Travaux pratiques sur l'API Openssl

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. C'est cette même distribution qu'intègre le serveur Apache pour offrir les services SSL.
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.

L'objectif de ces travaux pratiques 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.

Il vous ait demandé dans un premier temps de suivre pas à pas les exercices qui vous sont proposés et par la suite vous avez quartier libre pour les exercices qui vous feront plaisir.

Notation toutes les commandes et arguments associés seront notés dans la suite en ***italique gras*** (exemple : openssl> ***enc -in msg -out msge -e -des3***)

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_nom_eleve*** (choisissez un fichier qui contient des données textuelles). Ecrire la commande qui permet de le chiffrer et produit ainsi un fichier ***fic_nom_eleve.enc***
On peut alors vérifier que le message ***fic_nom_eleve.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_nom_eleve*** ...
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 enc -in fichier -out fichier.enc -e -des3
openssl enc -in fichier.enc -out fichier.dec -d -des3
Leur demander de diversifier leurs algorithmes. D'utiliser par exemple l'option –a pour produire un fichier lisible donc codé en base64

Question 2 :

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)

Solution :
timeout openssl enc -in fichier -out fichier.enc -e -des3
timeout openssl enc -in fichier.enc -out fichier.dec -d -des3

Idem avec –rc4, idea, des3 et …

Question 3 :
Pourquoi quand on applique deux fois la commande ***enc*** une fois au fichier en clair et une fois au fichier chiffré on 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 utile l'option –d il applique la même fonction sauf qu'il extrait du chiffrement les données structurelles.

Chiffrement asymétrique

Exercice 1 :

Génération de cle privée/public RSA :
Le format de sortie par défaut est du PEM (Privacy Enhanced Mail). A l'aide de l'option ***–outform*** ou ***-inform*** on peut cahnger le format. Deux formats sont supportés par cette option PEM et DER)
openssl
OpenSSL> ***genrsa*** (cette commande prend en entrée les paramètres positionés dans le fichier openssl.cnf)
OpenSSL> ***genrsa -help***
OpenSSL> ***genrsa 1024***
OpenSSL> ***genrsa -out key 1024***
OpenSSL> ***genrsa -des3 -out key 1024*** (la clé privé est chiffré avec un triple DES)

* creer un fichier nomme "rand.txt" contenant n'importe quoi...

OpenSSL> ***genrsa -des3 -out key -rand rand.txt 1024***

Verification de cle privee RSA :

OpenSSL> ***rsa -in key***
OpenSSL> ***rsa -in key -check***
OpenSSL> ***rsa -in key -check -modulus***
OpenSSL> ***rsa -in key -check -modulus -text***

Generation de cle publique RSA :

OpenSSL> ***rsa -in key -pubout -out pubkey***

Verification de cle publique RSA :

OpenSSL> ***rsa -pubin -in pubkey -text***

Certification

EXERCICE 2

Création d'un CA et de deux clients MIME :

*créer newcerts, serial et index.txt

CA :
Génération de la clé privé/public du CA dans le fichier ca.key
Tous les fichiers ainsi générés sont au format PEM.

OpenSSL> ***genrsa -out ca.key -des3 1024***

OpenSSL> ***rsa -pubout -check -text -in ca.key***

Génération d’un certificat autosigné. Le fichier de configuration openssl.cnf doit être dans le même répertoire. Le fichier ca.pem contient le certificat du CA.

OpenSSL> ***req -x509 -new -key ca.key -out ca.pem -days 365 -config openssl.txt -extensions CA_MIME***

Visualisation dans un format lisible le contenu du certificat.
OpenSSL> ***x509 -in ca.pem -purpose -text***

Génération des certificats pour les Clients MIME. Il s’agit ici bien entendu d’usager donc de certificat d’identité :

OpenSSL> ***genrsa -out cl.key -des3 1024***

Génération d’une requête de certificat au format PKCS#10 qui sera par la suite soumise au CA pour signature et produire ainsi un certificat d’usager. Vous remarquerez les options positionnées pour cette rubrique concerne des key usage pour un client de messagerie

OpenSSL> ***req -new -key cl.key -out cl.crs -config openssl.txt -extensions CLIENT_MIME***

Le CA va par la suite siggner la requête soumise. Trivial de déduire les différents paramètres !!

OpenSSL> ***ca -config openssl.txt -extensions CLIENT_MIME -cert ca.pem -keyfile ca.key -out cl.pem -infiles cl.crs***

No comment !!

OpenSSL> ***x509 -in cl.pem -purpose -text***

Idem pour le deuxième client :

OpenSSL> ***genrsa -out cl2.key -des3 1024***

OpenSSL> ***req -new -key cl2.key -out cl2.crs -config openssl.txt -extensions CLIENT_MIME***

OpenSSL> ***ca -config openssl.txt -extensions CLIENT_MIME -cert ca.pem -keyfile ca.key -out cl2.pem -infiles cl2.crs***

OpenSSL> ***x509 -in cl2.pem -purpose -text***

Chiffrement d'un message envoyer de cl a cl2

OpenSSL> ***smime -encrypt -in msg.txt -des3 -out mail cl2.pem***

OpenSSL> ***smime -decrypt -in mail -recip cl2.pem -inkey cl2.key -out maild***

Pour que l’exercice soit attrayant les élèves doivent s’amuser à s’échanger leur certificat. Et ainsi s’échanger des fichiers chiffrés.

Signature

Exercice 3 :

Faire idem exercice 2 mais en incluant la signature dans la commande smime.

Intégration ou importation du certificat au sein du client de messagerie (UA).

Exercice 4 :

Mettre votre clé privé et certificat au format pkcs#12 et l'intégrer à votre messagerie pour signer et déchiffrer. Intégrer les certificats des autres usagers pour leur envoyer des messages chiffrés. Il est nécessaire d'exporter le certificat client au format PKCS#12, format encapsulant à la fois le certificat signé et la clé privée. Cette exportation du format PEM vers le format PKCS#12 (extension p12) se fait avec la commande ***pkcs12***

Exemple avec la commande pkcs12 :
Openssl>***pkcs12 -export -inkey client-mime.key -in client-mime.pem -out client-mime.p12 -name "Certificat Client-mime"***

Deux mots de passe seront demandés au cours de cette manipulation : le premier est le mot de passe protégeant la clé privée client-mime.key, le second est le mot de passe qui protégera le certificat au format PKCS#12. Ce dernier sera demandé lors de l'importation du certificat dans le client de messagerie.
L'option –name permet de spécifier quel sera le nom désignant le certificat client dans le client de messagerie. Une fois ce fichier généré, il faut l'importer via le client de messagerie.

Bon courage !!!