- TP Cryptographie -
Confidentialité, Intégrité et Authenticité des Données

Rudy Gabert - Rémy Barras - Jérémy Rincker - Marie Michel

Sommaire

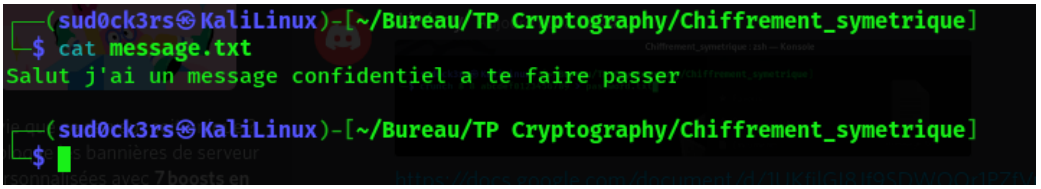
1. Chiffrement symétrique

1.1 AES

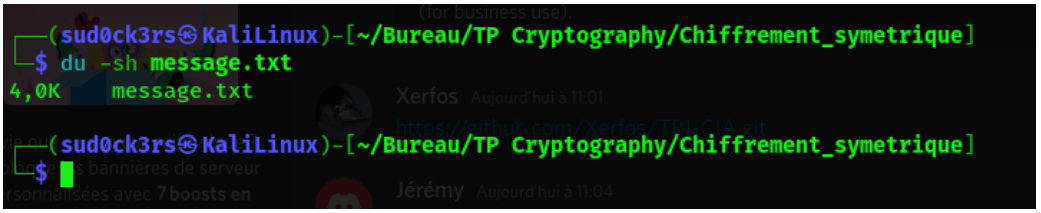
a. Chiffrement

Fichier léger :

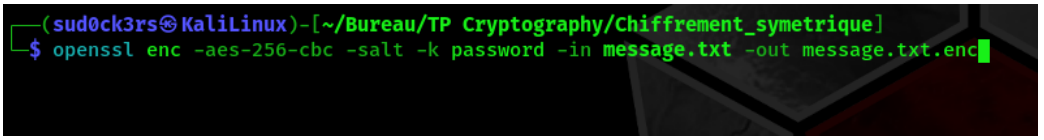
Message court à chiffrer :



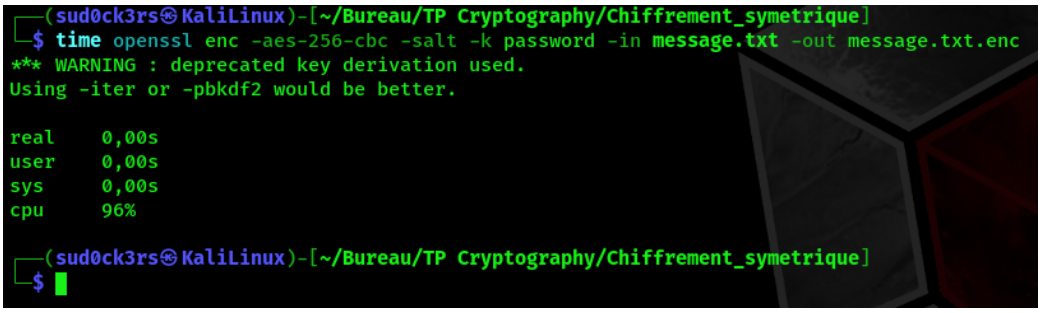
On peut voir avec la commande **du** la taille du fichier, qui est ici très léger :



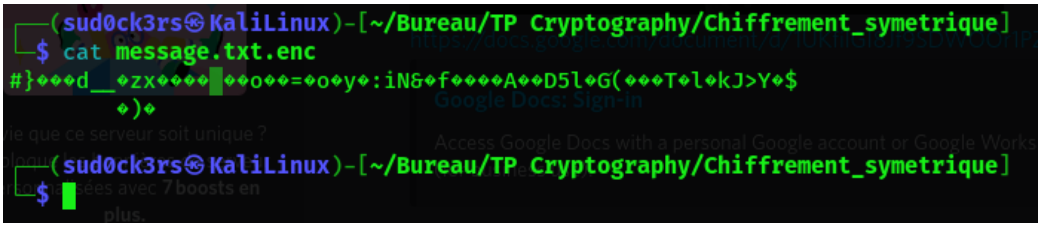
Chiffrement du fichier avec aes-256-cbc avec **openssl** :



On peut voir avec la commande **time** que l'encryption est instantanée (0s)
à ce niveau de précision:

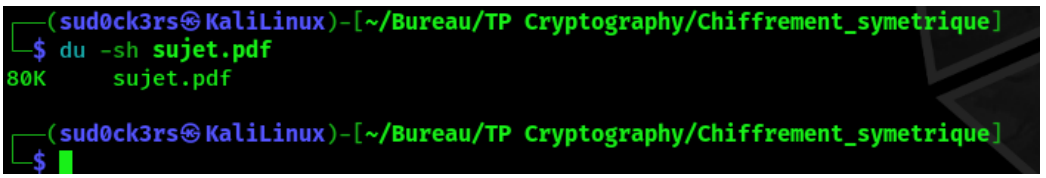


Le fichier est maintenant chiffré, voici son contenu :

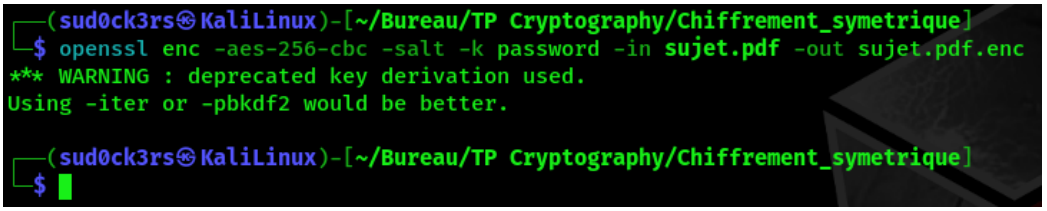


Fichier de taille moyenne type .PDF

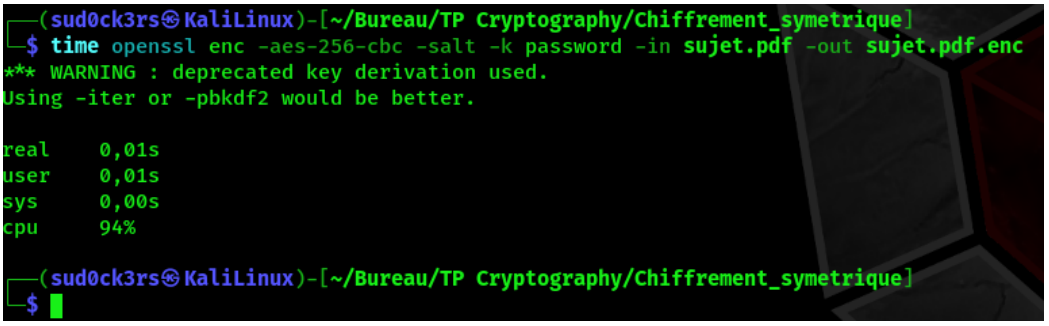
Même opération sur le sujet du TP :



Chiffrement du fichier en aes-256 avec **openssl** :

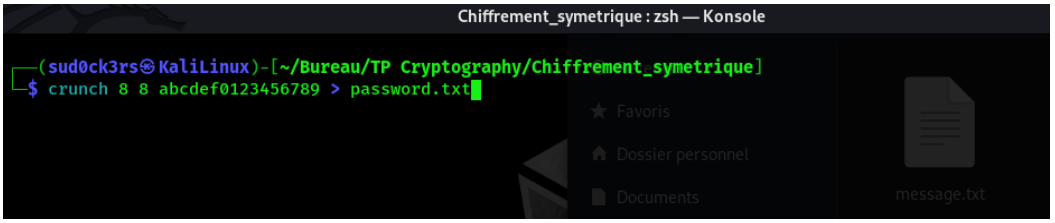


Commande **time**, le chiffrement est également presque instantané : (0.01s)



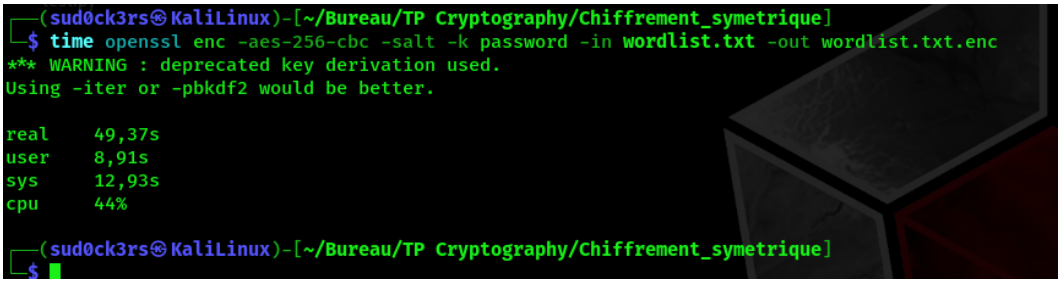
Fichier lourd

Même manipulation, mais avec un fichier lourd de plusieurs GigaBits, créé à partir de la commande **crunch** :



Nous avons généré une wordlist de 5.5 GB. Ce fichier nous permettra de mieu voir la différence de temps de chiffrement et de déchiffrement en symétrique et asymétrique.

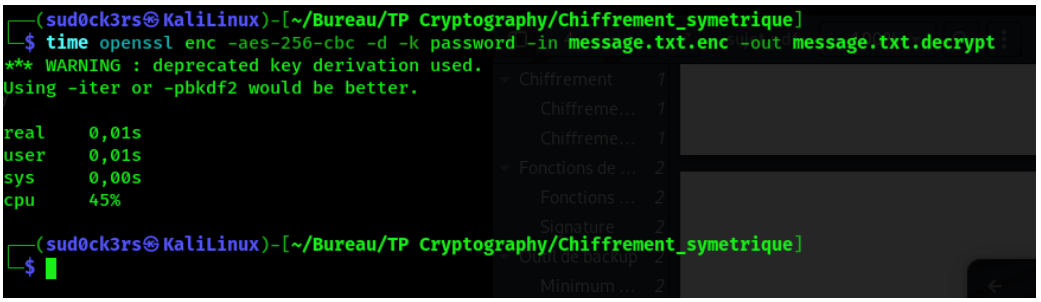
On observe que dans ce cas, le chiffrement est plus long, mais à échelle humaine il reste très court : **49,37s**



b. Déchiffrement

Fichier léger :

Déchiffrement du fichier message.txt.enc :



On observe que le déchiffrement d'un fichier si léger est presque instantané.

Fichier lourd :

On observe que pour le déchiffrement, le temps est légèrement plus long mais toujours acceptable : **56,80s**

Tableau comparatif :

Taille fichier	Durée chiffrement (secondes)	Durée déchiffrement (secondes)
4 ko	0,00s	0,00s
80 ko	0,01s	X
5,5 G	49,37s	56,80s

2. Chiffrement asymétrique

1.1 RSA

a. **Chiffrement**
b. **Déchiffrement**
c. **Tableau Comparatif**

3. Hachage

3.1 SHA256

Fichier lourd (iso Win10), on remarque que cela est rapide **17,11s**

```
daag-jeremy@jere:~/Téléchargements$ time cat Win10_21H2_French_x64.iso | openssl dgst -sha256
SHA2-256(stdin)= 2cc9731ee278666a632bdf5944105fc5f215f59ced98d75aeccf8185bd5bca3a

real    0m17,114s
user    0m14,399s
sys     0m6,651s
daag-jeremy@jere:~/Téléchargements$ du -sh Win10_21H2_French_x64.iso
5,4G    Win10_21H2_French_x64.iso
```

3.2 MD5

Même fichier, avec cet algorithme nous somme à **11,15s**

```
daag-jeremy@jere:~/Téléchargements$ time cat Win10_21H2_French_x64.iso | openssl dgst -MD5
MD5(stdin)= 74f15a7aaa171b91be317f1adf1cf815

real    0m11,158s
user    0m8,672s
sys     0m6,436s
daag-jeremy@jere:~/Téléchargements$ du -sh Win10_21H2_French_x64.iso
5,4G    Win10_21H2_French_x64.iso
```

3.3 SHA1

Ici, **10,63s**

```
daag-jeremy@jere:~/Téléchargements$ time cat Win10_21H2_French_x64.iso | openssl dgst -sha1
SHA1(stdin)= 37e5b758dea5256932f164d36dc0f7e88a747825

real    0m10,639s
user    0m6,991s
sys     0m6,363s
daag-jeremy@jere:~/Téléchargements$ du -sh Win10_21H2_French_x64.iso
5,4G    Win10_21H2_French_x64.iso
```

2.4 Tableau comparatif

Taille de fichier	SHA256	MD5	SHA1
5,4G	17,11s	11,5s	10,63s

On remarque donc que le temps diffère selon l'algorithme de hashage (plus long si plus puissant), néanmoins la différence n'étant pas impactante à l'utilisation on préfèrera utiliser SHA256 car plus fiable.
En effet les hashs md5 et sha1 sont connus pour être contournables facilement par brute force, il y a même des outils en ligne avec d'immenses banques de hashs pour les contourner.

Fichier modifié/corrompu :

On peut observer qu'avec une seule modification d'un caractère (rajout d'un "4" au début du texte, l'empreinte hash est complètement différente , le hash assure donc une intégrité de la donnée.

```
daag-jeremy@jere:~/Bureau$ cat texte.txt
4Lorem ipsum dolor sit amet. Est nobis accusamus aut sint corporis ut ipsa iste. Ut enim voluptatem a autem eius ut ra
tione corporis ea totam autem sit distinctio voluptatem cum fugiat autem ut natus iste. Qui iste assumenda ut cupidita
te sunt et perferendis autem aut autem asperiores sit numquam repellat. Eos sequi fuga eos tempore voluptatibus nam do
loribus tempora qui deserunt rerum.
daag-jeremy@jere:~/Bureau$ time cat texte.txt | openssl dgst -sha256
SHA2-256(stdin)= 91143011d7bad522f3177fe0c4d63dd23afe355fcbc948f5c22042f8e3ad520d

real    0m0,006s
user    0m0,008s
sys     0m0,001s
```

On a signé le fichier texte.txt, avec les commandes :
> gpg --gen-key
> gpg --sign texte.txt
Exemple d'une vérification d'authenticité et d'intégrité avec une clé publique :

```
daag-jeremy@jere:~/Bureau$ gpg --verify texte.txt.gpg
gpg: Signature faite le jeu. 10 nov. 2022 19:33:24 CET
gpg:                avec la clef RSA AA5E1F77350588EB9BCAA77391166C792FD8C5B6
gpg: Bonne signature de « Jérémy Rincker <exo@test.com> » [ultime]
```

Ici nous avons créé un hash du texte.txt et l'avons chiffré avec une clé privée RSA.
L'outil gpg nous permet d'assurer de manière fiable la traçabilité des données.
L'utilisateur devra déchiffrer le hash avec la clé publique pour le comparer avec le hash du fichier en clair, ce qui garantit l'intégrité.