web et sécurité

TP1 L'outil OpenSSL

Réaliser Par : Guettouche Islam & Rabhi Lounis

Exercices 01

1. Quelle version de TLS est utilisée ?
   Sur openSSL on trouve **TLSv1.**
2.

Google chrome

- **https://www.facebook.com/**

  - ✓ La version de TLS utilisée est : **TLS 1.2.**
  - ✓ La suite chiffrante est **ECDHE_ECDSA** with P-256 (a strong key exchange), and **AES_128_GCM** (a strong cipher).

ECDHE :
Algorithme d'échange de clés diffie-helman **asymétrique.**

ECDSA :
Algorithme de signature et de hachage : **asymétrique.**

AES_128_GCM :
Algorithme de chiffrement : **symétrique.**

- ✓ **Les objectifs :**
  L'Intégrité des données, la non répudiation, Garantir la confidentialité et l'authentification.

- **https://dpt-info-sciences.univ-rouen.fr**

  - ✓ La version de TLS utilisée est : **TLS 1.2.**
  - ✓ La suite chiffrante est **ECDHE_RSA** with P-256 (a strong key exchange), and **AES_256_GCM** (a strong cipher).

ECDHE :
Algorithme d'échange de clés diffie-helman **asymétrique.**

RSA :
Algorithme de signature et de hachage : **asymétrique.**

AES_256_GCM :
Algorithme de chiffrement : **symétrique.**

- ✓ **Les objectifs :**
  L'Intégrité des données, la non répudiation, Garantir la confidentialité et l'authentification.

https://www.globetrotter.ch/

- ✓ La version de TLS utilisée est : **TLS 1.0.**
- ✓ The connection to this site uses TLS 1.0 (an obsolete protocol), **RSA** (an obsolete key exchange), and **AES_128_CBC** with **HMAC-SHA1** (an obsolete cipher)

RSA :
Algorithme de signature et de hachage : **asymétrique.**

AES_128_CBC :
Algorithme de chiffrement : **symétrique.**

HMAC :
C'est pour l'authentification et l'intégrité des données.

SHA1 :
Fonctions de hachage.

- ✓ **Les objectifs :**
  L'Intégrité des données.

Firefox

- **https://www.facebook.com/**

  - ✓ La version de TLS utilisée est : **TLS 1.2.**
  - ✓ La suite chiffrante est **ECDHE_ECDSA** with P-256 (a strong key exchange), and **AES_128_GCM** (a strong cipher).

ECDHE :
Algorithme d'échange de clés diffie-helman **asymétrique.**

ECDSA :
Algorithme de signature et de hachage : **asymétrique.**

AES_128_GCM :
Algorithme de chiffrement : **symétrique.**

SHA256 :
Fonctions de hachage.

- ✓ **Les objectifs :**
  L'Intégrité des données, la non répudiation, Garantir la confidentialité et l'authentification.

- **https://dpt-info-sciences.univ-rouen.fr**

  - ✓ La version de TLS utilisée est : **TLS 1.2.**
  - ✓ La suite chiffrante est **ECDHE_ECDSA** with P-256 (a strong key exchange), and **AES_128_GCM** (a strong cipher).

ECDHE :
Algorithme d'échange de clés diffie-helman **asymétrique.**

RSA :
Algorithme de signature et de hachage : **asymétrique.**

AES_128_GCM :
Algorithme de chiffrement : **symétrique.**

SHA256 :
Fonctions de hachage.

✓ **Les objectifs :**

L'Intégrité des données, la non répudiation, Garantir la confidentialité et l'authentification.

https://www.globetrotter.ch/

✓ La version de TLS utilisée est : **TLS 1.0.**

DHE :

Protocole diffie helman pour échange de clé : **asymétrique.**

RSA :

Algorithme de signature et de hachage : **asymétrique.**

AES_128_CBC :

Algorithme de chiffrement : **symétrique.**

SHA128 :

Fonctions de hachage.

✓ **Les objectifs :**

L'Intégrité des données.

OpenSSL

- **https://www.facebook.com/**

  - ✓ La version de TLS utilisée est : **TLSv1**
  - ✓ L'algorithme de chiffrement est **AES128.**
  - ✓ La fonction de hachage est **sha 2.**

- **https://dpt-info-sciences.univ-rouen.fr**

  - ✓ La version de TLS utilisée est : **TLSv1**.
  - ✓ L'algorithme de chiffrement est **AES256.**
  - ✓ La fonction de hachage est **sha 2**.
  - ✓ L'algorithme de signature est **RSA**.
  - ✓ L'algorithme d'échange de clé est diffie helman **DHE.**

- **https://www.globetrotter.ch/**

  - ✓ La version de TLS utilisée est : **TLSv1**.
  - ✓ L'algorithme de chiffrement est **AES256.**
  - ✓ La fonction de hachage est **sha 2**.
  - ✓ L'algorithme de signature est **RSA**.
  - ✓ L'algorithme d'échange de clé est diffie helman **DHE.**

Remarque :

- Pour les algorithmes de chiffrement **symétriques** ils sont par bloc.
- Pour les algorithmes de chiffrement **asymétrique** les problèmes mathématiques sur lesquels sont fondés leur sécurité est le logarithme discret ainsi que la factorisation des nombres premiers.
  « difie helman résiste au problème de logarithme discret, RSA résiste au problème de factorisation de deux nombre premiers ».

- Les déférents objectifs sont :

  - ➢ L'intégrité des données, la non répudiation, l'authentification et la confidentialité.
  - ➢ L'intégrité des données est assurée par la fonction de hachage.
  - ➢ La confidentialité est assurée par le chiffrement.
  - ➢ L'authentification c'est grâce au HMAC.
  - ➢ Le non répudiation est assuré par l'algorithme de signature.

3. Ce n’est pas les mêmes suites chiffrantes entre les différents navigateurs et OpenSSL.

// Utilisation de whireshark

Les suites chiffrantes utilisées peuvent différer d'un client a l'autre parce que l'ors de l'envoie de paquet client_hello une liste de suites chiffrantes est envoyées au serveur par ordre de priorité, et le serveur réponds en choisissant la plus prioritaires des suites supporter par le serveur et le client, donc deux client différents peuvent envoyés des suites chiifrantes différentes selon leurs système.

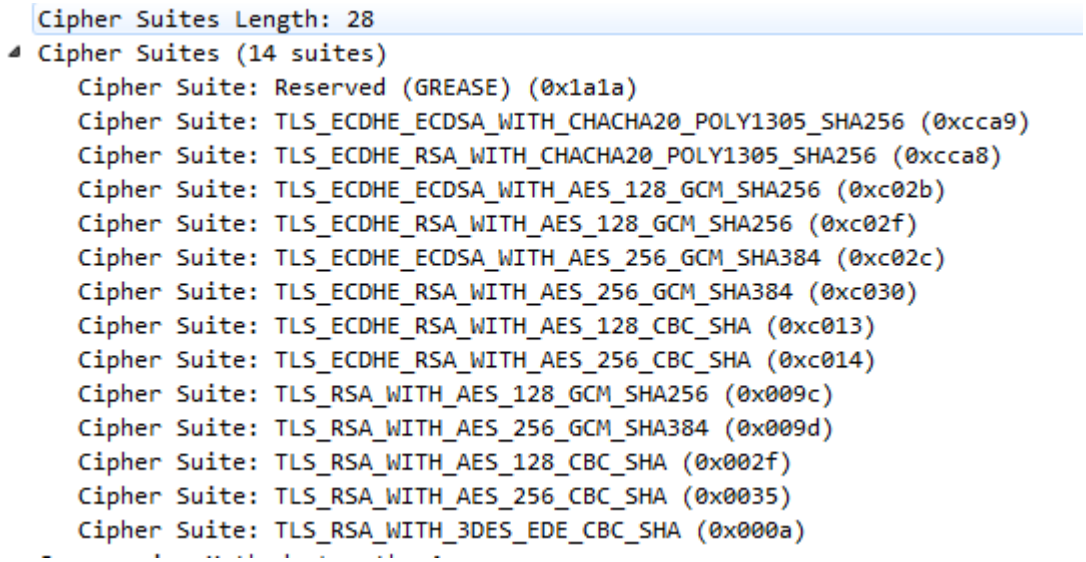


Figure 1 : Liste de suites chiffrantes supportées par le client.

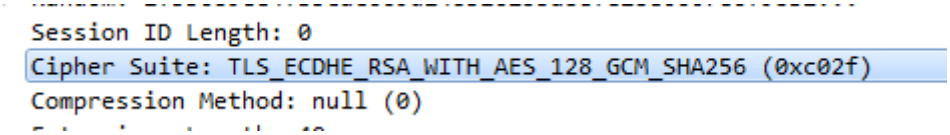


Figure 2 : La suite chiffrante choisis par le serveur.

4. Un **master-key** sert à l'échange des clés, cette dernières est générée aléatoirement.

5. L'erreur code 20 signifie qu’on ne peut pas déterminer le certificat de l'émetteur local, parce que on possède aucune certificat.

Exercices 02

- La version de OpenSSL.

```
OpenSSL 0.9.8h 28 May 2008
```

- Les caractéristiques de la machine.

Processeur :	Intel(R) Pentium(R) CPU 2020M @ 2.40GHz 2.40 GHz
Mémoire installée (RAM) :	6,00 Go (5,90 Go utilisable)
Type du système :	Système d'exploitation 64 bits

1. Pour chaque algorithme, commentez et comparez avec les autres. Vous pouvez comparer :

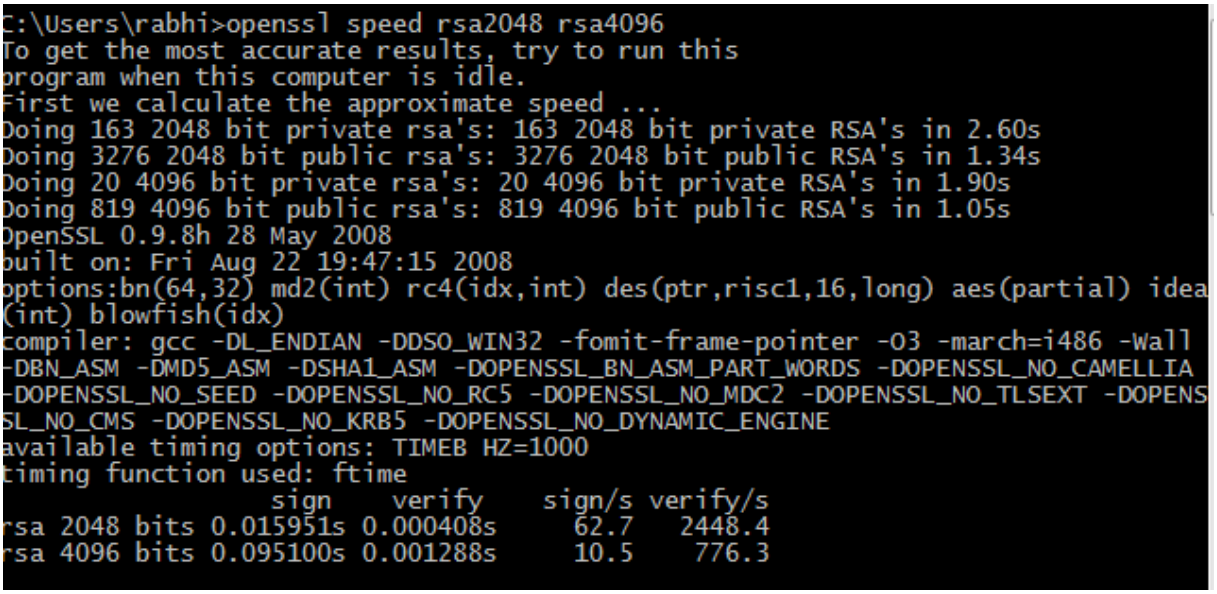
   - La vitesse des algorithmes symétriques entre eux.

```
C:\Users\rabhi>openssl speed des
To get the most accurate results, try to run this
program when this computer is idle.
First we calculate the approximate speed ...
Doing des cbc 20971520 times on 16 size blocks: 20971520 des cbc's in 11.88s
Doing des cbc 5242880 times on 64 size blocks: 5242880 des cbc's in 8.85s
Doing des cbc 1310720 times on 256 size blocks: 1310720 des cbc's in 10.30s
Doing des cbc 327680 times on 1024 size blocks: 327680 des cbc's in 10.91s
Doing des cbc 40960 times on 8192 size blocks: 40960 des cbc's in 11.70s
Doing des ede3 6990506 times on 16 size blocks: 6990506 des ede3's in 11.00s
Doing des ede3 1747626 times on 64 size blocks: 1747626 des ede3's in 7.65s
Doing des ede3 436906 times on 256 size blocks: 436906 des ede3's in 10.64s
Doing des ede3 109226 times on 1024 size blocks: 109226 des ede3's in 8.98s
Doing des ede3 13653 times on 8192 size blocks: 13653 des ede3's in 8.95s
OpenSSL 0.9.8h 28 May 2008
built on: Fri Aug 22 19:47:15 2008
options:bn(64,32) md2(int) rc4(idx,int) des(ptr,risc1,16,long) aes(partial) idea
(int) blowfish(idx)
compiler: gcc -DL_ENDIAN -DDSO_WIN32 -fomit-frame-pointer -O3 -march=i486 -Wall
-DBN_ASM -DMD5_ASM -DSHA1_ASM -DOPENSSL_BN_ASM_PART_WORDS -DOPENSSL_NO_CAMELLIA
-DOPENSSL_NO_SEED -DOPENSSL_NO_RC5 -DOPENSSL_NO_MDC2 -DOPENSSL_NO_TLSEXT -DOPENS
SL_NO_CMS -DOPENSSL_NO_KRB5 -DOPENSSL_NO_DYNAMIC_ENGINE
available timing options: TIMEB HZ=1000
timing function used: ftime
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
des cbc          28244.47k    37927.47k    32577.12k    30747.21k    28669.20k
des ede3         10164.31k    14628.31k    10508.07k    12451.01k    12502.28k
```

Remarque :

On constate que l'algorithme DES_CBC est plus rapide en chiffrement que le DES_EDE3 exmp :
DES_CBC chiffre plus de 20 millions de fichiers de taille 16 octet alors que DES_EDE3 ne chiffre que 7 millions dans une durée de 11 secondes.

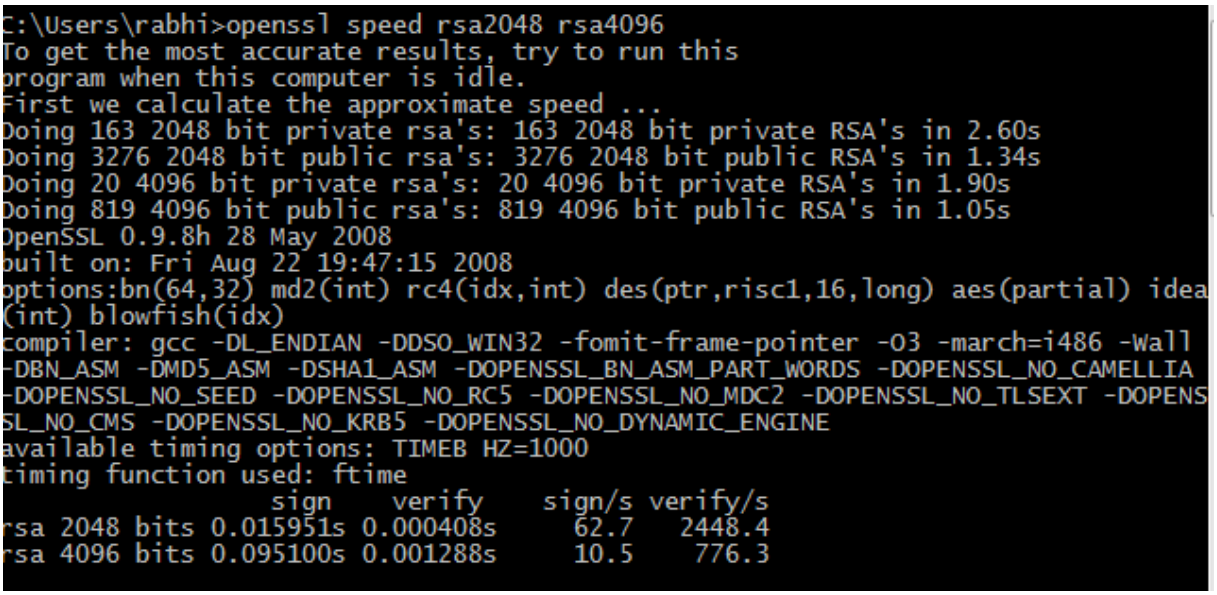
- La vitesse des algorithmes asymétriques entre eux.



Remarque :

On constate que l'algorithme RSA2048 est plus rapide en chiffrement que le RSA4096 et de même en déchiffrement exmp :
RSA2048 chiffre 163 bits en 2 secondes 60 par contre RSA4096 ne chiffre que 20 bits dans une durée de 1 secondes 90.

- La vitesse des algorithmes asymétriques en signature ou vérification de signature.



Remarque :

On constate que l'algorithme RSA2048 est plus rapide en chiffrement que le RSA4096 et de même en déchiffrement exmp :

RSA2048 signe plus de 62 bits en 1 secondes par contre RSA4096 ne signe que 10.5 bits dans une durée de 1 secondes.

- La vitesse de l'AES en fonction de la taille de clef.

```
C:\Users\rabhi>openssl speed aes-128-cbc aes-192-cbc
To get the most accurate results, try to run this
program when this computer is idle.
First we calculate the approximate speed ...
Doing aes-128 cbc 10485760 times on 16 size blocks: 10485760 aes-128 cbc's in 3.
65s
Doing aes-128 cbc 2621440 times on 64 size blocks: 2621440 aes-128 cbc's in 3.33
s
Doing aes-128 cbc 655360 times on 256 size blocks: 655360 aes-128 cbc's in 3.04s

Doing aes-128 cbc 163840 times on 1024 size blocks: 163840 aes-128 cbc's in 3.41
s
Doing aes-128 cbc 20480 times on 8192 size blocks: 20480 aes-128 cbc's in 3.62s
Doing aes-192 cbc 10485760 times on 16 size blocks: 10485760 aes-192 cbc's in 4.
08s
Doing aes-192 cbc 2621440 times on 64 size blocks: 2621440 aes-192 cbc's in 4.16
s
Doing aes-192 cbc 655360 times on 256 size blocks: 655360 aes-192 cbc's in 3.22s

Doing aes-192 cbc 163840 times on 1024 size blocks: 163840 aes-192 cbc's in 2.38
s
Doing aes-192 cbc 20480 times on 8192 size blocks: 20480 aes-192 cbc's in 4.18s
OpenSSL 0.9.8h 28 May 2008
built on: Fri Aug 22 19:47:15 2008
options:bn(64,32) md2(int) rc4(idx,int) des(ptr,risc1,16,long) aes(partial) idea
(int) blowfish(idx)
compiler: gcc -DL_ENDIAN -DDSO_WIN32 -fomit-frame-pointer -O3 -march=i486 -Wall
-DBN_ASM -DMD5_ASM -DSHA1_ASM -DOPENSSL_BN_ASM_PART_WORDS -DOPENSSL_NO_CAMELLIA
-DOPENSSL_NO_SEED -DOPENSSL_NO_RC5 -DOPENSSL_NO_MDC2 -DOPENSSL_NO_TLSEXT -DOPENS
SL_NO_CMS -DOPENSSL_NO_KRB5 -DOPENSSL_NO_DYNAMIC_ENGINE
available timing options: TIMEB HZ=1000
timing function used: ftime
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128 cbc      46028.03k    50427.46k    55115.69k    49142.40k    46307.52k
aes-192 cbc      41110.55k    40358.95k    52086.98k    70551.79k    40127.28k
```

Remarque :

La vitesse de chiffrement est plus grande lorsque la taille de la clef est petite.

.

- La vitesse de l'AES si vous utilisez l'EVP : openssl speed -evp aes-128-cbc

```
C:\Users\Guettouche>openssl speed  aes-128-cbc
To get the most accurate results, try to run this
program when this computer is idle.
First we calculate the approximate speed ...
Doing aes-128 cbc 41943040 times on 16 size blocks: 41943040 aes-128 cbc's in 6.83s
Doing aes-128 cbc 10485760 times on 64 size blocks: 10485760 aes-128 cbc's in 6.56s
Doing aes-128 cbc 2621440 times on 256 size blocks: 2621440 aes-128 cbc's in 6.50s
Doing aes-128 cbc 655360 times on 1024 size blocks: 655360 aes-128 cbc's in 6.47s
Doing aes-128 cbc 81920 times on 8192 size blocks: 81920 aes-128 cbc's in 6.53s
OpenSSL 0.9.8h 28 May 2008
built on: Fri Aug 22 19:47:15 2008
options:bn(64,32) md2(int) rc4(idx,int) des(ptr,risc1,16,long) aes(partial) idea(int) blowfish(idx
compiler: gcc -DL_ENDIAN -DDSO_WIN32 -fomit-frame-pointer -O3 -march=i486 -Wall -DBN_ASM -DMD5_ASM
BN_ASM_PART_WORDS -DOPENSSL_NO_CAMELLIA -DOPENSSL_NO_SEED -DOPENSSL_NO_RC5 -DOPENSSL_NO_MDC2 -DOPE
SL_NO_CMS -DOPENSSL_NO_KRB5 -DOPENSSL_NO_DYNAMIC_ENGINE
available timing options: TIMEB HZ=1000
timing function used: ftime
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128 cbc      98184.15k   102315.69k   103212.65k   103723.13k   102801.57k
```

```
C:\Users\Guettouche>openssl speed -evp aes-128-cbc
To get the most accurate results, try to run this
program when this computer is idle.
First we calculate the approximate speed ...
Doing aes-128-cbc 20971520 times on 16 size blocks: 83886080 aes-128-cbc's in 14.57s
Doing aes-128-cbc 20971520 times on 64 size blocks: 20971520 aes-128-cbc's in 13.36s
Doing aes-128-cbc 20971520 times on 256 size blocks: 5242880 aes-128-cbc's in 13.09s
Doing aes-128-cbc 20971520 times on 1024 size blocks: 1310720 aes-128-cbc's in 12.97s
Doing aes-128-cbc 20971520 times on 8192 size blocks: 163840 aes-128-cbc's in 12.97s
OpenSSL 0.9.8h 28 May 2008
built on: Fri Aug 22 19:47:15 2008
options:bn(64,32) md2(int) rc4(idx,int) des(ptr,risc1,16,long) aes(partial) idea(int) blowfish(idx
compiler: gcc -DL_ENDIAN -DDSO_WIN32 -fomit-frame-pointer -O3 -march=i486 -Wall -DBN_ASM -DMD5_ASM
BN_ASM_PART_WORDS -DOPENSSL_NO_CAMELLIA -DOPENSSL_NO_SEED -DOPENSSL_NO_RC5 -DOPENSSL_NO_MDC2 -DOPE
SL_NO_CMS -DOPENSSL_NO_KRB5 -DOPENSSL_NO_DYNAMIC_ENGINE
available timing options: TIMEB HZ=1000
timing function used: ftime
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128-cbc      92100.27k   100477.41k   102558.06k   103459.28k   103507.16k
```

Remarque :

La vitesse de l'AES est plus grande sans l'utilisation de l'option EVP.

Exercices 03

- On suivant les étapes annoncé et en utilisant la commande openssl enc -d -base64 -in joint.txt -out groupes.pdf on a pu récupérer un fichier PDF similaire à celui trouvant sur notre boite mail.
- La commande ne demande pas de mot de passe car l'algorithme Base64 ne nécessite pas un password.
- Le fichier est constitué d'un ensemble de caractères qui sont :
  De 'A' a 'Z', De 'a' a 'z', {0.1…..9} et les symboles {+, /}.
- La taille du fichier décodé est plus petite que la taille du fichier encodé.
- L'utilité de base 64 est de transmettre les fichiers Binaire qui sont en pièce jointe dans un mail d'une manière optimal.

Exercices 04

```
C:\Users\Guettouche\Desktop\M 2 GIL\Web Sec\Nouveau dossier\Web_et_s-curit--TP01>openssl enc -p -aes128 -a -in clair1.txt -out chiffre1.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
salt=ADF71C15F54D75C0
key=5A623A4DD09B3307F0FE8A223900A698
iv =EA4B2A6D631B4D285FA805A5FF2F3093
```

1.

- openssl enc -p -aes128 -a -in clair1.txt -out chiffre1.txt
  enc : commande pour encoder un fichier.
  -p : pour afficher le (salt, key, iv).
  -aes128 : algorithme de chiffrement.
  -a : c'est pour indiquer à openSSL que les données cryptées sont en mode base64.
  -in : pour indiquer les données à cryptées.
  Clair1.txt : le fichier contenant le texte en claire.
  -out : pour indiquer le fichier contenant les données chiffrées.
  chiffre1.txt : le fichier contenant le texte chiffré.

- **SALT :** est utilisé pour dériver une clé du mot de passe.

KEY : c’est la clef pour chiffrer le mot de passe.
IV : fait à peu près la même chose qu'un sel, et permet d’utiliser le même mot de passe pour crypter plusieurs messages différents.

2. On constate que le salt et le key et le iv ont changé car le salt et le iv sont générer aléatoirement.

3. On constate que le fichier chiffré a une taille supérieure à celle du fichier clair,Est cela revient au fait que le fichier chiffré a étais générer à partir d’un algorithme aes128 et comporte le salt et le key et le iv.

4. La clef secrète est utilisée pour chiffrer et déchiffrer un message donné.