Réseau et Sécurité
TP 2 - OpenSSL

www. openssl. org/

Dans ce TP nous travaillerons sur l'implémentation de primitives cryptographique OpenSSL. OpenSSL est un projet proposant des implémentation cryptographiques de complète et de qualité. La prise de décision technique du projet est gérée par le Comité technique OpenSSL et la gouvernance du projet est gérée par le Comité de gestion OpenSSL. Une communoté d'expert gravite autour de ce projet afin d'en garantir la qualité. OpenSSL bénéficie d'une licence de type Apache, ce qui signifie essentiellement que vous êtes libre de l'obtenir et de l'utiliser à des fins commerciales et non commerciales (sous réserve de quelques conditions). Une liste des vulnérabilités, ainsi que les versions dans lesquelles elles ont été découvertes et les corrections est facliement accesible : `https://www.openssl.org/news/vulnerabilities.html`.

Remarque. Plusieurs exercices de ce TP sont à faire en binôme. Certains accès nécessitent d'être super utilisateur.

Ce TP est à réaliser **dans une VM**. Connectez-vous par exemple à votre VM myISIMA. Lancez la sur `https://my.isima.fr/fr/vm/` puis connectez vous en ssh :

```
ssh <login>@vm-<login>.local.isima.fr
```

Syntaxe général :

openssl commande [options] [arguments]

Les commandes que nous utiliserons dans ce TP incluent : *passwd, enc, genrsa, x509, dgst, req, verify.*

Utilisation d'OpenSSL

1. OpenSSL est généralement préinstallé sur un environement Linux. Installez openssl si cela n'est pas déjà fait.

Convertion en base64

Définition Base64 est un codage utilisant 65 caractères imprimables (les 26 lettres majuscules, les 26 lettres minuscules, les 10 chiffres, le caractère '+', le caractère '/' et le caractère spécial ' ='). Base64 permet d'échanger des données en réduisant les problèmes d'encodage des caractères spéciaux.

Syntaxe : pour encoder en base64, on utilise :

openssl enc -base64 -in fichier

Pour décoder de la base64, on utilise :

openssl enc -base64 -d -in fichier

1. Encodez un mot de passe (fictif) avec base64 et donnez le à votre binôme. Votre binôme devra retrouver le mot de passe.

2. Est-ce que base64 est un moyen sûr de protéger un mot de passe ?

Fichiers de mots de passe

Définition Le fichier /etc/shadow contient des informations sur les mots de passe des utilisateurs. Il est, par défaut, lisible uniquement par l'administrateur.

man 5 passwd

Définition Le fichier /etc/shadow contient des informations sur les mots de passe des utilisateurs. Il est, par défaut, lisible uniquement par l'administrateur.

man 5 shadow

1. Créez un utilisateur pass1 avec un mot de passe quelconque en utilisant la commande `adduser`.

man 8 adduser

2. Identifiez le champ contenant le mot de passe hacahé de l'utilisateur.

Chiffrement

Syntaxe : Pour le chiffrement, on utilise la commande enc. Pour le déchiffrement, on utilise la commande enc et l'option -d. Pour utiliser DES, on utilise l'option -des. Pour utiliser le triple DES, on utilise l'option -des3. Ne les utilisez pas ! Utilisez AES !

Syntaxe :

openssl enc -help

1. Trouvez la liste des chiffrements proposé par *openssl enc*.
2. Chiffrez un fichier quelconque et déchiffrez le avec le bon mot de passe.

   Remarque : Si le mot de passe est incorrecte, openssl le détecte directement et ne déchiffre pas le fichier. Nous allons chercher à savoir comment.

3. Comparez les tailles des fichiers clairs et chiffrés. Expliquez la différence.
4. Prenez deux fichiers très différents f1 et f2. Chiffrez puis déchiffrez ces deux fichiers avec le bon mot de passe, et en utilisant l'option *-nopad*. Vous obtenez des fichiers que vous nommerez respectivement f1b et f2b.
5. Avec l'outil xxd (par exemple), étudiez la différence entre f1 et f1b, ainsi qu'entre f2 et f2b. Expliquez les différences.
6. Prenez à présent un fichier f3. Chiffrez-le puis déchiffrez-le en utilisant un mauvais mot de passe, et en utilisant l'option -nopad. Vous obtenez un fichier que vous nommerez f3b. Examinez le fichier f3b (seulement). Sans comparer f3 et f3b, comment pouvez-vous déterminer que f3b a été déchiffré de manière incorrecte ?

RSA

Syntaxe : La génération de clefs se fait en utilisant :

openssl genrsa -out clef-privée taille

1. Créez une clef privée private1.pem de 2048 bits, sans chiffrement. Créez une clef privée private2.pem de 2048 bits, avec chiffrement.

2. Récupérez les informations sur chacune de ces deux clefs.

   Syntaxe : Pour récupérer la clef publique associée à une clef, on utilise :

openssl rsa -in clef-privée *-pubout -out* clef-publique

3. Créez la clef publique associée à chaque clef privée.

4. Faut-il conserver la clef publique en clair ? Est-ce que votre clef publique est sauvegardée en clair ?

   Syntaxe : Pour afficher des informations sur une clef publique, il faut utiliser l'option `-pubin` pour indiquer que le fichier d'entrée comprend seulement des informations publiques.

5. Récupérez des informations sur les deux clefs publiques.

Chiffrement et déchiffrement

Syntaxe : Pour chiffrer avec une clef publique RSA, on utilise :

openssl pkeyutl -encrypt -in fichier-clair -out fichier-chiffré -pubin -inkey clef-publique

Pour déchiffrer, on utilise l'option *-decrypt*.

1. Chiffrez et déchiffrez un petit fichier.

2. Chiffrez un gros fichier. Que se passe-t'il ? Pourquoi ?

3. Demandez à votre binôme sa clef publique. Chiffrez un court message avec sa clef publique. Transmettez-le lui, et demandez-lui de le déchiffrer.

   Syntaxe : Pour chiffrer avec un algorithme *alg*, on utilise :

openssl enc -alg -in fichier-clair *-out* fichier-chiffré

   Pour déchiffrer avec ce même algorithme, on utilise :

openssl enc -alg -d -in fichier-clair *-out* fichier-chiffré

4. Demandez à votre binôme sa clef publique. Chiffrez un long message avec un algorithme de chiffrement symétrique (comme AES), en utilisant un mot de passe. Chiffrez le mot de passe avec la clef publique de votre binôme. Transmettez à votre binôme le mot de passe chiffré et le message chiffré. Demandez-lui de déchiffrer le message.

Signature

Pour hacher un fichier, on utilise :

openssl dgst -sha256 -out fichier-hache.sha256 fichier-clair

Syntaxe : Pour signer un haché, on utilise :

openssl pkeyutl -sign -in haché *-inkey* clef *-out* signature

1. Est-ce correcte d'utiliser md5 comme fonction de hachage ? Si oui, justifiez pourquoi. Sinon, trouver une alternative.

2. Faut-il utiliser la clef publique ou la clef privée pour une signature ?

   Syntaxe : Pour vérifier une signature, on utilise :

openssl pkeyutl -verify -in signature *-pubin -inkey* clef-publique *-out* haché2

3. Signez un message.

4. Vérifiez sa signature. Vous pourrez utiliser l'outil diff.

   Syntaxe :

man diff

5. Créez trois messages. Signez ces trois messages. Modifiez-en un ou deux légèrement. Transmettez à votre binôme les trois messages ainsi que les signatures correspondantes. Demandez-lui de déterminer quels sont les messages qui ont été modifiés.

Création de certificats

Syntaxe : Pour demander un certificat, il faut créer une requête de la manière suivante :

openssl req -new -key clef -out requête

Remarque : la clef doit être suffisamment grande pour pouvoir signer le certificat. Une clef de 2048 bits convient généralement.

1. Créer une requête de certificat appelée user-request.pem concernant une entité appelée user. Cette requête est en attente de signature par une autorité de certification.

   Syntaxe : Pour visualiser une requête de certificat ou un certificat, on utilise :

openssl x509 -in certificat *-text -noout*

2. Visualiser le contenu de votre requête de certificat.

3. Créer une paire de clefs pour l'autorité de certification. Créer une requête de certificat nommée ca-request.pem concernant l'entité de certification appelée CA.

   Syntaxe : Pour autosigner une requête de certificat, on utilise :

openssl x509 -req -in requête *-signkey* clef-privee *-out* certificat

4. Autosigner la requête de certificat de CA dans un certificat nommé ca-certificat.pem.

5. Visualiser le certificat ca-certificat.pem.

   Syntaxe : Pour signer une requête de certificat, on utilise :

openssl x509 -days duree *-CAserial* serial *-CA* certificat *-CAkey* clef *-in* requête *-req -out* certificat

   Le fichier serial contient un nombre enregistré en hexadécimal (c'est-à-dire un nombre pair de chiffres hexadécimaux).

6. Signer la requête de certificat user-request.pem pour produire un certificat user-certificat.pem.

   Syntaxe : Pour vérifier un certificat, on utilise :

openssl verify -CAfile ca-certificat certificat

7. Vérifier le certificat user-certificat.pem.

8. Modifier le contenu du certificat user-certificat.pem et tentez de le vérifier.

Chaine de certification

1. Créez deux autorités de certifications CA1 et CA2. CA1 est l'autorité de certification racine, et CA2 est certifiée par CA1.

2. Créez une requête de certificat et signez la par CA2.
3. Vérifiez la chaîne complète de certification du certificat que vous venez de créer

Mettre en place TLS 1.3

1. Tester ces commandes afin de vérifier que ces sites soient bien en TLS1.2 et TLS1.3.

   ```
   openssl s_client -connect www.google.com:443 -tls1_2
   openssl s_client -connect www.google.com:443 -tls1_3
   openssl s_client -connect ayesh.me:443 -tls1_3
   ```

Cloner le git :

```
git clone https://github.com/testssl/testssl.sh.git
```

2. Tester le script `testssl.sh` sur `https://uca.fr`.
3. Essayez de vous connecter à `localhost` ou `127.0.0.1`. Si cela ne fonctionne pas, résolvez le problème.
4. Essayez de vous connecter à `https://127.0.0.1`. Si cela ne fonctionne pas, résolvez le problème en utilisant les commandes suivantes puis expliquez ce qu'elles font :

   ```
   a2ensite default-ssl
   a2enmod ssl
   systemctl restart apache2
   ```

   Si nécessaire, vous pouvez `unset` les variables `http_proxy` et `https_proxy`. A présent tenter à nouveau de vous connecter à `https://127.0.0.1`, cela devrait fonctionner.
5. Ouvrir ce fichier : `/etc/apache2/mods-available/ssl.conf` et enlever la version de TLS 1.3. Vérifier avec `testssl.sh` que les changements sont effectifs (attention à relancer le service `apache2`).
6. Maintenant mettre `TLS 1.3` est par défaut `SSLProtocol -all +TLSv1.3 -TLSv1.2` Vérifier que cela fonctionne.
7. Jouer avec les paramétres cryptographiques :
   — Authoriser uniquement les `Ciphersuites` suivants `TLS_CHACHA20_POLY1305_SHA256` et `TLS_AES_128_GCM_SHA256`.
   — Ajouter `TLS_AES_256_GCM_SHA384`

   Vérifier chaque fois que les bons options sont activées.

   Faire de même pour les courbes elliptiques(`Curves`) :
   — `SSLOpenSSLConfCmd Curves X25519:sect571r1:secp521r1:secp384r1`
   — `SSLOpenSSLConfCmd Curves X25519:secp521r1:secp384r1:prime256v1`

ATTENTION. Faire un `service apache2 restart` entre chaque manipulation.