Fiche TP - Hash, chiffrement, utilisation et limitations.

Cette fiche TP Vise à accompagner le cour sur le hachage et le chiffrement. Le contenu sera déroulé en parallèle du cour pour illustrer le propos.

> **Prérequis :** Les TP ci-dessous seront réalisés sur openSSL.

Installation sur debian : `apt install openssl`

Installation sur windows : `winget install openssl`

TP1 - Hashage

Objectif :

- Comprendre le fonctionnement des fonctions de hachage.
- Voir les différences entre les algorithmes.
- Observer un effet "avalanche"

Exercice 1 : Génération des empreintes et tests de hash

Créer un fichier texte qui contiendra un message simple :

```
echo "Bonjour les BTS SIO/CIEL !" > message.txt
```

```
C:\Users\ochabran\Downloads\ssl>echo "Bonjour les BTS SIO/CIEL !" > message.txt

C:\Users\ochabran\Downloads\ssl>dir
 Volume in drive C is OS
 Volume Serial Number is AEFA-B6D4

 Directory of C:\Users\ochabran\Downloads\ssl

02/02/2026  02:12 PM    <DIR>          .
02/02/2026  02:11 PM    <DIR>          ..
02/02/2026  02:12 PM                31 message.txt
               1 File(s)             31 bytes
               2 Dir(s)  233,901,596,672 bytes free
```

Calculer un premier hash :

```
openssl dgst -sha1 message.txt
```

```
C:\Users\ochabran\Downloads\ssl>openssl dgst -sha1 message.txt
SHA1(message.txt)= 24bd1e66049b0444c914f96a34742585f259c887
```

copier / coller le fichier en message1.txt et calculer également son hash.

```
C:\Users\ochabran\Downloads\ssl>openssl dgst -sha1 message.txt
SHA1(message.txt)= 24bd1e66049b0444c914f96a34742585f259c887

C:\Users\ochabran\Downloads\ssl>openssl dgst -sha1 message1.txt
SHA1(message1.txt)= 24bd1e66049b0444c914f96a34742585f259c887
```

> **info** : On constate bien que si même donnée d'entrée, même donnée de sortie.

Tester ensuite avec des algorithmes plus complexe ayant une clé suppérieure.

```
openssl dgst -sha256 message.txt
openssl dgst -sha512 message.txt
```

```
C:\Users\ochabran\Downloads\ssl>openssl dgst -sha1 message.txt
SHA1(message.txt)= 24bd1e66049b0444c914f96a34742585f259c887

C:\Users\ochabran\Downloads\ssl>openssl dgst -sha256 message.txt
SHA2-256(message.txt)= 9c2a9ea5c98627002c48fec5c431331db4e928788b2c724f261ea2c435546872

C:\Users\ochabran\Downloads\ssl>openssl dgst -sha512 message.txt
SHA2-512(message.txt)= b3e33e5b7ada48733b8cfe547c7199121be0605cb52e48b6bb7fd12e8d6e2632f48b8bd813eaeb54ba5d6726ce1f0517f
fd34867c77fa474b885f713aa72a593
```

> **Info** : On constate bien la différence de complexité d'un algo à l'autre.

Tester maintenant l'effet "Boule de neige".

Modifier un caractère dans le fichier, en changeant par exemple le '!' en '?'. Recalculer le hash.

```
C:\Users\ochabran\Downloads\ssl>openssl dgst -sha1 message.txt
SHA1(message.txt)= 24bd1e66049b0444c914f96a34742585f259c887

C:\Users\ochabran\Downloads\ssl>openssl dgst -sha1 message.txt
SHA1(message.txt)= 00efb0cbc8cfbde27ed88a5b03dbddbe10aa9d10
```

> **Info** : On constate bien que toute modification, aussi infime soit-elle, entraîne une modification conséquente du hash.

Exercice 2 : Limitation du hash sur les mots de passe faibles.

Chiffrer un mot de passe avec un algorithme volontairement vulnérable dans un fichier texte :

```
echo -n "azerty" | md5sum
```

```
root@STE-OCH-PO01:/mnt/c/Users/ochabran# echo -n "azerty123" | md5sum
882baf28143fb700b388a87ef561a6e5  -
```

Copier la chaîne de caractères sans le ' -' dans un fichier texte appelé par exemple 'password.txt'

Installer '***hashcat***'.

Récupérer ou éditer un dictionnaire simple (il y en a un fourni en pièce jointe de cette page).

```
root@STE-OCH-PO01:/mnt/c/Users/ochabran/Downloads/ssl# ll
total 0
drwxrwxrwx 1 root root 512 Feb  2 14:50 ./
drwxrwxrwx 1 root root 512 Feb  2 14:11 ../
-rwxrwxrwx 1 root root  73 Feb  2 14:48 dictionnaire.txt*
-rwxrwxrwx 1 root root  89 Feb  2 14:24 message.sha256*
-rwxrwxrwx 1 root root  26 Jan 23 16:16 message.txt*
-rwxrwxrwx 1 root root  31 Feb  2 14:12 message1.txt*
-rwxrwxrwx 1 root root  33 Feb  2 14:46 password.txt*
root@STE-OCH-PO01:/mnt/c/Users/ochabran/Downloads/ssl#
```

Pour casser le hash, il suffit d'exécuter la commande suivante :

```
hashcat -m 0 password.txt dictionnaire.txt
```

```
Approaching final keyspace - workload adjusted.

ab4f63f9ac65152575886860dde480a1:azerty

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
Hash.Target......: ab4f63f9ac65152575886860dde480a1
Time.Started.....: Mon Feb  2 14:53:58 2026 (0 secs)
Time.Estimated...: Mon Feb  2 14:53:58 2026 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (dictionnaire.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:     3102 H/s (0.11ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 10/10 (100.00%)
Rejected.........: 0/10 (0.00%)
Restore.Point....: 0/10 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: root -> toto
Hardware.Mon.#1..: Util:  8%

Started: Mon Feb  2 14:53:57 2026
Stopped: Mon Feb  2 14:54:00 2026
```

> **Info** : On voit ici que le mot de passe a été correctement deviné et que cela à pris moins d'une seconde.

Nous pouvons en conclure que l'utilisation de mots de passe simple et d'algorithmes obsolètes est à proscrire.

TP2 - Chiffrement symétrique

Objectif :

- Comprendre AES et les modes de chiffrement.
- Manipuler une clé symétrique.
- Observer les différences entre ECB et CBC.

Exercice 1 : chiffrer et déchiffrer un fichier

> **Conseil** : Proposer aux étudiants de chiffrer un message, puis d'échanger leurs messages et se donner la clé sur un support à part ou à l'oral.

Chiffrer le fichier 'message.txt'. Il va demander un mot de passe qui fera office de clé de chiffrement.

```
openssl enc -aes-256-cbc -salt -in message.txt -out message.enc
```

```
C:\Users\ochabran\Downloads\ssl>openssl enc -aes-256-cbc -salt -in message.txt -out message.enc
enter AES-256-CBC encryption password:

Verifying - enter AES-256-CBC encryption password:

*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

```
root@STE-OCH-PO01:/mnt/c/Users/ochabran/Downloads/ssl# cat message.enc
Salted__◆t◆
          ◆◆◆◆◆◆◆iX◆◆}]◆4◆ښsI◆o◆
```

Déchiffrer le message :

```
openssl enc -aes-256-cbc -d -salt -in message.enc -out message.dec
```

```
root@STE-OCH-PO01:/mnt/c/Users/ochabran/Downloads/ssl# openssl enc -aes-256-cbc -d -salt -in message.enc -out message.de
c
enter AES-256-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

```
root@STE-OCH-PO01:/mnt/c/Users/ochabran/Downloads/ssl# cat message.dec
"bonjour les BTS ciels"
```

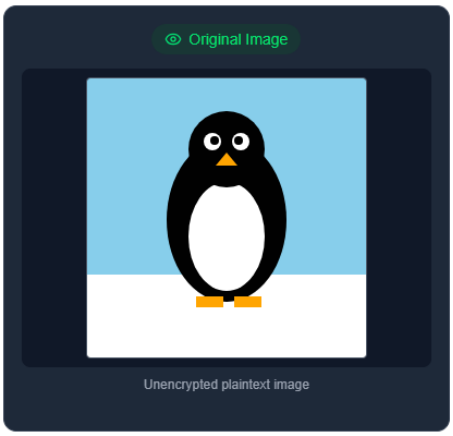
Exercice 2 : ECB vs CBC

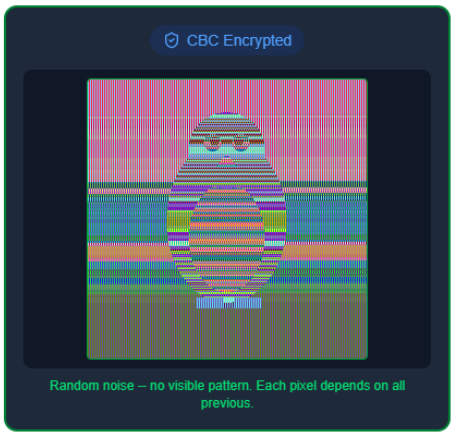
Selon les cas, certaines méthodes de chiffrements sont plus efficaces que d'autres.

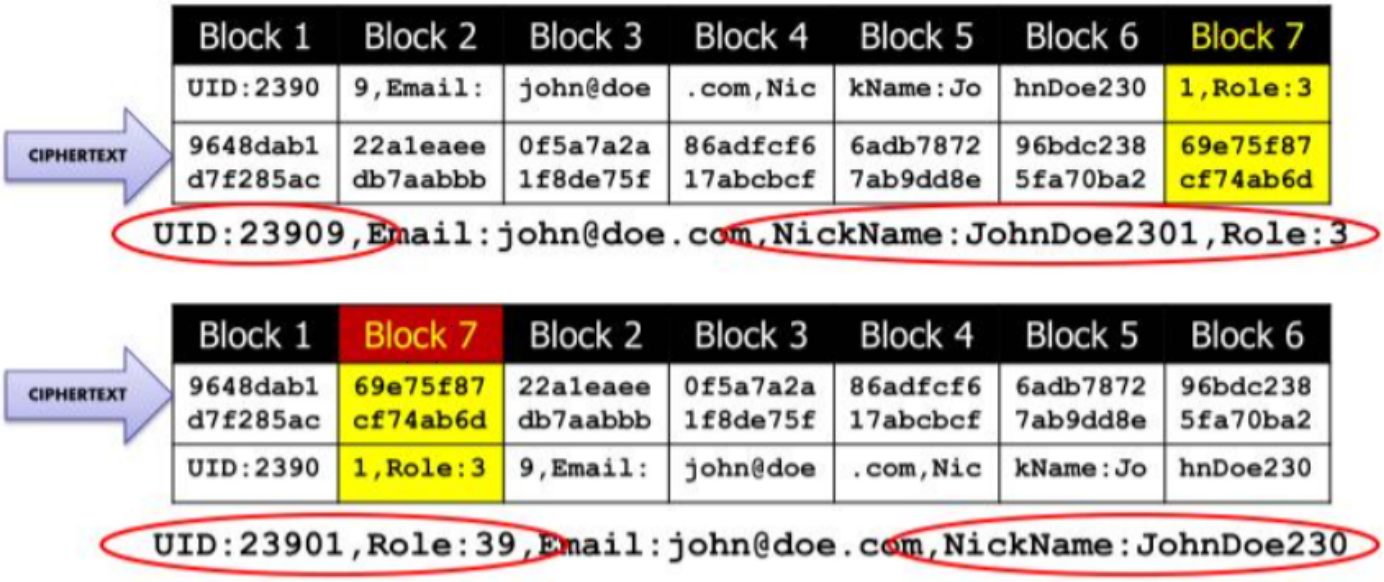
Récupérer l'image fournie dans le TP ou la télécharger. Puis créer deux version chiffrées de cette image.

```
openssl enc -aes-256-ecb -in Tux-PNG-Pic.png -out image1.enc
openssl enc -aes-256-cbc -in Tux-PNG-Pic.png -out image2.enc
```

Le résultat est qu'il est facile avec le SCB, d'isoler des paterns avec un editeur hexadécimal ou avec un visualiseur d'images chiffrées.







TP3 - Chiffrement asymétrique

Objectif :

- Générer une paire de clés RSA.
- Chiffrer/déchiffrer.
- Signer/vérifier.

> **Conseil** : Proposer aux étudiants d'échanger les clés publiques avec un binôme, puis de se faire passer un message chiffré.

Exercice 1 : Générer un couple de clé privée/publique

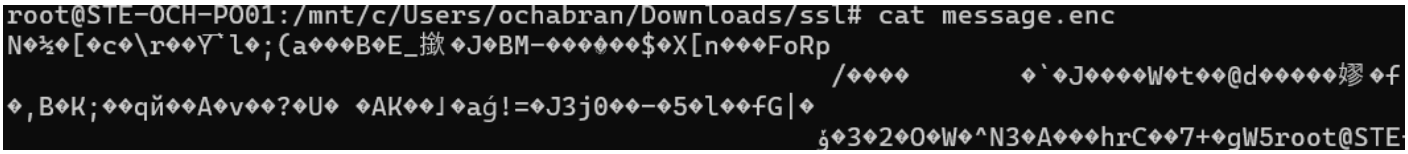
Générer le couple de clé avec les commandes suivantes :

```
openssl genrsa -out private.key 2048
openssl rsa -in private.key -pubout -out public.key
```

Échanger les clé ou procéder soi-même au chiffrement / déchiffrement.

Chiffrer le message avec la clé publique :

```
openssl pkeyutl -encrypt -inkey public.key -pubin -in message.txt -out message.enc
```



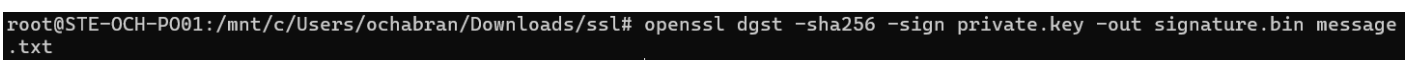
Déchiffrer ou faire déchiffrer par son binôme avec la clé privée :

```
openssl pkeyutl -decrypt -inkey private.key -in message.enc -out message_decrypt.txt
```

Exercice 2 : Signature numérique de document

Signer un document avec la clé privée :

```
openssl dgst -sha256 -sign private.key -out signature.bin message.txt
```



```
root@STE-OCH-PO01:/mnt/c/Users/ochabran/Downloads/ssl# cat signature.bin
◆◆+ ◆k1fw◆◆◆◆F◆X◆7◆E^7(◆{5◆◆◆D
                          '◆U◆◆
                               ◆◆◆◆GTN◆6r
◆◆
  ◆◆◆M1◆◆◆!o◆◆◆M◆◆ ◆◆◆◆<◆eVL◆◆◆◆◆◆◆y◆◆◆1◆DL1u◆◆4
                                                ◆◆c◆◆
◆* ◆C◆odFM◆!x◆◆L'|+◆!$◆_◆◆~◆
```

La signature (.bin) et le message (.txt) sont transmis au binôme ou vérifié par soi-même avec la clé publique.

```
openssl dgst -sha256 -verify public.key -signature signature.bin message.txt
```

```
root@STE-OCH-PO01:/mnt/c/Users/ochabran/Downloads/ssl# openssl dgst -sha256 -verify public.key -signature signature.bin
message.txt
Verified OK
```

> **Info** : Le message 'verified OK' signifie que l'identité de l'émeteur du fichier est vérifié. Mais également que le message en question n'a pas été altéré.

En effet, faire une modification sur le message.txt et relancer la commande :

```
root@STE-OCH-PO01:/mnt/c/Users/ochabran/Downloads/ssl# openssl dgst -sha256 -verify public.key -signature signature.bin
message.txt
Verification failure
40373D822A720000:error:02000068:rsa routines:ossl_rsa_verify:bad signature:../crypto/rsa/rsa_sign.c:430:
40373D822A720000:error:1C880004:Provider routines:rsa_verify:RSA lib:../providers/implementations/signature/rsa_sig.c:77
4:
```

Revision #2
Created 2026-07-17 22:33:10 UTC by Olivier
Updated 2026-07-17 22:35:43 UTC by Olivier