Cybersécurité - TP 1

GIBOZ Alexandre, MAURICE Romain, AINOUZ Nicolas

INFO1 2022-2025

Exercice 1 - Empreintes numériques (en anglais, digest ou hash)

1. **En utilisant la commande openssl list -digest-algorithms ou openssl list -digest-commands, déterminer les primitives de hachage supportées.**

Les primitives de hachage supportées sont les suivantes :

```
RSA-MD4 => MD4
RSA-MD5 => MD5
RSA-RIPEMD160 => RIPEMD160
RSA-SHA1 => SHA1
RSA-SHA1-2 => RSA-SHA1
RSA-SHA224 => SHA224
RSA-SHA256 => SHA256
RSA-SHA3-224 => SHA3-224
RSA-SHA3-256 => SHA3-256
RSA-SHA3-384 => SHA3-384
RSA-SHA3-512 => SHA3-512
RSA-SHA384 => SHA384
RSA-SHA512 => SHA512
RSA-SHA512/224 => SHA512-224
RSA-SHA512/256 => SHA512-256
RSA-SM3 => SM3
BLAKE2b512
BLAKE2s256
id-rsassa-pkcs1-v1_5-with-sha3-224 => SHA3-224
id-rsassa-pkcs1-v1_5-with-sha3-256 => SHA3-256
id-rsassa-pkcs1-v1_5-with-sha3-384 => SHA3-384
id-rsassa-pkcs1-v1_5-with-sha3-512 => SHA3-512
MD4
md4WithRSAEncryption => MD4
MD5
MD5-SHA1
md5WithRSAEncryption => MD5
ripemd => RIPEMD160
RIPEMD160
ripemd160WithRSA => RIPEMD160
rmd160 => RIPEMD160
SHA1
sha1WithRSAEncryption => SHA1
SHA224
sha224WithRSAEncryption => SHA224
SHA256
```

```
sha256WithRSAEncryption => SHA256
SHA3-224
SHA3-256
SHA3-384
SHA3-512
SHA384
sha384WithRSAEncryption => SHA384
SHA512
SHA512-224
sha512-224WithRSAEncryption => SHA512-224
SHA512-256
sha512-256WithRSAEncryption => SHA512-256
sha512WithRSAEncryption => SHA512
SHAKE128
SHAKE256
SM3
sm3WithRSAEncryption => SM3
ssl3-md5 => MD5
ssl3-sha1 => SHA1
whirlpool
```

2. **Créer un Fichier texte contenant un petit paragraphe (toto.txt), et donner les empreintes numériques obtenues avec les primitives : md5, sha1, sha256, sha512. La commande à utiliser est : openssl dgst - -out <Fichier qui va stocker l'empreinte> <Fichier à hasher>.**

On créer un fichier contenant un petit paragraphe :

```
echo "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec euismod, nisl eget ultricies ultrices.
```

On récupère les empreintes numériques :

```
openssl dgst -md5 -out toto_md5.txt toto.txt
openssl dgst -sha1 -out toto_sha1.txt toto.txt
openssl dgst -sha256 -out toto_sha256.txt toto.txt
openssl dgst -sha512 -out toto_sha512.txt toto.txt
```

Les résultats sont les suivants (dans l'ordre) :

```
MD5(toto.txt)= 32bcb585dd3d0bb3ac979d09ecb5a0c9
SHA1(toto.txt)= 12fe55b03216d59a362c0c961a2d4b75f830a2bb
SHA2-256(toto.txt)= 89623168d1165956b148d5ec9cd39cad5e022b109d5fa5472b9a6e04bab5301d
SHA2-512(toto.txt)= 28d4a43d704c51a179dc19db1f0eded9198e57154aba86687ec728b8787fd72c9ee774d72793990c61a4931
```

3. **Donner la taille de chaque empreinte (en bits).**

Les tailles des empreintes sont les suivantes :

```
MD5(toto.txt)= 128 bits
SHA1(toto.txt)= 160 bits
SHA2-256(toto.txt)= 256 bits
SHA2-512(toto.txt)= 512 bits
```

4. **Modifier un caractère dans le Fichier texte, et calculer à nouveau les empreintes numériques. Que constatez-vous ? Quelle conclusion ?**

On modifie le fichier :

```
sed -i 's/L/l/' toto.txt
```

Les nouvelles empreintes numériques sont les suivantes :

```
MD5(toto.txt)= 7867f7eb0bf3d4dc827f210588855786
SHA1(toto.txt)= 3da409e50ff4cca900ba595a9eca1ec1e833542a
SHA2-256(toto.txt)= 1db41cb459acfd9df9a7353a7a114c96fa801f6d55ffd78801c4642d7ddc32ce
SHA2-512(toto.txt)= c3c4ddc71fdc823016c72f93467076199868d0903115227a2b0a8ba2da993f6f14a483f9dad863638ac3d8e
```

On constate que les empreintes numériques sont totalement différentes, alors que le fichier n'a été modifié que d'un seul caractère.

Cela démontre tout l'intérêt du hashage. A condition d'utiliser un algorithme sécurisé et robuste (sans collisions aucune), il permet de vérifier l'intégrité de fichiers. Ce contrôle d'intégrité est très utilisé, que ça soit pour du téléchargement ou des bases de données de hash d'antivirus.

Le hashage est supposé être unilatéral, c'est à dire qu'il est impossible de retrouver le fichier d'origine à partir de son empreinte numérique. Cependant, cela peut être déjoué avec des dictionnaires/rainbow tables. Dans le cadre de crack de mots de passe (ou l'on possède un hash d'un mot de passe à trouver en clair), on hash chaque mot de passe dans un dictionnaire avec le même algorithme. Si les deux hash sont identiques, alors on a trouvé le mot de passe.

5. **L'éditeur peut mettre à disposition une empreinte numérique comme une sorte de checksum pour vous permettre de vérifier l'intégrité d'un Fichier téléchargé (le Fichier téléchargé est bien celui attendu et qu'il n'ait pas été altéré en cours de route)** a) **Télécharger un petit Fichier (non volumineux) et son empreintes SHA256 en utilisant par exemple le lien : https://archive.apache.org/dist/openofice/4.1.3/binaries/SDK/**

Le lien ne fonctionne plus, je vais donc télécharger le gestionnaire de dépendance "Composer" à l'adresse suivante: "https://getcomposer.org/download/"

On télécharge le fichier et son empreinte SHA256 :

```
wget https://getcomposer.org/download/latest-stable/composer.phar
wget https://getcomposer.org/download/latest-stable/composer.phar.sha256
```

b) **Comparer l'empreinte de l'éditeur à celle calculée par openSSL.**

L'empreinte dans le fichier "composer.phar.sha256" est la suivante :

```
9256c4c1c803b9d0cb7a66a1ab6c737e48c43cc6df7b8ec9ec2497a724bf44de
```

On calcule l'empreinte SHA256 du fichier "composer.phar" téléchargé :

```
openssl dgst -sha256 composer.phar
```

On obtient le résultat suivant :

```
SHA2-256(composer.phar)= 9256c4c1c803b9d0cb7a66a1ab6c737e48c43cc6df7b8ec9ec2497a724bf44de
```

c) **Quelle conclusion ?**

Les hash sont les même. On peut donc en déduire que le fichier téléchargé est bien le bon.

Comme mentionné précédemment, le hashage est très utile pour vérifier l'intégrité d'un fichier téléchargé.

Exercice 2 - Chiffrement symétrique

1. **En utilisant la commande openssl list -cipher-commands, déterminer les algorithmes de chiffrement et les modes associés supportés par openssl.**

Les algorithmes de chiffrement et les modes associés supportés par openssl sont les suivants :

```
aes-128-cbc       aes-128-ecb       aes-192-cbc       aes-192-ecb
aes-256-cbc       aes-256-ecb       aria-128-cbc      aria-128-cfb
aria-128-cfb1     aria-128-cfb8     aria-128-ctr      aria-128-ecb
aria-128-ofb      aria-192-cbc      aria-192-cfb      aria-192-cfb1
aria-192-cfb8     aria-192-ctr      aria-192-ecb      aria-192-ofb
aria-256-cbc      aria-256-cfb      aria-256-cfb1     aria-256-cfb8
aria-256-ctr      aria-256-ecb      aria-256-ofb      bf
bf-cbc            bf-cfb            bf-ecb            bf-ofb
camellia-128-cbc  camellia-128-ecb  camellia-192-cbc  camellia-192-ecb
camellia-256-cbc  camellia-256-ecb  cast              cast-cbc
cast5-cbc         cast5-cfb         cast5-ecb         cast5-ofb
des               des-cbc           des-cfb           des-ecb
des-ede           des-ede-cbc       des-ede-cfb       des-ede-ofb
des-ede3          des-ede3-cbc      des-ede3-cfb      des-ede3-ofb
des-ofb           des3              desx              rc2
rc2-40-cbc        rc2-64-cbc        rc2-cbc           rc2-cfb
rc2-ecb           rc2-ofb           rc4               rc4-40
seed              seed-cbc          seed-cfb          seed-ecb
seed-ofb          sm4-cbc           sm4-cfb           sm4-ctr
sm4-ecb           sm4-ofb
```

2. **Classer ces algorithmes en algorithmes par bloc et par flot**

Les algorithmes par bloc sont les suivants :

```
AES (Advanced Encryption Standard)
DES (Data Encryption Standard)
3DES (Triple Data Encryption Standard)
Camellia
IDEA (International Data Encryption Algorithm)
CAST5 (CAST-128)
SEED (KISA SEED)
```

```
BF (Blowfish)
RC2 (Rivest Cipher 2)
RC4 (Rivest Cipher 4)
RC5 (Rivest Cipher 5)
```

Les algorithmes par flot sont les suivants :

```
ChaCha20
Salsa20
```

3. **Les chiffrement symétrique a besoin d'une clef partagée. Cette clef peut être générée par un mot de passe ou donnée de manière explicite.** a) **Chiffrer le Fichier créé précédemment (toto.txt) en toto.txt.enc en utilisant l'algorithme AES avec une taille de clef de 128bits générée par mot de passe et le mode CBC. La commande à utiliser est : openssl enc -aes.... -p. L'option -p à la finn permet d'afficher plus de détails.**

On chiffre le fichier "toto.txt" :

```
openssl enc -aes-128-cbc -in toto.txt -out toto.txt.enc -p
```

On obtient par la suite un scanner qui nous demande de spécifier un mot de passe. On entre le mot de passe "away".

Les données suivantes nous sont retournées:

```
salt=BD22A34116F41C82
key=2F39440CE407BA1C68F0FD3848A58EFB
iv =3267F3CA7752ABD5AA9260E8AAC01911
```

b) **Qu'est ce que le salt ? expliquer son rôle ? comment il est généré ? comment il est stocké ?**

Le salt est une suite de caractères aléatoires qui est ajoutée au mot de passe avant le hashage. Il ajoute de l'entropie au mot de passe.

Il permet d'éviter les attaques par dictionnaire et par force brute.

Il est généré de manière aléatoire par le programme (openssl).

Il est, en général, stocké en association avec les données hachées ou dérivées.

Dans notre cas, le salt est `BD22A34116F41C82` .

c) **Quelles sont la clef, l'IV et leurs tailles générées automatiquement par le mot de passe ?**

La clef est `2F39440CE407BA1C68F0FD3848A58EFB` et le vecteur d'initialisation est `3267F3CA7752ABD5AA9260E8AAC01911` .

Leurs tailles respectives sont de 128 bits (AES).

d) **Déchiffrer le fichier toto.txt.enc en toto.dec en utilisant les trois méthodes : via le mot de passe à taper suite à la commande, en précisant le salt et le mot de passe comme argument, ou en précisant l'IV et la clef**

comme argument. La commande à utiliser est : openssl enc -d -aes ....

On déchiffre le fichier "toto.txt.enc" en utilisant le mot de passe :

```
openssl enc -d -aes-128-cbc -in toto.txt.enc -out toto.dec -p
```

On déchiffre le fichier "toto.txt.enc" en utilisant le salt et le mot de passe :

```
openssl enc -d -aes-128-cbc -in toto.txt.enc -out toto.dec -p -S BD22A34116F41C82
```

On déchiffre le fichier "toto.txt.enc" en utilisant l'IV et la clef :

```
openssl enc -d -aes-128-cbc -in toto.txt.enc -out toto.dec -p -K 2F39440CE407BA1C68F0FD3848A58EFB -iv 3267F
```

e) **Comparer les deux fichiers toto.txt et toto.dec avec la commande : diff toto.txt toto.dec. Quelle conclusion ?**

On compare les fichiers "toto.txt" et "toto.dec" :

```
diff toto.txt toto.dec
```

La commande ne retourne rien, ce qui signifie que les deux fichiers sont identiques.

On peut en conclure que le chiffrement et le déchiffrement ont bien fonctionné, et que l'intégrité à été préservée, car le fichier avant et après chiffrement sont identiques.

f) **Tenter de déchiffrer le fichier toto.enc en utilisant un mauvais mot de passe. Comment réagit openSSL?**

On déchiffre le fichier "toto.txt.enc" en utilisant le mot de passe "esipe" (mauvais mot de passe) :

```
openssl enc -d -aes-128-cbc -in toto.txt.enc -out toto.dec -p
```

Le résultat suivant est retourné par la commande :

```
salt=BD22A34116F41C82
key=12DAE481E21A39BDE09AD65227772202
iv =023A3F861C061C67DDBD3F130AF5B1F8
bad decrypt
4057CFD4577F0000:error:1C800064:Provider routines:ossl_cipher_unpadblock:bad decrypt:../providers/implement
```

Si l'on ouvre le fichier "toto.dec", on constate que le fichier est rempli de caractères aléatoires:

```
��N厄�O��%␖y⍰t�>i;x/<��j�W8y&,���kX���h�f
```

g) **Effectuer le chiffrement et le déchiffrement du fichier toto.txt en utilisant une clef explicite de 128 bits exprimée par 32 chiffres hexa et un vecteur d'initialisation de 128 bits exprimé par 32 chiffres hexa.**

On chiffre le fichier "toto.txt" :

```
openssl enc -aes-128-cbc -K 0123456789abcdef0123456789abcdef -iv 0123456789abcdef0123456789abcdef -in toto.
```

On déchiffre le fichier "toto_encrypted.txt" :

```
openssl enc -d -aes-128-cbc -K 0123456789abcdef0123456789abcdef -iv 0123456789abcdef0123456789abcdef -in to
```

On compare le contenu de "toto.txt" et "toto_decrypted.txt" :

```
diff toto.txt toto_decrypted.txt
```

```
┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl enc -aes-128-cbc -K 0123456789abcdef0123456789abcdef -iv 0123456789abcdef0123456789abcdef -in toto.txt -out toto_encrypted.txt

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl enc -d -aes-128-cbc -K 0123456789abcdef0123456789abcdef -iv 0123456789abcdef0123456789abcdef -in toto_encrypted.txt -out toto_decrypted.txt

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ tail toto_decrypted.txt
lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec euismod, nisl eget ultricies ultrices.

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ diff toto.txt toto_decrypted.txt
```

La commande ne retourne rien, ce qui signifie que les deux fichiers sont identiques.

h) **Tester le chiffrement et le déchiffrement d'une image**

On chiffre l'image suivante. Le mot de passe est "away":

```
openssl enc -aes-128-cbc -in img_to_encrypt.jpeg -out img_encrypted.jpeg
```

On déchiffre l'image:

```
openssl enc -d -aes-128-cbc -in img_encrypted.jpeg -out img_decrypted.jpeg
```

On compare les hash des deux images:

```
md5sum img_to_encrypt.jpeg
md5sum img_decrypted.jpeg
```

Les hash sont identiques, ce qui signifie que l'intégrité de l'image est préservée.

```
┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl enc -aes-128-cbc -in img_to_encrypt.jpeg -out img_encrypted.jpeg

enter AES-128-CBC encryption password:
Verifying - enter AES-128-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl enc -d -aes-128-cbc -in img_encrypted.jpeg -out img_decrypted.jpeg

enter AES-128-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ md5sum img_to_encrypt.jpeg
md5sum img_decrypted.jpeg
1f18f6795bc3be470644fa059225c2bb  img_to_encrypt.jpeg
1f18f6795bc3be470644fa059225c2bb  img_decrypted.jpeg
```

Exercice 3 : Chiffrement asymétrique RSA

1. **Générer une clef privée de module de taille 1024 bits avec la commande : openssl genrsa -out maClef.pem 1024, et visualiser le contenu du fichier ainsi généré (cat maClef.pem).**

On génère une clef privée de module de taille 1024 bits :

```
openssl genrsa -out maClef.pem 1024
```

2. **Afficher les paramètres de la clef privée RSA avec la commande openssl rsa -in maClef.pem -text -noout, et commenter le résultat.**

On affiche les paramètres de la clef privée RSA :

```
└─$ openssl rsa -in maClef.pem -text -noout
Private-Key: (1024 bit, 2 primes)
modulus:
    00:9e:24:44:51:c3:9e:ad:f8:6c:38:43:66:db:3e:
    54:a4:61:fe:3d:c6:7d:e1:26:98:89:0a:de:26:75:
    a1:fc:f2:d5:9b:18:13:18:b1:f2:da:a0:5c:f4:99:
    13:f2:81:e6:2a:da:69:f3:e6:23:aa:bb:52:8e:f0:
    a4:31:6f:97:fa:14:c7:3e:64:f0:80:32:1f:b0:ea:
    9e:e3:3f:1f:0c:0b:c3:8d:fe:94:28:63:29:cc:65:
    c0:48:7b:18:f0:6b:0e:90:c1:d2:aa:76:2c:dd:4c:
    6c:00:30:8e:10:12:aa:e1:c6:30:c7:4e:c7:47:de:
    ba:a1:55:1a:84:27:0c:23:e5
publicExponent: 65537 (0x10001)
privateExponent:
    4a:7a:48:16:bd:eb:70:81:8b:ee:1f:88:44:7f:21:
    7b:dd:8e:d0:67:38:07:9e:96:be:0b:f9:3d:e1:ae:
    ad:a3:6c:08:44:19:52:4b:14:55:f4:aa:72:a3:d9:
    6a:1f:d5:57:9f:4f:c1:07:45:0a:f6:77:d6:4b:ec:
    f8:3f:57:32:39:a5:b7:79:8e:0a:66:f3:bf:71:d5:
    3e:6f:a9:0f:1c:d0:f0:a6:d0:bb:ed:32:3e:b8:77:
    6d:93:c8:6a:b1:e9:20:88:01:52:b0:5b:15:5f:d4:
    4c:ed:bb:04:11:ca:3e:75:da:9c:13:04:fa:6f:8f:
    a0:7e:92:11:fd:4f:03:e1
prime1:
    00:cc:4a:0c:db:56:9d:ef:4f:e5:b0:c2:a1:52:c1:
    90:db:02:2e:5f:79:41:ee:2d:4e:dd:e5:c0:56:8b:
    76:a8:c8:4a:14:34:a7:40:d8:c7:9b:80:6d:b3:0f:
    1d:b3:13:41:f8:74:ff:3c:3f:de:ad:04:d3:4f:5c:
    e0:7f:33:ce:7d
prime2:
    00:c6:2b:d9:f7:1a:dd:f6:f3:ab:d3:e4:ee:55:d6:
    94:04:17:3b:25:4d:41:a0:92:20:5d:4f:c0:ce:91:
    1f:00:4e:e5:fb:c6:48:88:a5:3a:9e:16:39:12:03:
    13:25:34:f8:9b:d1:8b:cb:17:38:52:03:f7:fd:46:
    a9:b0:cf:9f:89
exponent1:
    00:b3:70:73:bc:91:67:66:56:1a:9d:d9:47:54:66:
    7b:73:33:86:42:0c:43:52:0e:f0:10:4e:87:54:3d:
    69:fb:b4:fa:04:a7:7f:25:a1:84:2a:dd:72:fd:ed:
    d0:1b:84:55:d5:04:07:51:07:94:f3:0a:a3:05:39:
    c2:63:58:c1:0d
exponent2:
    00:c5:dc:1e:a2:23:c7:1e:bc:28:01:46:7f:d8:60:
    88:24:95:75:4e:47:16:91:55:94:ce:d5:c4:31:fa:
    9f:33:55:93:36:90:c0:f6:fd:d1:ca:e9:71:4d:d4:
    98:16:fc:0e:84:b4:f1:98:72:b4:9e:de:ab:89:cc:
    35:bc:fd:6f:d1
coefficient:
    00:b9:19:08:cc:a2:dc:db:fa:4c:39:7a:8f:4e:bc:
    59:25:5e:85:4c:8c:c3:24:bd:04:0e:87:e3:90:e6:
    08:6a:e9:8b:12:90:cd:ba:ae:59:8e:24:2c:d9:3f:
```

La première ligne indique que la taille de la clef est de 1024 bits. "2 primes" indique qu'il y a deux facteurs premiers dans la clef.

Les différents modules sont affichés, représentant des valeurs dérivées des nombres premiers.

3. **Déterminer la clef publique, la clef privée et les deux nombres premiers.**

On détermine la clef publique avec la commande suivante :

```
openssl rsa -in maClef.pem -pubout -out maClef.pub
```

On obtient un fichier "maClef.pub" contenant la clef publique:

```
┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ cat maClef.pub
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCeJERRw56t+Gw4Q2bbPlSkYf49
xn3hJpiJCt4mdaH88tWbGBMYsfLaoFz0mRPygeYq2mnz5iOqu1KO8KQxb5f6FMc+
ZPCAMh+w6p7jPx8MC8ON/pQoYynMZcBIexjwaw6QwdKqdizdTGwAMI4QEqrhxjDH
TsdH3rqhVRqEJwwj5QIDAQAB
-----END PUBLIC KEY-----
```

La commande suivante permet d'obtenir la clef privée. Cependant, le fichier "maClef.pem" contient déjà la clef privée.

```
openssl rsa -in maClef.pem -out maClef.priv
```

```
┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ diff maClef.pem maClef.priv

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$
```

Les deux nombres premiers sont affichés dans le fichier "maClef.pem" :

```
openssl rsa -in maClef.pem -text -noout | awk '/prime1:/{getline; print} /prime2:/{getline; print}'
```

On obtient les valeurs suivantes:

```
00:cc:4a:0c:db:56:9d:ef:4f:e5:b0:c2:a1:52:c1: (= 217478085165881973593960816363007611137)
00:c6:2b:d9:f7:1a:dd:f6:f3:ab:d3:e4:ee:55:d6: (= 215216864580500029508825281394181271990)
```

4. **Il n'est pas prudent de stocker la clef privée en clair. Pour chiffrer le fichier maClef.pem, 3 algorithmes de chiffrement symétrique sont possibles : des, des3 et idea.** a) **Chiffrer le fichier maClef.pem avec la commande openssl rsa -in maClef.pem -des3 -out maClef.pem, et visualiser et commenter son impact.**

Une fois la commande entrée, un scanner est ouvert afin de demander un mot de passe. On choisit le mot de passe "away".

Le nouveau fichier "maClef.pem" est chiffré (différent de son contenu précédent):

```
┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl rsa -in maClef.pem -des3 -out maClef.pem
writing RSA key
Enter pass phrase:
Verifying - Enter pass phrase:

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ cat maClef.pem
-----BEGIN ENCRYPTED PRIVATE KEY-----
MIIC1DBOBgkqhkiG9w0BBQ0wQTApBgkqhkiG9w0BBQwwHAQI6k4LwmuJUXoCAggA
MAwGCCqGSIb3DQIJBQAwFAYIKoZIhvcNAwcECMBIVPJVFeOCBIICgH7LPqMej0Vx
4YZEiEaDVUlg3gzXdg4SUYLuwW2e6L3+iYFhZUxUZ3DH94tFZOa6rw3guqUWdW4J
OlTHYYFT0pXGB6BMlAr4E12EM6zkmBvIhP8EPyzEc1hZlsm5yzf94ArOH2S1AfRM
/nopWzTvHyKKUimF6YbjbmPUz76NGwBqWI2oAH79Ee7U2Spmm0Whfi9NAzvymW6k
OZPkvCvmzy1tnMys/EYFXpzcxxJ5vmJq1IxP9QlxnRW3XVTJHhCZ0E149bY0O0d+
YUvT0ixjzzvRvflcof8sEgZ2JXYqcFj2MD7DxdJQBTyHt3zeDxrJHIY3yWVRgqzH
3c3I8skImwQICy2PWloaqTChf0fhhDXcZwZUkmNQ4bm8nWlerbayFqvLjmlanADP
U4cfuVI4x+Kjkv1T4vDYQ1e+QvJdDR/NsvVT4P0fRWFigxuB6YXqazlaPW5dVBvl
jInaaNLu944lcBQtyRVxcL7BN5UqKrXl+tDAeV9LkFTk+WwjlfMPcSsX1+Q8X+L+
Xv+IJGV+GJJ14hnbE/zYRirB+Af+zIK0X0RDfs7kvJPNJ3udqnbbysrEVuxNq9AH
ZKCwvhP9rlaN4F3NUYtrxdOD3foxBwTh5LOIUteHPVF8cO1T3fyiRqdWfNGhA9dK
YTuJKt++7uxs8oBMBtif+I5rO+2TjbvbOa4iLl+rpnT1zUTcgboCCazxWW8m3+ot
BjkO761/3LZoGBzE1OhTFJ0yjmG6A2jdlbnbK3KkXb4/B7Vg7hRAdg2bxbGPlf3p
7kDYYU7SAzTZiwTwAxQ+QOM7HWQiXDGUhZ1jp4VxHasFKewOt7yKsZr6×0rBXQDK
VdepKXjeRD4=
-----END ENCRYPTED PRIVATE KEY-----
```

5. **Générer la clef publique RSA à partir de la clef privée avec la commande openssl rsa -in maClef.pem -pubout -out maClefPublique.pem, et commenter le contenu du fichier ainsi généré.**

Une fois le mot de passe entré, on obtient la même clef publique qu'auparavant:

```
┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl rsa -in maClef.pem -pubout -out maClefPublique.pem
Enter pass phrase for maClef.pem:
writing RSA key

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ diff maClef.pub maClefPublique.pem

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$
```

6. **Pour chiffrer avec une clef publique RSA, on utilise la commande : openssl rsautl -encrypt -pubin -inkey maClefPublique.pem -in -out . Pour déchiffrer, l'option -decrypt est utilisée.** a) **Effectuer le chiffrement et le déchiffrement du fichier toto.txt.**

On chiffre le fichier "toto.txt" avec la commande suivante :

```
openssl rsautl -encrypt -pubin -inkey maClefPublique.pem -in toto.txt -out toto_encrypted.txt
```

On déchiffre le fichier "toto_encrypted.txt" avec la commande suivante :

```
openssl rsautl -decrypt -inkey maClef.pem -in toto_encrypted.txt -out toto.dec
```

b) **Comparer les deux fichiers toto.txt et toto.dec.**

Les fichiers sont identiques. On a bien conservé l'intégrité de toto.txt:

```
┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl rsautl -decrypt -inkey maClef.pem -in toto_encrypted.txt -out toto.dec

The command rsautl was deprecated in version 3.0. Use 'pkeyutl' instead.
Enter pass phrase for maClef.pem:

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ diff toto.txt toto.dec

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$
```

c) **Avec la clef publique d'un autre groupe, chiffrer votre fichier toto.txt et envoyer le résultat à ce groupe pour le déchiffrer. Commenter le résultat.**

TODO

Exercice 4 : Signatures numériques

1. **En utilisant la commande openssl rsautl -sign -in <fichier où l'empreinte est stockée> -inkey <fichier de la clef privé> -out , générer la signature du fichier toto.txt.**

On génère la signature du fichier "toto.txt" avec la commande suivante :

```
openssl rsautl -sign -in toto.txt -inkey maClef.pem -out toto_signature.txt
```

2. **En utilisant la commande penssl rsautl -verify -in -pubin -inkey -out <fichier où l'empreinte est stockée>, vérifier la signature du fichier toto.txt.**

On vérifie la signature du fichier "toto.txt" avec la commande suivante :

```
openssl rsautl -verify -in toto_signature.txt -pubin -inkey maClefPublique.pem -out toto_signature_verified
```

On récupère bien le contenu du fichier "toto.txt" à l'identique:

```
"the quieter you become, the more you are able to hear"
┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl rsautl -verify -in toto_signature.txt -pubin -inkey maClefPublique.pem -out toto_signature_verified.txt

The command rsautl was deprecated in version 3.0. Use 'pkeyutl' instead.

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ diff toto.txt toto_signature_verified.txt

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$
```

Exercice 5 : Certificats - Projet

Dans ce projet, vous allez générer un fake serveur d'amazon sécurisé. Un fake certificat est nécessaire pour le fake serveur. Ce fake certificat sera signé par un fake CA.

```
┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl genrsa 2048 > servtest.pem

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl req -new -key servtest.pem > servtest.csr
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]:FR
State or Province Name (full name) [Some-State]:Paris
Locality Name (eg, city) []:Paris
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ESIPE
Organizational Unit Name (eg, section) []:amazon.com
Common Name (e.g. server FQDN or YOUR name) []:amazon.com
Email Address []:admin@amazon.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:esipe123
An optional company name []:amazon.com

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl genrsa -des3 2048 > ca.pem
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

```
┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl genrsa -des3 2048 > ca.pem
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl req -new -x509 -days 365 -key ca.pem > ca.crt

Enter pass phrase for ca.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]:FR
State or Province Name (full name) [Some-State]:Paris
Locality Name (eg, city) []:Paris
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ESIPE
Organizational Unit Name (eg, section) []:EISC
Common Name (e.g. server FQDN or YOUR name) []:cert_CA
Email Address []:admin@cert_CA.fr

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ openssl x509 -req -in servtest.csr -out servtest.crt -CA ca.crt -CAkey ca.pem -CAcreateserial -CAserial ca.srl

Certificate request self-signature ok
subject=C = FR, ST = Paris, L = Paris, O = ESIPE, OU = amazon.com, CN = amazon.com, emailAddress = admin@amazon.com
Enter pass phrase for ca.pem:
```

On modifie le fichier de configuration "sites-available":

```
        # 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

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        # For most configuration files from conf-available/, which are
        # enabled or disabled at a global level, it is possible to
        # include a line for only one particular virtual host. For example the
        # following line enables the CGI configuration for this host only
        # after it has been globally disabled with "a2disconf".
        #Include conf-available/serve-cgi-bin.conf

ServerName amazon.com
Redirect "/" "https://amazon.com/"

</VirtualHost>
```

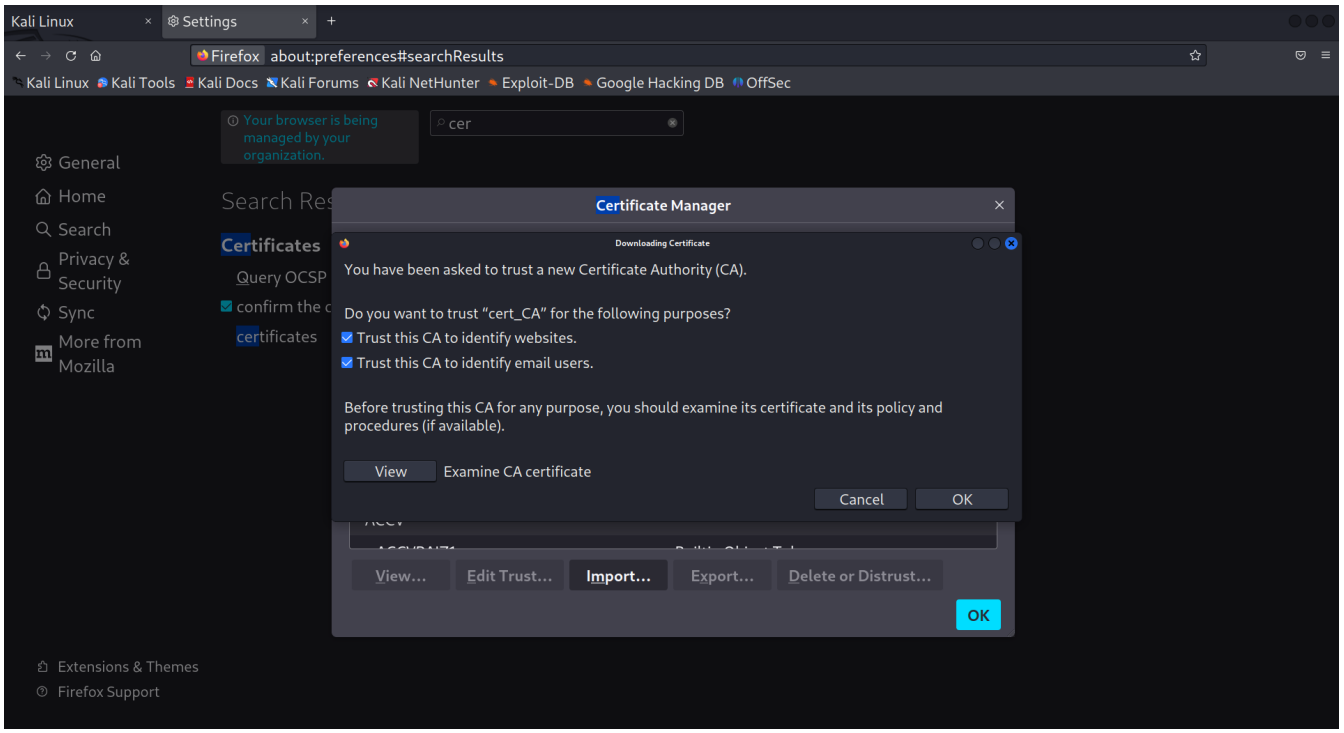
On modifier le fichier de configuration "default-ssl.conf":

```
        #    this only for browsers where you know that their SSL implementation
        #    works correctly.
        #   Notice: Most problems of broken clients are also related to the HTTP
        #   keep-alive facility, so you usually additionally want to disable
        #   keep-alive for those clients, too. Use variable "nokeepalive" for this.
        #   Similarly, one has to force some clients to use HTTP/1.0 to workaround
        #   their broken HTTP/1.1 implementation. Use variables "downgrade-1.0" and
        #   "force-response-1.0" for this.
        BrowserMatch "MSIE [2-6]" \
                nokeepalive ssl-unclean-shutdown \
                downgrade-1.0 force-response-1.0
        ServerName amazon.com

        SSLEngine on

        SSLCertificateFile      /etc/ssl/certs/servtest.crt
        SSLCertificateKeyFile /etc/ssl/private/servtest.pem
</VirtualHost>
~
```

On déplace nos certificats dans les bons répertoires

```
┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ ls -la /etc/ssl/certs | grep -i test

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ ls
ca.crt  ca.pem  ca.srl  servtest.crt  servtest.csr  servtest.pem

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ sudo mv servtest.crt /etc/ssl/certs

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ sudo mv servtest.pem /etc/ssl/private

┌──(kali㉿kali)-[~/Documents/TP-Cyber]
└─$ ls -la /etc/ssl/certs | grep -i test
-rw-r--r-- 1 kali kali   1285 Jun 12 02:55 servtest.crt
```

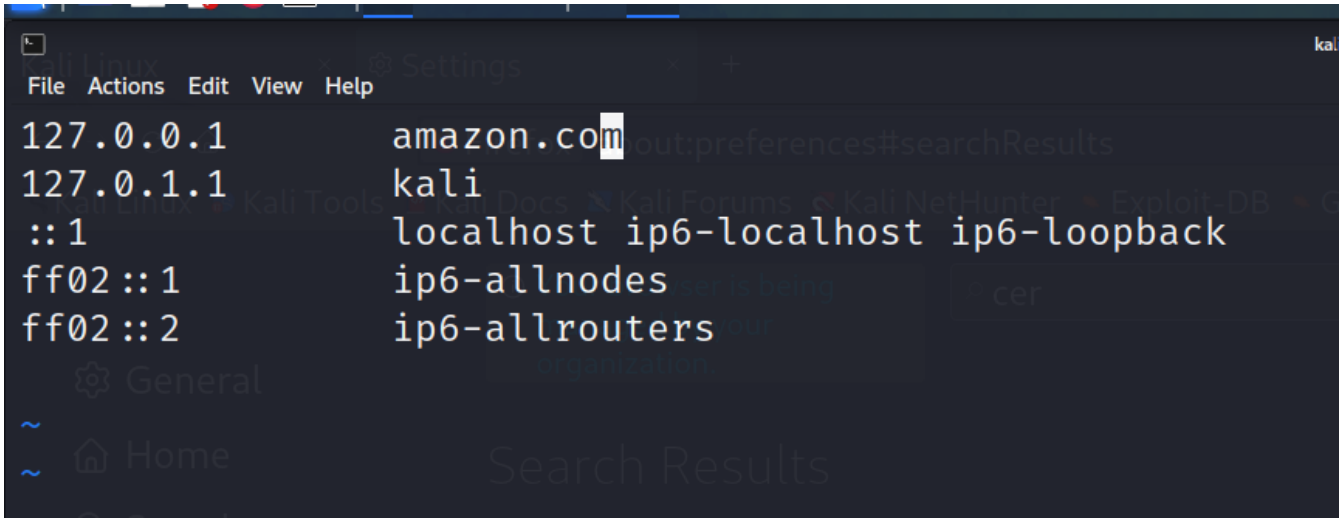
On importe le certificat dans le navigateur web de la machine



On lance les commandes suivantes en tant que super user:

```
a2enmod ssl
a2enmod headers
a2ensite default-ssl
apache2ctl configtest
systemctl restart apache2
```

On modifie le fichier "/etc/hosts":



L'exercice a été validé par le chargé de TP.