AGUILAR_Flavien_Cryptographie_TP

Flavien AGUILAR

12/10/2020

Chiffrement :

Générateur aléatoire :

a) Un générateur aléatoire est un dispositif permettant de générer une suite de nom où de caractère dont il n'existe aucun lien déterministe.

b) Pour générer un élément aléatoire de 16 bits, nous allons utiliser la commande :

```
[miiruki@detp:~/Cryptographie/TP $] openssl rand -base64 2
S3M=
```

Nous utilisone l'option -base64 2 car nous générons 2 octets de caractères ce qui correspond à 16bits.

Pour générer une chaine aléatoire de 256 bits, nous allons utiliser la même commande mais avec l'option -base64 32 :

```
[miiruki@detp:~/Cryptographie/TP $] openssl rand -base64 32
Yy8lbv7VnH+McLDcdMn3Z07DKxU950VUqgKh5Rs0xmE=
```

Pratique :

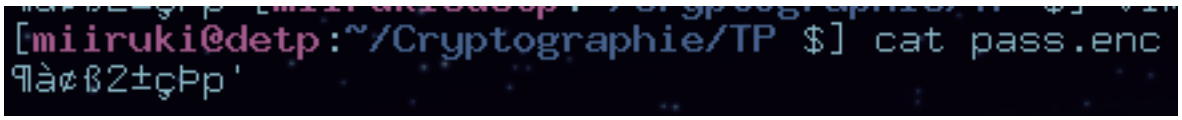
c) Pour générer une clé aes256 de 256 bits, nous allons utiliser la commande :

```
[miiruki@detp:~/Cryptographie/TP $] openssl aes256 -k bonjour -nosalt -
    P
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
key=2CB4B1431B84EC15D35ED83BB927E27E8967D75F4BCD9CC4B25C8D879AE23E18
iv =6FAAB891433D768BEE116110C816B4C2
```

Lorsque mon voisin tape la même commande, la clé est exactement la même. Nous pouvons en conclure que si le mot de passe est généré aléatoirement, la clé de chiffrement sera elle aussi aléatoire.

d) Le mot de passe controle bien la notion d'aléatoire car lorsque nous retrons un mot de passe aléatoire après l'option -k de la commande, la clé sera impossible à retrouver.

e) Nous pouvons chiffrer le contenu d'un fichier avec la commande :



f) Pour déchiffrer du texte, nous allons maintenant utiliser la commande :

```
[miiruki@detp:~/Cryptographie/TP $] openssl enc -aes256 -d -in pass.enc
    -out pass.decrypt -nosalt
enter aes-256-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better
```

Nous pouvons vérifier le contenu du fichier pass.decrypt :

```
[miiruki@detp:~/Cryptographie/TP $] cat pass.decrypt
alexandre le nul
```

g) Lorsque nous générons une clé avec un salt, le contenu de la clé change à chaque fois :

```
[miiruki@detp:~/Cryptographie/TP $] openssl aes256 -k bonjour -P
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=493AB31B0D2E7372
key=D3B4386A43E8B7A9AD7F3760234F61AD8BB69CF485CCC144B8E03B1ED30AB16B
iv =FE08E302D2AFE8D5F87CCCCA4AEEF66C

[miiruki@detp:~/Cryptographie/TP $] openssl aes256 -k bonjour -P
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=DD7BE5A004BBB056
key=093A0A4B93C11B96767BCFF9F4A2DA7185561970DEAB002A6B7BDAA661D52F72
iv =405DCA767089BB46F38BFAA5C9F900B
```

h) Le vecteur d'initialisation dépend aussi du salt car il se modifie aussi en fonction du salt.

i) Le générateur aléatoire est controlé par le salt qui va changer à chaque fois.

j) Lorsque nous visualisons le contenu d'un fichier encrypté avec le salt :

Nous voyons que la chaine de caractère commence par Salted__ .

Le salt est affiché en clair dans le hexdump du fichier :

```
[miiruki@detp:~/Cryptographie/TP $] hexdump pass.enc
0000000 6153 746c 6465 5f5f c0df fcf6 e076 41c9
0000010 6f98 a536 b894 bcd0 03e8 b312 c911 55b6
0000020 c751 ea3e 1a4d 08c6 8248 d341 f5b7 f74d
0000030
```

La première ligne correspond bien à notre salt, mais les octets sont stockés différement dans notre mémoire d'ou l'affichage ci-dessus.

Clés asymétriques :

a) Pour générer une clé RSA de 2048 bits, nous allons utiliser la commande :

```
[miiruki@detp:~ $] openssl genrsa -out key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
...........................+++++
.....................................................................................................

e is 65537 (0x010001)
```

Nous pouvons visualiser le contenu du fichier key :

```
[miiruki@detp:~ $] cat key
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAvrQ0PYwsirkiqAi5+8CsKT6bJtW7KvAGjctTPwOu16Z5/qXr
KfTO3mrDjuTpowlhzZUW5q351fxgg86yBgKUk911e5ukS628bymKcgIzvszUmpOn
Pndkrp0knfpbzuuqydp/D8SnHTzay5Blt1V7WXM2YlP5iYTbHvpci1qPg+0lSJxk
34P7akQLCqxTuYJ+uh5J7ao1EaJhxS7BmhqVH32hahxSWHtmauu6jYAnE7ZuRlQN
zqWWz6Ysj00TbiAmLf/N5gRs5sepaToSjzdateVvjy4KAzzDkE/fUA0WiMPzOKFj
iXl20rtuas7DdP64caFnSdcrxT0kb4uqrAH99QIDAQABAoIBAFfoNZSRgw7HXKRc
sW7NWA8DHv390/NtBXhjW5JWnkgmI2IxFYAXqyx3xoGeH9zZvE3KBe58EAiF9++K
XiHLGrY2hatfCZMhGRWc7oazYSr4vKAspBF1c+HR6kkwWRW/KNE3eQ7JtccSgEEJ
5Pln3B5Vr6IP151+sotLGVHbgwBqiuDEu2fa+eEuSiilonXXx6wO33H8W9kQnaqJ
aSh4QTNarkWsCDfV76vQJYqBsTxAXH38DsqZ6YSm0X48YDz7X4eHuy6/cljkFtzo
WWSSuJ63YBDSNimW2+I2DwB6dTs5+COI6o5jGeZwIaVCg/MyzxBWf9IcdWQeet0q
UTgWNgECgYEA4VhTgm/xHRNOJnXzJUghWCc5sYJhyp3/3RmHv9MLFThYcsCgjyXm
JdENhyqwTcfCqeqz3TPwlFTOoL/EM50AtzFxz9HJDIuKrMrkssuFSqpqA1WzEECQ
A4+z6gMWDpsKErM5KpcbmWWze94uzWgN4QehJ8CldHq/cjVrub4+wTUCgYEA2KV/
EN5TatZohJz+6cYeG77RUA+HJFNcoqROA/KsPOBzp6JULD4bOw+oiCM2YvdM20TK
xhA8VrTv3YZwXQjM5G3BvujfoJ7XGOnK9PvpRO1u7OjFgTX2roluX+qJDsLnuhH7
alJ/sNbsDN/wo/TrQ2oFiN1oFugo3OwRr0BwocECgYEA34XMeBLvuOaAW3gd8bol
kRp/iMsKk3+736XGWIlPcgINAe5+lN+D9bjy0UOP0+c0+XScauttVMT2nM77IDM5
FX3PJdQyt+XmbVVoGs3z7kcxlxM35gQUlePCE3pyM5SoHjp2lv+CWdteeJzAEfGH
6tG1kgfuVqa3JeNFct35w40CfzA8qQ4YcnpXQu5EBn4Gnq2OzwwEaRKtfPK5oqU+
```

```
Yxsq3pV0j0lKisWbtuqY7IMcjQBFUODIvKaRotawGFSeoQA7b+gR8bjRpRLm+0mQ
zV+LfidBrLgYS9SQxQKWNe4Td4T2fAQwWj4FnIlBU1MuNTPNZ24R3hs7ld2aslVT
A8ECgYEAyQj5+Y/B/VvfIXD3VVNsj1f5ACZ1L5NN/m9BRq02quMH2Q9A7czbl9pz
TpeT2mNSlhp05/EBKvEd/Hi3b6hs45DIES7xABY5wc7gm37FV9xAWt8vSIgf5Xr9
HCr74OXgaP/Q8/nbDZqqhJfLxQ0rGcLBgr5PmwgKf3OteZWyTI0=
-----END RSA PRIVATE KEY-----
```

b) Pour générer une clé RSA chiffrée avec une passphrase, nous allons utiliser la commande :

```
[miiruki@detp:~/Cryptographie/TP $] openssl genrsa -des3 -out key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.................+++++
.............+++++
e is 65537 (0x010001)
Enter pass phrase for key:
Verifying - Enter pass phrase for key:
```

Nous pouvons visualiser le contenu du fichier :

```
[miiruki@detp:~/Cryptographie/TP $] cat key
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,6C23ED368B11C290

kOTO/k1XQLjNM7XQLX/2XcZT2ZA1w2t/Fy5sOV7zcK01+ECXXxnDlkagI6gkGqci
9fRb1s9NhclcMLrHQ+67kFbGRyjzbARJ0mVyl8D3ZP2Q83hTHZlBwv9bIwEBS0Py
lGxmK/7NjVILh5toHpo/6QHY6uLB74yut98lhu2PhVCXcguWP/G8OLOlFl99lHOW
kDjE0YKJlAWiTuHkoKVKrF5cB5Xhg2ZOULnOJAaz1JaiG8/3jMULco67x+LjICx4
96GjWKX9NFvXjKGPhvde6dRrtFqagL9yFg7iF9PZkCOWKyqA+PJtaRj2R/Kz1nXN
6Sq55mFudy5t96ADMu2ielR9QMH+HvBo0TGM0TvFq8bKxeqFEuPQu0v7f1xUStGI
SPVC01G208C7Sz23KYWLyLLZpVHsI/zgjF8ZSPwKLTjJx47RLS5NTdef8X956DNf
JZzQ3Q7WWh32E54rUf6BSVNDm6OIVKaC7G5zWvAZkTRv9G88gdhFFo9NDERWR/6T
WJnQ3cBnkLnnnv3WHexOT+uLOadtq6gWGm+v6ukKJyWu/vq94OMbdd0YiBNmgb0g
BNPL1IIsaEc26xw+4cHaYG8DN40HRhX/HpjGLTWgrY9//9kAqUMjDlZ2psL/y6d5
g38POQgPjVx0jDt5ncjyCAW103PIuDmHdWOS3thqvsWLcfgziGmjyRnku8kKQ363
KYvLcIp9F2ncheDIgcFwummvvkNTlzqH9xqqH46T2G/R7H9hiie+C5/0mPMiB9nn
jLTvvaZrty/7lWwzhhoA8Aj0Me6C98tmjb1nR+DR+0Z/FbN+FxrDlJ2D4qVZ5Q++
cRBrZVlqGolugP/MsBdfOi/bkSnRqf3VUrVbuJHZlQe/pCddcTZwJ4gv9OqlQ4e+
NsKgYIArBAnKlskfRxXw+85D24WQVGgZFQTgLD7S9OvosUrUyRKEbi0/mlYKBOoU
saHuA6Gj7/CEJ8nAg6Pc0DcgCnLo4JARlS+WZ8tESAwTcFa8+62a7eL0l3yVTpU1
2D4Q3LjEKzeSip7QqXzBgVlyVmFI+8j8jcReKIBNQUyR0yczuFjot2FhKJxUK0iJ
fJHJw9/n1M9mmLl0jKXITTFb4DpE3/hKrYZ85o5xTh+NiAlGR69X0LoQ7Wf9GpC0
X1TKx14tLsdwY6hB0hWMc9D1DOFlzpY/KHmzo3sLneOp4dQHoWG/lce47/tCng8Q
YRXswZNovqRqYyMbJ/iDAsyq7qf8icDIR4YE/FMS5hsAq8kD2K9qQ7n7hqdSlgBH
Y5PKhPkzcj4D9Vcr185Is9aSK+YRKZCBcOREaPYZmFiu3HNbR9j2sE2EMlRzUaP+
v9pU98NMEwvHVOe3dx688OodLYQG8UN6qak1cnBddUiEc+k2lAmgnHgvmW3AQ/jO
```

```
Sg2kaDyuyjOxuzXiAhK2kw6Bbp4rl0RJwYaGy4msq2Tv7Yst1t041DcUyu2Qq0XU
cz1cXCziKWQC4jjIq7NGlNz9CaFhCxCPAn+KlCaWz40X/UIUkxL0N4G6xJ6Z4eEZ
eiF5YGU77br5dHvA2kIINjWS+91sIft8nHBeCLrdWPLkYN/oddYaXg==
-----END RSA PRIVATE KEY-----
```

c) Nous pouvons générer la clé publique à partir de la clé privé avec la commande :

```
[miiruki@detp:~/Cryptographie/TP $] openssl rsa -in key -pubout -out
    key.public
Enter pass phrase for key:
writing RSA key
```

d) Nous allons maintenant chiffrer un message avec la clé publique :

```
[miiruki@detp:~/Cryptographie/TP $] openssl rsautl -in pass.txt -out
    pass.enc -inkey key.public -pubin -encrypt
```

Nous pouvons maintenant décrypter le fichier à partir de la clé publique :

```
[miiruki@detp:~/Cryptographie/TP $] openssl rsautl -in pass.enc -out
    pass.decrypt -inkey key -decrypt
Enter pass phrase for key:
```

Nous pouvons vérifier le contenu du fichier :

```
[miiruki@detp:~/Cryptographie/TP $] cat pass.decrypt
fichier en clair
```

Message en digest :

a) Nous calculons l'empreinte du fichier pass.txt :

```
[miiruki@detp:~/Cryptographie/TP $] openssl dgst pass.txt
SHA256(pass.txt)= 2
    d9cf5c002e9d6a13b37486f22c6dc5105d609e07a9f7476a7c9de3e4c5e3c12
```

b) Nous modifions un seul bit à l'intérieur du fichier et nous recalculons l'empreinte :

```
[miiruki@detp:~/Cryptographie/TP $] openssl dgst pass.txt
SHA256(pass.txt)= 4
    cd0ec9459db25bde91bf2921fa03aacd571d71fe7784add8e1f55385da662b1
```

c) Si un pirate peut accèder à un fichier et à son empreinte, il pourra modifier l'empreinte pour ensuite faire passer un fichier pontentiellement malveillant. Pour remédier à ça, nous pouvons signer l'empreinte du fichier, de ce fait, même si l'attaquant arrive à récupérer l'empreinte et le fichier, il ne pourra pas reproduire notre signature et de ce fait ne pourra pas envoye de fichiers malveillants.

d) Nous générons et signons l'empreinte du fichier :

```
[miiruki@detp:~/Cryptographie/TP $] openssl dgst -out emp -sign key
    pass.txt
```

e) Nous pouvons vérifier la signature avec la commande :

```
[miiruki@detp:~/Cryptographie/TP $] openssl dgst -verify key.public -
    signature emp pass.txt
Verified OK
```

f) Si un pirate accède au fichier et à la signature, il va pouvoir modifier la signature du fichier de sorte à ce qu'elle corresponde à celle d'unfichier modifié.

g) Sur un serveur dont les fichiers systèmes risquent d'être compromis, les empreintes signées des fichiers vont permettre de vérifier l'authenticité des donénes. Il faudrait idéallement stockés ce signatures sur un serveur distant dont les fichiers ne risquent pas d'être compromis.