

## Cryptographie Symétrique Serhrouchni

Ces travaux pratiques sont basés sur Openssl.  
Openssl est l'Api de crypto la plus répandue. Elle est open source avec licence GPL. On trouvera toutes les informations la concernant à l'adresse : <http://www.openssl.org>  
A titre d'exemple Openssl intègre également le protocole SSL/TLS. C'est cette même distribution qu'intègre le serveur Apache pour offrir les services SSL/TLS.  
Openssl offre un mode commande où l'on peut interagir avec l'API. C'est ce mode que l'on utilisera tout simplement en tapant openssl. Malheureusement la documentation est suffisamment pauvre. Pour accéder à l'usage des différentes commandes, il suffira lorsqu'on est sur openssl de taper le nom de la commande suivie de -help. Vous trouverez également sur le site un manuel relatif aux commandes de Openssl.  
L'objectif de ces travaux pratiques est de vous familiariser avec les services de base de la sécurité, chiffrement et hachage.

Notation toutes les commandes et arguments associés seront notés dans la suite en **italique gras** (exemple : openssl> **genrsa -out mykey -des3 1024** )

### Partie 1 : Chiffrement symétrique

La commande qui vous permet d'utiliser le chiffrement symétrique est la commande **enc** (en tapant **enc -help** vous aurez toutes les options associées)

#### **Question 1 :**

Soit un fichier donné **fic** (choisissez un fichier qui contient des données textuelles). Ecrire la commande qui permet de le chiffrer et produit ainsi un fichier **fic.enc**

On peut alors vérifier que le message **fic.enc** est bien inintelligible...

Ce même message est transmis à votre camarade qui pourra également le déchiffrer.

On peut alors vérifier que le message ainsi déchiffré est bien identique à **fic** ...

Il vous est bien sûr de comprendre l'ensemble des manipulations associées.

Il vous est demandé de diversifier les algorithmes. D'utiliser également l'option -a pour produire un fichier chiffrer lisible donc codé en base64 (lisible ne veut pas dire en clair).

#### **Solution :**

**Openssl> aes128 -in fic -out fic.enc -e**

**Openssl> aes128 -in fic.enc -out fic.dec -d**

Diversifier les algorithmes. Utiliser par exemple l'option -a pour produire un fichier lisible donc codé en base64

### Question 2 :+

Chiffrez le fichier exécutable **openssl.exe** avec un algorithme symétrique et un mode quelconques, puis déchiffrez le cryptogramme associé.

### Solution:

**aes128 -in openssl.exe -out ope -e -k 123456789**

Ouvrez avec notepad+ ope, choisissez le mode d'affichage hexa. Vous constaterez dans l'entête les 16 premiers octets qui composent le **Salted**. Sur la base de celui-ci et l'option **-k**, sont générés la clé et le vecteur d'initialisation sur 128 bits.

```
C:\>cd OpenSSL-New
C:\OpenSSL-New>cd bin
C:\OpenSSL-New\bin>aes256 -in openssl.exe -out ope -e -k 123456789
'aes256' n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.
C:\OpenSSL-New\bin>openssl
OpenSSL> aes256 -in openssl.exe -out ope -e -k 123456789
OpenSSL> aes256 -in openssl.exe -out ope -e -k 123456789 -p
salt=E3A15EDEEF717DED
key=34E2A2CF7C045E7E3E6AA45C4CD3FC4741B6FC6E02E8C7951240D65988D28CCC
iv =BE63A3CD179D5D69FE69158B82570FFD
OpenSSL> aes256 -out ope.exe -in ope -d -iv BE63A3CD179D5D69FE69158B82570FFD -K
34E2A2CF7C045E7E3E6AA45C4CD3FC4741B6FC6E02E8C7951240D65988D28CCC
OpenSSL> md5 ope.exe openssl.exe
MD5(ope.exe)= d7952814f94b5282d6e26d415b803007
MD5(openssl.exe)= d7952814f94b5282d6e26d415b803007
OpenSSL> aes256 -in openssl.exe -out opes -e -iv BE63A3CD179D5D69FE69158B82570FFD
-K 34E2A2CF7C045E7E3E6AA45C4CD3FC4741B6FC6E02E8C7951240D65988D28CCC
OpenSSL> md5 ope opes
MD5(ope)= e4e8bee08d94a9de7c35edff47ead642
MD5(opes)= e4e8bee08d94a9de7c35edff47ead642
OpenSSL>
```

### Question 3 :

Comparez les tailles des fichiers clairs et chiffrés (en supprimant le Salted). Donnez une explication sur la différence de ces tailles.

### Solution :

Editer le fichier chiffré avec **notepad+** et choisissez le mode Hexa. Vous observez au niveau des 16 premiers octets l'ajout du Salt sur la base duquel ont été calculé la clé et le vecteur d'initialisation.

### Question 4 :

Déchiffrer un cryptogramme en utilisant un mauvais mot de passe. Qu'observez-vous ?

**Question 5 :**

Chiffrer un fichier avec AES128 en mode CBC en utilisant les options iv et -K. Le vecteur d'initialisation et la clé doivent être donnés en hexadécimal.

Déchiffrer le cryptogramme obtenu.

**Question 6 :**

Même exercice que précédemment avec DES-EIE.

**Solution :**

Pas possible !!!

**Question 7 :**

Déchiffrer le fichier secret.enc. Il est chiffré en DES-CBC dont le vecteur d'initialisation est « ABABABABABABABAB » et la clé en base64 est « QUJBQkFCQUJBQkFCQUJBQg== ».

Que pouvez-vous dire de la clé et de son codage en base64.

**Question 8 :**

Donner des éléments temporels comparatifs entre les opérations de chiffrement et de déchiffrement et ensuite entre différents algorithmes (conseil : choisissez un fichier avec une taille conséquente). La commande time n'est pas supportée par Windows mais par Linux !!!

Utiliser la commande speed de openssl sous windows.

**Solution :**

***time openssl enc -in fichier -out fichier.enc -e -des3***

***time openssl enc -in fichier.enc -out fichier.dec -d -des3***

Idem avec -rc4, idea, des3 et ...

**Question 9 :**

Pourquoi quand on applique deux fois la commande **enc** une fois au fichier en clair et une fois au fichier chiffré on n'obtient pas un résultat en clair.

**Réponse :**

Au fichier chiffré on rajoute des entêtes qui ne font pas partie des données. Ainsi qu'on applique deux fois de suite la commande avec l'option **-e** il chiffre et les données et les entêtes. Quand on utilise l'option **-d** il applique la même fonction sauf qu'il extrait du chiffrement les données structurées.

**Question 10 :**

Chiffrer un fichier avec AES-128-CBC et déchiffrer avec AES-128-EDE en utilisant la même clé. Qu'observez-vous ?

## Commandes

**Openssl enc -ciphername [-in nomfichier] [-out nomfichier] [-pass arg] [-e] [-d] [-a] [-A]**

**[-k motdepasse] [-kfile nomfichier] [-S salt] [-K clé] [-iv IV] [-p] [-P] [-bufsize nombre]**

**[-debug]**

**-in nomfichier** --- Le fichier d'entrée, l'entrée standard par défaut.

**-out nomfichier** --- Le fichier de sortie, la sortie standard par défaut.

**-pass arg** --- le fichier des mots de passe d'entrée. Pour plus d'information sur le format de **arg** référez-vous à la section **ARG. DE MOT DE PASSE** d'[openssl](#)

**-salt** --- Utilise un ``salt" (littéralement rajouter du sel) dans les routines de dérivation de clé. Cette option devra **TOUJOURS** être utilisée, sauf si la compatibilité avec d'anciennes versions d'OpenSSL ou SSLeay est requise.

**-nosalt** --- Ne pas utiliser de ``salt" dans les routines de dérivation de clé. Ceci est la valeur par défaut pour assurer la compatibilité avec d'anciennes versions d'OpenSSL et SSLeay.

**-e** --- Encodage des données d'entrée : ceci est la valeur par défaut.

**-d** --- Décodage des données d'entrée.

**-a** --- Traitement base64 des données. Ceci signifie que lors d'un encodage, les données sont encodées base64 après le traitement et lors d'un décodage, les données sont décodées base64 avant le traitement.

**-A** --- avec l'option **-a**, le traitement base64 est effectué sur une ligne.

**-k motdepasse** --- le mot de passe à partir duquel la clé sera dérivée. Ceci est pour assurer la compatibilité avec d'anciens versions d'OpenSSL et SSLeay, mais remplacé par l'argument **-pass**.

**-kfile fichier** --- lit le mot de passe pour la dérivation de la clé à partir de la première ligne de **nomfichier**. Ceci est pour assurer la compatibilité avec d'anciens versions d'OpenSSL, mais remplacé par l'argument **-pass**.

**-S salt** --- Le salt à utiliser. Sous forme de chaîne de caractères composée de caractères hexadécimaux uniquement.

**-K clé** --- La clé à utiliser : sous forme de chaîne de caractères composée de caractères hexadécimaux uniquement. Si uniquement la clé est spécifiée, le IV doit être spécifié en plus en utilisant l'option **-iv**. Si une clé et un mot de passe sont spécifiés, la clé donnée avec l'option **-K** est prise et le IV est généré à partir du mot de passe. Cela manque probablement de sens que de spécifier et la clé et le mot de passe.

**-iv IV** --- Le IV à utiliser : sous forme de chaîne de caractères composée de caractères hexadécimaux uniquement. Si uniquement la clé est spécifiée avec l'option **-K**, le IV doit être défini explicitement. Si un mot de passe est spécifié, le IV est généré à partir de ce mot de passe.

**-p** --- affiche la clé et le IV utilisés.

**-P** --- affiche la clé et le IV utilisés puis sort immédiatement : aucun encodage ni décodage n'est fait.

**-bufsize nombre** --- affecte la taille du tampon d'entrée/sortie.

**-debug** --- débogue le BIOs utilisé pour l'entrée/sortie.

**openssl genrsa** [-out *nomfichier*] [-passout *arg*] [-des] [-des3] [-idea] [-f4] [-3] [-rand *fichier(s)*] [*nombrebits*]

**-out nomfichier**

Le fichier de sortie. Si cet argument n'est pas spécifié, la sortie standard est utilisée.

**-passout arg**

le fichier des mots de passe de sortie. Pour plus d'information sur le format de **arg** référez-vous à la section **ARGUMENTS DE MOT DE PASSE** d'openssl.

**-des|-des3|-idea**

Ces options encodent la clé privée avec l'encodage DES, triple DES, ou IDEA respectivement avant la sortie du résultat. Si aucune de ces options n'est présente, aucun encodage n'est utilisé. Si l'encodage utilise un mot de passe, il est demandé à l'invité de commande s'il n'est pas précisé via l'argument **-passout**.

**-F4|-3**

L'exposant public à utiliser, soit 65537 ou 3. La valeur par défaut est 65537.

**-rand fichier(s)**

Un ou plusieurs fichiers contenant des données aléatoires servant à initialiser le générateur de nombres aléatoires, ou encore un socket EGD. Plusieurs fichiers peuvent être spécifiés utilisant un caractère dépendant de l'OS. Ce séparateur est ; pour MS-Windows, , pour OpenVMS, et : pour tous les autres.

*nombrebits*

La taille de la clé privée à générer en bits. Cette option doit être placée en dernier. Si elle n'est pas présente, une valeur de 512 est utilisée.

**openssl dgst** [-md5|-md4|-md2|-sha1|-sha|-mdc2|-ripemd160|-dss1] [-c] [-d] [-hex] [-binary] [-out *nomfichier*] [-sign *nomfichier*] [-verify *nomfichier*] [-prverify *nomfichier*] [-signature *nomfichier*] [*fichier...*]

**-c**

affiche le sommaire par groupe de deux chiffres séparés par colonnes, uniquement applicable si le format de sortie **hex** est employé.

**-d**

affiche l'information de débogage BIO.

**-hex**

sommaire est à afficher en hexadécimal brut. Ceci est le cas par défaut pour les sommaires ``normaux" en opposition aux sommaires digitaux.

**-binary**

affiche le sommaire sous forme binaire.

**-out nomfichier**

nom de fichier de sortie, la sortie standard par défaut.

**-sign nomfichier**

signature digitale du sommaire utilisant la clé privée trouvée dans ``nomfichier".

**-verify nomfichier**

vérifie la signature utilisant la clé publique de ``nomfichier". Le résultat est soit ``Verification OK" soit ``Verification Failure".

**-prverify nomfichier**

vérifie la signature utilisant la clé privée de ``nomfichier".

**-signature nomfichier**

la signature à vérifier.

**-rand fichier(s)**

un ou plusieurs fichiers contenant des données aléatoires servant comme racine au générateur de nombres pseudoaléatoires, ou alors un socket EGD. Plusieurs fichiers peuvent être spécifiées utilisant un caractère dépendant de l'OS. Ce séparateur est ; pour MS-Windows, , pour OpenVMS, et : pour tous les autres.

**fichier...**

le fichier ou les fichiers à traiter. Par défaut, l'entrée standard est utilisée.