LAB 3 : La Cryptographie avec OpenSSL

Objectif

L'objectif de ce lab est de vous familiariser avec les services de base de la sécurité. Notamment le chiffrement symétrique, le chiffrement asymétrique, le hachage, la signature numérique et sa vérification, et la certification.

Ces travaux pratiques sont basés sur la suite logicielle OpenSSL.
OpenSSL est une boîte à outils cryptographiques implémentant les protocoles SSL et TLS qui offre une bibliothèque de programmation en C permettant de réaliser des applications client/serveur sécurisées s'appuyant sur SSL/TLS.

Rendu

Vous êtes invités à remettre, sur votre Google Classroom, Un compte rendu avec les commandes que vous avez lancées et les résultats obtenus avec des captures d'écran. Un seul rendu est à remettre par groupe.

1. CHIFFREMENT SYMETRIQUE

L'algorithme RC4

La commande qui vous permet d'utiliser le chiffrement symétrique est la commande

```
#openssl enc –in fichier_clair –out resultat_obtenu
```

- Créer le fichier fichier_nom_eleve contenant du texte clair

```
#gedit fichier_nom_eleve
```

- Chiffrer ce fichier avec l'algorithme RC4

```
# openssl enc –rc4 -in fichier_nom_eleve -out fichier_chiff_algo
```

NB : openSSL nous donne la possibilité de donner un mot de passe en entrée : étant donné le mot de passe, openSSL dérive une clé de chiffrement.

- Vérifier que le message dans fichier_nom_eleve.enc est bien inintelligible

Ce fichier chiffré est transmis à votre camarade qui pourra également le déchiffrer.

- Ecrire la commande qui permet de le déchiffrer et produit ainsi le fichier

  #openssl enc –RC4 -d –in fichier_chiff_algo -out fichier_dechiff_rc
- Vérifier alors que le message déchiffré est bien identique au fichier initial

  #diff fichier_nom_eleve fichier_dechiff_rc

L'algorithme DES

Pour chiffrer le fichier fichier_nom_eleve avec l'algorithme DES avec clé explicite :

openssl enc -des -in fichier_nom_eleve -out fichier_chiff_des -k 0123456789ABCDEF

- Quelle est la commande qui permet de déchiffrer fichier_chiff_des ?
- Vérifier que le fichier déchiffré est identique au fichier initial.

2. CHIFFREMENT ASYMETRIQUE

Le chiffrement asymétrique RSA demande l'utilisation d'une paire de clés : une clé publique et une clé privée qu'il faut tout d'abord les générer.

Génération de clé privée/publique RSA

En openSSL, les clés RSA générées sont stockées dans un fichier avec l'extension .pem (Privacy Enhanced Mail). L'instruction à utiliser est *openssl genrsa*.

Utiliser cette instruction pour :

- Générer une paire de clés de taille 1024 bits et stocker-la dans le fichier **rsakey.pem**

  #openssl genrsa -out rsakey.pem 1024
- Afficher le fichier en utilisant la commande **cat**. Qu'est ce que vous remarquez ?
- Une façon de visualiser les clés en format complet est d'utiliser la commande **rsa**. Afficher alors les clés en format hexadécimal, en supprimant la sortie normalement produite par l'instruction rsa.

  #openssl rsa –in rsakey.pem –text -noout
- Extraire la clé publique de la clé privée et sauvegarder le résultat dans le fichier **rsapubkey.pem**

  #openssl rsa –in rsakey.pem –pubout –out rsapubkey.pem

Chiffrement de la clé RSA par l'algorithme

Nous allons maintenant utiliser l'algorithme AES256 pour chiffrer la clé privée.

- Ecrire la commande qui permet de chiffrer le fichier **rsakey.pem** et produit ainsi un fichier **rsakeyencaes.pem.**

  #openssl enc –AES256 -in rsakey.pem -out rsakeyencaes.pem

Remarque : le fichier qu'on veut chiffrer est celui qui contient la paire de clé privée/publique ! Ceci fera en sorte qu'à chaque fois qu'on veut utiliser l'information stockée dans le fichier de clés, on devra mettre le mot de passe.

Chiffrement/déchiffrement de données avec RSA

- Ecrire la commande qui permet de chiffrer le fichier initial **fichier_nom_eleve** avec la clé publique **rsapubkey.pem** et produit ainsi un fichier **fichier_nom_eleve.rsaenc** *(utiliser l'instruction openssl rsautl).*

#openssl rsautl –pubin –inkey rsapubkey.pem –in fichier_nom_eleve –encrypt –out fihcier_chiff_rsa

- Ecrire la commande qui permet de déchiffrer le fichier **fichier_nom_eleve.rsaenc** et produit ainsi un fichier **fichier_nom_eleve.rsadec.**

#openssl rsautl –inkey rsakey.pem –in fichier_chiff_rsa.rsa –decrypt –out fichier_dechiff_rsa

- Vérifier l'égalité des deux fichiers **fichier_nom_eleve** et **fichier_dechiff_rsa**.

3. SIGNATURE NUMERIQUE

Génération d'une empreinte d'un fichier

Pour signer un document on calcule d'abord une empreinte de ce document. L'instruction à utiliser pour calculer l'empreinte est :

#openssl dgst <-algo> -out <sortie> <entree>

- Calculer la valeur de l'empreinte du fichier **fichier_nom_eleve** avec l'algorithme MD5 et la mettre dans un fichier **fichier_nom_eleve.md5**.
- Quelle est la taille de cette empreinte ?
- Calculer la valeur de l'empreinte du même fichier avec l'algorithme SHA1 et la mettre dans un fichier **fichier_nom_eleve.sha1**.
- Quelle est la taille de cette empreinte ?
- Comparer le résultat des deux fonctions de hachage. Qu'est-ce que vous remarquez ?

Signature d'un fichier

Signer un document revient à signer son empreinte. L'instruction à utiliser dans ce cas est :

```
#openssl rsautl –sign –in empreinte_fichier –inkey rsaprivkey.pem – out fichier_sig
```

- Signer le fichier **fichier_nom_eleve.sha1** et mettre de résultat le fichier **fichier_sig**. Dans ce cas, quel est la clé que vous devez utiliser pour signer ?
- Il reste ensuite à vérifier que l'empreinte ainsi produite dans le fichier **fichier_nom_eleve.sha1** est la même que l'on peut calculer. Utiliser l'instruction *openssl rsautl -verify*.

```
#openssl rsautl –verify –in fichier_sig -out empreinte2 –pubin – inkey rsapubkey.pem
```

- Quel est la clé que vous devez utiliser pour vérifier la signature du fichier fichier_sig?

4. CERTIFICAT NUMERIQUE

Génération de la clé privée

- Générer une paire de clés RSA de taille 1024 bits et stocker le résultat dans le fichier **server_cle.pem**

Génération d'une requête de création d'un certificat

Créer un fichier de demande de signature de certificat (CSR Certificate Signing Request) *:*

```
#openssl req –new -key serveur_cle.pem –out serveur_cert.pem
```

Signature du certificat

Afin de signer le certificat deux possibilités sont offertes :

- Auto signer le certificat
- Signer le certificat par une autorité de certification (AC)

Auto signature d'un certificat

- Signer le fichier **server.cert** à l'aide de la clé privée contenant dans le fichier **server_cle.pem** et stocker le résultat dans le fichier **server_cert.crt**. Le certificat doit avoir une période de validité d'un an.

```
#openssl req –new – x509 –days 365 –key serveur_cle.pem –out serveur_cert.crt
```

- Afficher le contenu du certificat en format texte *:*

```
#openssl x509 -in serveur_cert.crt -text -noout
```

Signature par une autorité de certification (AC)

- La première étape consiste à générer une clé privée RSA pour l'AC de taille 2048 bits et de stocker le résultat dans le fichier **cakey.pem**

```
#openssl genrsa -out cakey.pem 2048
```

- Générer un certificat pour l'AC ayant une période de validité 730 jours et stocker le résultat dans le fichier **ca.crt**.

```
#openssl req -new -x509 -days 730 -key cakey.pem –out ca_crt
```

Rq: ca.crt est le certificat auto signé de l'autorité de certification qui va permettre de signer les certificats créés.

- Signer la demande du certificat du serveur (le fichier **server.csr**) par l'autorité de certification AC en utilisant l'instruction suivante :

```
#openssl x509 –req –in server.csr –out server.crt –CA ca.crt -CAkey cakey.pem
-CAcreateserial -CAserial ca.srl
```

Rq : Le certificat signé est le fichier **ca.srt**