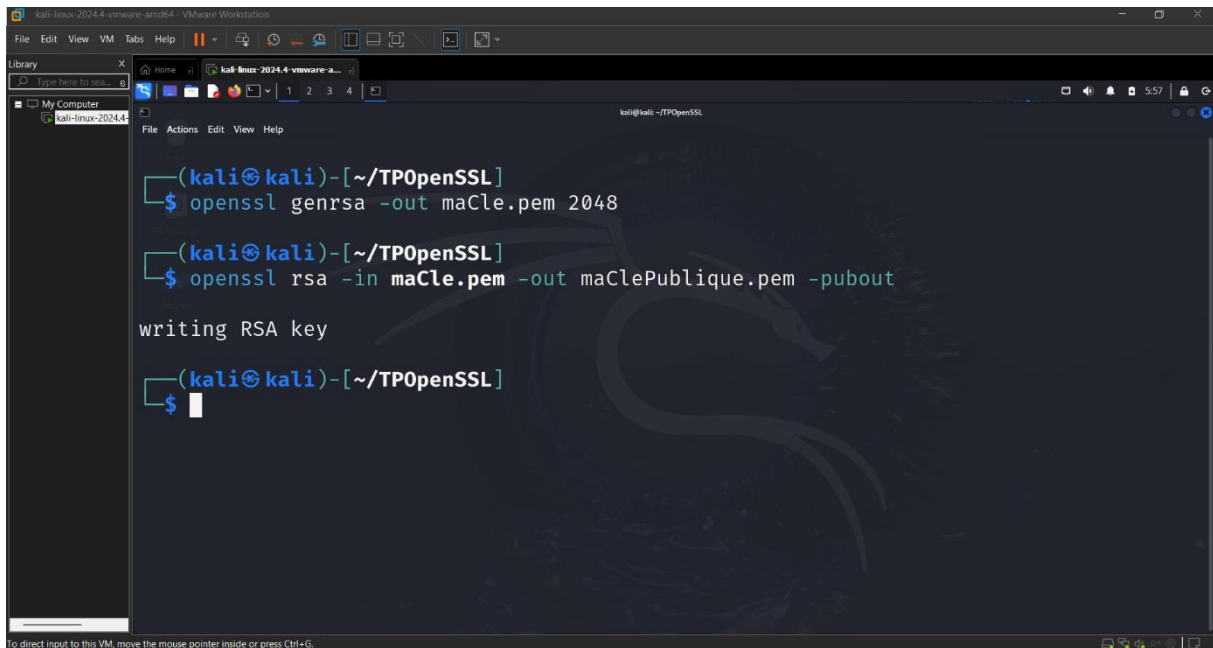


```
(kali@kali)-[~/TPOpenSSL]
$ diff message messageDechiffre

(kali@kali)-[~/TPOpenSSL]
$
```

4.1. Génération d'une paire de clés RSA On peut générer une paire de clés RSA avec la commande `genrsa` de `openssl`

4.2. Exploration de la partie publique



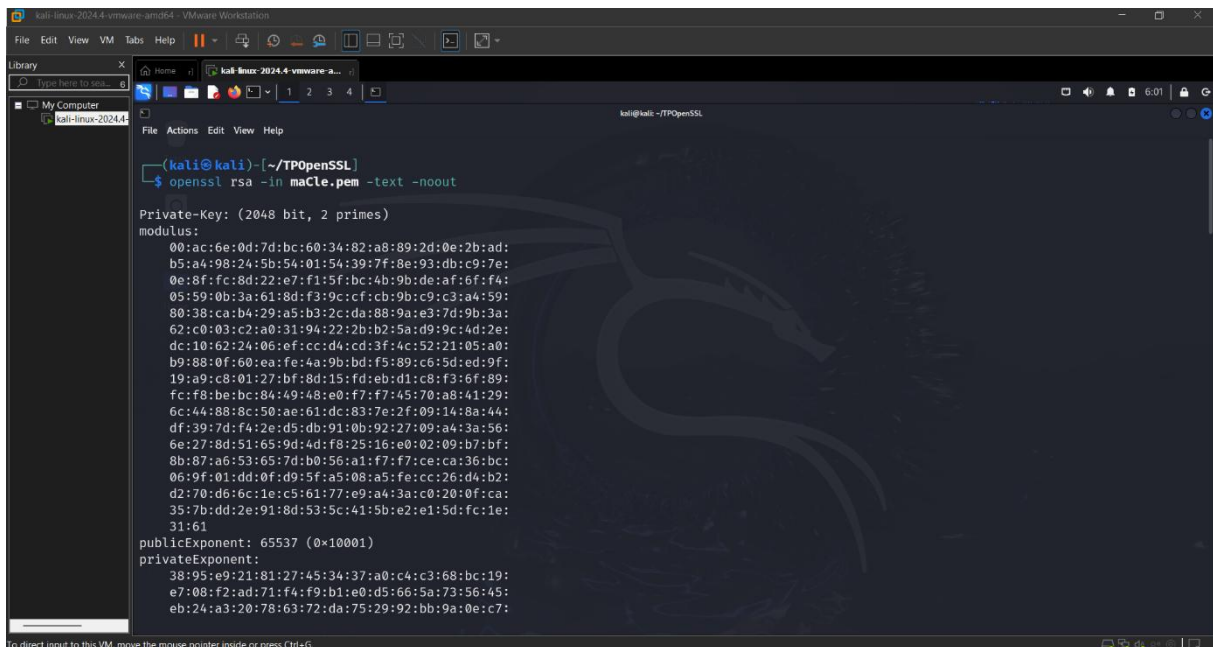
```
(kali@kali)-[~/TP0openssl]
$ openssl genrsa -out maCle.pem 2048

(kali@kali)-[~/TP0openssl]
$ openssl rsa -in maCle.pem -out maClePublique.pem -pubout

writing RSA key

(kali@kali)-[~/TP0openssl]
$
```

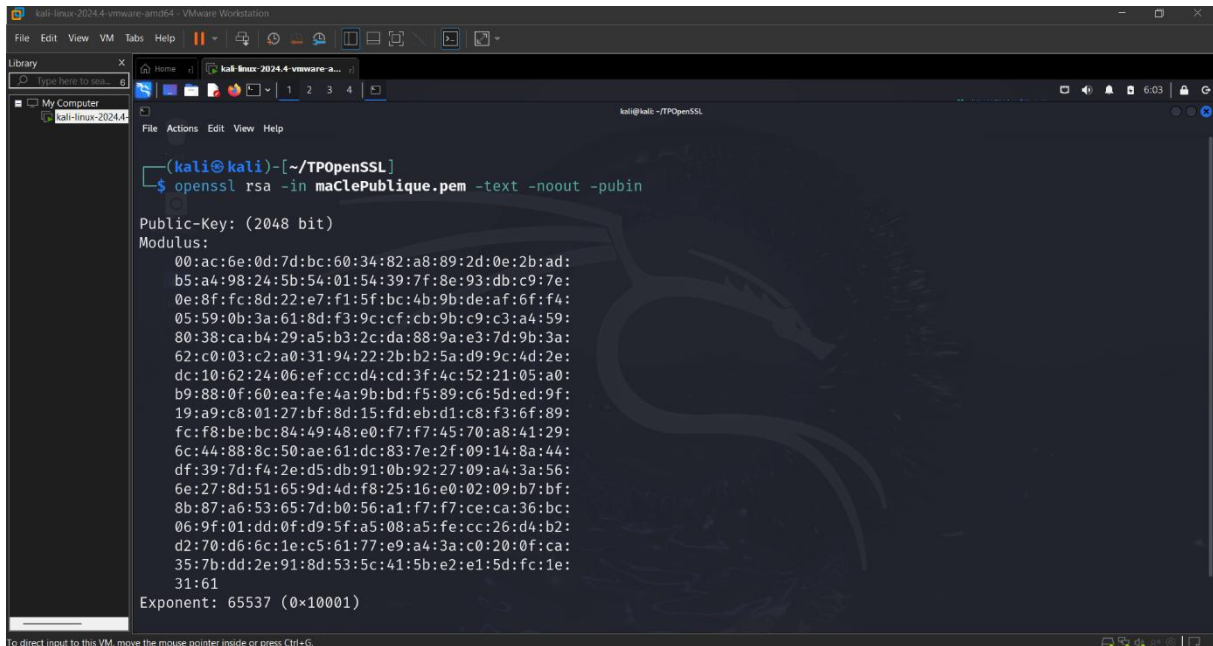
Question 3. Afficher le contenu des fichiers `maCle.pem` et `maClePublique.pem` avec la commande `cat`. Remarquer les marqueurs de début et de fin. 4.2. Visualisation des clés RSA La commande `rsa` permet de visualiser le contenu d'un fichier au format PEM contenant une paire de clés RSA.



```
(kali@kali)-[~/TP0openssl]
$ openssl rsa -in maCle.pem -text -noout

Private-Key: (2048 bit, 2 primes)
modulus:
00:ac:6e:0d:7d:bc:60:34:82:a8:89:2d:0e:2b:ad:
b5:a4:98:24:5b:54:01:54:39:7f:8e:93:db:c9:7e:
0e:8f:fc:8d:22:e7:f1:5f:bc:4b:9b:de:af:6f:f4:
05:59:0b:3a:61:8d:f3:9c:cf:cb:9b:c9:c3:a4:59:
80:38:ca:b4:29:a5:b3:2c:da:88:9a:e3:7d:9b:3a:
62:c0:03:c2:a0:31:94:22:2b:b2:5a:d9:9c:4d:2e:
dc:10:62:24:06:ef:cc:d4:cd:3f:4c:52:21:05:a0:
b9:88:0f:60:ea:fe:4a:9b:bd:f5:89:c6:5d:ed:9f:
19:a9:c8:01:27:bf:8d:15:fd:eb:d1:c8:f3:6f:89:
fc:f8:be:bc:84:49:48:e0:f7:f7:45:70:a8:41:29:
6c:44:88:8c:50:ae:61:dc:83:7e:2f:09:14:8a:44:
df:39:7d:f4:2e:d5:db:91:0b:92:27:09:a4:3a:56:
6e:27:8d:51:65:9d:4d:f8:25:16:e0:02:09:b7:bf:
8b:87:a6:53:65:7d:b0:56:a1:f7:f7:ce:ca:36:bc:
06:9f:01:dd:0f:d9:5f:a5:08:a5:fe:cc:26:d4:b2:
d2:70:d6:6c:1e:c5:61:77:e9:a4:3a:c0:20:0f:ca:
35:7b:dd:2e:91:8d:53:5c:41:5b:e2:e1:5d:fc:1e:
31:61
publicExponent: 65537 (0x10001)
privateExponent:
38:95:e9:21:81:27:45:34:37:a0:c4:c3:68:bc:19:
e7:08:f2:ad:71:f4:f9:b1:e0:d5:66:5a:73:56:45:
eb:24:a3:20:78:63:72:da:75:29:92:bb:9a:0e:c7:
```

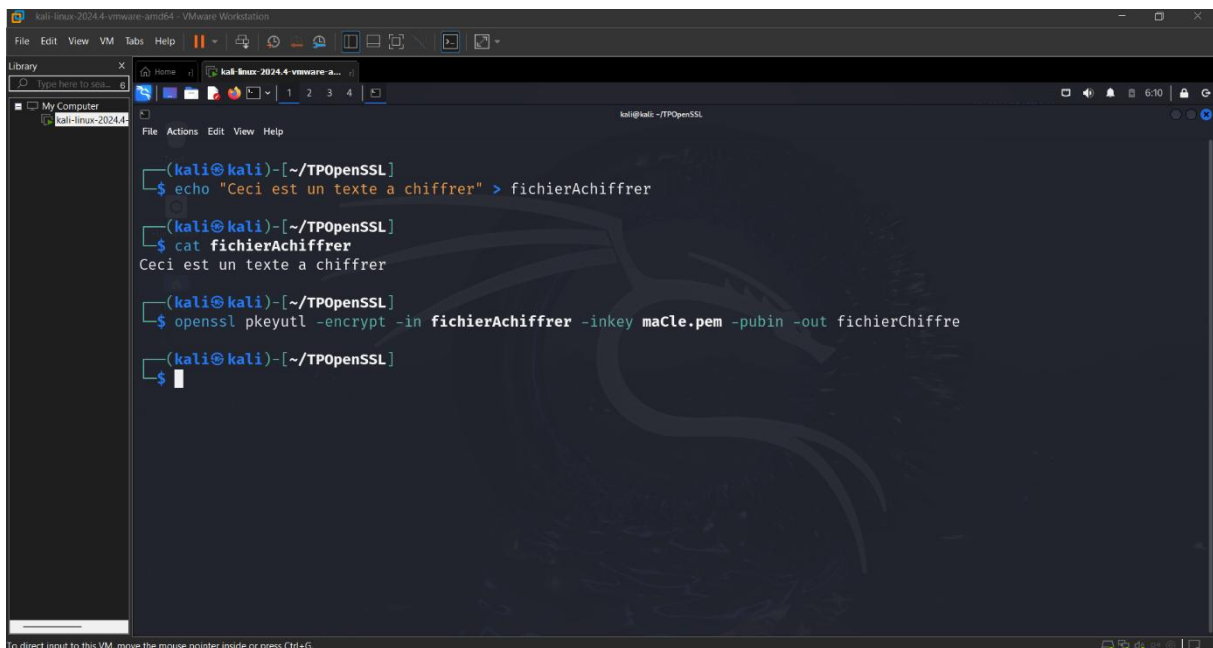
Question 4. Avec la commande `rsa` visualiser la clé publique. Attention vous devez préciser l'option `-pubin`, car seule la partie publique figure dans le fichier `maClePublique.pem`



```
(kali@kali)-[~/TPOpenSSL]
$ openssl rsa -in maClePublique.pem -text -noout -pubin

Public-Key: (2048 bit)
Modulus:
00:ac:6e:0d:7d:bc:60:34:82:a8:89:2d:0e:2b:ad:
b5:a4:98:24:5b:54:01:54:39:7f:8e:93:db:c9:7e:
0e:8f:fc:8d:22:e7:f1:5f:bc:4b:9b:de:af:6f:f4:
05:59:0b:3a:61:8d:f3:9c:cf:cb:9b:c9:c3:a4:59:
80:38:ca:b4:29:a5:b3:2c:da:88:9a:e3:7d:9b:3a:
62:c0:03:c2:a0:31:94:22:2b:b2:5a:d9:9c:4d:2e:
dc:10:62:24:06:ef:cc:d4:cd:3f:4c:52:21:05:a0:
b9:88:0f:60:ea:fe:4a:9b:bd:f5:89:c6:5d:ed:9f:
19:a9:c8:01:27:bf:8d:15:fd:eb:d1:c8:f3:6f:89:
fc:f8:be:bc:84:49:48:e0:f7:f7:45:70:a8:41:29:
6c:44:88:8c:50:ae:61:dc:83:7e:2f:09:14:8a:44:
df:39:7d:f4:2e:d5:db:91:0b:92:27:09:a4:3a:56:
6e:27:8d:51:65:9d:4d:f8:25:16:e0:02:09:b7:bf:
8b:87:a6:53:65:7d:b0:56:a1:f7:f7:ce:ca:36:bc:
06:9f:01:dd:0f:d9:5f:a5:08:a5:fe:cc:26:d4:b2:
d2:70:d6:6c:1e:c5:61:77:e9:a4:3a:c0:20:0f:ca:
35:7b:dd:2e:91:8d:53:5c:41:5b:e2:e1:5d:fc:1e:
31:61
Exponent: 65537 (0x10001)
```

4.3. Chiffrement/Déchiffrement de données par RSA On peut chiffrer des données avec une clé RSA. Pour cela on utilise la commande `pkeyutl`



```
(kali@kali)-[~/TPOpenSSL]
$ echo "Ceci est un texte a chiffrer" > fichierAchiffrer

(kali@kali)-[~/TPOpenSSL]
$ cat fichierAchiffrer
Ceci est un texte a chiffrer

(kali@kali)-[~/TPOpenSSL]
$ openssl pkeyutl -encrypt -in fichierAchiffrer -inkey maCle.pem -pubin -out fichierChiffre

(kali@kali)-[~/TPOpenSSL]
$
```

Question 5: Chiffrer un fichier texte puis déchiffrer avec la méthode RSA. 5. Hachage et Signature numérique de fichiers

```
(kali㉿kali)-[~/TP0penSSL]
$ echo "Ceci est un texte exemple pour le hachage" > fichier.txt

(kali㉿kali)-[~/TP0penSSL]
$ openssl dgst -sha256 -out empreinte.sha256 fichier.txt

(kali㉿kali)-[~/TP0penSSL]
$
```

Question 6: Calculer l’empreinte d’un fichier de votre choix. Signer un document revient à signer son empreinte. Pour cela, on utilise l’option `-sign` de la commande `pkeyutl`

```
(kali㉿kali)-[~/TP0penSSL]
$ openssl dgst -sha256 -binary -out empreinte.bin fichier.txt

(kali㉿kali)-[~/TP0penSSL]
$ ls -l empreinte.bin
-rw-rw-r-- 1 kali kali 32 Mar 29 06:22 empreinte.bin

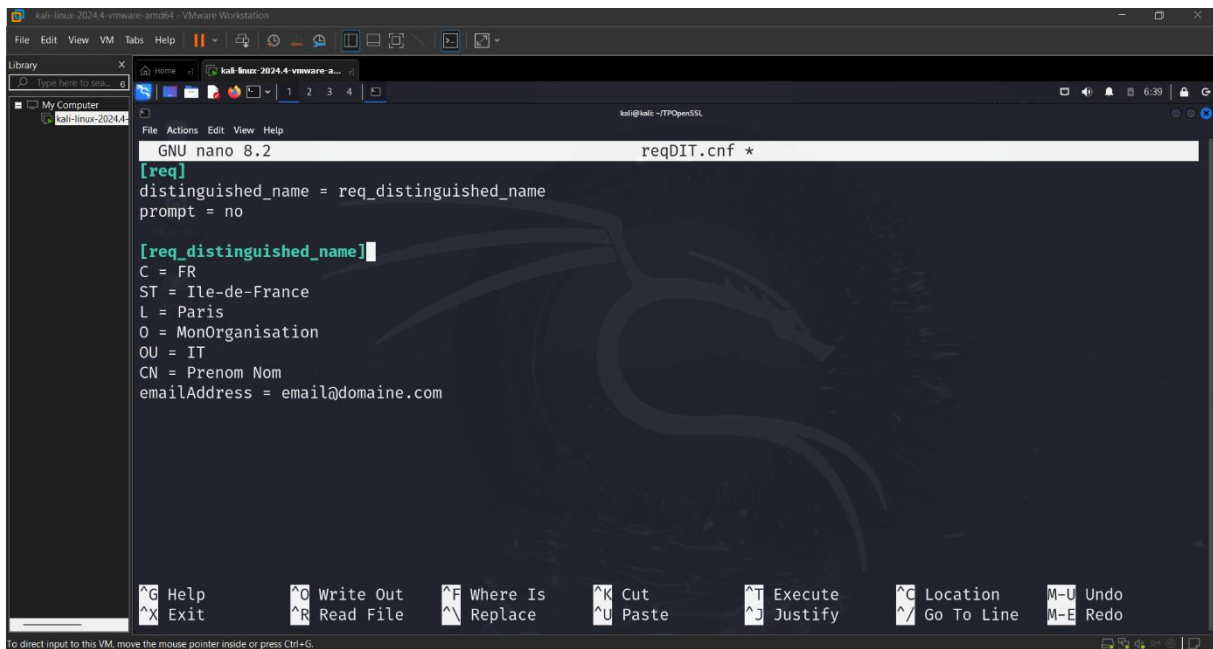
(kali㉿kali)-[~/TP0penSSL]
$ openssl pkeyutl -sign -in empreinte.bin -inkey maCle.pem -out signature.bin -pk
eyopt digest:sha256

(kali㉿kali)-[~/TP0penSSL]
$ openssl pkeyutl -verify -in empreinte.bin -sigfile signature.bin -inkey maClePub
lique.pem -pubin -pkeyopt digest:sha256
Signature Verified Successfully

(kali㉿kali)-[~/TP0penSSL]
$
```

Question 7: Signer le fichier de votre choix, puis vérifier la signature. Il est possible de produire directement une empreinte signée

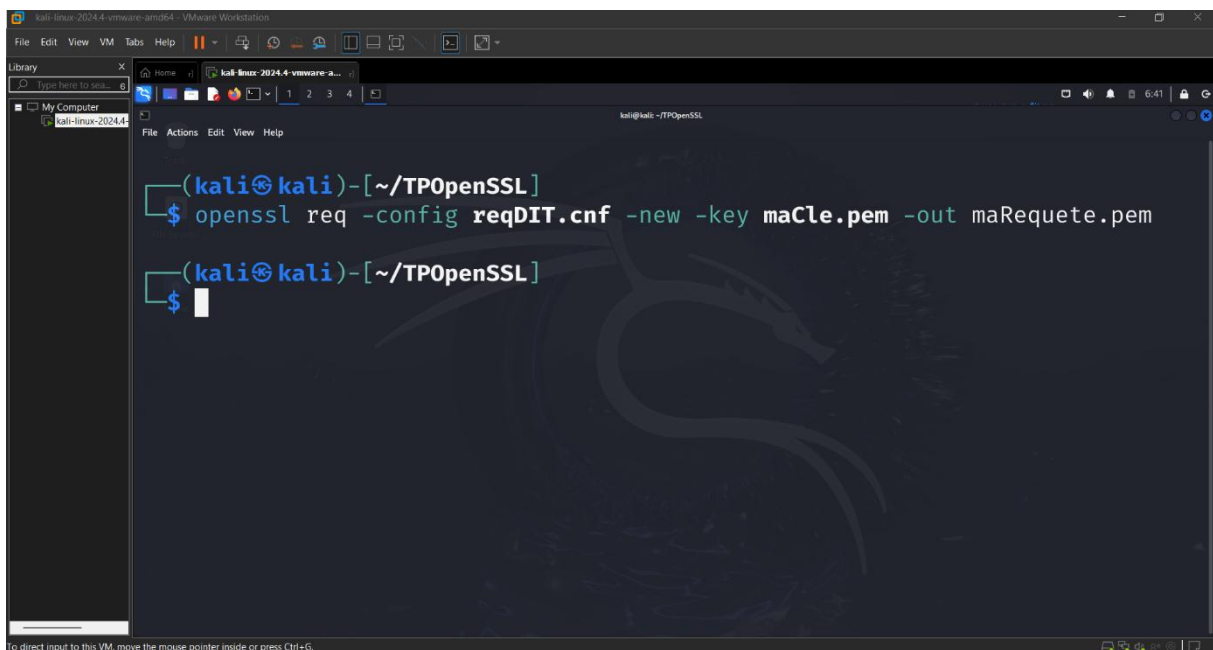
1. Créez un fichier de configuration nommé reqDIT.cnf :



```
GNU nano 8.2 reqDIT.cnf *
[req]
distinguished_name = req_distinguished_name
prompt = no

[req_distinguished_name]
C = FR
ST = Ile-de-France
L = Paris
O = MonOrganisation
OU = IT
CN = Prenom Nom
emailAddress = email@domaine.com
```

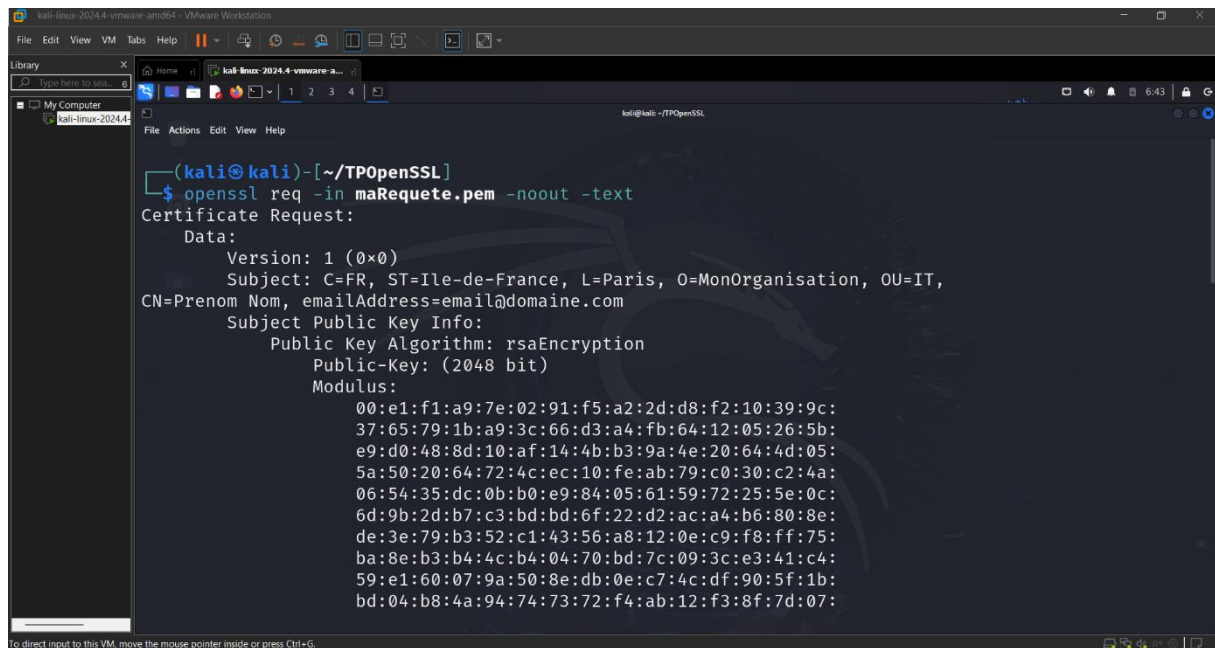
2. Générez la requête de certificat :



```
(kali@kali)-[~/TPOpenSSL]
$ openssl req -config reqDIT.cnf -new -key maCle.pem -out maRequete.pem

(kali@kali)-[~/TPOpenSSL]
$
```

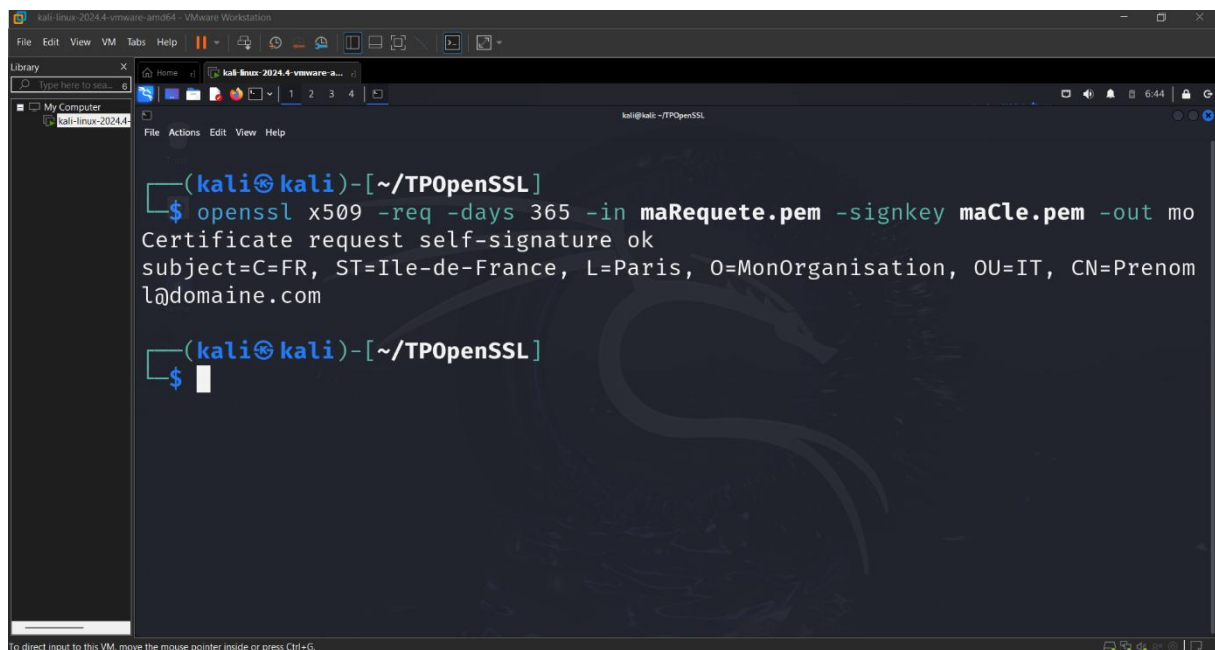
6.3. Vérification de la requête



```
(kali㉿kali)-[~/TPOpenSSL]
$ openssl req -in maRequete.pem -noout -text
Certificate Request:
Data:
  Version: 1 (0x0)
  Subject: C=FR, ST=Ile-de-France, L=Paris, O=MonOrganisation, OU=IT,
  CN=Prenom Nom, emailAddress=email@domaine.com
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (2048 bit)
    Modulus:
      00:e1:f1:a9:7e:02:91:f5:a2:2d:d8:f2:10:39:9c:
      37:65:79:1b:a9:3c:66:d3:a4:fb:64:12:05:26:5b:
      e9:d0:48:8d:10:af:14:4b:b3:9a:4e:20:64:4d:05:
      5a:50:20:64:72:4c:ec:10:fe:ab:79:c0:30:c2:4a:
      06:54:35:dc:0b:b0:e9:84:05:61:59:72:25:5e:0c:
      6d:9b:2d:b7:c3:bd:bd:6f:22:d2:ac:a4:b6:80:8e:
      de:3e:79:b3:52:c1:43:56:a8:12:0e:c9:f8:ff:75:
      ba:8e:b3:b4:4c:b4:04:70:bd:7c:09:3c:e3:41:c4:
      59:e1:60:07:9a:50:8e:db:0e:c7:4c:df:90:5f:1b:
      bd:04:b8:4a:94:74:73:72:f4:ab:12:f3:8f:7d:07:
```

6.4. Auto-signature (pour tests)

Si vous voulez créer un certificat auto-signé (sans passer par une autorité de certification) :

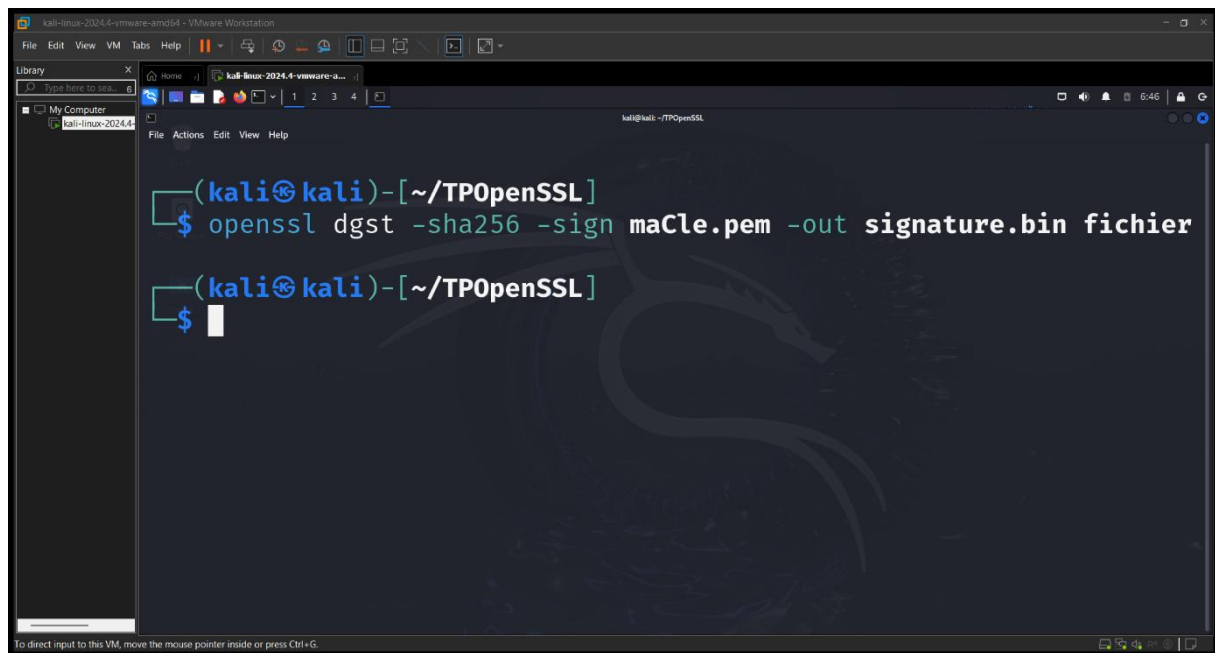


```
(kali㉿kali)-[~/TPOpenSSL]
$ openssl x509 -req -days 365 -in maRequete.pem -signkey maCle.pem -out mo
Certificate request self-signature ok
subject=C=FR, ST=Ile-de-France, L=Paris, O=MonOrganisation, OU=IT, CN=Prenom
l@domaine.com

(kali㉿kali)-[~/TPOpenSSL]
$
```

6.5. Utilisation du certificat

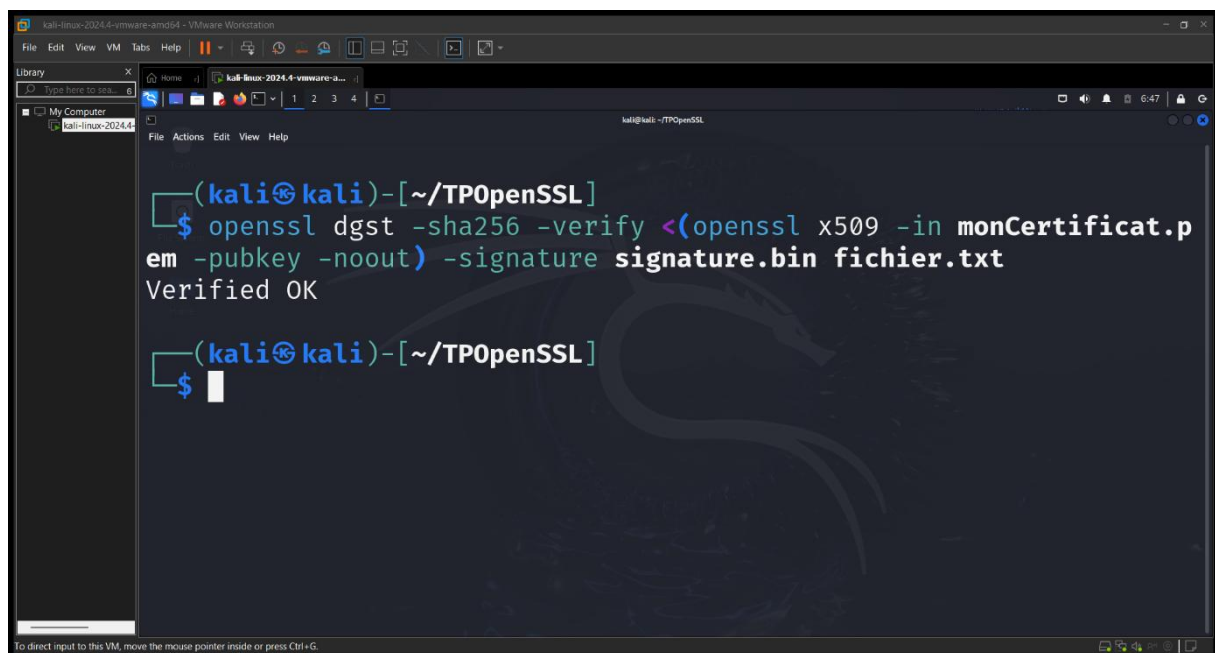
Pour signer un fichier :



```
(kali@kali)-[~/TP0openSSL]
$ openssl dgst -sha256 -sign maCle.pem -out signature.bin fichier
```

The screenshot shows a terminal window within a VMware Workstation. The prompt is `(kali@kali)-[~/TP0openSSL]`. The command `openssl dgst -sha256 -sign maCle.pem -out signature.bin fichier` has been entered and executed. The terminal background features a Kali Linux dragon logo.

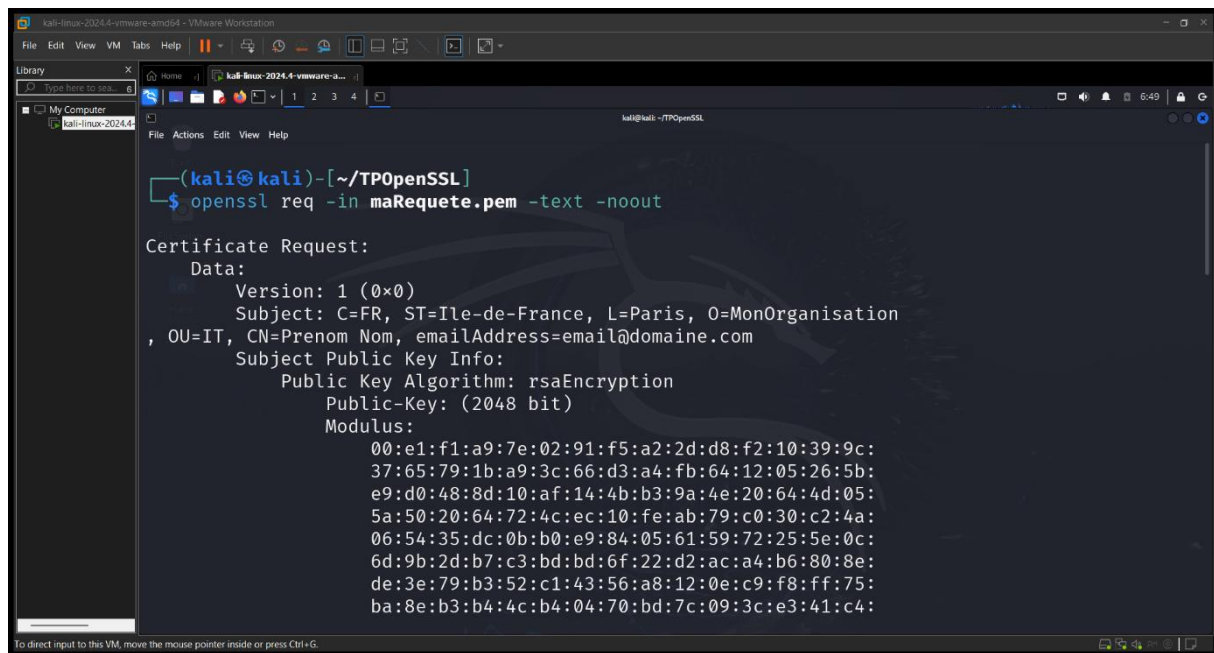
Pour vérifier avec le certificat :



```
(kali@kali)-[~/TP0openSSL]
$ openssl dgst -sha256 -verify <(openssl x509 -in monCertificat.pem -pubkey -noout) -signature signature.bin fichier.txt
Verified OK
```

The screenshot shows the same terminal window. The prompt is `(kali@kali)-[~/TP0openSSL]`. The command `openssl dgst -sha256 -verify <(openssl x509 -in monCertificat.pem -pubkey -noout) -signature signature.bin fichier.txt` has been entered and executed, resulting in the output `Verified OK`.

Question 9. Afficher le fichier produit maRequete.pem avec la commande cat.

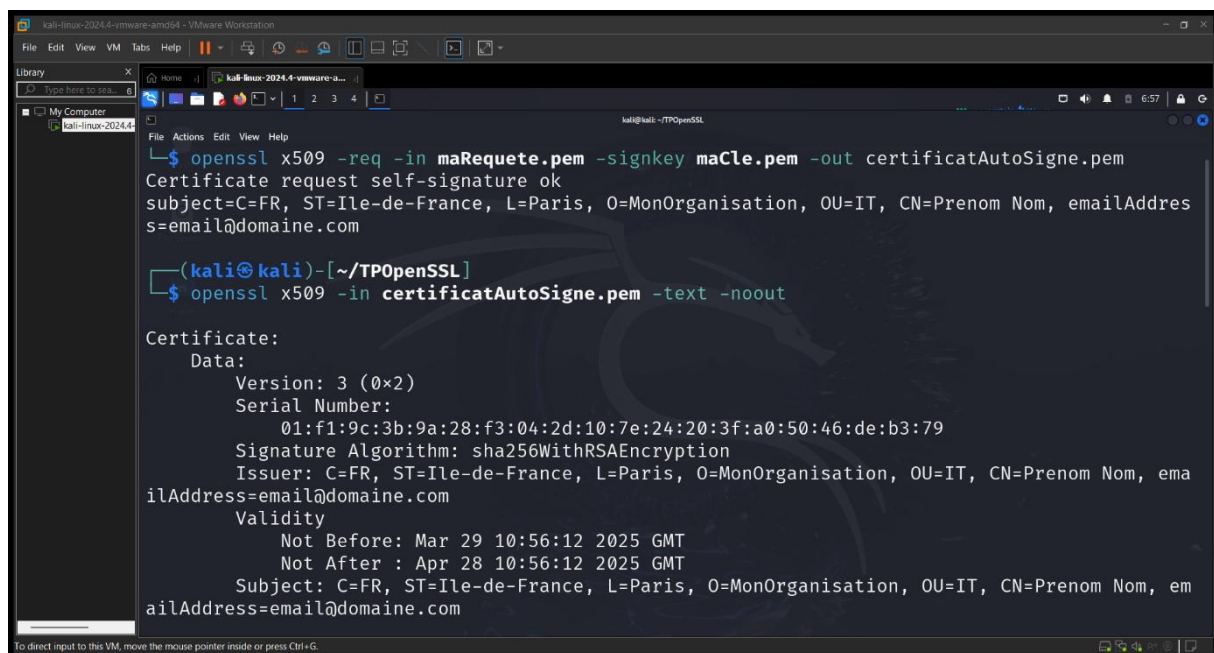


```
(kali㉿kali)-[~/TP0openssl]
$ openssl req -in maRequete.pem -text -noout

Certificate Request:
Data:
  Version: 1 (0x0)
  Subject: C=FR, ST=Ile-de-France, L=Paris, O=MonOrganisation
, OU=IT, CN=Prenom Nom, emailAddress=email@domaine.com
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (2048 bit)
    Modulus:
      00:e1:f1:a9:7e:02:91:f5:a2:2d:d8:f2:10:39:9c:
      37:65:79:1b:a9:3c:66:d3:a4:fb:64:12:05:26:5b:
      e9:d0:48:8d:10:af:14:4b:b3:9a:4e:20:64:4d:05:
      5a:50:20:64:72:4c:ec:10:fe:ab:79:c0:30:c2:4a:
      06:54:35:dc:0b:b0:e9:84:05:61:59:72:25:5e:0c:
      6d:9b:2d:b7:c3:bd:bd:6f:22:d2:ac:a4:b6:80:8e:
      de:3e:79:b3:52:c1:43:56:a8:12:0e:c9:f8:ff:75:
      ba:8e:b3:b4:4c:b4:04:70:bd:7c:09:3c:e3:41:c4:
```

Question 10 : Expliquer les différents éléments contenus dans cette requête. La clé privée du sujet y figure-t-elle ?

6.3. Auto-signer une requête (le demandeur devient CA)



```
(kali㉿kali)-[~/TP0openssl]
$ openssl x509 -req -in maRequete.pem -signkey maCle.pem -out certificatAutoSigne.pem
Certificate request self-signature ok
subject=C=FR, ST=Ile-de-France, L=Paris, O=MonOrganisation, OU=IT, CN=Prenom Nom, emailAdress=s=email@domaine.com

(kali㉿kali)-[~/TP0openssl]
$ openssl x509 -in certificatAutoSigne.pem -text -noout

Certificate:
Data:
  Version: 3 (0x2)
  Serial Number:
    01:f1:9c:3b:9a:28:f3:04:2d:10:7e:24:20:3f:a0:50:46:de:b3:79
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: C=FR, ST=Ile-de-France, L=Paris, O=MonOrganisation, OU=IT, CN=Prenom Nom, emailAdress=s=email@domaine.com
  Validity
    Not Before: Mar 29 10:56:12 2025 GMT
    Not After : Apr 28 10:56:12 2025 GMT
  Subject: C=FR, ST=Ile-de-France, L=Paris, O=MonOrganisation, OU=IT, CN=Prenom Nom, emailAdress=s=email@domaine.com
```

Question 11: Expliquer les différents éléments contenus dans le certificat.

6.4. Demande de signature de certificat auprès de la CA **Résumé des fichiers**

Éléments contenus dans un certificat X.509

Un certificat numérique (comme `certificatAutoSigne.pem` ou `PereUbuCertif.pem`) contient plusieurs champs standardisés. Voici leur signification :

1. En-tête et version

- **Format** : Généralement en **PEM** (-----BEGIN CERTIFICATE-----) ou **DER** (binaire).
- **Version** : X.509 v1, v2 ou v3 (la plus courante).

2. Informations sur le sujet (Subject)

Identifie l'entité (personne, serveur, organisation) à qui le certificat est délivré :

- **CN (Common Name)** → Nom du domaine (ex: `example.com`) ou de l'utilisateur.
- **O (Organization)** → Société (ex: Père Ubu Inc.).
- **OU (Organizational Unit)** → Département (ex: IT Security).
- **L (Locality)** → Ville (ex: Paris).
- **ST (State/Province)** → Région (ex: Ile-de-France).
- **C (Country)** → Code pays (ex: FR).
- **emailAddress** → Contact (ex: `admin@pere-ubu.com`).

3. Informations sur l'émetteur (Issuer)

Décrit l'Autorité de Certification (CA) qui a signé le certificat :

- Pour un certificat **auto-signé**, Issuer = Subject.
- Pour un certificat émis par une CA externe (ex: Père Ubu CA), l'émetteur est différent.

4. Période de validité

- **Not Before** → Date de début (ex: Jan 1 00:00:00 2024 GMT).
- **Not After** → Date d'expiration (ex: Dec 31 23:59:59 2024 GMT).

5. Clé publique du sujet

- **Algorithm** → RSA/ECC/DSA.

- **Public Key** → La clé publique liée au certificat.
 - Pour RSA : Modulus et Exponent.
 - Pour ECC : Curve et Pubkey.

6. Extensions (X.509 v3)

- **Key Usage** → Utilisations autorisées (Digital Signature, Key Encipherment, etc.).
- **Subject Alternative Name (SAN)** → Noms alternatifs (ex: DNS:example.com).
- **Basic Constraints** → Indique si le certificat est une CA (CA:TRUE/CA:FALSE).
- **CRL Distribution Points** → URL pour vérifier les révocations.

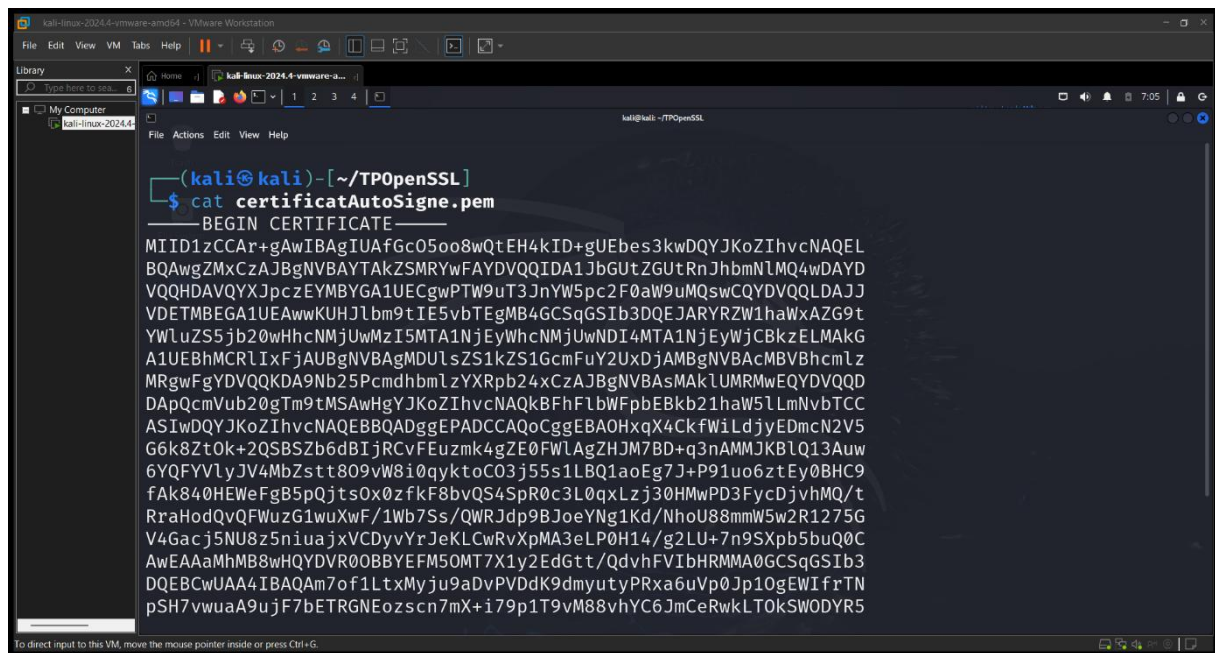
7. Signature numérique

- **Algorithm** → Ex: sha256WithRSAEncryption.
- **Valeur** → Signature de la CA pour authentifier le certificat.

RESUME DU FICHIER:

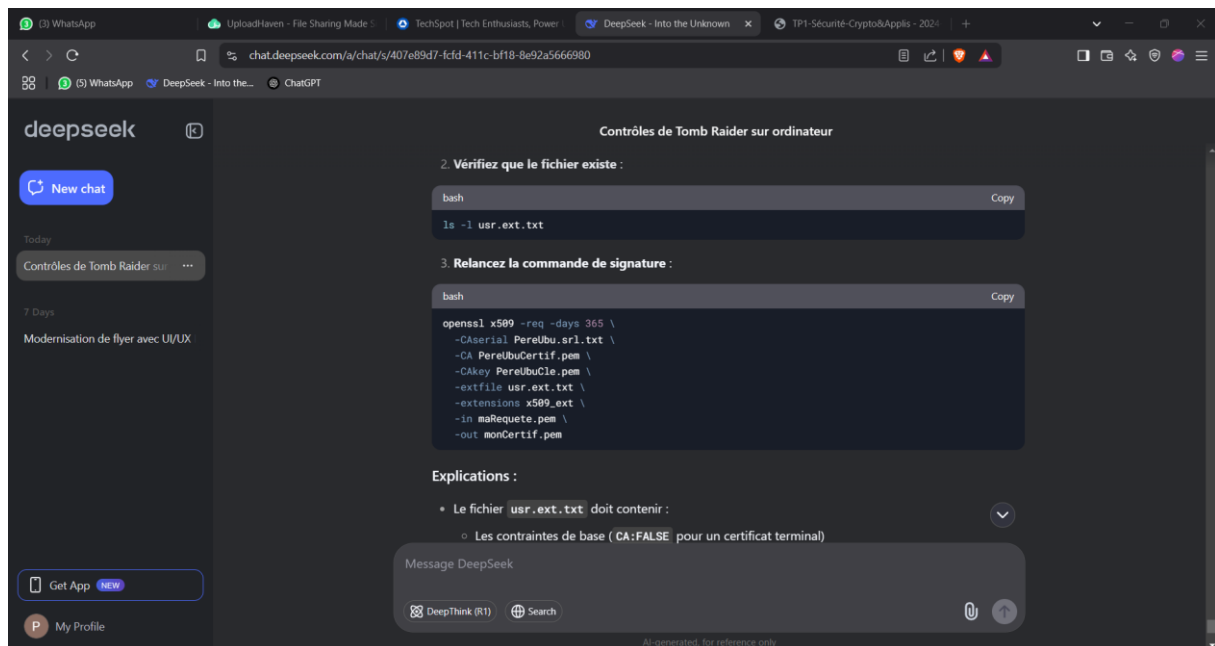
Fichier	Rôle
maCle.pem	Clé privée du demandeur (à garder secrète).
maRequete.pem	CSR (contient la clé publique + infos).
PereUbuCertif.pem	Certificat de la CA Père Ubu.
PereUbuCle.pem	Clé privée de la CA (utilisée pour signer).
certificatSigne.pem	Certificat final signé par la CA.

Question 12: Afficher le certificat produit avec la commande cat.

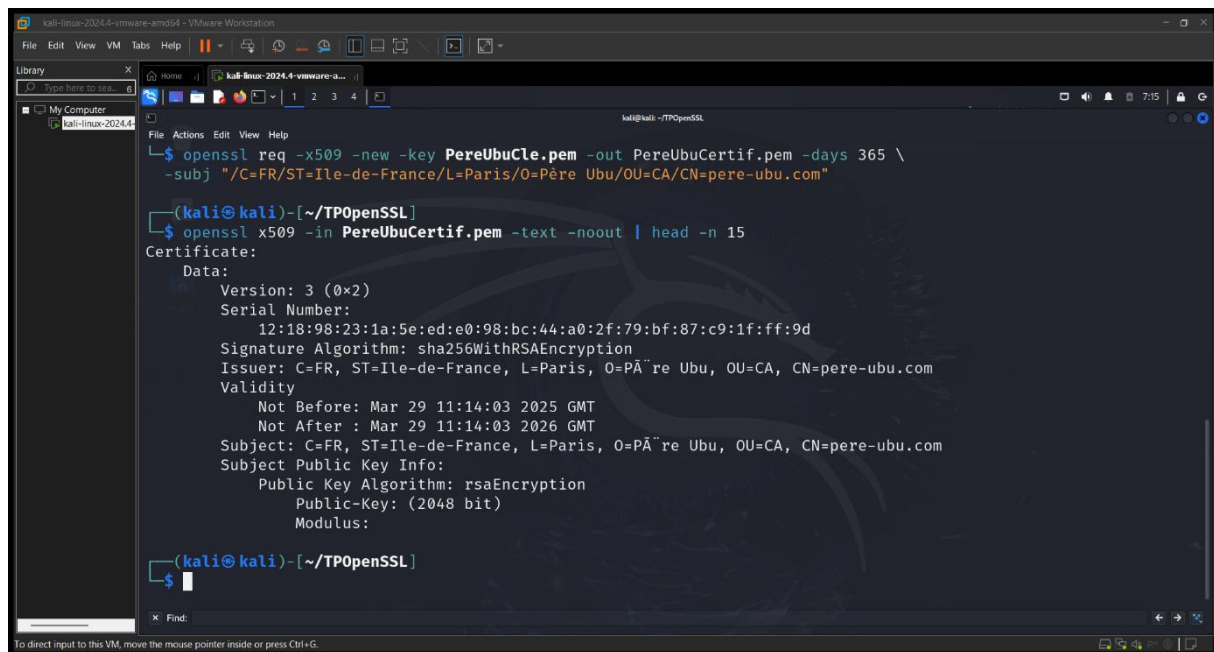


```
(kali@kali) - [~/TP0openssl]
$ cat certificatAutoSigne.pem
-----BEGIN CERTIFICATE-----
MIID1zCCAr+gAwIBAgIUAFGcO5oo8wQtEH4kID+gUEbes3kwDQYJKoZIhvcNAQEL
BQAwgZMx CzA JBgNVBAYTAkZSMRYwFAYDVQQIDA1JbGUtZG9tRnJhbmNlMQ4wDAYD
VQQHDAVQYXJpczEYMBYGA1UECgwPTW9uT3JnYW5pc2F0aW9uMQswCQYDVQQLDAJJ
VDEtMBEGA1UEAwwKUUhJLm9tIE5ybTE5vTE5vTE5vTE5vTE5vTE5vTE5vTE5vTE5v
YWVlZS5jb20wHhcNMjI1MTA1NjE5WWhcNMjI1MTA1NjE5WWhcNMjI1MTA1NjE5WWhc
A1UEBHMCRlIx FjAUBgNVBAGMDUlsZS1kZS1GcmFuY2UxZjA1MBgNVBACMBVBhcmLz
MRgwFgYDVQKDA9Nb25PcmduY2UxZjA1MBgNVBAsMAkUUMRMwEQYDVQKDA9Nb25Pcmdu
DAPQcmVub20gTm9tMSAwHgYJKoZIhvcNAQkBFhFlbW9uT3JnYW5pc2F0aW9uMQsw
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAAOHxqX4CkFWiLdJyEDmcN2V5
G6k8ZtOk+2QSBZb6dBIjRCvFEuzmk4gZE0FWLAgZHM7BD+q3nAMMJKBLQ13Auw
6YQFYVlyJV4MbZstt809vW8i0qyktoC03j55s1LBQ1aoEg7J+P91uo6ztEy0BHC9
fAk840HEWefgB5pQjts0x0zfKf8bvQS4Spr0c3L0qxLzj30HMwPD3FycDjvhMQ/t
RrahodQvQFWuzG1wuXwF/1Wb7Ss/QWRJdp9BJoeYNg1Kd/NhoU88mmW5w2R1275G
V4Gacj5NU8z5niaujxVCDyYrJeKLCwRvXpMA3eLP0H14/g2LU+7n9SXPb5buQ0C
AwEAAAMhMB8wHQYDVR00BBYEFM5OMT7X1y2EdGt/QdvhFVibHRMMA0GCSqGSIb3
DQEBChUAA4IBAQA7of1LtXMyju9aDvPVDdK9dmyutyPRxa6uVp0Jp10gEWIfrTN
pSH7vwuaA9uJf7bETRGNcozscn7mX+i79p1T9vM88vhYC6JmCeRwLTK0kSWODYR5
```

Question 13: Après avoir récupéré le certificat de l'autorité, ainsi que sa paire de clés RSA, chercher quelle est la date d'expiration du certificat et la taille de la clé



Vérification de certificats : vérifier la validité d'un certificat avec la commande `verify`. Pour cela, il est nécessaire de disposer du certificat de l'autorité qui l'a émis.



```
kali@kali: ~/TPOpenSSL
$ openssl req -x509 -new -key PereUbuCle.pem -out PereUbuCertif.pem -days 365 \
-subj "/C=FR/ST=Ile-de-France/L=Paris/O=Père Ubu/OU=CA/CN=pere-ubu.com"

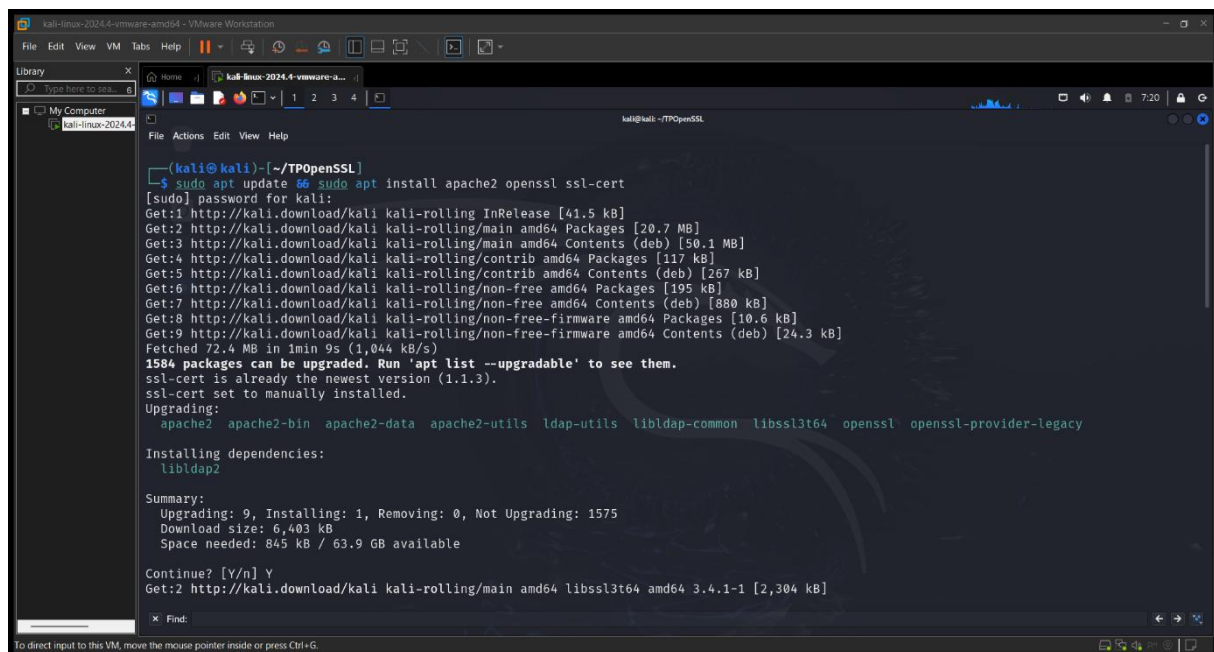
(kali@kali)-[~/TPOpenSSL]
$ openssl x509 -in PereUbuCertif.pem -text -noout | head -n 15
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            12:18:98:23:1a:5e:ed:e0:98:bc:44:a0:2f:79:bf:87:c9:1f:ff:9d
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=FR, ST=Ile-de-France, L=Paris, O=Père Ubu, OU=CA, CN=pere-ubu.com
        Validity
            Not Before: Mar 29 11:14:03 2025 GMT
            Not After : Mar 29 11:14:03 2026 GMT
        Subject: C=FR, ST=Ile-de-France, L=Paris, O=Père Ubu, OU=CA, CN=pere-ubu.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:

(kali@kali)-[~/TPOpenSSL]
$
```

7. Home work :

7.1. Mettre en place un serveur Apache-SSL (passer de http à https)

1. Installation des paquets nécessaires



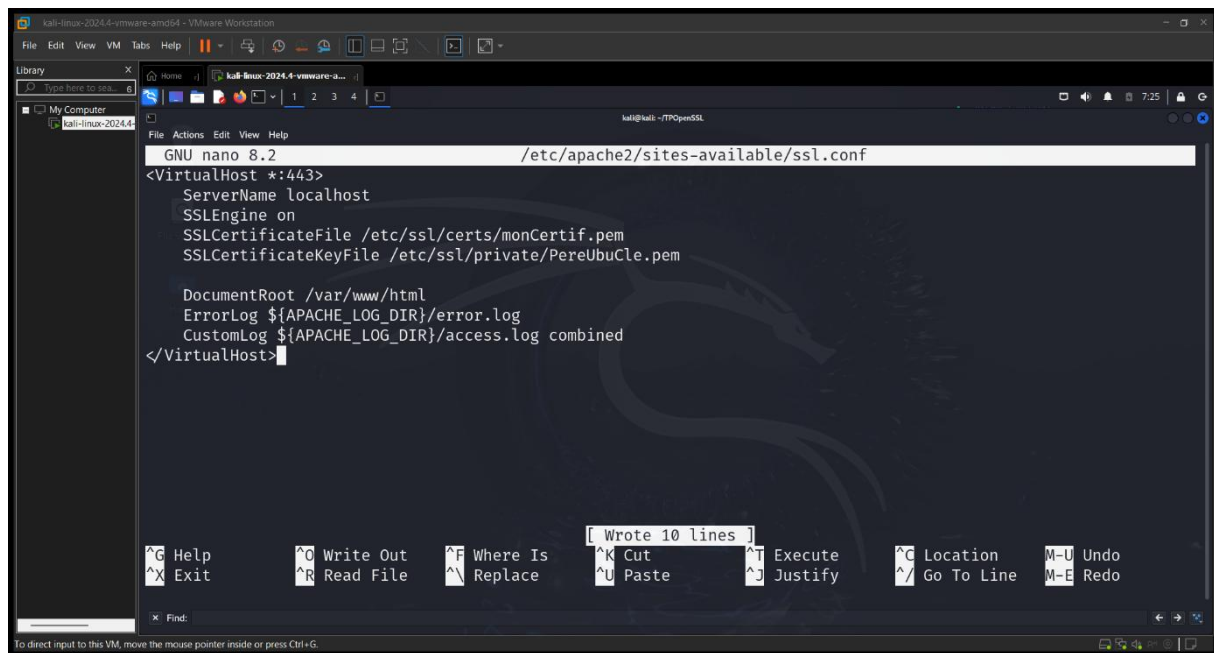
```
kali@kali: ~/TPOpenSSL
$ sudo apt update && sudo apt install apache2 openssl ssl-cert
[sudo] password for kali:
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [20.7 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [50.1 MB]
Get:4 http://kali.download/kali kali-rolling/contrib amd64 Packages [117 kB]
Get:5 http://kali.download/kali kali-rolling/contrib amd64 Contents (deb) [267 kB]
Get:6 http://kali.download/kali kali-rolling/non-free amd64 Packages [195 kB]
Get:7 http://kali.download/kali kali-rolling/non-free amd64 Contents (deb) [880 kB]
Get:8 http://kali.download/kali kali-rolling/non-free-firmware amd64 Packages [10.6 kB]
Get:9 http://kali.download/kali kali-rolling/non-free-firmware amd64 Contents (deb) [24.3 kB]
Fetched 72.4 MB in 1min 9s (1,044 kB/s)
1584 packages can be upgraded. Run 'apt list --upgradable' to see them.
ssl-cert is already the newest version (1.1.3).
ssl-cert set to manually installed.
Upgrading:
  apache2  apache2-bin  apache2-data  apache2-utils  ldap-utils  libldap-common  libssl3t64  openssl  openssl-provider-legacy

Installing dependencies:
  libldap2

Summary:
  Upgrading: 9, Installing: 1, Removing: 0, Not Upgrading: 1575
  Download size: 6,403 kB
  Space needed: 845 kB / 63.9 GB available

Continue? [y/n] Y
Get:2 http://kali.download/kali kali-rolling/main amd64 libssl3t64 amd64 3.4.1-1 [2,304 kB]
Get:3 http://kali.download/kali kali-rolling/main amd64 openssl amd64 3.4.1-1 [2,304 kB]
```

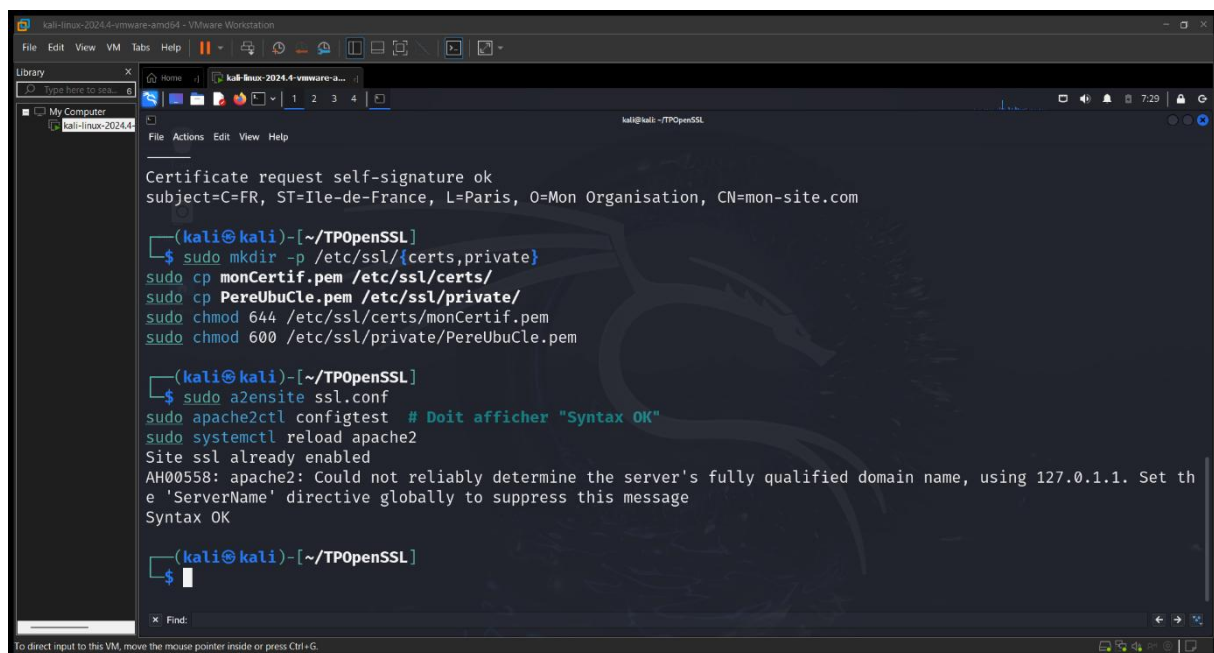
2. Activation du module SSL Apache



```
GNU nano 8.2 /etc/apache2/sites-available/ssl.conf
<VirtualHost *:443>
    ServerName localhost
    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/monCertif.pem
    SSLCertificateKeyFile /etc/ssl/private/PereUbuCle.pem

    DocumentRoot /var/www/html
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

5. Activation et test



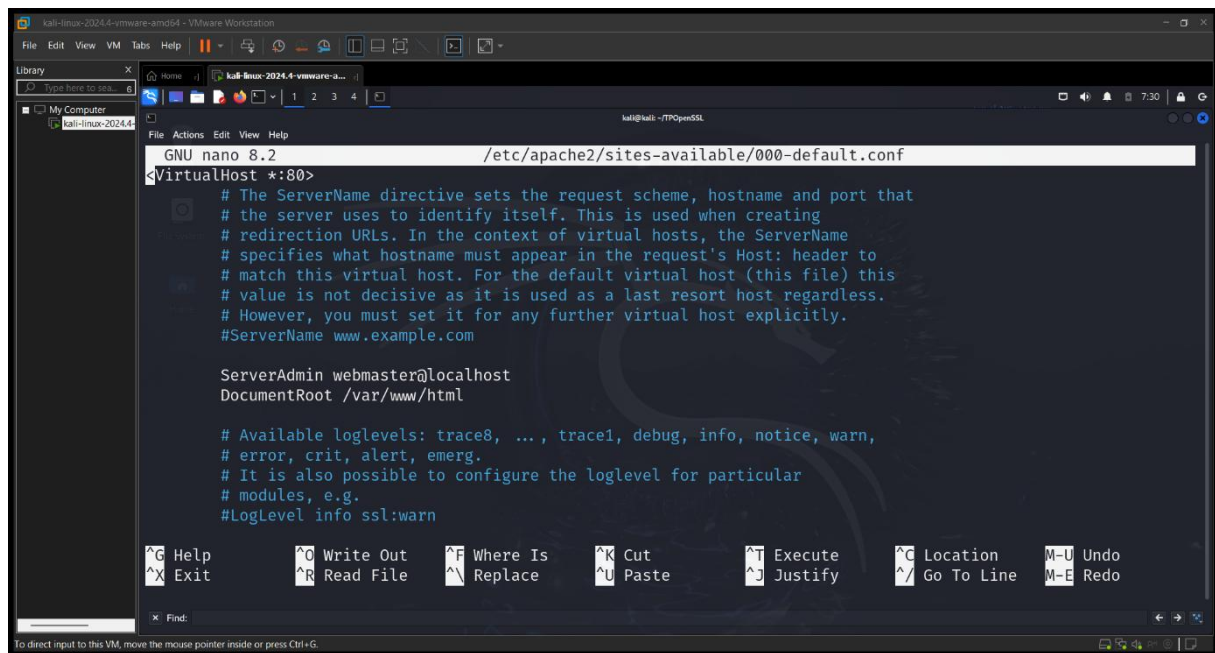
```
Certificate request self-signature ok
subject=C=FR, ST=Ile-de-France, L=Paris, O=Mon Organisation, CN=mon-site.com

(kali@kali)-[~/TP0penSSL]
$ sudo mkdir -p /etc/ssl/{certs,private}
$ sudo cp monCertif.pem /etc/ssl/certs/
$ sudo cp PereUbuCle.pem /etc/ssl/private/
$ sudo chmod 644 /etc/ssl/certs/monCertif.pem
$ sudo chmod 600 /etc/ssl/private/PereUbuCle.pem

(kali@kali)-[~/TP0penSSL]
$ sudo a2ensite ssl.conf
$ sudo apache2ctl configtest # Doit afficher "Syntax OK"
$ sudo systemctl reload apache2
Site ssl already enabled
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message
Syntax OK

(kali@kali)-[~/TP0penSSL]
$
```

7. Redirection HTTP → HTTPS (optionnel)



```
GNU nano 8.2 /etc/apache2/sites-available/000-default.conf
<VirtualHost *:80>
# The ServerName directive sets the request scheme, hostname and port that
# the server uses to identify itself. This is used when creating
# redirection URLs. In the context of virtual hosts, the ServerName
# specifies what hostname must appear in the request's Host: header to
# match this virtual host. For the default virtual host (this file) this
# value is not decisive as it is used as a last resort host regardless.
# However, you must set it for any further virtual host explicitly.
#ServerName www.example.com

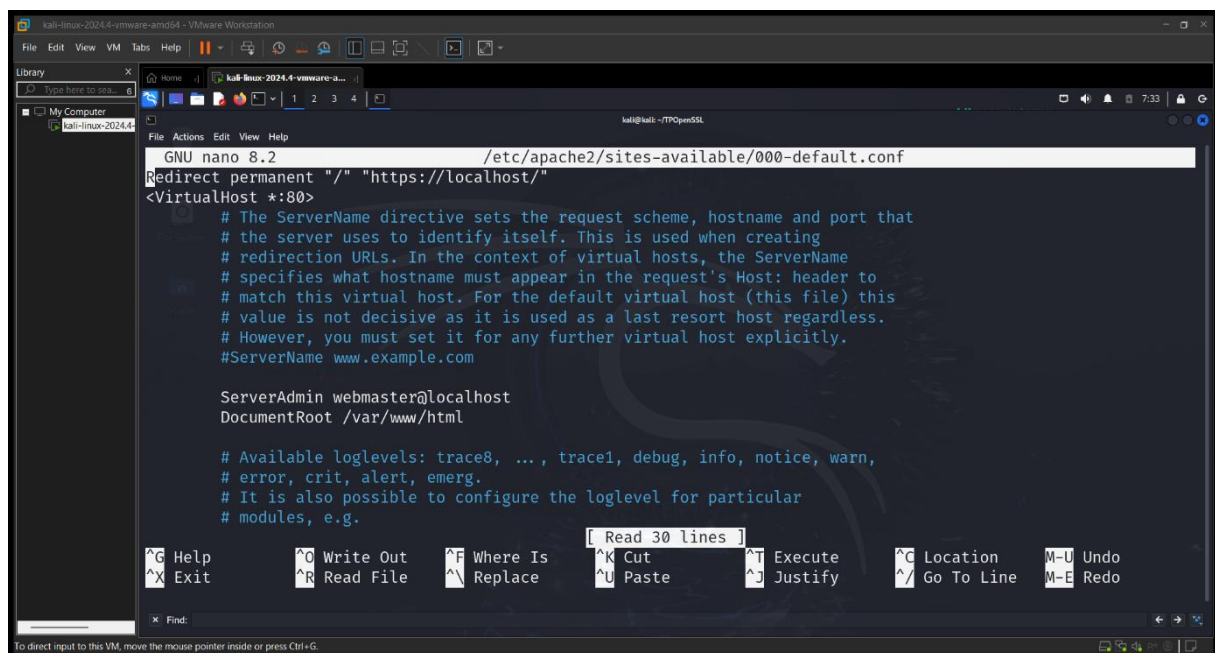
ServerAdmin webmaster@localhost
DocumentRoot /var/www/html

# Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
# error, crit, alert, emerg.
# It is also possible to configure the loglevel for particular
# modules, e.g.
#LogLevel info ssl:warn

^G Help      ^O Write Out  ^F Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/_ Go To Line  M-E Redo

Find:
```

Ajoutez avant </VirtualHost> :



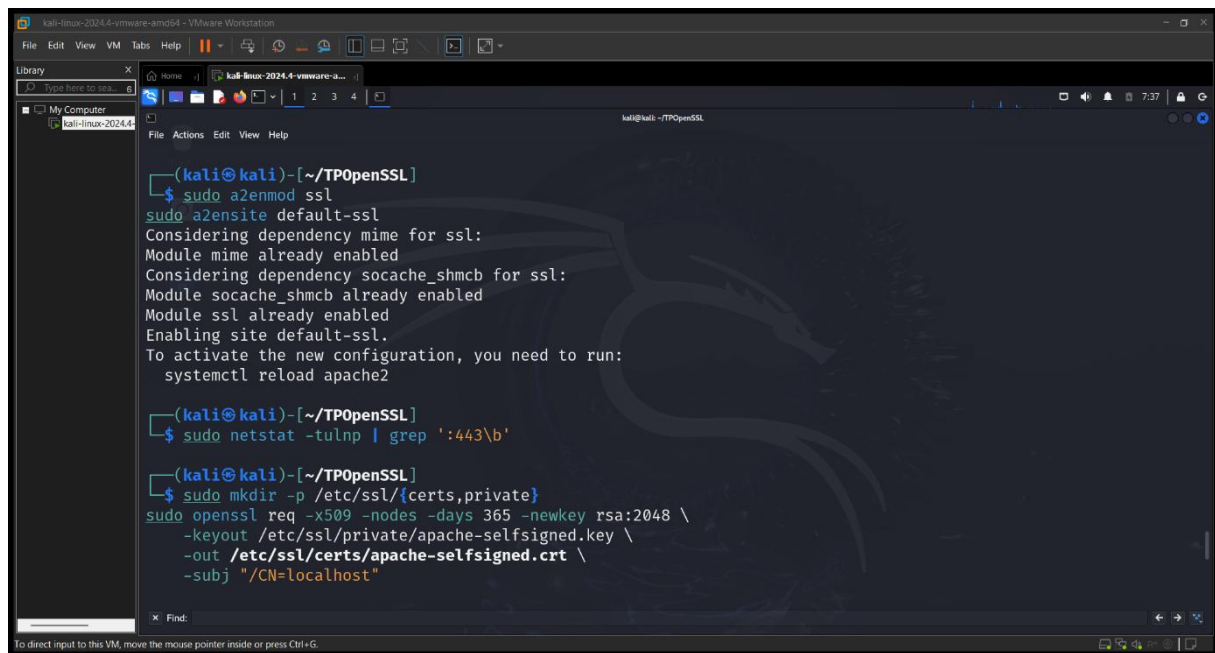
```
GNU nano 8.2 /etc/apache2/sites-available/000-default.conf
Redirect permanent "/" "https://localhost/"
<VirtualHost *:80>
# The ServerName directive sets the request scheme, hostname and port that
# the server uses to identify itself. This is used when creating
# redirection URLs. In the context of virtual hosts, the ServerName
# specifies what hostname must appear in the request's Host: header to
# match this virtual host. For the default virtual host (this file) this
# value is not decisive as it is used as a last resort host regardless.
# However, you must set it for any further virtual host explicitly.
#ServerName www.example.com

ServerAdmin webmaster@localhost
DocumentRoot /var/www/html

# Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
# error, crit, alert, emerg.
# It is also possible to configure the loglevel for particular
# modules, e.g.

[ Read 30 lines ]
^G Help      ^O Write Out  ^F Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/_ Go To Line  M-E Redo

Find:
```



```
(kali@kali)-[~/TP0penSSL]
$ sudo a2enmod ssl
sudo a2ensite default-ssl
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Module socache_shmcb already enabled
Module ssl already enabled
Enabling site default-ssl.
To activate the new configuration, you need to run:
    systemctl reload apache2

(kali@kali)-[~/TP0penSSL]
$ sudo netstat -tulnp | grep ':443\b'

(kali@kali)-[~/TP0penSSL]
$ sudo mkdir -p /etc/ssl/{certs,private}
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout /etc/ssl/private/apache-selfsigned.key \
-out /etc/ssl/certs/apache-selfsigned.crt \
-subj "/CN=localhost"
```