

Travaux pratiques –TP 2

Services de sécurité, mécanismes, algorithmes avec *API Openssl*

Partie 1 : OpenSSL

OpenSSL est une boîte à outils cryptographiques implémentant les protocoles SSL et TLS. Il offre :

- Une bibliothèque de programmation en C permettant de réaliser des applications client/serveur sécurisées s'appuyant sur SSL/TLS
- Une commande en ligne (OpenSSL) permettant
 - o la création de clés RSA, DSA (signature)
 - o la création de certificats X509
 - o le calcul d'empreintes (MD5, SHA, RIPEMD160, ...)
 - o le chiffrement et déchiffrement (RSA, DES, IDEA, RC2, RC4, Blowfish, ...)
 - o la réalisation de tests de clients et serveurs SSL/TLS
 - o la signature et le chiffrement de courriers (S/MIME)

La syntaxe générale de la commande openssl est

`$ Openssl <commande> <option>`

On trouvera toutes les informations la concernant à l'adresse : <http://www.openssl.org>

L'objectif de ce TP est de vous familiariser avec les services de base de la sécurité, chiffrement, hachage et signature.

- a) Télécharger l'outil openssl avec la commande `apt-get install openssl`

```
-(mar. mars 24 14:10:31)--(latitude-3590:~)-
[nk] $ sudo apt install openssl
openssl est déjà la version la plus récente (3.5.5-1~deb13u1).
openssl passé en « installé manuellement ».
Les paquets suivants ont été installés automatiquement et ne sont plus nécessaires :
  linux-image-6.12.63+deb13-amd64  linux-image-6.12.69+deb13-amd64
Veuillez utiliser « sudo apt autoremove » pour les supprimer.

Sommaire :
  Mise à niveau de : 0. Installation de : 0Supprimé : 0. Non mis à jour : 0
```

- b) Vérifier la bonne installation de l'outil
- c) Répondez aux questions suivantes :
1. Quelle est la version d'openssl installée ?

```
-(mar. mars 24 14:11:36)--(latitude-3590:~)-
[nk] $ openssl -v
OpenSSL 3.5.5 27 Jan 2026 (Library: OpenSSL 3.5.5 27 Jan 2026)
```

2. Lister tous les algorithmes de cryptographie présents dans l'outil ?

```

-(mar. mars 24 14:17:29)--(latitude-3590:~)-
[nk] $ openssl --help
help:

Standard commands
asn1parse      ca          ciphers      cmp
cms            crl         crl2pkcs7    dgst
dhparam        dsa         dsaparam     ec
ecparam        enc         engine       errstr
fipsinstall    gensa      genpkey      genrsa
help           info       kdf          list
mac            nseq       ocsf         passwd
pkcs12         pkcs7      pkcs8        pkey
pkeyparam      pkeyutl    prime        rand
rehash         req        rsa          rsautl
s_client       s_server   s_time       sess_id
skeyutl        smime      speed        spkac
srp            storeutl   ts           verify
version        x509

Message Digest commands (see the `dgst' command for more details)
blake2b512     blake2s256 md4           md5
rmd160         sha1        sha224        sha256
sha3-224       sha3-256    sha3-384      sha3-512
sha384         sha512      sha512-224    sha512-256
shake128       shake256     sm3

Cipher commands (see the `enc' command for more details)
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   base64
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

```

d) Tapez la commande suivante et expliquer le résultat : `openssl ciphers -v`

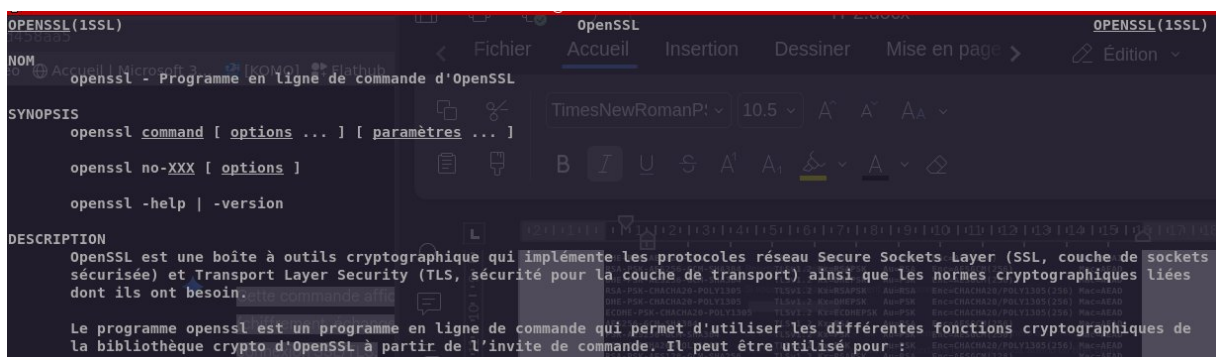

```

-(mar. mars 24 14:18:45)- (latitude-3590:~)-
[nk] $ openssl ciphers -v
TLS_AES_256_GCM_SHA384 TLSv1.3 Kx=any Au=any Enc=AESGCM(256) Mac=AEAD
TLS_CHACHA20_POLY1305_SHA256 TLSv1.3 Kx=any Au=any Enc=CHACHA20/POLY1305(256) Mac=AEAD
TLS_AES_128_GCM_SHA256 TLSv1.3 Kx=any Au=any Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESGCM(256) Mac=AEAD
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH Au=RSA Enc=AESGCM(256) Mac=AEAD
DHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=DH Au=RSA Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH Au=ECDSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
ECDHE-RSA-CHACHA20-POLY1305 TLSv1.2 Kx=ECDH Au=RSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
DHE-RSA-CHACHA20-POLY1305 TLSv1.2 Kx=DH Au=RSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
ECDHE-ECDSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESGCM(128) Mac=AEAD
ECDHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=ECDH Au=RSA Enc=AESGCM(128) Mac=AEAD
DHE-RSA-AES128-GCM-SHA256 TLSv1.2 Kx=DH Au=RSA Enc=AESGCM(128) Mac=AEAD
ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AES(256) Mac=SHA384
ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH Au=RSA Enc=AES(256) Mac=SHA384
DHE-RSA-AES256-SHA256 TLSv1.2 Kx=DH Au=RSA Enc=AES(256) Mac=SHA256
ECDHE-ECDSA-AES128-SHA256 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AES(128) Mac=SHA256
ECDHE-RSA-AES128-SHA256 TLSv1.2 Kx=ECDH Au=RSA Enc=AES(128) Mac=SHA256
DHE-RSA-AES128-SHA256 TLSv1.2 Kx=DH Au=RSA Enc=AES(128) Mac=SHA256
ECDHE-ECDSA-AES256-SHA TLSv1.2 Kx=ECDH Au=ECDSA Enc=AES(256) Mac=SHA1
ECDHE-RSA-AES256-SHA TLSv1.2 Kx=ECDH Au=RSA Enc=AES(256) Mac=SHA1
DHE-RSA-AES256-SHA TLSv1.2 Kx=DH Au=RSA Enc=AES(256) Mac=SHA1
ECDHE-ECDSA-AES128-SHA TLSv1.2 Kx=ECDH Au=ECDSA Enc=AES(128) Mac=SHA1
ECDHE-RSA-AES128-SHA TLSv1.2 Kx=ECDH Au=RSA Enc=AES(128) Mac=SHA1
DHE-RSA-AES128-SHA TLSv1.2 Kx=DH Au=RSA Enc=AES(128) Mac=SHA1
RSA-PSK-AES256-GCM-SHA384 TLSv1.2 Kx=RSAPSK Au=RSA Enc=AESGCM(256) Mac=AEAD
DHE-PSK-AES256-GCM-SHA384 TLSv1.2 Kx=DHEPSK Au=PSK Enc=AESGCM(256) Mac=AEAD
RSA-PSK-CHACHA20-POLY1305 TLSv1.2 Kx=RSAPSK Au=RSA Enc=CHACHA20/POLY1305(256) Mac=AEAD
DHE-PSK-CHACHA20-POLY1305 TLSv1.2 Kx=DHEPSK Au=PSK Enc=CHACHA20/POLY1305(256) Mac=AEAD
ECDHE-PSK-CHACHA20-POLY1305 TLSv1.2 Kx=ECDHEPSK Au=PSK Enc=CHACHA20/POLY1305(256) Mac=AEAD
AES256-GCM-SHA384 TLSv1.2 Kx=RSA Au=RSA Enc=AESGCM(256) Mac=AEAD
PSK-AES256-GCM-SHA384 TLSv1.2 Kx=PSK Au=PSK Enc=AESGCM(256) Mac=AEAD
PSK-CHACHA20-POLY1305 TLSv1.2 Kx=PSK Au=PSK Enc=CHACHA20/POLY1305(256) Mac=AEAD
RSA-PSK-AES128-GCM-SHA256 TLSv1.2 Kx=RSAPSK Au=RSA Enc=AESGCM(128) Mac=AEAD
DHE-PSK-AES128-GCM-SHA256 TLSv1.2 Kx=DHEPSK Au=PSK Enc=AESGCM(128) Mac=AEAD
AES128-GCM-SHA256 TLSv1.2 Kx=RSA Au=RSA Enc=AESGCM(128) Mac=AEAD
PSK-AES128-GCM-SHA256 TLSv1.2 Kx=PSK Au=PSK Enc=AESGCM(128) Mac=AEAD
AES256-SHA256 TLSv1.2 Kx=RSA Au=RSA Enc=AES(256) Mac=SHA256
AES128-SHA256 TLSv1.2 Kx=RSA Au=RSA Enc=AES(128) Mac=SHA256
ECDHE-PSK-AES256-CBC-SHA384 TLSv1.2 Kx=ECDHEPSK Au=PSK Enc=AES(256) Mac=SHA384
ECDHE-PSK-AES256-CBC-SHA TLSv1.2 Kx=ECDHEPSK Au=PSK Enc=AES(256) Mac=SHA1
SRP-RSA-AES-256-CBC-SHA SSLv3 Kx=SRP Au=RSA Enc=AES(256) Mac=SHA1
SRP-AES-256-CBC-SHA SSLv3 Kx=SRP Au=SRP Enc=AES(256) Mac=SHA1

```

Cette commande affiche la liste détaillée et ordonnée de toutes les méthodes de sécurité (chiffrement, échange de clés, authentification) que votre système sait utiliser pour établir une connexion SSL/TLS.

e) Pour connaître toutes les fonctionnalités de openssl : `man openssl`



Partie 2 : Fonctions de hachage

a) Créer le fichier `Plain.txt` contenant la phrase « travaux pratiques »

```
-(mar. mars 24 14:46:28)--(latitude-3590:~)-  
[nk] $ echo "travaux pratiques" >> Plain.txt  
-(mar. mars 24 14:46:52)--(latitude-3590:~)-  
[nk] $ cat Plain.txt  
travaux pratiques
```

- b) Générer le digest (l'empreinte) en SHA1 et MD5 pour votre fichier.

```
-(mar. mars 24 14:46:56)--(latitude-3590:~)-  
[nk] $ md5sum Plain.txt  
2d4a4b04cae4ed5de873af34aeaad108 Plain.txt  
-(mar. mars 24 14:47:16)--(latitude-3590:~)-  
[nk] $ sha1sum Plain.txt  
5b3664d38a586bee479860ab0be57b3ce7c10d9c Plain.txt
```

- c) Modifier le contenu du fichier en remplaçant seulement le caractère **t** par **T** du mot travaux. Générer de nouveau le digest du fichier. Comparer le résultat avec la question b). Conclure.

```
-(mar. mars 24 14:51:38)--(latitude-3590:~)-  
[nk] $ sed -i 's/travaux/Travaux/g' Plain.txt  
-(mar. mars 24 14:51:57)--(latitude-3590:~)-  
[nk] $ md5sum Plain.txt  
6147895edb7f31b647f19a62a46b8830 Plain.txt  
-(mar. mars 24 14:52:03)--(latitude-3590:~)-  
[nk] $ sha1sum Plain.txt  
b610ad00ec541f544ddc42d290b3644fdc41983b Plain.txt
```

Le digest du fichier a changé, car son contenu a changé.

- d) Télécharger le fichier `usa.rar` donné par le chargé du TP. Décompresser le fichier et générer le digest en MD5 pour chaque fichier dans `usa.rar`. Qu'en pensez-vous ?

```
-(mar. mars 24 14:56:29)--(latitude-3590:~/Documents)  
[nk] $ md5sum */*  
008ee33a9d58b51cfeb425b0959121c9 ./message1.bin  
008ee33a9d58b51cfeb425b0959121c9 ./message2.bin  
aff950cab4c0265e21d401db15f1026d ./st_img1.tif  
aff950cab4c0265e21d401db15f1026d ./st_img2.tif
```

Malgré que les `.bin` et les `.tif` sont des noms différents ils ont le même md5 donc probablement que message1 et message2 soit la même chose avec simplement un renommage, et `st_img1.tif` et `st_img2.tif` qui ont également le même md5 sont également la même chose avec un renommage.

- e) Générer le digest en sha 256 pour chaque fichier dans `usa.rar`. Que concluez-vous

```
-(mar. mars 24 14:56:45)--(latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE)  
[nk] $ sha256sum */*  
54bcb9a4fda31e4f254303e3959acd5e420ad18a80949d56a3000c3716fbd1a0 ./message1.bin  
90774a6455a2bdb7d106e533923ecbefe81392ca55bed0ce81cfab2c1a7f0afe ./message2.bin  
753e0dd54f28c4f7009b9c0b18a68aed175416bd8b7d134858264586eaac56f0 ./st_img1.tif  
753e0dd54f28c4f7009b9c0b18a68aed175416bd8b7d134858264586eaac56f0 ./st_img2.tif
```

A la différence du md5, message1 et message2 ne sont pas le même sha256. Le Sha256 est donc plus efficace et précis.

Partie 3 : Chiffrement symétrique

La commande `openssl enc` permet de chiffrer et déchiffrer des messages. Plus d'informations sur cette commande peuvent être trouvées en tapant `openssl enc -h`

- a) Créer un fichier texte `Plain.txt` et y écrire : « travaux pratiques ». Chiffrer ce fichier avec l'algorithme DES-CBC et sauvegarder le fichier sous le nom `Cipher.txt`. Utiliser l'option `-k` pour saisir le mot de passe symétrique. Vous pouvez utiliser l'option `-base64` pour la lisibilité.

```
-(mar. mars 24 16:06:59)- (latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_secu
rite_des_reseaux/26-03-24/usa)-
[nk] $ echo "travaux pratique" > Plain.txt
-(mar. mars 24 16:07:08)- (latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_secu
rite_des_reseaux/26-03-24/usa)-
[nk] $ openssl enc -des-cbc -k toto -base64 -in Plain.txt -out Cipher.txt -provider legacy -provider def
ault
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

- b) Ouvrir le fichier chiffré. Qu'observez-vous au niveau des premiers caractères ? À quoi cela sert-il ?

```
-(mar. mars 24 16:08:24)- (latitude-3590:~/Documents/UTEC
rite_des_reseaux/26-03-24/usa)-
[nk] $ cat Cipher.txt
U2FsdGVkX19LjdVzcoMuo+6tn2s6lNXUqN83b84ARDn1Gu9fOC9kNQ==
```

- c) Déchiffrer le fichier chiffré en lui donnant le nom `NewPlain.txt` et vérifier qu'on retombe bien sur le fichier de départ. Pour vérifier : `diff Plain.txt NewPlain.txt -q`

```
-(mar. mars 24 16:07:25)- (latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_secu
rite_des_reseaux/26-03-24/usa)-
[nk] $ openssl enc -d -des-cbc -k toto -base64 -in Cipher.txt -out NewPlain.txt -provider legacy -provid
er default
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
-(mar. mars 24 16:07:44)- (latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_secu
rite_des_reseaux/26-03-24/usa)-
[nk] $ diff Plain.txt NewPlain.txt -q
```

Ils ne sont pas différents.

- d) Reprendre la question a) et c) mais cette fois ci en indiquant l'option `-p`. Détailler les informations obtenues.

```
-(mar. mars 24 16:10:00)- (latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_secu
rite_des_reseaux/26-03-24/usa)-
[nk] $ openssl enc -des-cbc -k toto -base64 -in Plain.txt -out Cipher.txt -p -provider legacy -provider
default
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=5D32DB6CC0818469
key=B37AE1021D1E0998
iv =84970172BA453603
-(mar. mars 24 16:10:09)- (latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_secu
rite_des_reseaux/26-03-24/usa)-
[nk] $ openssl enc -d -des-cbc -k toto -base64 -in Cipher.txt -out NewPlain.txt -p -provider legacy -pro
vider default
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=5D32DB6CC0818469
key=B37AE1021D1E0998
iv =84970172BA453603
```

Ça nous donne le salt , la clé et l'iv,

Voici à quoi cela correspond :

salt : Une valeur aléatoire ajoutée à votre mot de passe pour rendre le chiffrement plus robuste face aux attaques par dictionnaire.

key : La véritable clé binaire (en hexadécimal) dérivée de votre mot de passe. C'est elle qui "verrouille" réellement les données.

iv (Vecteur d'Initialisation) : Un nombre aléatoire utilisé pour s'assurer que si vous chiffrez deux fois le même texte avec le même mot de passe, le résultat (le Cipher.txt) sera différent.

- e) Chiffrer le `Plain.txt` encore une fois (`NewCipher.txt`). Comparer les fichiers `Cipher.txt` et `NewCipher.txt`. Justifiez.

```
-(mar. mars 24 16:10:23)-((latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_sécurité_des_reseaux/26-03-24/usa)-
[nk]$ openssl enc -des-cbc -k toto -base64 -in Plain.txt -out NewCipher.txt -p -provider legacy -p-provider default
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=499D64AC7FA06802
key=64E003DE0D5F50B9
iv =51763239C6FD0969
-(mar. mars 24 16:12:40)-((latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_sécurité_des_reseaux/26-03-24/usa)-
[nk]$ diff Cipher.txt NewCipher.txt -q
Les fichiers Cipher.txt et NewCipher.txt sont différents
```

ils ne seront pas identiques car OpenSSL génère un nouveau sel (salt) aléatoire à chaque fois. Ce sel modifie la clé technique et le vecteur d'initialisation (IV) dérivés de votre mot de passe, produisant un fichier chiffré unique.

- f) Refaire ce test deux nouvelles fois en ajoutant l'option `-nosalt`. Comparer et expliquer les résultats obtenus dans b).

```
-(mar. mars 24 16:14:47)-((latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_sécurité_des_reseaux/26-03-24/usa)-
[nk]$ openssl enc -des-cbc -k toto -base64 -in Plain.txt -out Cipher.txt -p -nosalt -provider legacy -p-provider default
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
key=31F7A65E315586AC
iv =198BD798B6629CE4
-(mar. mars 24 16:15:00)-((latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_sécurité_des_reseaux/26-03-24/usa)-
[nk]$ openssl enc -des-cbc -k toto -base64 -in Plain.txt -out NewCipher.txt -p -nosalt -provider legacy -p-provider default
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
key=31F7A65E315586AC
iv =198BD798B6629CE4
-(mar. mars 24 16:15:08)-((latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_sécurité_des_reseaux/26-03-24/usa)-
[nk]$ diff Cipher.txt NewCipher.txt -q
Non, ils ne seront pas différents
```

Les deux fichiers chiffrés sont identiques car la clé et l'IV seront générés uniquement à partir du mot de passe, sans variation aléatoire. Le diff ne trouve donc aucune différence.

- g) Grâce à la commande `rand`, créer un fichier `GrandFichier.txt` avec des données aléatoires, d'une taille d'environ 100 Mo. Utiliser la commande `time` pour calculer le temps de chiffrement de votre fichier en utilisant DES, 3DES, AES-CBC (Prendre le temps user). Comparer les résultats ?

```
-(mar. mars 24 16:17:55)- (latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_securite_des_reseaux/26-03-24/usa)-
[nk] $ openssl rand -out GrandFichier.txt 100000000
-(mar. mars 24 16:17:56)- (latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_securite_des_reseaux/26-03-24/usa)-
[nk] $ time openssl enc -des-cbc -k test -in GrandFichier.txt -out /dev/null -provider legacy -provider default
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

real    0m1,639s
user    0m1,590s
sys     0m0,048s
-(mar. mars 24 16:18:19)- (latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_securite_des_reseaux/26-03-24/usa)-
[nk] $ time openssl enc -des3 -k test -in GrandFichier.txt -out /dev/null -provider legacy -provider default
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

real    0m4,376s
user    0m4,267s
sys     0m0,102s
-(mar. mars 24 16:18:27)- (latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_securite_des_reseaux/26-03-24/usa)-
[nk] $ time openssl enc -aes-256-cbc -k test -in GrandFichier.txt -out /dev/null
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

real    0m0,198s
user    0m0,165s
sys     0m0,033s
```

DES est un algorithme simple mais clé très courte (obsolète).

3DES applique le DES trois fois de suite, ce qui triple le travail du CPU.

AES-256 est le plus rapide c'est le plus complexe, les processeurs modernes possèdent des instructions matérielles (AES-NI) pour l'accélérer.

Partie 4 : Chiffrement asymétrique

Dans cet exercice, l'intérêt est de chiffrer un document avec une clé publique que seul son créateur pourra déchiffrer.

- a) Générer une paire de clés RSA 2048 bits que vous nommerez `PrivateKeyMyName.priv`.
Ex : `PrivateKeyDupont.priv`. Utilisez l'option `-p`.

```
-(mar. mars 24 16:25:34)- (latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_securite_des_reseaux/26-03-24/usa)-
[nk] $ openssl genrsa -out PrivateKeySanchez.priv 2048
```

- b) Extraire ensuite la clé publique que vous nommerez `PublicKeyDupont.pub`

```
rite_des_reseaux/26-03-24/usa)-
[nk] $ openssl rsa -in PrivateKeySanchez.priv -pubout -out PublicKeySanchez.pub
writing RSA key
```

- c) Envoyer à votre voisin votre clé publique (par ssh ou clé usb)

```
rite_des_reseaux/26-03-24/usa)-
[nk] $ scp PublicKeySanchez.pub admkactus@192.168.1.242:/home/admkactus/
PublicKeySanchez.pub 100% 451 35.0KB/s 00:00
```

- d) Chiffrer le fichier `Plain.txt` avec la clé publique de votre collègue


```
-(mar. mars 24 16:31:23)--(Latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_secu
rite_des_reseaux/26-03-24/usa)-
[nk] $ openssl pkeyutl -encrypt -pubin -inkey PublicKeyVoisin.pub -in Plain.txt -out Plain.enc
```

- e) Déchiffrer le fichier que vous avez reçu avec votre clé privée

```
-(mar. mars 24 16:34:41)--(Latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_secu
rite_des_reseaux/26-03-24/usa)-
[nk] $ openssl pkeyutl -decrypt -inkey PrivateKeySanchez.priv -in fichierVoisin.enc -out fichierVoisin.t
xt
-(mar. mars 24 16:35:24)--(Latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_secu
rite_des_reseaux/26-03-24/usa)-
[nk] $ cat fichierVoisin.txt
Bonjour voisin !
```

- f) Chiffrer le fichier `GrandFichier.txt` avec la clé publique de votre collègue. Vous devez remarquer un problème. Comment l'expliquez-vous ?

```
-(mar. mars 24 16:35:34)--(Latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analyse_de_la_secu
rite_des_reseaux/26-03-24/usa)-
[nk] $ openssl pkeyutl -encrypt -pubin -inkey PublicKeyVoisin.pub -in GrandFichier.txt -out Grandfichier
.enc
Public Key operation error
40A78A67137F0000:error:0200006E:rsa routines:ossl_rsa_padding_add_PKCS1_type_2_ex:data too large for key
size:../crypto/rsa/rsa_pk1.c:132:
```

Le chiffrement asymétrique (RSA) est limité mathématiquement par la taille de la clé. Avec une clé de 2048 bits, vous ne pouvez chiffrer qu'un message d'environ 245 octets maximum.

Partie 5 : Chiffrement symétrique/asymétrique et signature numérique

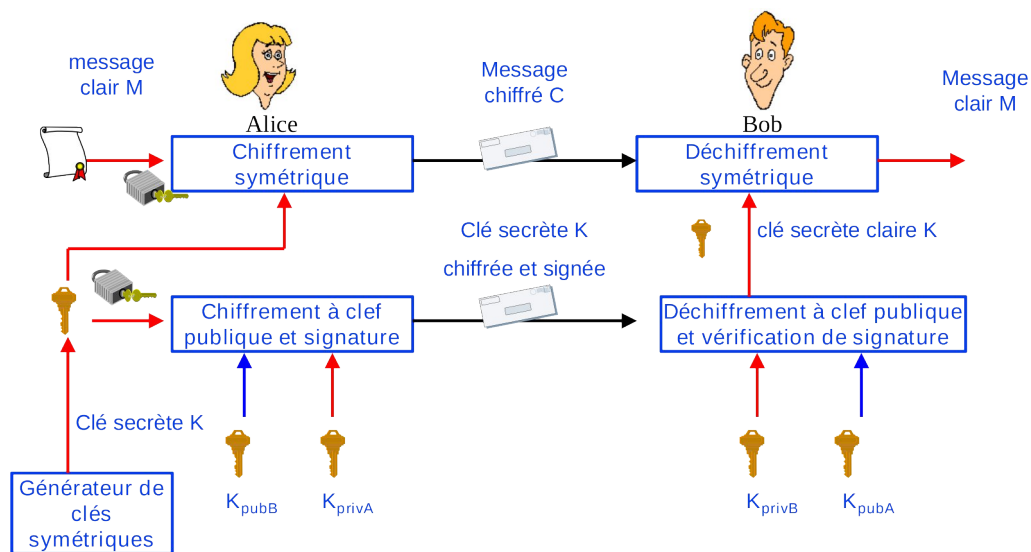


Figure 1. Scénario sécurité

- a) Expliquer le scénario ci-dessus.

Alice génère une clé de session (K). Elle chiffre son message avec K (symétrique). Pour que Bob puisse lire, elle chiffre K avec la clé publique de Bob (K_{pubB}) et signe le tout avec sa propre clé privée (K_{privA}).

- b) Quels sont les différents services de sécurité réalisés dans ce scénario.

Confidentialité Chiffrement symétrique du message + asymétrique de la clé.

Authenticité/Non-répudiation (Signature avec la clé privée d'Alice).

Intégrité : le hash signé garantit que rien n'a été modifié.

- c) Générer avec la commande `rand` une clé symétrique `SymKey.key` de taille 128 bits. Encoder la clé en hex.

```
-(mar. mars 24 16:40:04)--(latitude-359
rite_des_reseaux/26-03-24/usa) -
[nk] $ openssl rand -out SymKey.key 16
-(mar. mars 24 16:43:56)--(latitude-359
rite_des_reseaux/26-03-24/usa) -
[nk] $ openssl rand -hex 16
24053734c653d241f4cd9901b6df8763
```

- d) Visualiser la clé symétrique en binaire en utilisant la commande linux : `xxd`
`xxd -b -c 8 SymKey.key`

```
-(mar. mars 24 16:44:42)--(latitude-3590:~/Documents/UTEC-Master_M2I_2025-2026/UE2.34_Analys
rite_des_reseaux/26-03-24/usa) -
[nk] $ xxd -b -c 8 SymKey.key
00000000: 11011101 01010000 01100011 10101101 10011000 10011101 00101010 00101011  .Pc...5+
00000008: 11111011 01110011 10011100 10000111 00010110 00011001 10111111 01110101  .S.....u
```

- e) Réaliser le scénario ci-dessus en échangeant d'une manière sûre la clé symétrique avec votre collègue. Pour information, on signe le hash d'un fichier et non pas le fichier lui-même.

Commandes utiles

`$ time openssl commande` : pour visualiser le temps d'exécution d'une commande

`$ diff file1.txt file2.txt -q` : pour comparer le contenu de deux fichiers

`$ openssl genrsa -out <fichier_rsa.priv> <size>` : génère la clé privé RSA de taille size.

`$ openssl rsa -in <fichier_rsa.priv> -des3 -out <fichier.pem>` : chiffre la clef privé RSA avec l'algorithme DES3. Vous pouvez utiliser DES, 3DES, IDEA, etc.

`$ openssl rsa -in <fichier_rsa.priv> -pubout -out <fichier_rsa.pub>` : stocke la partie publique dans un fichier à part

`$ openssl enc <-algo> -in <claire.txt> -out <chiffre.enc>`: pour le chiffrement de claire.txt avec l'algorithme spécifié (`openssl enc -help` pour avoir la liste des possibilités ou bien `openssl list-cipher-commands`) dans un fichier chiffre.enc. On peut ajouter `-k` comme option et saisir la clé comme une passphrase

`$ openssl enc <-algo> -in <chiffre> -d -out <claire>` : pour le déchiffrement.

`$ openssl dgst <-algo> -out <sortie> <entrée>` : pour hacher un fichier. L'option `<-algo>` est le choix de l'algorithme de hachage (sha, sha1, dss1, md2 ,md4, md5, ripemd160).

`$ openssl rand -out <clé.key> <nombre_octets>` : pour générer un nombre aléatoire de taille nombre_octets (utiliser l'option `-base 64` pour la lisibilité).

\$ openssl aes-256-cbc -in <claire.txt> -out <chiffre.enc> -e -k <clé.key> : pour chiffrer un fichier avec l'AES.

\$ openssl rsautl -encrypt -pubin -inkey <rsa.pub> -in <clair.txt> -out <chiffre.enc> : chiffrer fichier.txt avec la RSA en utilisant la clef publique rsa.pub

\$ openssl rsautl -decrypt -inkey <rsa.priv> -in <chiffre.enc> -out <fichier.txt> : pour déchiffrer le fichier fic.dec

\$ openssl rsautl -sign -inkey <ras.priv> -in <fichier.txt> -out <fic.sig> : pour générer une signature.

\$ openssl rsautl -verify -pubin -inkey <rsa.pub> -in fic fic.sig : pour vérifier une signature.

- \$openssl aes-256-cbc -in <fichier.txt> -out <fichier.enc> -e -k clé.key : pour chiffrer un fichier avec l'algorithme AES utilisant une clé symétrique.

- \$openssl aes-256-cbc -in <fichier.txt> -out <fichier.enc> -d -k clé.key : pour déchiffrer un fichier avec l'algorithme AES utilisant une clé symétrique.

COMMANDES STANDARDS

asn1parse

Traitement d'une séquence ASN.1.

ca

Gestion Certificate Authority (CA).

ciphers

Détermination de la description de la suite de chiffrement.

crl

Gestion Certificate Revocation List (CRL).

crl2pkcs7

Conversion CRL vers PKCS#7.

dgst

Calcul signature message (MD5).

dh

Gestion des paramètres Diffie-Hellman. Obsolète par **dhparam**.

dsa

Gestion données DSA.

dsaparam

Génération paramètres DSA.

enc

Chiffrement.

errstr

Conversion numéro d'erreur vers descriptif texte (String).

dhparam

Génération et gestion de paramètres Diffie-Hellman.

gendh

Génération de paramètres Diffie-Hellman. Obsolète par **dhparam**.

genssa

Génération de paramètres DSA.

genrsa

Génération de paramètres RSA.

passwd

Génération de mots de passe hashés.

pkcs7

Gestion données PKCS#7.

rand

Génère octets pseudo-aléatoires.

req

Gestion X.509 Certificate Signing Request (CSR).

rsa

Gestion données RSA.

rsautl

Utilitaire RSA pour signature, vérification, chiffrement, et déchiffrement.

s_client

Ceci fournit un client SSL/TLS générique qui sait établir une connexion transparente avec un serveur distant parlant SSL/TLS. Étant seulement prévu pour des propos de test, il n'offre qu'une interface fonctionnelle rudimentaire tout en utilisant en interne la quasi-totalité des fonctionnalités de la librairie **ssl** d'OpenSSL.

s_server

Ceci fournit un client SSL/TLS générique qui accepte des connexions transparentes provenant de clients qui parlent SSL/TLS. Étant seulement prévu pour des propos de test, il n'offre qu'une interface fonctionnelle rudimentaire tout en utilisant en interne la quasi-totalité des fonctionnalités de la librairie **ssl** d'OpenSSL. Il fournit à la fois son propre protocole orienté commandes en ligne pour le test de fonctions SSL et une facilité de réponse simple HTTP pour émuler un serveur internet qui gère SSL/TLS.

s_time

Horloger de connexions SSL.

sess_id

Gestion des données de session SSL.

smime

Traitement mails S/MIME.

speed

Mesure la vitesse de l'algorithme.

verify

Vérification du certificat X.509.

version

Information sur la version d'OpenSSL.

x509

Gestion de données pour le certificat X.509.

COMMANDES DE SIGNATURE DE MESSAGE

md2

Signature MD2

md5

Signature MD5

mdc2

Signature MDC2

rmd160

Signature RMD-160

sha

Signature SHA

sha1

Signature SHA-1

COMMANDES D'ENCODAGE ET DE CHIFFREMENT

base64

Chiffrement Base64

bf bf-cbc bf-cfb bf-ecb bf-ofb

Chiffrement Blowfish

cast cast-cbc

Chiffrement CAST

cast5-cbc cast5-cfb cast5-ecb cast5-ofb

Chiffrement CAST5

des des-cbc des-cfb des-ecb des-edc des-edc-cbc des-edc-cfb des-edc-ofb des-ofb

Chiffrement DES

des3 desx des-edc3 des-edc3-cbc des-edc3-cfb des-edc3-ofb

Chiffrement Triple-DES

idea idea-cbc idea-cfb idea-ecb idea-ofb

Chiffrement IDEA

rc2 rc2-cbc rc2-cfb rc2-ecb rc2-ofb

Chiffrement RC2

rc4

Chiffrement RC4

rc5 rc5-cbc rc5-cfb rc5-ecb rc5-ofb

Chiffrement RC5