TP – OPENSSL

EXERCICE 1

— Pour connaître toutes les fonctionnalité ́es de openSSL : man openssl.

```
File  Actions  Edit  View  Help
OPENSSL(1SSL)                    OpenSSL                    OPENSSL(1SSL)

NAME
       openssl - OpenSSL command line tool

SYNOPSIS
       openssl command [ command_opts ] [ command_args ]

       openssl list [ standard-commands | digest-commands | cipher-commands |
       cipher-algorithms | digest-algorithms | public-key-algorithms]

       openssl no-XXX [ arbitrary options ]
```

— Pour afficher les sous-commandes disponibles : openssl help.

```
kali@kali:~/Desktop/openssl$ openssl help
Standard commands
asn1parse         ca                ciphers           cms
crl               crl2pkcs7         dgst              dhparam
dsa               dsaparam          ec                ecparam
enc               engine            errstr            gendsa
genpkey           genrsa            help              list
nseq              ocsp              passwd            pkcs12
pkcs7             pkcs8             pkey              pkeyparam
pkeyutl           prime             rand              rehash
req               rsa               rsautl            s_client
s_server          s_time            sess_id           smime
speed             spkac             srp               storeutl
ts                verify            version           x509
```

— Pour afficher une description d'une commande : openssl commande -h.

```
kali@kali:~/Desktop/openssl$ openssl sha256 -help
Usage: sha256 [options] [file ... ]
  file ... files to digest (default is stdin)
 -help              Display this summary
 -list              List digests
 -c                 Print the digest with separating colons
 -r                 Print the digest in coreutils format
 -out outfile       Output to filename rather than stdout
```

Avec le -h ça ne fonctionne pas pour moi, j'utilise -help

— L'option "openssl speed" permet de tester les capacité ́es du système pour encrypter des données avec les différents algorithmes de cryptage pendant une p ́période donnée.

```
-engine val       Use engine, possibly a hardware device
kali@kali:~/Desktop/openssl$ openssl speed
Doing md4 for 3s on 16 size blocks: 13557938 md4's in 2.99s
Doing md4 for 3s on 64 size blocks: 10806531 md4's in 3.00s
Doing md4 for 3s on 256 size blocks: 6290316 md4's in 2.99s
Doing md4 for 3s on 1024 size blocks: ^C
```

— Il est possible d'effectuer le test sur un algorithme précis : "openssl speed sha256".

```
kali@kali:~/Desktop/openssl$ openssl speed sha256
Doing sha256 for 3s on 16 size blocks: 11026192 sha256's in 2.99s
Doing sha256 for 3s on 64 size blocks: 5251371 sha256's in 2.99s
Doing sha256 for 3s on 256 size blocks: ^C
kali@kali:~/Desktop/openssl$
```

— Il est possible également de tester les performances SSL d'un serveur distant : "openssl

s_time -connect google.fr:443 -www / -new -ssl3"

```
kali@kali:~/Desktop/openssl$ openssl s_time -connect google.fr:443 -www / -new
Collecting connection statistics for 30 seconds
**********

20 connections in 0.09s; 222.22 connections/user sec, bytes read 1027999
20 connections in 31 real seconds, 51399 bytes read per connection
kali@kali:~/Desktop/openssl$
```

Pour moi, ça ne fonctionne pas avec l'option -ssl3

— Pour encoder une chaîne de caractères en base64, utiliser la commande suivante : "echo -n "mon mot de passe" | openssl base64"oubien"echo -n "mon mot de passe" | openssl enc -base64"

```
kali@kali:~/Desktop/openssl$ echo -n "mon mot de passe" | openssl base64
bW9uIG1vdCBkZSBwYXNzZQ==
kali@kali:~/Desktop/openssl$ echo -n "mon mot de passe" | openssl enc -base64
bW9uIG1vdCBkZSBwYXNzZQ==
kali@kali:~/Desktop/openssl$
```

— Pour encoder une chaîne de caractères en md5, utiliser la commande suivante : "echo -n 'mon mot de passe' | openssl md5"

```
kali@kali:~/Desktop/openssl$ echo -n 'mon mot de passe' | openssl md5
(stdin)= cd0f3f235f32250d7d1d43cb9d7f8a03
```

— Pour décoder une chaîne de caractères : "echo -n "bW9uIG1vdCBkZSBwYXNzZQ==" | openssl enc -base64 -d"

```
kali@kali:~/Desktop/openssl$ echo "bW9uIG1vdCBkZSBwYXNzZQ==" | openssl enc -base64 -d
mon mot de passekali@kali:~/Desktop/openssl$
```

Il faut enlever le -n sinon ça ne peut pas fonctionner !

— Générer une chaîne aléatoire de 128 octets encodée en BASE64 "openssl rand -base64 128"

```
kali@kali:~/Desktop/openssl$ openssl rand -base64 128
59hNInQTlJKMlGEb7gGHoR+y7e81lQSYWwOYyrQ8V/hy7+6g5vVbAS6jM51kjyG5
Zy8BjDbvac9HIZBQ+sL3+34UsJICo36rK6VGWJP25hLV+B++LaIICaUxgUwENfUd
imXyJW+CTbhgXlsyG5XyMW9/kUX8u9aJcmvlopUEkgU=
```

— Générer une chaîne aléatoire de 16 octets encodée en HEX "openssl rand -hex 16"

```
kali@kali:~/Desktop/openssl$ openssl rand -hex 16
0cb9c463ea79b6ba8e68d12bcb19d9c6
```

— Générer des mots de passe cryptés ”openssl passwd MyPasswd”

```
kali@kali:~/Desktop/openssl$ openssl passwd MyPasswd
GaIaLoPw1cDhE
```

— Générer des mots de passe en utilisant MD5-based password algorithm”openssl passwd -1 MyPasswd”

```
kali@kali:~/Desktop/openssl$ openssl passwd -1 MyPasswd
$1$vkxZudVK$K8T1WI9Ai0eaqBaBpv4mQ/
```

(Appuyez-vous toujours sur l’aide de l’outil pour plus d’informations)

EXERCICE 2

1. CHIFFREMENT SYMETRIQUE

(a) Dans un répertoire TP2, créer un fichier fichier.txt contenant le message suivant : ”Bonjour les gars !”.

```
kali@kali:~/Desktop$ echo "Bonjour les gars !" > TP2/fichier.txt
```

(b) En utilisant la commande ”man openssl”, essayez de parcourir l’aide de notre bibliothèque.”openssl enc -help” permet de lister les algorithmes symétriques supportés par ”openssl”. Ou bien lancer cette commande ”openssl list-cipher-commands”.

```
kali@kali:~/Desktop/TP2$ openssl enc -ciphers Supported ciphers
Supported ciphers:
-aes-128-cbc               -aes-128-cfb               -aes-128-cfb1

-aes-128-cfb8              -aes-128-ctr               -aes-128-ecb

-aes-128-ofb               -aes-192-cbc               -aes-192-cfb
```

(c) Choisissez un algorithme symétrique et chiffrer votre fichier ”openssl enc <algo> -in <fichier.txt> -out <fichier.enc>”.

```
kali@kali:~/Desktop/TP2$ openssl enc -sm4 -in fichier.txt -out fichier.en
c
enter sm4-cbc encryption password:
Verifying - enter sm4-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

Algo choisi sm4 et mdp : sm4

(d) Pour déchiffrer le fichier, on ajoute l'option "-d" : "openssl enc <algo> -in <fi- chier.enc> -d -out <fichier1.txt> ". Comparez "<fichier.txt>"et "<fichier1.txt>".

```
kali@kali:~/Desktop/TP2$ openssl enc -sm4 -in fichier.enc -d -out fichier1.txt
enter sm4-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
kali@kali:~/Desktop/TP2$ diff fichier.txt fichier1.txt
kali@kali:~/Desktop/TP2$
```

Tout est ok car il n'y a aucune différence entre les deux fichiers.

(e) Pour améliorer le chiffrement de votre fichier, utilisez l'option "-salt".

```
kali@kali:~/Desktop/TP2$ openssl enc -sm4 -salt -in fichier.txt -out fichier_salt.enc
enter sm4-cbc encryption password:
Verifying - enter sm4-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
kali@kali:~/Desktop/TP2$ diff fichier.enc fichier_salt.enc
1c1
< Salted__
\ No newline at end of file
---
> Salted__
\ No newline at end of file
kali@kali:~/Desktop/TP2$
```

Il y a bien une différence entre les deux façons d'encoder

(f) Ajouter l'option "-a" lors du chiffrement pour avoir un format ASCII.

```
kali@kali:~/Desktop/TP2$ openssl enc -aes128 -a -in fichier.txt -out fichier_a.enc
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:~/Desktop/TP2$ openssl enc -aes128 -in fichier_a.enc -d -out fichier2.txt
enter aes-128-cbc decryption password:
bad magic number
```

J'ai essayé avec différents algorithmes et sur différentes vm cependant à chaque fois cette erreur apparaît. En cherchant sur internet il semble ce soit dû à des mises à jour python ou UNIX.

(g) Pour afficher le salt, la clef et le vecteur d'initialisation, utiliser l'option "-p".

```
kali@kali:~/Desktop/TP2$ openssl enc -sm4 -in fichier_salt.enc -p -out fichier2.txt
enter sm4-cbc encryption password:
Verifying - enter sm4-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=BDBED6CA56681C0F
key=A32226C76DA9A8A065704D53206F97E4
iv =988BC47A90DF1A863672E0243A8CFF86
```

2. CHIFFREMENT ASYMETRIQUE

(a) Générez votre paire de clefs RSA ”openssl genrsa -out CLEF 4092”.

```
kali@kali:~/Desktop/TP2$ openssl genrsa -out CLEF 4092
Generating RSA private key, 4092 bit long modulus (2 primes)
.....................................................................++++
.....................................................................+
+++
e is 65537 (0×010001)
kali@kali:~/Desktop/TP2$
```

(b) Pour protéger votre paire de clefs par un mot de passe, vous pouvez utiliser un algorithme symétrique comme DES3. ”openssl rsa -in CLEF -des3 -out CLEF”

```
kali@kali:~/Desktop/TP2$ openssl rsa -in CLEF -des3 -out CLEF
writing RSA key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

Mot de passe choisi : MOTdepasse25

(c) Pour visualiser votre paire de clef, vous procédez comme suit : ”openssl rsa -in CLEF -text -noout”. L’option -text demande l’affichage décodé de la paire de clefs. L’option -noout supprime la sortie normalement produite par la commande rsa. Les différents éléments de la clef sont affichés en hexadécimal. On peut distinguer le modulus, l’exposant public (qui par défaut est toujours 65537), l’exposant privé, les nombres premiers facteurs du modulus, plus trois autres nombres qui servent à optimiser l’algorithme de déchiffrement.

```
kali@kali:~/Desktop/TP2$ openssl rsa -in CLEF -text -noout
Enter pass phrase for CLEF:
RSA Private-Key: (4092 bit, 2 primes)
modulus:
    0b:5d:06:86:04:30:ee:3f:04:0e:92:d5:a7:9c:17:
    47:6a:e1:30:2a:a2:be:7e:d1:65:8f:f2:a1:4e:a4:
    19:40:65:d0:56:d8:cc:2c:14:50:b7:a4:76:90:23:
    8e:26:00:cb:7b:c4:3b:44:29:92:46:61:00:d4:c3:
    e9:29:cc:9a:4d:3c:14:7a:32:fe:f9:0c:90:04:a6:
```

(d) La CLEF générée ne peut servir que pour le déchiffrement et la signature. Pour pouvoir chiffrer les fichiers, nous devons faire appel à la clef publique. Pour ce faire, exécuter cette commande : ”openssl rsa -in CLEF -pubout -out CLEF_pub”

```
kali@kali:~/Desktop/TP2$ openssl rsa -in CLEF -pubout -out CLEF_pub
Enter pass phrase for CLEF:
writing RSA key
```

(e) Rien que pour vérifier les entêtes des différents fichiers générés, utiliser la commande ”cat”, et décerner la différence.

```
kali@kali:~/Desktop/TP2$ cat CLEF_pub
-----BEGIN PUBLIC KEY-----
MIICITANBgkqhkiG9w0BAQEFAAOCAg4AMIICCQKCAgALXQaGBDDuPwQOktWnnBdH
```

```
kali@kali:~/Desktop/TP2$ cat CLEF
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,809029187E729E43

XtNK0wcZpltb+1L3MFUWZ0oFC6i1WewxwOCri+T2ILm5d+VpCbw/Ggv0cWmuW3fY
```

La différence est qu’il y a une clé publique et une clé privée. On a mis dans un fichier à part la partie publique de la CLEF.

(f) La partie publique d’une paire de clef RSA est publique, et à ce titre peut être communiquée à n’importe qui. La CLEF contient la partie privée de la clef, et ne peut donc pas être communiqué tel quel (même si elle est protégée par pwd). Avec l’option -pubout on peut exporter la partie publique d’une clef.

La partie privée est donc CLEF et la partie publique est CLEF_pub

(g) Procédons au chiffrement de notre fichier ”<fichier.txt>” comme suit : ”openssl rsautl -encrypt -pubin -inkey CLEF_pub -in fichier. txt -out fichier.enc”.

```
kali@kali:~/Desktop/TP2$ openssl rsautl -encrypt -pubin -inkey CLEF_pub -in fichier.txt -out
 fichier.enc
kali@kali:~/Desktop/TP2$
```

(h) Pour déchiffrer : ”openssl rsautl -decrypt -inkey CLEF -in fichier.enc -out fichier2.txt”.

```
kali@kali:~/Desktop/TP2$ openssl rsautl -decrypt -inkey CLEF -in fichier.enc -out fichier2.t
xt
Enter pass phrase for CLEF:
```

```
kali@kali:~/Desktop/TP2$ cat fichier2.txt
Bonjour les gars !
```

(i) On signe le fichier ”<fichier.txt>” par la clef privée CLEF : ”openssl rsautl -sign -inkey CLEF -in fich.txt -out fichier.sign »

```
kali@kali:~/Desktop/TP2$ openssl rsautl -sign -inkey CLEF -in fichier.txt -out fichier.sign
Enter pass phrase for CLEF:
kali@kali:~/Desktop/TP2$
```

C’est fichier.txt et pas fich.txt

(j) On peut vérifier la signature du fichier en procédant par le chemin inverse :”openssl rsautl -verify -pubin -inkey CLEF_pub -out fichier3.txt -in fichier.sign”

```
kali@kali:~/Desktop/TP2$ openssl rsautl -verify -pubin -inkey CLEF_pub -out fichier3.txt -in
 fichier.sign
```

```
kali@kali:~/Desktop/TP2$ cat fichier3.txt
Bonjour les gars !
```

(k) Signature et vérification d'un fichier condensé avec sha256 : "openssl dgst -sha256 -sign CLEF -out fich_hash_sign fich.txt"

```
kali@kali:~/Desktop/TP2$ openssl dgst -sha256 -sign CLEF -out fich_hash_sign fichier.txt
Enter pass phrase for CLEF:
```

C'est fichier.txt et pas fich.txt

(l) Pour pouvoir vérifier la bonne signature du message et avoir comme rendu "Verified OK", on procède comme suit : "openssl dgst -sha256 -verify CLEF_pub -signature fich_hash_sign fich.txt"

```
kali@kali:~/Desktop/TP2$ openssl dgst -sha256 -verify CLEF_pub -signature fich_hash_sign fic
hier.txt
Verified OK
```

C'est fichier.txt et pas fich.txt

(m) Bien évidemment, on peut aussi hacher le fichier sans faire appel à la signature.

```
kali@kali:~/Desktop/TP2$ cat fich_hash_sign
SHA256(fichier.txt)= a644ba9c54c8567baa2899af84a201d043376fb1719b86dc19d08b332eea4993
kali@kali:~/Desktop/TP2$
```