TP de bases cryptographiques
M2 SIRAV, SSI

Résumé

Le but de ce TD est de découvrir les fonctionnalités offertes sous Linux par OpenSSL, liées à la cryptographie. Pour vous aider, faites appel au manuel :

— `man openssl`

— `openssl help` (pour voir la liste des commandes possibles)

— `man cmde` (pour une commande de OpenSSL : `openssl cmde`), ou `help cmde` (si vous êtes sous l'environnement OpenSSL)

ou allez sur `https://www.openssl.org`. Selon la version de OpenSSL installée, les commandes et leurs paramètres peuvent varier. Pour information, la version courante est 3.0/3.1 (`openssl version`).

1 Actions préliminaires

Avant d'utiliser OpenSSL, il faut utiliser de petits fichiers ayant les caractéristiques suivantes (à créer vous-même) :

— `f.txt` contenant un tout petit texte (une phrase).

— `longf.txt` contenant un texte de plusieurs lignes.

— `1.txt` contenant exactement 48 octets.

— `2.txt` contenant exactement 24 octets, différent de `1.txt`.

— `12.txt` contenant la concaténation de `1.txt` puis `2.txt` (donc exactement 72 octets).

— `21.txt` contenant la concaténation de `2.txt` puis `1.txt` (donc exactement 72 octets).

2 Questions d'encodage

1. Quels sont les encodages les plus couramment utilisés en informatique ?

   Solution: UTF-8 et ASCII.

2. Lorsque vous avez créé les fichiers précédents, vous n'avez pas précisé l'encodage. Quel encodage par défaut le système a-t-il utilisé ?
   Aidez-vous de la commande `file -i`.

   Solution: Généralement ASCII.

   Un encodage fréquent dans le traffic Internet est Base64. Renseignez-vous sur cet encodage.

3. Quelles sont les spécificités de Base64 ?

Solution: Là où les encodages usuels encodent une suite binaire par blocs de 8 bits, Base64 encode par blocs de 6 bits (avec donc 2^{64} possibilités, d'où le nom).
Contrairement à UTF-8 et ASCII, Base64 est **propre** : chaque chaîne de bits peut être transformé en un encodage unique, et chaque encodage peut être transformé en une chaîne de bits unique (c'est un encodage **bijectif**).

4. Pourquoi cet encodage est-il utilisé dans le traffic Internet ?

Solution: Parce que le traffic Internet est généralement chiffré : la sortie d'un chiffrement est une chaîne de bits aléatoire, or avec UTF-8 ou ASCII, certaines chaînes de bits ne peuvent pas être encodées en caractères, d'où l'usage de Base64.

5. Existe-t-il des encodages par blocs de 8 bits adaptés au traffic Internet ?

Solution: Oui, Latin-1 par exemple. Mais ils sont moins utilisés.

Généralement, les librairies qui génèrent des messages pour le traffic Internet permettent de l'encoder en Base64. C'est le cas d'OpenSSL (avec l'option `-base64`).
On peut aussi obtenir un encodage Base64 avec une fonction indépendante : la commande `base64`.

1. encodez le fichier `f.txt` en Base64 et enregistrez le résultat dans le fichier `f.b64`.
2. essayez de retrouver le résultat, ou au moins de recalculer les premiers caractères, pour bien comprendre le fonctionnement de Base64. La commande `hexdump` (ou `hd`) peut aider.
3. décodez le fichier `f.b64` sans stocker le résultat et vérifiez que c'est bien l'original.

3 Chiffrement symétrique

1. Quelle différence entre un chiffrement symétrique et un encodage ?

Solution: Dans le fond, un chiffrement ressemble à un encodage secret. L'encodage n'est lui pas secret. On note bien qu'on n'avait pas besoin d'une clef pour encoder en Base64.

La commande `enc` de OpenSSL permet de faire du chiffrement symétrique. Étudiez le chiffrement à partir du chiffrement par blocs AES. Vous pouvez utiliser l'option `-nosalt` pour obtenir un résultat plus facile à interpréter (si vous utilisez un mot de passe plutôt qu'une clef), et éventuellement l'option `-a` pour ne pas avoir un résultat binaire.

2. Quelles sont les tailles de clef possibles pour AES ?

Solution: 128, 192 et 256 bits.
La version avec une clef de 128 bits est la mieux étudiée et est la version que je recommande. Nous étudierons celle-là.

3. Utilisez la version avec 128 bits de clef. `openssl enc` permet de donner la clef en paramètre de la commande, au format hexadécimal. Rappelez-vous le format hexadécimal : quels sont les caractères hexadécimaux ? Combien de caractères y a-t-il ? À combien de bits correspond un caractère hexadécimal ?

Solution: Caractères : 1… 9 a… f
16 caractères
Nombre de bits : $\log_2(16) = 4$ bits

4. Combien de caractères hexadécimaux fera donc la clef d'AES ? Choisissez-en une (plus ou moins aléatoirement…).

Solution: $128/4 = 32$

5. Chiffrer le fichier `1.txt` en mode ECB, le résultat étant enregistré dans le fichier `1.ecb`. Combien de blocs fait `1.txt` ?

Solution: 3 blocs.

6. Chiffrer le fichier `2.txt` en mode ECB, le résultat étant enregistré dans le fichier `2.ecb`. Combien de blocs fait `1.txt` ?

Solution: 1,5 blocs.

7. Chiffrer le fichier `12.txt` en mode ECB, le résultat étant enregistré dans le fichier `12.ecb`. Combien de blocs fait `12.txt` ?

Solution: 4,5 blocs.

8. Comparez les résultats obtenus. Pourquoi ne faut-il pas utiliser le mode ECB (*i.e.* pas de mode) ?

Solution: Ce n'est pas sécurisé. Ici ce qu'on observe, c'est que le chiffré de `12.ecb` est la concaténation des chiffrés de `1.ecb` et `2.ecb`.

9. Chiffrer le fichier `21.txt` en mode ECB, le résultat étant enregistré dans le fichier `21.ecb`. Combien de blocs fait `21.txt` ?

Solution: 4,5 blocs.

10. Comparez les résultats obtenus. Quelle différence par rapport à `12.ecb` ?

Solution: C'est un peu différent. Le début de `21.ecb` est le début de `2.ecb` (1 bloc en commun). Le reste de `21.ecb` n'a rien à voir avec `1.ecb` ni avec `2.ecb`. C'est parce que `2.ecb` est le chiffré de 1,5 blocs, complété à 2 blocs avec du rembourrage. Le 2e bloc de chiffré de `2.ecb` est donc le chiffré de 0,5 bloc de clair et 0,5 bloc de rembourrage, tandis que le 2e bloc de `21.ecb` est le chiffré de 0,5 bloc de clair de `2.ecb` et de 0,5 bloc de clair de `1.ecb` !
De plus, dans la suite, tout est décalé d'un demi bloc dans le clair de `21.ecb` par rapport au clair de `1.ecb`, donc ce ne sont pas les mêmes blocs qui sont chiffrés par AES-128, et donc pas les mêmes blocs de chiffré.

11. Mêmes questions avec CBC et avec CTR. Attention, ces modes utilisent une IV (option `-iv`). Rappelez-vous des propriétés désirables sur cette IV pour avoir de la sécurité.

Solution: L'IV est un nonce : elle ne doit être utilisée qu'au plus une fois avec une clef donnée.

12. En mode CTR, chiffrez `f.txt` dans un fichier `f.ctr`. Puis, chiffrez `f.ctr` avec la **même** clef et la **même** IV. Qu'observez-vous ? Est-ce une propriété acceptable de sécurité ? À quoi sert l'IV ?

Solution: On observe qu'en mode CTR à IV fixe, le chiffrement = le déchiffrement. C'est un problème : un attaquant qui peut chiffrer peut aussi déchiffrer, ce qui pose un gros problème de sécurité dans la plupart des cas. C'est à ça que sert l'IV qui doit être un nonce : si on appelle la fonction de chiffrement une 2e fois, il faut que ce soit obligatoirement avec une IV différente pour éviter que ça ne déchiffre. Le nonce rend la fonction non-déterministe.

Notez en passant une propriété importante à retenir : **combiner des cryptosystèmes n'ajoute pas nécessairement de la sécurité, et peut même en retirer**. C'est d'ailleurs pour cette raison que le successeur de DES fut TripleDES et pas DoubleDES : DoubleDES était moins sûr que DES, et même trivial à attaquer.

Par exemple, si vous stockez des données D en format signées/hachées/chiffrées $C_{obs}(D)$ avec un cryptosystème obsolète C_{obs} dans une base de données, et que vous voulez mettre à jour pour utiliser un cryptosystème à jour C_{OK}, il est déconseillé de le faire sur les données signées/hachées/chiffrées : $C_{OK}(C_{obs}(D))$ peut avoir une mauvaise sécurité, on ne sait pas, alors que $C_{OK}(D)$ a une bonne sécurité (c'est étudié pour).

4 Chiffrement asymétrique

Génération de clefs asymétriques : OpenSSL propose différents algorithmes pour calculer des couples de clefs asymétriques. Utilisez l'algorithme RSA : la commande `genpkey` (ou `genrsa`) sert à engendrer une clef privée ; la commande `rsa` sert à engendrer une clef publique.

La commande `pkeyutl` (ou `rsautl`) sert à chiffrer avec ces clefs.

1. Engendrez une clef privée (secrète) et stockez-la dans le fichier `ma_rsa.sk`.
2. Engendrez la clef publique associée, et stockez-la dans le fichier `ma_rsa.pk`. Quelle différence notable avec la clef privée ?

Solution: La clef publique est nettement plus petite.

3. Quelles informations peuvent être obtenues sur ces clefs ? (option `-text` de la commande `rsa`)
4. Chiffrez le fichier `f.txt` avec la clef publique et stockez le résultat dans `f.cipher`. Déchiffrez-le avec la clef privée et vérifiez le résultat.
5. Signez le fichier `f.txt` à l'aide de la clef privée, et stockez le résultat dans `f.sign`. Attention : suivant la version de OpenSSL, il se peut que la signature ne puisse être faite que sur une donnée hachée. Il vous faudra alors commencer par la partie hachage ci-après avant de revenir sur la signature. Dans ce cas, hachez `f.txt` avec SHA-256 dans un fichier `f.sha256`, puis signez `f.sha256`.
6. Vérifiez le fichier signé (option -verify) à l'aide de la clef privée, puis de la clef publique.

5 Hachage

OpenSSL propose également des fonctions de hachage grâce à la commande `dgst`. Notez que sous Linux, les commandes `md5sum` et `sha*sum` (où * est 1, 224, 256, 384 ou 512) ont aussi cette vocation.

1. Calculez le hachage du fichier `longf.txt` avec l'algorithme MD5.

2. Calculez le hachage du fichier `longf.txt` avec divers algorithmes SHA*.
3. Comparez les résultats.

> **Solution:** Les meilleurs algorithmes ont un haché de plus grande taille.

4. Étudiez les conséquences du changement d'un seul caractère dans le fichier initial (par exemple changer une lettre minuscule en majuscule).

> **Solution:** Le haché n'a rien à voir, c'est tout le principe.

5. Le hachage des mots de passe sous Linux s'effectue par un algorithme classique. Consulter votre mot de passe haché dans le fichier `/etc/shadow` : sa version hachée est précédée du login, d'un chiffre symbolisant l'algorithme utilisé (1 pour MD5, 5 pour SHA-256, 6 pour SHA-512), et du "sel". Quel algorithme est utilisé ?

> **Solution:** Chez moi c'est SHA-256 par défaut.

6 Certificats

OpenSSL propose également la préparation et l'utilisation de certificats. Pour cela, il faut au préalable avoir engendré une clef privée, par exemple avec l'algorithme RSA (réutiliser les clefs engendrées dans les questions précédentes). La commande de requête ou de création de certificats pour OpenSSL est `req`.

1. À l'aide d'une clef privée existante, engendrez une requête de nouveau certificat et stockez-la dans le fichier `req.pem`. Répondez relativement sérieusement aux questions posées.
2. Que faut-il faire normalement avec une telle requête ?

> **Solution:** Il faut l'envoyer à une autorité de certification (CA) qui nous renverra un certificat signé avec sa clef privée, reconnu de confiance par tout le monde et qui prouvera le lien entre notre identité et notre clef publique.

3. Engendrez un certificat auto-signé avec l'algorithme/commande `x509`, valable 5 jours (`-req` permet que l'entrée de la commande soit une requête). Vérifiez-le ensuite. Quelles autres options peuvent être utilisées ?

   Normalement, générer le certificat est le travail d'une autorité de certification.

   Sur Internet, il s'agit souvent d'entreprises externes qui ont la charge de générer les certificats.

   Il se peut que vous soyez un jour en charge vous-même d'une autorité de certification ! En effet, il est de plus en plus courant qu'une entreprise ait sa propre CA en interne, en particulier pour une entreprise multi-sites, et que les canaux sécurisés entre 2 machines de cette entreprise soient établis à partir de cette CA interne. Dans ce cas, la requête de certificat faite par la machine 1 doit être envoyée à la machine de la CA interne (par un canal sécurisé quelconque de l'entreprise), et la personne en charge de la CA interne doit vérifier que cette demande de certificat est légitime, puis générer et signer le certificat, et renvoyer ce certificat à la machine 1 qui pourra l'intégrer dans ses propres certificats.

   Pour ajouter une nouvelle CA considérée "de confiance" (par exemple, on veut ajouter la CA interne de l'entreprise sur toutes les machines de l'entreprise) pour que ses certificats soient acceptés, on l'ajoute généralement au niveau du navigateur Web.

4. Cherchez comment ajouter une nouvelle CA dans votre navigateur Web favori.

> **Solution:** Dans Chromium par exemple, il faut aller dans les paramètres. Puis sous "Confidentialité et sécurité". Puis sous "Sécurité". Puis sous "Gérer les certificats". Là, vous pouvez par exemple rajouter une CA dans l'onglet "Autorités" en cliquant sur le bouton "Importer".

5. Une autre option est d'ajouter la CA directement dans le système d'exploitation. Cherchez comment faire sur votre OS favori.

> **Solution:** Sous la plupart des systèmes Unix par exemple, il y a un répertoire `/usr/share/ca-certificates`. Vous pouvez ajouter un répertoire `/usr/share/ca-certificates/extra` et y ajouter votre certificat (fichier `.crt` par exemple).
> Il peut être nécessaire de changer aussi un fichier de config avec la commande :
> `echo "/usr/share/ca-certificates/extra/new.crt" >> /etc/ca-certificates.conf`
> Il faut ensuite recharger la liste de CA pour que le changement soit effectif :
> `update-ca-certificates`