Travaux pratiques

OPENSSL

Objectifs

La cryptologie est la science du secret. Elle se compose de deux discipline: la cryptographie et de la cryptanalyse. La cryptographie est l'art de la conception des écritures secrètes tandis que la cryptanalyse est l'art de les casser. L'objectif de ce TP n'est pas de faire de vous des experts en cryptologie (le cours 4MMCRY est la pour ça) mais de vous faire découvrir OPENSSL pour comprendre les grands principes de la cryptographie: *chiffrement (symétrique ou asymétrique), hachage, code d'authentification.*

Keywords and phrases Cryptographie, chiffrement, hachage
Digital Object Identifier TP1

1 Chiffrement symétrique

1.1 Construire une clef de chiffrement

Pour construire une clef de chiffrement symétrique, vous avez deux options. On peut utiliser la commande `dd` avec comme fichier d'entrée `/dev/random`:

```
dd if=/dev/random of=key1 bs=1 count=XX
```

avec `XX` le nombre d'octets que vous souhaitez ajouter à votre clef. Pour votre clef, il suffit d'afficher le fichier `key` avec la commande `hexdump`.
L'autre alternative est d'utiliser le générateur de nombre pseudo-aléatoire d'openssl.

```
openssl rand -out key2 XX
```

Pour visualiser votre clef, utiliser `hexdump` sur le fichier `key2`. Si vous souhaitez ne pas devoir passer par `hexdump` pour visualiser votre clef, il existe une option avec `rand` qui vous permet d'obtenir directement votre clef en hexadécimal.
⚠ Avec `dd` et `rand` il faut entrer un nombre d'octets et pas un nombre de bits !

1.2 Chiffrer un fichier

Nous allons apprendre à chiffrer un fichier de deux façons différentes avec la commande `enc` d'`openssl`. La première méthode est plutôt pénible pour un être humain mais elle est relativement simple à comprendre en terme de fonction mise en œuvre. La deuxième méthode est simple pour un être humain mais beaucoup plus difficile à comprendre.

1.2.1 Avec une clef et un vecteur

Construire une clef de 128 bits et un vecteur d'initialisation de 128 bits en utilisant la commande `rand` d'`openssl`.
Maintenant on peut chiffrer en utilisant la clef et le vecteur d'initialisation que vous avez générer, en utilisant la commande `enc` d'`openssl`.

```
openssl enc -e -cipher -in clear -out ciphertext -K KEY -iv IV
```

Avant d'utiliser cette commande, il faut bien faire attention aux options:

- `-cipher` est le nom du système de chiffrement que vous utilisez.
- `clear` est le nom du fichier que vous allez chiffrer.
- `KEY` est la clef en hexadécimal.
- `IV` est le vecteur d'initialisation en hexadécimal.

⚠ On ne veut pas des noms de fichier !

Après avoir réussi à chiffrer un fichier, il est important de vérifier qu'on peut effectivement le déchiffrer. Si chiffrer est réalisé avec l'option `-e`, déchiffrer est obtenu avec `-d`.

```
openssl enc -d -cipher -in ciphertext -out clear.1 -K KEY -iv IV
```

Il faut utiliser la même clef et le même vecteur que lors du chiffrement. Il ne reste plus qu'a vérifier que les fichiers `clear` et `clear.1` sont bien identiques avec `diff`:

```
diff clear clear.1
```

1.2.2 Avec un mot de passe

La syntaxe est beaucoup plus simple plus besoin de donner une clef et un vecteur d'initialisation en hexadécimal (le rêve pour l'utilisateur), il suffit juste de choisir un mot de passe.

```
openssl enc -e -cipher -in clear -out ciphertext
```

Le déchiffrement est réalisé avec l'option `-d` et en donnant le mot de passe utilisant lors du chiffrement.
Pour l'instant, vous ne savez rien sur les algorithmes de chiffrement que nous avons utilisés. Par une série d'exercice, nous allons essayer de comprendre comment fonctionne.

Exercice 1: Analyse différentielle

Commençons avec un peu *d'analyse différentielle* pour comprendre comment un chiffrement peut fonctionner. Nous allons créer un fichier que nous allons chiffrer. Puis nous allons changer un octet du fichier de départ et recommencer le chiffrement pour voir comment la différence c'est propagée dans le nouveau fichier chiffré. Dans les exercices 1 et 2, vous utiliserez toujours la méthode 1.2.1 de chiffrement (en donnant la clef et le vecteur d'initialisation). Vous disposez d'un script `differential.bash`, il vous faudra compléter ce script pour répondre aux questions suivantes.

Question 2 Que peut-on déduire sur le fonctionnement des chiffrement `-des-ecb` et `-aes-128-ecb` à partir de l'analyse différentielle ? (Les tailles de clef sont différentes pour ces deux systèmes)

Question 3 La même analyse est-elle concluante pour `rc4` ?

Question 4 Appliquer ce script au chiffrement `-des-cbc`. Que peut-on en déduire sur ces deux systèmes de chiffrement ?

Exercice 2: Propagation des erreurs

Un système de chiffrement permet de protéger la confidentialité des données et pas l'intégrité ! Il est quand même intéressant de voir l'impact d'une modification d'un fichier chiffré sur le déchiffrement.

Question 5 Renommer le fichier `differential.bash` en `error.bash`. Puis modifier `error.bash` pour pouvoir modifier un octet d'un fichier chiffré et ensuite le déchiffrer. Vous partirez une nouvelle fois d'un fichier `text` de 64 octets et de `text.enc` qui aura été chiffré avec `des-cbc`. Pour comparer les fichiers dans votre boucle, je vous recommande d'utiliser:

```
xxd text
xxd tmp.enc
echo "==================================="
```

Qu'observez vous pour `des.cbc` ?

Question 6 La même question pour `rc4` ?

Exercice 3: Mode opératoire

Les systèmes de chiffrement `-aes-128-ecb`, `-aes-256-cbc`, `-aes-128-ctr`... sont tous basés sur l'algorithme **AES** (**A**dvanced **E**ncryption **S**tandart). Le nombre qui suit indique la taille de la clef à utilisé en bits. Enfin le sigle qui termine le nom du système désigne le mode opératoire utilisé: ECB, CBC, CTR.... Ainsi `-camellia-256-ofb` vous indique que l'on utilise l'algorithme CAMELLIA avec des clefs de 256 bits en mode OFB.

Question 7 Analyser la structure du fichier `Q7` avec `hexdump`. Vous savez qu'il a été chiffré avec l'AES. À votre avis avec quel mode opératoire et quelle taille de clef a-t-il été chiffré ?

Question 8 Même question mais pour les fichiers `Q81` et `Q82`. Tous les deux ont été chiffrés avec l'AES et avec le même mode. À votre avis avec quel mode opératoire et quelle taille de clef a-t-il été chiffré ? Quelle erreur a-t-elle pu être commise ?

Question 9 Chiffrer le fichier `Q9` avec successivement `-aes-128-cbc` et `-aes-128-ctr` pour la même clef et le même vecteur d'initialisation pour obtenir les fichiers `Q9.cbc` et `Q9.ctr`. Êtes vous surpris par la taille de ces fichiers par rapport à la taille de `Q9` ? Pour obtenir la taille d'un fichier, vous pouvez utiliser `ls` ou `wc -c`.

Question 10 Que pensez vous de l'option `-nopad` par rapport à la question précédente ?

Question 11 Chiffrer le fichier `Q11` avec `-aes-128-ctr`. Vous obtenez un fichier chiffré `Q11.enc`. Chiffrer le fichier `Q11.enc` avec `-aes-128-ctr` avec la même clef et le même vecteur d'initialisation pour obtenir `Q11.enc.bis`. Êtes vous surpris du résultat obtenu ?

Exercice 4: Mot de passe

Maintenant plus besoin de donner une clef et un vecteur d'initialisation, vous pouvez utiliser un mot de passe.

Question 12 Chiffrer le fichier `Q12` avec `-aes-128-cbc` en entrant un mot de passe une première fois pour obtenir un fichier chiffré `Q10.enc`. Répéter cette opération avec le même mot de passe une seconde fois, vous obtiendrez un fichier chiffré `Q10b.enc`. Comparer les deux fichiers chiffrés ainsi obtenus. Le résultat est-il surprenant ?

Question 13 Quand vous chiffrez avec un mot de passe une clef et un vecteur d'initialisation sont-ils créés par `openssl` ou le mot de passe suffit-il à tout faire ? Existe-t-il une option de la commande `enc` d'`openssl` qui pourrait vous permettre de la vérifier ?

Question 14 Voici 2 fichiers `Q141` et `Q142` qui ont été chiffrés avec `-aes-128-cbc` et le mot de passe `qwerty` (*indice la version d'`openssl` utilisée pour chiffrer n'est certainement pas la même*). Que contiennent ces fichiers ? Que pensez vous de l'option `-md` ?

Exercice 5 (Niveau M2): Malléabilité

Question A Créer un fichier `A` qui se chiffre en un fichier `B` qui sera `pdf` valide en utilisant `-aes-128-ctr`. (Cette question a déjà été utilisée en examen.)

Question B Créer un fichier `A` qui se chiffre en un fichier `B` qui sera `pdf` valide en utilisant `-aes-128-ctr`. Les 16 premiers octets de `A` doivent être identiques à ceux de `B`.

Question C Créer un fichier `A` qui se chiffre en un fichier `B` qui sera un `pdf` valide. Utiliser le chiffrement `-aes-128-ctr`. Ce fichier `A` devra être détecter par la commande `file` comme étant `zip`.

2 Hachage et authentification

`openssl` permet aussi de calculer des empreintes comme `md5sum`. On peut même calculer des codes d'authentification.

2.1 Calcul d'une empreinte

Pour calculer l'empreinte d'un fichier c'est très simple (c'est vraiment la commande la plus simple d'`openssl`), on utilise la commande `dgst` au lieu de `enc`.

```
openssl dgst -hash filename
```

Quelques explications sur cette commande:

- `-hash` est le nom de la fonction de hachage cryptographique que vous souhaitez utiliser comme `-md5` ou `-sha256`.
- `filename` est le nom du fichier.

⚠ nous somme bien d'accord qu'une fonction de hachage a toujours des collisions!

2.2 Calcul d'un code d'authentification

On utilise la commande `dgst` avec l'option `-hmac`.

```
openssl dgst -sha256 -hmac KEY filename
```

`KEY` est la valeur de la clef (en hexadécimal).

Exercice 6: Effet avalanche

L'effet avalanche consiste à prendre un fichier et modifier un octet. Ensuite, on compare l'empreinte du fichier modifié avec l'empreinte d'origine.

Question 15 En vous inspirant de `differential.bash`, créez un script `avalanche.bash` pour analyser `-sha256`. Qu'observez vous sur les empreintes que vous avez obtenues ?

Question 16 Calculer le code d'authentification du fichier du sujet `TP3.pdf` avec `-hmac -sha256` avec la clef `5921c827c1bc5245f4f8f835b6253f8b`.