Fascicule 3 : La Cryptographie avec OpenSSL

Travail réalisé par :

HARBAOUI Wassim

KEFI Mohamed Mehdi

GUEDIDI Raed

LASSOUED Tesnim Rafia

Objectif

L'objectif de ce lab est de vous familiariser avec les services de base de la sécurité. Notamment le chiffrement symétrique, le chiffrement asymétrique, le hachage, la signature numérique et sa vérification, et la certification.

Ces travaux pratiques sont basés sur la suite logicielle OpenSSL.
OpenSSL est une boîte à outils cryptographiques implémentant les protocoles SSL et TLS qui offre une bibliothèque de programmation en C permettant de réaliser des applications client/serveur sécurisées s'appuyant sur SSL/TLS.

1. CHIFFREMENT SYMETRIQUE

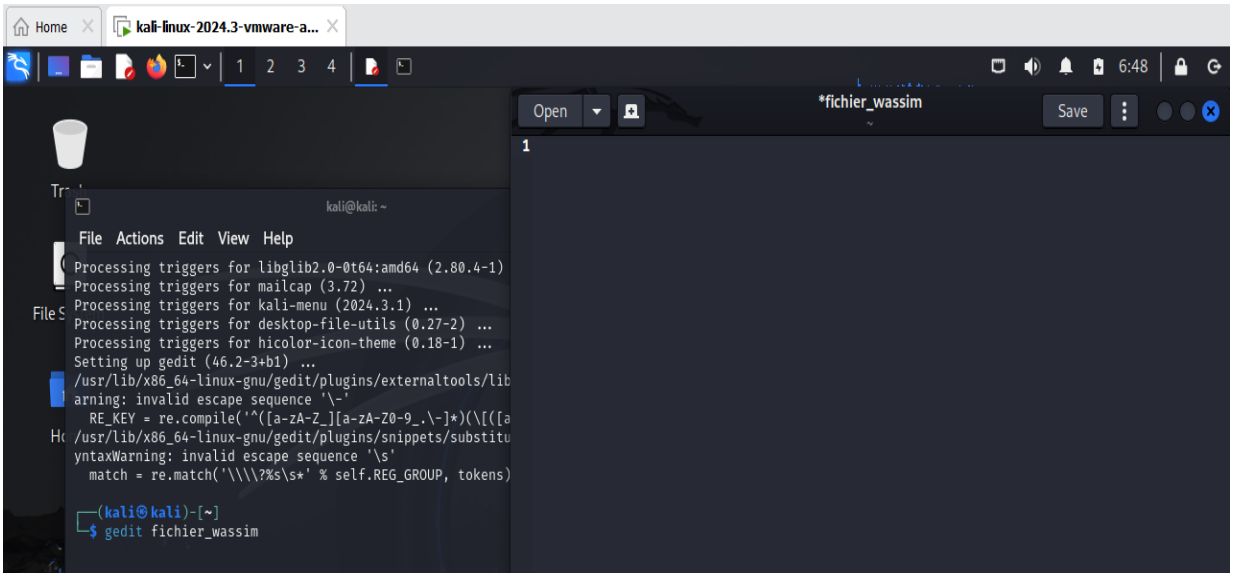
L'algorithme RC4

La commande qui vous permet d'utiliser le chiffrement symétrique est la commande

```
#openssl enc –in fichier_clair –out resultat_obtenu
```

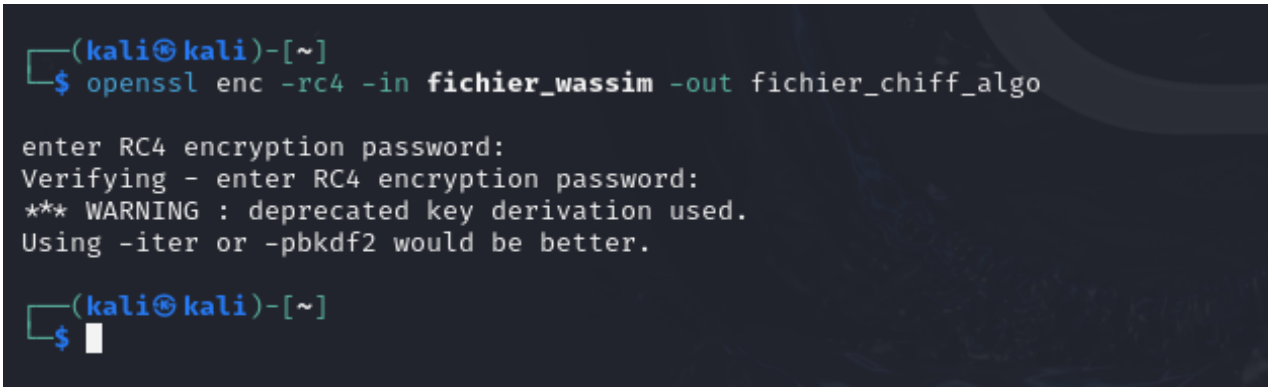
- Créer le fichier fichier_nom_eleve contenant du texte clair

```
#gedit fichier_nom_eleve
```



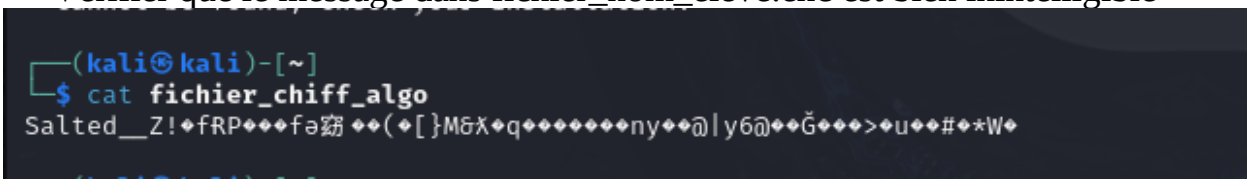
- Chiffrer ce fichier avec l'algorithme RC4

openssl enc –rc4 -in fichier_nom_eleve -out fichier_chiff_algo



Le chiffrement symétrique utilise une clé dérivée d'un mot de passe pour chiffrer et déchiffrer le fichier.

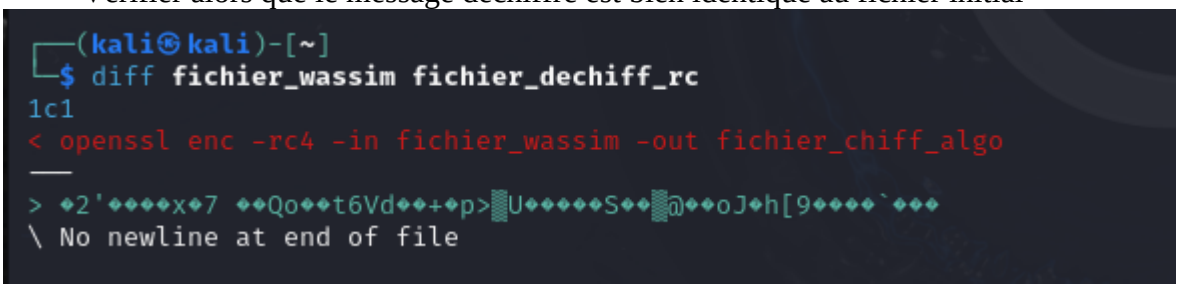
- Vérifier que le message dans fichier_nom_eleve.enc est bien inintelligible



Vérifier l'inintelligibilité : *Nous pouvons essayer d'afficher le fichier chiffré avec cat fichier_chiff_algo pour confirmer qu'il est illisible.*

- Ecrire la commande qui permet de le déchiffrer et produit ainsi le fichier

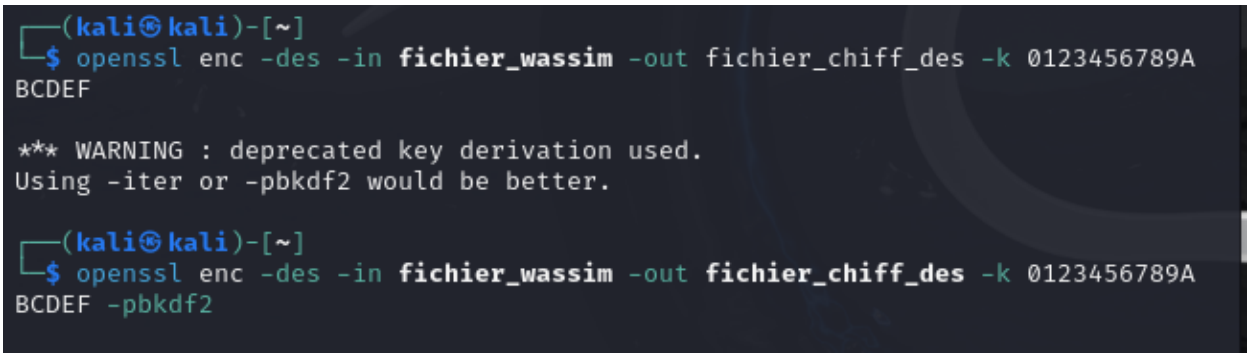
- Vérifier alors que le message déchiffré est bien identique au fichier initial



L'algorithme DES

Pour chiffrer le fichier fichier_nom_eleve avec l'algorithme DES avec clé explicite :

openssl enc -des -in fichier_nom_eleve -out fichier_chiff_des -k 0123456789ABCDEF



- **openssl enc** : Utilise l'outil de chiffrement/déchiffrement d'OpenSSL.

-des : Indique que l'algorithme de chiffrement utilisé est DES.

-in fichier_nom_eleve : Remplacez fichier_nom_eleve par le nom de votre fichier contenant le texte clair à chiffrer.

-out fichier_chiff_des : Remplacez fichier_chiff_des par le nom que vous souhaitez donner au fichier chiffré.

-k 0123456789ABCDEF : Spécifie une clé de chiffrement explicite (ici, une clé de 16 caractères).

- Quelle est la commande qui permet de déchiffrer fichier_chiff_des ?

```
┌──(kali㉿kali)-[~]
└─$ openssl enc -des -d -in fichier_chiff_des -out fichier_dechiff_des -k 012
3456789ABCDEF -pbkdf2
```

-d : Indique que vous souhaitez déchiffrer le fichier.
-in fichier_chiff_des : Remplacez fichier_chiff_des par le nom du fichier chiffré.
-out fichier_dechiff_des : Remplacez fichier_dechiff_des par le nom que vous souhaitez donner au fichier déchiffré.

- Vérifier que le fichier déchiffré est identique au fichier initial.

```
┌──(kali㉿kali)-[~]
└─$ diff fichier_wassim fichier_dechiff_des
```

diff : Compare les fichiers spécifiés.

fichier_wassim : Le fichier original contenant le texte clair.

fichier_dechiff_des : Le fichier qui a été déchiffré.

NB :

La commande diff ne renvoie aucune sortie, cela signifie que les fichiers sont identiques. En revanche, si des différences sont signalées, cela indique que le fichier déchiffré ne correspond pas au fichier d'origine.

2. CHIFFREMENT ASYMETRIQUE

Le chiffrement asymétrique RSA demande l'utilisation d'une paire de clés : une clé publique et une clé privée qu'il faut tout d'abord les générer.

Génération de clé privée/publique RSA

En openSSL, les clés RSA générées sont stockées dans un fichier avec l'extension .pem (Privacy Enhanced Mail). L'instruction à utiliser est *openssl genrsa*.

Utiliser cette instruction pour :

- Générer une paire de clés de taille 1024 bits et stocker-la dans le fichier **rsakey.pem**

```
┌──(kali㉿kali)-[~]
└─$ openssl genrsa -out rsakey.pem 1024
```

-

- Afficher le fichier en utilisant la commande **cat**. Qu’est ce que vous remarquez ?

Ceci affiche la clé privée en format PEM.

Une façon de visualiser les clés en format complet est d'utiliser la commande **rsa**.Afficher alors les clés en format hexadécimal, en supprimant la sortienormalement produite par l'instruction rsa.

```
┌──(kali㉿kali)-[~]
└─$ openssl rsa -in rsakey.pem -text -noout

Private-Key: (1024 bit, 2 primes)
modulus:
    00:ca:83:97:9b:a2:8e:dc:4c:2d:71:84:9f:18:bd:
    8e:bd:0a:75:22:ab:de:9e:28:b4:4a:45:95:74:93:
    e1:6b:c2:89:f8:76:59:69:60:97:0a:5d:7a:a5:f8:
    8a:72:b1:0a:2e:5f:09:10:34:7c:65:26:dd:18:9d:
    c6:79:52:05:6d:67:72:66:af:70:41:ca:88:2b:c5:
    ed:e6:4d:0c:3e:a9:48:0c:6a:b6:30:71:65:2b:d6:
    81:82:76:9d:4d:50:a9:09:c1:fb:af:1e:d9:c2:5b:
    00:9a:85:1f:f2:64:2f:d5:7a:1d:d7:60:0c:4e:94:
    f7:dc:55:cc:bb:28:9c:dc:7b
publicExponent: 65537 (0×10001)
privateExponent:
    00:c6:dd:5a:2f:76:fe:1a:ce:a3:81:69:de:61:89:
    23:75:f7:a2:f7:1d:17:4a:f0:65:ba:d9:81:de:2c:
    a9:9c:0c:39:38:47:df:9d:5f:40:72:e9:40:db:62:
    a9:71:de:f0:44:81:06:81:67:49:10:7b:8d:79:f2:
    d5:47:a5:87:b5:1b:66:68:2f:e8:c9:82:3d:0b:38:
    c9:03:49:fa:6e:bc:ea:6c:96:7a:f3:0a:a7:89:92:
    78:c3:40:a6:05:83:44:dc:5c:3e:4e:fa:65:d2:d9:
    6b:d7:e5:52:dc:f8:0d:55:98:f2:6d:f7:44:61:07:
    b1:87:c4:a4:c9:36:a3:65:41
prime1:
    00:f2:59:68:a0:2a:02:c4:8c:e5:e6:12:6f:39:2f:
    c9:5b:15:2d:7e:6a:af:dd:34:ee:2e:7d:ac:0e:64:
    30:96:96:cf:91:37:15:12:33:a1:ce:a5:0e:c6:13:
    eb:d1:5b:28:e8:10:52:a5:18:84:a1:c6:23:fb:2d:
    ab:fc:5c:81:e9
prime2:
    00:d5:eb:c6:0a:c0:6c:c5:d5:08:25:80:3e:88:3b:
```

- Extraire la clé publique de la clé privée et sauvegarder le résultat dans le fichier **rsapubkey.pem**

```
┌──(kali㉿kali)-[~]
└─$ openssl rsa -in rsakey.pem -pubout -out rsapubkey.pem

writing RSA key
```

Chiffrement de la clé RSA par l'algorithme

Nous allons maintenant utiliser l'algorithme AES256 pour chiffrer la clé privée.

Ecrire la commande qui permet de chiffrer le fichier **rsakey.pem** et produit ainsiun fichier **rsakeyencaes.pem.**

```
┌──(kali㉿kali)-[~]
└─$ openssl enc -aes256 -in rsakey.pem -out rsakeyencaes.pem -pbkdf2

enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:
```

-pbkdf2 : Cette option utilise PBKDF2 pour la dérivation de clé, ce qui renforce la sécurité de votre chiffrement.

Chiffrement/déchiffrement de données avec RSA

- Ecrire la commande qui permet de chiffrer le fichier initial **fichier_nom_eleve** avec la clé publique **rsapubkey.pem** et produit ainsi un fichier **fichier_nom_eleve.rsaenc** *(utiliser l'instruction openssl rsautl).*

```
┌──(kali㉿kali)-[~]
└─$ openssl pkeyutl -pubin -inkey rsapubkey.pem -in fichier_wassim -encrypt -
out fichier_chiff_rsa
```

Ecrire la commande qui permet de déchiffrer le fichier **fichier_nom_eleve.rsaenc** et produit ainsi un fichier **fichier_nom_eleve.rsadec.**

```
┌──(kali㉿kali)-[~]
└─$ openssl pkeyutl -inkey rsakey.pem -in fichier_chiff_rsa -decrypt -out fic
hier_dechiff_rsa
```

3. SIGNATURE NUMERIQUE

Génération d'une empreinte d'un fichier

- Calculer la valeur de l'empreinte du fichier **fichier_nom_eleve** avec l'algorithme MD5 et la mettre dans un fichier **fichier_nom_eleve.md5**.
- Quelle est la taille de cette empreinte ?

```
┌──(kali㉿kali)-[~]
└─$ openssl dgst -md5 -out fichier_wassim.md5 fichier_wassim

┌──(kali㉿kali)-[~]
└─$ cat fichier_wassim.md5

MD5(fichier_wassim)= 3fbfe293196c4a9aad00f18d67df298b
```

- Calculer la valeur de l'empreinte du même fichier avec l'algorithme SHA1 et la mettre dans un fichier **fichier_nom_eleve.sha1**.

```
┌──(kali㉿kali)-[~]
└─$ openssl dgst -sha1 -out fichier_wassim.sha1 fichier_wassim
```

- Quelle est la taille de cette empreinte ?

```
┌──(kali㉿kali)-[~]
└─$ openssl dgst -sha1 fichier_wassim
SHA1(fichier_wassim)= a6a745c968a7c30f9364cf20f70c74cf8447854e
```

La taille de l'empreinte d'un hachage dépend de l'algorithme utilisé. Pour le cas de SHA-1, l'empreinte produite est de 160 bits, ce qui équivaut à 20 octets. Voici comment on obtient cette information :

Calcul de la taille de l'empreinte

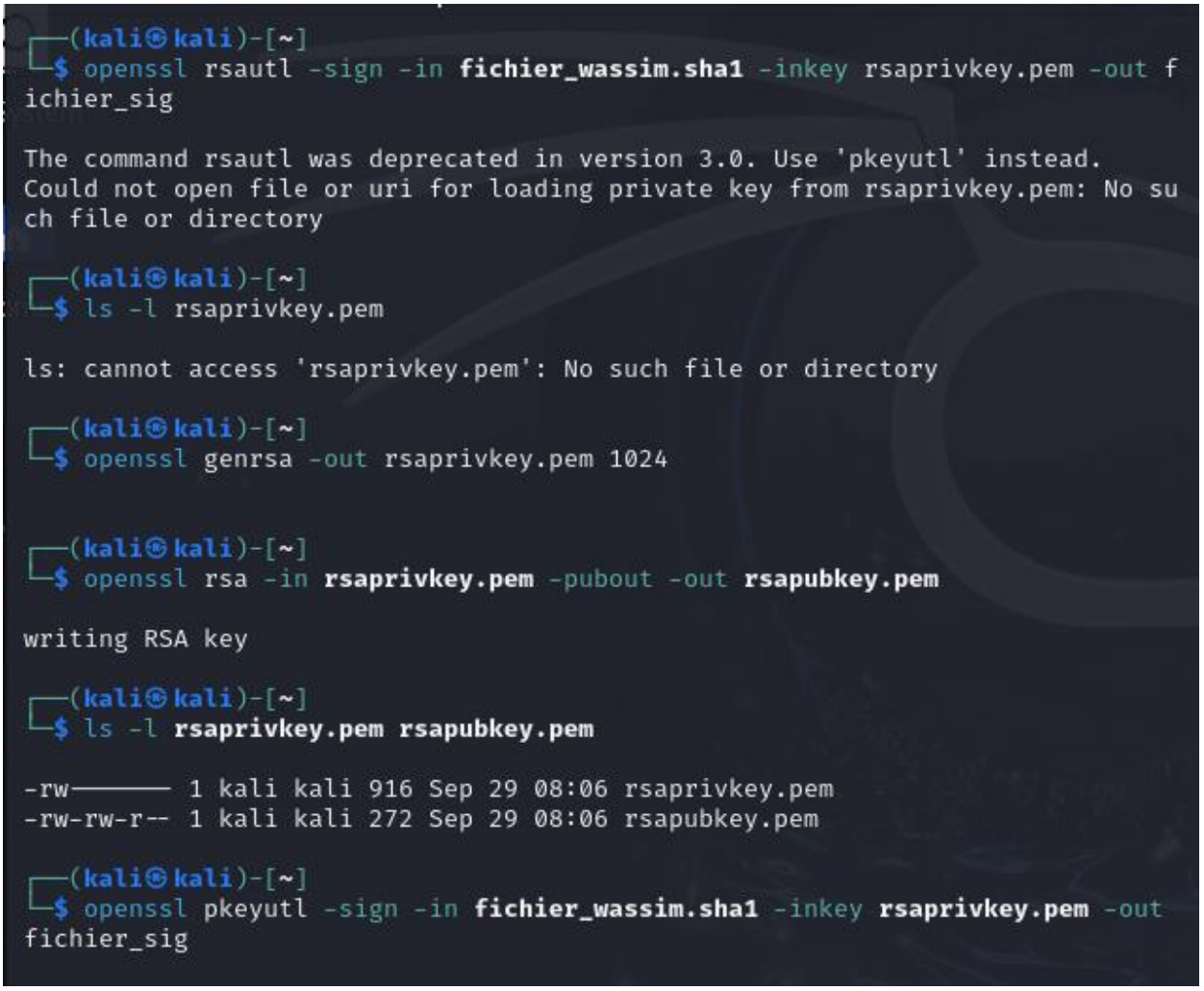
1. **Bits et octets** :
   - 1 octet = 8 bits.
   - Pour SHA-1, la taille de l'empreinte est spécifiée comme étant 160 bits.
2. **Conversion en octets** :
   - Pour convertir des bits en octets, vous divisez le nombre de bits par 8.
   - Donc, pour SHA-1 : Taille en octets=160 bits8=20 octets\text{Taille en octets} = \frac{160 \text{ bits}}{8} = 20 \text{ octets}Taille en octets=8160 bits=20 octets

- Comparer le résultat des deux fonctions de hachage. Qu'est-ce que vous remarquez ?

Les deux empreintes sont différentes, car MD5 et SHA1 utilisent des algorithmes différents

Signature d'un fichier

Signer un document revient à signer son empreinte. L'instruction à utiliser dans ce cas est :



1 - Générez une clé privée RSA.

2-Générez la clé publique correspondante.

3-Utilisez la clé privée pour signer votre empreinte.

4-Vérifiez la création de la signature.

- Signer le fichier **fichier_nom_eleve.sha1** et mettre de résultat le fichier **fichier_sig**. Dans ce cas, quel est la clé que vous devez utiliser pour signer ?
- Il reste ensuite à vérifier que l'empreinte ainsi produite dans le fichier **fichier_nom_eleve.sha1** est la même que l'on peut calculer. Utiliser l'instruction *openssl rsautl -verify*.

#openssl rsautl –verify –in fichier_sig -out empreinte2 –pubin – inkey rsapubkey.pem

```
┌──(kali㉿kali)-[~]
└─$ openssl pkeyutl -verify -in fichier_sig -inkey rsapubkey.pem -out emprein
te2

Could not find private key from rsapubkey.pem
401764430D7F0000:error:1608010C:STORE routines:ossl_store_handle_load_result:
unsupported:../crypto/store/store_result.c:151:
401764430D7F0000:error:1608010C:STORE routines:ossl_store_handle_load_result:
unsupported:../crypto/store/store_result.c:151:
pkeyutl: Error initializing context
```

4. CERTIFICAT NUMERIQUE

Génération de la clé privée

- Générer une paire de clés RSA de taille 1024 bits et stocker le résultat dans le fichier **server_cle.pem**

```
┌──(kali㉿kali)-[~]
└─$ openssl genrsa -out server_cle.pem 1024
```

Génération d'une requête de création d'un certificat

Créer un fichier de demande de signature de certificat (CSR Certificate Signing Request) *:*

```
┌──(kali㉿kali)-[~]
└─$ openssl req -new -key server_cle.pem -out server_cert.pem

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) [AU]:TN
State or Province Name (full name) [Some-State]:Tunis
Locality Name (eg, city) []:Tunis
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Esprit
Organizational Unit Name (eg, section) []:Esprit
Common Name (e.g. server FQDN or YOUR name) []:Wassim
Email Address []:wessharbaoui@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:kali
An optional company name []:esprit
```

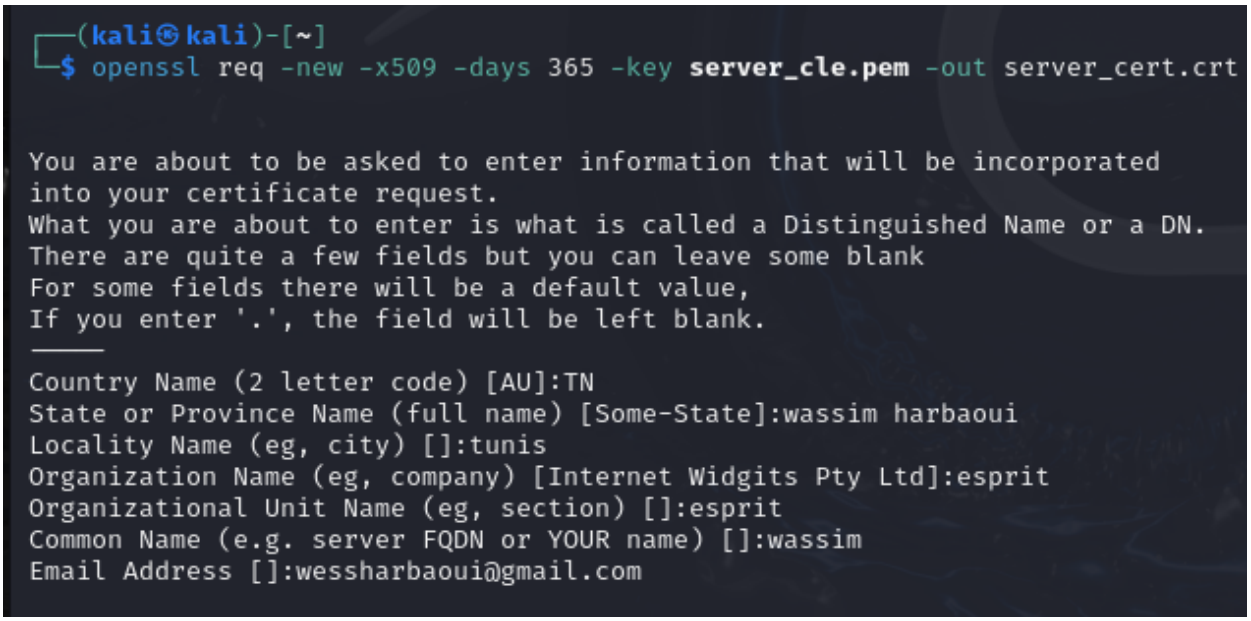
Signature du certificat

Afin de signer le certificat deux possibilités sont offertes :

- Auto signer le certificat
- Signer le certificat par une autorité de certification (AC)

Auto signature d'un certificat

- Signer le fichier **server.cert** à l'aide de la clé privée contenant dans le fichier **server_cle.pem** et stocker le résultat dans le fichier **server_cert.crt**. Le certificat doit avoir une période de validité d'un an.



- Afficher le contenu du certificat en format texte *:*

```
┌──(kali㉿kali)-[~]
└─$ openssl x509 -in server_cert.crt -text -noout

Certificate:
    Data:
        Version: 3 (0×2)
        Serial Number:
            05:1f:d7:d8:4f:3e:50:b1:9a:c4:4f:d7:5c:48:0a:5c:0d:3f:49:35
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=TN, ST=wassim harbaoui, L=tunis, O=esprit, OU=esprit, CN=w
ssim, emailAddress=wessharbaoui@gmail.com
        Validity
            Not Before: Sep 29 12:17:54 2024 GMT
            Not After : Sep 29 12:17:54 2025 GMT
        Subject: C=TN, ST=wassim harbaoui, L=tunis, O=esprit, OU=esprit, CN=
assim, emailAddress=wessharbaoui@gmail.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:9d:39:c0:70:af:22:18:d1:ed:33:76:ae:b3:b2:
                    45:6c:d9:41:14:0e:67:a9:a1:ce:ea:6c:8a:bd:52:
                    4a:16:90:db:38:ba:46:ff:72:35:e7:d6:fd:36:e6:
                    9b:1b:8c:31:d3:3a:f4:76:8e:0c:07:2d:79:42:d0:
                    e0:23:1a:8c:a1:ab:36:9a:3d:4e:96:79:d8:49:43:
                    d4:98:dc:6d:e5:8b:d4:f4:f1:2a:70:70:c1:52:62:
                    ab:31:93:98:17:b7:f5:a1:37:f5:b7:7f:35:85:74:
                    10:ef:81:3d:3e:ec:51:69:83:f9:fa:f2:14:84:33:
                    7e:78:c0:e3:b5:90:39:6e:29
                Exponent: 65537 (0×10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                46:BF:FA:AA:7D:B1:BF:A2:3D:F5:32:7C:DC:97:6F:CC:2D:68:30:E8
            X509v3 Authority Key Identifier:
                46:BF:FA:AA:7D:B1:BF:A2:3D:F5:32:7C:DC:97:6F:CC:2D:68:30:E8
            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        99:c5:5c:fd:2b:cb:b5:1f:82:cf:e8:84:28:dc:24:02:4f:60:
```

Signature par une autorité de certification (AC)

- La première étape consiste à générer une clé privée RSA pour l'AC de taille 2048 bits et de stocker le résultat dans le fichier **cakey.pem**

```
┌──(kali㉿kali)-[~]
└─$ openssl genrsa -out cakey.pem 2048
```

- Générer un certificat pour l'AC ayant une période de validité 730 jours et stocker le résultat dans le fichier **ca.crt**.

```
┌──(kali㉿kali)-[~]
└─$ openssl req -new -x509 -days 730 -key cakey.pem -out ca.crt

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) [AU]:tn
State or Province Name (full name) [Some-State]:wassim harbaoui
Locality Name (eg, city) []:tunis
Organization Name (eg, company) [Internet Widgits Pty Ltd]:esprit
Organizational Unit Name (eg, section) []:esprit
Common Name (e.g. server FQDN or YOUR name) []:wassim
Email Address []:wessharbaoui@gmail.com
```

Rq: ca.crt est le certificat auto signé de l'autorité de certification qui va permettre de signer les certificats créés.

- Signer la demande du certificat du serveur (le fichier **server.csr**) par l'autorité de certification AC en utilisant l'instruction suivante :

```
┌──(kali㉿kali)-[~]
└─$ openssl x509 -req -in server_cert.pem -out server.crt -CA ca.crt -CAkey c
akey.pem -CAcreateserial -CAserial ca.srl

Certificate request self-signature ok
subject=C=TN, ST=Tunis, L=Tunis, O=Esprit, OU=Esprit, CN=Wassim, emailAddress
=wessharbaoui@gmail.com
```

Rq : Le certificat signé est le fichier **ca.srt**