Travaux pratiques –TP 1
Services de sécurité, mécanismes, algorithmes avec *API Openssl*

Partie 1 : OpenSSL

OpenSSL est une boîte à outils cryptographiques implémentant les protocoles SSL et TLS. Il offre :

- Une bibliothèque de programmation en C permettant de réaliser des applications client/serveur sécurisées s'appuyant sur SSL/TLS
- Une commande en ligne (OpenSSL) permettant
  - la création de clés RSA, DSA (signature)
  - la création de certificats X509
  - le calcul d'empreintes (MD5, SHA, RIPEMD160, ...)
  - le chiffrement et déchiffrement (RSA, DES, IDEA, RC2, RC4, Blowfish, ...)
  - la réalisation de tests de clients et serveurs SSL/TLS
  - la signature et le chiffrement de courriers (S/MIME)

La syntaxe générale de la commande openssl est

```
$ Openssl  <commande>  <option>
```

On trouvera toutes les informations la concernant à l'adresse : http://www.openssl.org

L'objectif de ce TP est de vous familiariser avec les services de base de la sécurité, chiffrement, hachage et signature.

a) Télécharger l'outil openssl avec la commande *apt-get install openssl*
b) Vérifier la bonne installation de l'outil
c) Répondez aux questions suivantes :
   1. Quelle est la version d'openssl installée ?
   2. Lister tous les algorithmes de cryptographie présents dans l'outil ?
d) Tapez la commande suivante et expliquer le résultat : *openssl ciphers –v*
e) Pour connaître toutes les fonctionnalités de openSSL : *man openssl*

Partie 2 : Fonctions de hachage

a) Créer le fichier Plain.txt contenant la phrase « travaux pratiques »
b) Générer le digest (l'empreinte) en SHA1 et MD5 pour votre fichier.
c) Modifier le contenu du fichier en remplacer seulement le caractère t par T du mot travaux. Générer de nouveau le digest du fichier. Comparer le résultat avec la question b). Conclure.
d) Télécharger le fichier usa.rar donné par le chargé du TP. Décompresser le fichier et générer le digest en MD5 pour chaque fichier dans usa.rar. Qu'en pensez-vous ?

Partie 3 : Chiffrement symétrique

La commande openssl enc permet de chiffrer et déchiffrer des messages. Plus d'informations sur cette commande peuvent être trouvées en tapant openssl enc -h

a) Créer un fichier texte Plain.txt et y écrire : « travaux pratiques ». Chiffrer ce fichier avec l'algorithme DES-CBC et sauvegarder le fichier sous le nom Cipher.txt. Utiliser l'option –k pour saisir le mot de passe symétrique. Vous pouvez utiliser l'option –base64 pour la lisibilité.
b) Ouvrir le fichier chiffré. Qu'observez-vous au niveau des premiers caractères ? À quoi cela sert-il ?

c) Déchiffrer le fichier chiffré en lui donnant le nom NewPlain.txt et vérifier qu'on retombe bien sur le fichier de départ. Pour vérifier : diff Plain.txt NewPlain.txt -q
d) Reprendre la question a) et c) mais cette fois ci en indiquant l'option –p. Détailler les informations obtenues.
e) Chiffrer le Plain.txt encore une fois (NewCipher.txt). Comparer les fichiers Cipher.txt et NewCipher.txt. Justifiez.
f) Refaire ce test deux nouvelles fois en ajoutant l'option -nosalt. Comparer et expliquer les résultats obtenus dans b).
g) Grâce à la commande rand, créer un fichier GrandFichier.txt avec des données aléatoires, d'une taille d'environ 100 Mo. Utiliser la commande time pour calculer le temps de chiffrement de votre fichier en utilisant DES, 3DES, AES-CBC (Prendre le temps user). Comparer les résultats ?

Partie 4 : Chiffrement asymétrique

Dans cet exercice, l'intérêt est de chiffrer un document avec une clé publique que seul son créateur pourra déchiffrer.

a) Générer une paire de clés RSA 2048 bits que vous nommerez PrivateKeyMyName.priv. Ex : PrivateKeyDupont.priv. Utilisez l'option –p.
b) Extraire ensuite la clé publique que vous nommerez PublicKeyDupont.pub
c) Envoyer à votre voisin votre clé publique (par ssh ou clé usb)
d) Chiffrer le fichier Plain.txt avec la clé publique de votre collègue
e) Déchiffrer le fichier que vous avez reçu avec votre clé privée
f) Chiffrer le fichier GrandFichier.txt avec la clé publique de votre collègue. Vous devez remarquer un problème. Comment l'expliquez-vous ?

Partie 5 : Chiffrement symétrique/asymétrique et signature numérique

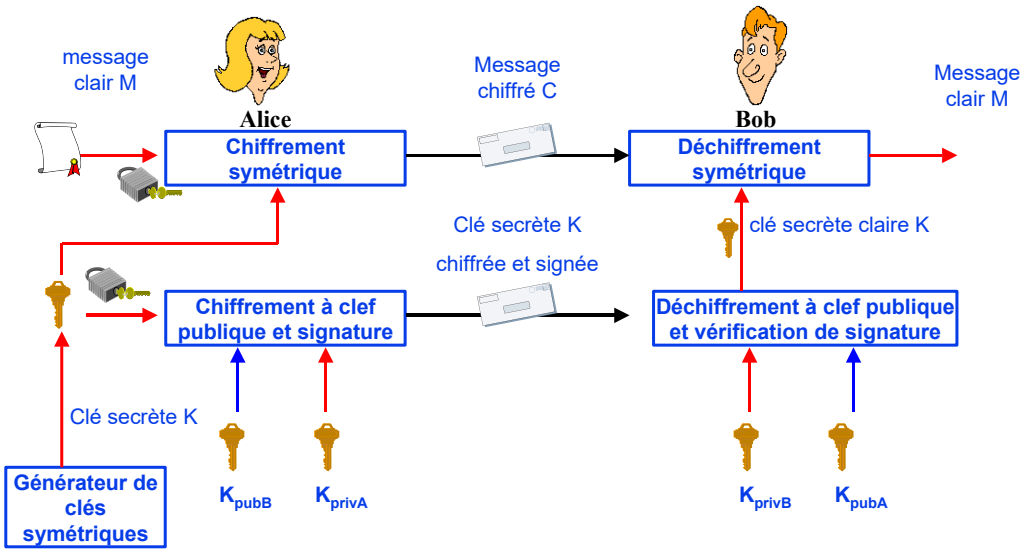

Figure 1. Scénario sécurité

a) Expliquer le scénario ci-dessus.
b) Quels sont les différents services de sécurité réalisés dans ce scénario.
c) Générer avec la commande rand une clé symétrique SymKey.key de taille 128 bits. Encoder la clé en hex.
d) Visualiser la clé symétrique en binaire en utilisant la commande linux : xxd
xxd –b –c 8 SymKey.key
e) Réaliser le scénario ci-dessus en échangeant d'une manière sûre la clé symétrique avec votre collègue. Pour information, on signe le hash d'un fichier et non pas le fichier lui-même.

Commandes utiles

$ time openssl commande : pour visualiser le temps d'exécution d'une commande
$ diff file1.txt file2.txt –q : pour comparer le contenu de deux fichiers

$ openssl genrsa -out <fichier_rsa.priv> <size> : génére la clé privé RSA de taille size.

$ openssl rsa -in <fichier_rsa.priv> -des3 -out <fichier.pem> : chiffre la clef privé RSA avec l'algorithme DES3. Vous pouvez utiliser DES, 3DES, IDEA, etc.

$ openssl rsa -in <fichier_rsa.priv> -pubout -out <fichier_rsa.pub> : stocke la partie publique dans un fichier à part

$ openssl enc <-algo> -in <claire.txt> -out <chiffre.enc>: pour le chiffrement de claire.txt avec l'algorithme spécifié (openssl enc –help pour avoir la liste des possibilités ou bien openssl list-cipher-commands) dans un fichier chiffre.enc. On peut ajouter –k comme option et saisir la clé comme une passphrase

$ openssl enc <-algo> -in <chiffre> -d -out <claire> : pour le déchiffrement.

$ openssl dgst <-algo> -out <sortie> <entrée> : pour hacher un fichier. L'option <-algo> est le choix de l'algorithme de hachage (sha, sha1, dss1, md2 ,md4, md5, ripemd160).

$ openssl rand -out <clé.key> <nombre_octets> : pour générer un nombre aléatoire de taille nombre_octets (utiliser l'option -base 64 pour la lisibilité).

$ openssl aes-256-cbc -in <claire.txt> -out <chiffre.enc> -e -k <clé.key> : pour chiffrer un fichier avec l'AES.

$ openssl rsautl -encrypt `-pubin` -inkey <rsa.pub> -in <clair.txt> -out <chiffre.enc> : chiffrer fichier.txt avec la RSA en utilisant la clef publique rsa.pub

$ openssl rsautl -decrypt -inkey <rsa.priv> -in <chiffre.enc> -out <fihcier.txt> : pour déchiffrer le fichier fic.dec

$ openssl rsautl -sign -inkey <ras.priv> -in <fichier.txt> -out <fic.sig> : pour générer une signature.

$ openssl rsautl -verify -pubin -inkey <rsa.pub> -in fic fic.sig : pour vérifier une signature.

- *$openssl aes-256-cbc -in <fichier.txt> -out <fichier.enc> -e -k clé.key* : pour chiffrer un fichier avec l'algorithme AES utilisant une clé symétrique.

- *$openssl aes-256-cbc -in <fichier.txt> -out <fichier.enc> -d –k clé.key* : pour déchiffrer un fichier avec l'algorithme AES utilisant une clé symétrique.

COMMANDES STANDARDS

asn1parse
: Traitement d'une séquence ASN.1.

ca
: Gestion Certificate Authority (CA).

ciphers
: Détermination de la description de la suite de chiffrement.

crl
: Gestion Certificate Revocation List (CRL).

crl2pkcs7
: Conversion CRL vers PKCS#7.

dgst
: Calcul signature message (MD5).

dh
: Gestion des paramètres Diffie-Hellman. Obsolète par **dhparam**.

dsa
: Gestion données DSA.

dsaparam
: Génération paramètres DSA.

enc
: Chiffrement.

errstr
: Conversion numéro d'erreur vers descriptif texte (String).

dhparam
: Génération et gestion de paramètres Diffie-Hellman.

gendh
: Génération de paramètres Diffie-Hellman. Obsolète par **dhparam**.

gendsa
: Génération de paramètres DSA.

genrsa
: Génération de paramètres RSA.

passwd
: Génération de mots de passe hashés.

pkcs7
: Gestion données PKCS#7.

rand
: Génère octets pseudo-aléatoires.

req
: Gestion X.509 Certificate Signing Request (CSR).

rsa
: Gestion données RSA.

rsautl
: Utilitaire RSA pour signature, vérification, chiffrement, et déchiffrement.

s_client
: Ceci fournit un client SSL/TLS générique qui sait établir une connexion transparente avec un serveur distant parlant SSL/TLS. Étant seulement prévu pour des propos de test, il n'offre qu'une interface fonctionnelle rudimentaire tout en utilisant en interne la quasi-totalité des fonctionnalités de la librairie **ssl** d'OpenSSL.

s_server
: Ceci fournit un client SSL/TLS générique qui accepte des connexions transparentes provenant de clients qui parlent SSL/TLS. Étant seulement prévu pour des propos de test, il n'offre qu'une interface fonctionnelle rudimentaire tout en utilisant en interne la quasi-totalité des fonctionnalités de la librairie **ssl** d'OpenSSL. Il fournit à la fois son propre protocole orienté commandes en ligne pour le test de fonctions SSL et une facilité de réponse simple HTTP pour émuler un serveur internet qui gère SSL/TLS.

s_time

Horlogeur de connections SSL.

sess_id
Gestion des données de session SSL.

smime
Traitement mails S/MIME.

speed
Mesure la vitesse de l'algorithme.

verify
Vérification du certificat X.509.

version
Information sur la version d'OpenSSL.

x509
Gestion de données pour le certificat X.509.

COMMANDES DE SIGNATURE DE MESSAGE

md2
Signature MD2

md5
Signature MD5

mdc2
Signature MDC2

rmd160
Signature RMD-160

sha
Signature SHA

sha1
Signature SHA-1

COMMANDES D'ENCODAGE ET DE CHIFFREMENT

base64
Chiffrement Base64

bf bf-cbc bf-cfb bf-ecb bf-ofb
Chiffrement Blowfish

cast cast-cbc
Chiffrement CAST

cast5-cbc cast5-cfb cast5-ecb cast5-ofb
Chiffrement CAST5

des des-cbc des-cfb des-ecb des-ede des-ede-cbc des-ede-cfb des-ede-ofb des-ofb
Chiffrement DES

des3 desx des-ede3 des-ede3-cbc des-ede3-cfb des-ede3-ofb
Chiffrement Triple-DES

idea idea-cbc idea-cfb idea-ecb idea-ofb
Chiffrement IDEA

rc2 rc2-cbc rc2-cfb rc2-ecb rc2-ofb
Chiffrement RC2

rc4
Chiffrement RC4

rc5 rc5-cbc rc5-cfb rc5-ecb rc5-ofb
Chiffrement RC5