Lycée Voltaire

T.P : Sécurité des communications

N.S.I

1 Présentation de openSSL

1.1 Protocole SSL

Le protocole SSL (Secure Socket Layer) a été développé par la société Netscape Communications Corporation pour permettre aux applications client/serveur de communiquer de façon sécurisée. SSL a été universellement adopté sur le World Wide Web pour authentifier et chiffrer les communications entre clients et serveurs.

Une session SSL se déroule en deux temps :

1. une phase de poignée de mains (handshake) durant laquelle le client et le serveur s'identifient, conviennent du système de chiffrement et d'une clé qu'ils utiliseront par la suite.
2. la phase de communication proprement dite durant laquelle les données échangées sont compressées, chiffrées et signées.

L'identification durant la poignée de mains est assurée à l'aide de certificats X509.

1.2 openSSL

openSSL (http://www. openssl .org) est une boîte à outils cryptographiques qui va nous permettre :
— la création de clés RSA, DSA (signature) ;
— la création de certificats X509 ;
— le calcul d'empreintes (MD5, SHA, RIPEMD160, ...) ;
— le chiffrement et déchiffrement (DES, IDEA, RC2, RC4, Blowfish, ...) ;
— la réalisation de tests de clients et serveurs SSL/TLS ;
— la signature et le chiffrement de courriers (S/MIME).
Pour connaître toutes les fonctionnalités de **openSSL** taper dans un terminal : `man openssl`

2 Chiffrement symétrique

openssl permet d'utiliser plusieurs algorithmes de chiffrement symétrique.
Pour obtenir une liste des systèmes disponibles taper dans un terminal : `openssl enc -ciphers`

Exercice n° 1

1. Créer un fichier `toto`. Contenant un texte de votre choix.
2. Ouvrir un terminal et en utilisant la commande `cd` aller dans le répertoire contenant le fichier `toto`.
3. Pour chiffrer un fichier en utilisant l'algorithme "blowfish" en mode CBC, on utilise :

   ```
   openssl enc -bf-cbc -in toto -out toto.chiffre
   ```

   Taper le code ci-dessus et observer le contenu du fichier.
   (Avec notepad++ ou avec la commande :
   `cat toto.chiffre`
4. Pour déchiffrer un message, on utilise :

   ```
   openssl enc -bf-cbc -d -in toto.chiffre -out toto.dechiffre
   ```

5. Y a t-il une différence de taille entre les fichiers `toto` et `toto.dechiffre` ?
6. Comparer le contenu des fichiers `toto` et `toto.dechiffre` (Le faire manuellement et en utilisant la commande : `diff toto toto.dechiffre`)

Exercice n° 2

On a chiffré le fichier top_secret.chiffre avec l'algorithme "blowfish" en mode CBC

1. Télécharger et regarder ce que contient le fichier : `top_secret.chiffre`
2. Déchiffrer le message à l'aide de la clef : `Password1234`

3 Chiffrement asymétrique - RSA avec openSSL

Génération d'une paire de clés
Pour générer une paire de clés RSA de 2048 bits, stockée dans le fichier `maCle.pem`, on saisit la commande suivante :

```
openssl genrsa -out maCle.pem 2048
```

Exercice n° 3

Pourquoi parle t-on d'une **PAIRE** de clefs ?

Le fichier `maCle.pem` obtenu est un fichier au format PEM (Privacy Enhanced Mail, format en base 64).
Pour visualiser la clef privée saisir la commande : `cat maCle.pem`
Pour visualiser la clef publique saisir la commande : `openssl rsa -in maCle.pem -pubout`

Exercice n° 4

1. La commande : `openssl rsa -in maCle.pem -pubout -out maClePublique.pem` permet d'enregistrer uniquement la clef publique dans le fichier `maClePublique.pem`.

2. La commande : `cat maClePublique.pem` permet de visualiser la clef publique.

3. Pourquoi le fichier **meCle.pem** ne doit pas être diffusé ?

3.1 Chiffrement/Déchiffrement de données avec RSA

Voici la commande pour chiffrer des données avec une clé publique RSA.

```
openssl rsautl -encrypt -in <fichier_entree> -inkey <clePublique> -pubin -out <fichier_sortie>
```

où :
— **fichier_entree** est le fichier des données à chiffrer. Attention, ce fichier ne doit pas avoir une taille excessive (ne doit pas dépasser 116 octets pour une clé de 1024 bits) ;
— **clePublique** est le fichier contenant la clé RSA publique.
— **fichier_sortie** est le fichier chiffré.

Pour déchiffrer, on remplace l'option `-encrypt` par `-decrypt`. Le fichier contenant la clé doit évidemment contenir la partie privée. Ce qui donne :

```
openssl rsautl -decrypt -in <fichier_entree> -inkey <cle> -out <fichier_sortie>
```

Exercice n° 5

1. M'envoyer votre clef publique par mail afin que je puisse la publier.
2. Suivre les instructions sur : http://bfourlegnie.com/clefpublique
3. Ecrire un message **gentil** et de moins de 116 octets dans un fichier txt. Chriffrer ce fichier et l'envoyer par mail à une personne dont vous avez la clef publique.(voir : http://bfourlegnie.com/clefpublique)
4. Déchiffrer le (ou les) message(s) que l'on vous envoie.
5. Comparer la taille du fichier déchiffré avec celui chiffré. (donner le facteur multiplicatif)

4 Utilisation des deux chiffrements

Exercice n° 6

Télécharger le fichier : operation_espadon.chiffre que j'ai chiffré avec une clef de chiffrement symétrique utilisant l'algorithme "blowfish" en mode CBC.
Vous allez recevoir par mail un fichier `clef.txt` que j'ai chiffré avec un chiffrement asymétrique RSA et contient la clef qui permet de déchiffrer `operation_espadon`.
Pour réaliser ce chiffrement asymétrique RSA, j'ai utilisé votre clef publique disponible sur http://bfourlegnie.com/clefpublique.

Qu'est-ce qui se cache derrière **operation_espadon.chiffre** ? (J'espère que vous avez conservé précieusement votre clef privée !)

Exercice n° 7

1. Choisir une clef pour un chiffrement symétrique.
2. Créer un fichier audio qui décrit et explique vos souhaits d'orientation post-bac.(moins de 1 minute).
3. Chiffrer ce fichier avec la clef symétrique en utilisant l'algorithme "blowfish" en mode CBC.
   (Nommé le fichier chiffré : `VotreNom.chiffre`)
4. Copier votre clef dans un fichier texte et chiffrer le fichier avec ma clef publique.
   Nommer ce fichier chiffré : (`clef.chiffre`)
5. M'envoyer par mail, les deux fichiers chiffrés.

5 Authentification

Il n'est possible de signer que de petits documents. Pour signer un gros document on calcule d'abord une empreinte de ce document (avec une fonction de hachage). La commande `dgst` permet de le faire.

```
openssl dgst <hachage> −out <empreinte> <fichier_entree>
```

où `hachage` est une fonction de hachage. Avec `openssl`, plusieurs fonctions de hachage sont proposées dont :
— MD5 (option -md5), qui calcule des empreintes de 128 bits,
— SHA1 (option -sha1), qui calcule des empreintes de 160 bits,
— SHA256 (option -sha256), qui calcule des empreintes de 256 bits,
— RIPEMD160 (option -ripemd160), qui calcule des empreintes de 160 bits.
Signer un document revient à signer son empreinte avec sa clé privée. Pour cela, on utilise l'?option `-sign` de la commande `rsautl`

```
openssl rsautl -sign -in <empreinte> -inkey <cle> -out <signature>
```

On envoie cette signature ainsi que le document qui a servie à produire l'empreinte.
Le destinataire recoit la signature et le document. Pour vérifier l'authentification, il doit :

1. Utiliser la clef publique de l'expéditeur pour avoir l'empreinte.
   ```
   openssl rsautl -verify -in <signature> -pubin -inkey <cle> -out <empreinte1>
   ```

2. Vérifier que le document recu fournit la même empreinte :
   ```
   openssl dgst <hachage> -out <empreinte2> <fichier_entree>

   diff <empreinte2> <empreinte1>
   ```

Exercice n° 8

Récupérez l'archive exo8.zip qui contient deux fichiers accompagnés d'une signature. Vous avez à disposition :
— fichier : `quinquin.txt`, signature : `signature1`
— fichier : `mirabeau.txt`, signature : `signature2`
— La partie publique de la clé RSA ayant produit la signature : uneClePublique.pem.
De ces deux fichiers, lequel n'est pas authentifiable ?