Université Côte d'Azur — L3 Info, L3 MI, L3 ST
Cryptographie et calcul symbolique — 2023-24

TP

Introduction à OpenSSL

OpenSSL est une boîte-à-outils cryptographique qui implémente les protocoles réseau Secure Sockets Layer (SSL v2/v3, couche de sockets sécurisés) et Transport Layer Security (TLS v1.3, sécurité pour la couche de transport) ainsi que les standards cryptographiques liés dont ils ont besoin. Il est actuellement utilisé par plus des deux-tiers des sites-web. Suite à la découverte de failles (la principale étant *Heartbleed* en 2014), il a été *forké* par openBSD en LibreSSL [1]. OpenSSL comporte 2 bibliothèques écrites en `C` (une de cryptographie générale et une implémentant le protocole SSL). Il contient aussi le programme `openssl`, un outil de ligne de commande pour utiliser les différentes fonctions cryptographiques de la librairie `crypto` d'OpenSSL à partir du `shell`. Voici un aperçu de ses différents usages :

- chiffrement et déchiffrement symétrique
- chiffrement et déchiffrement asymétrique
- création de paramètres des clefs
- signature digitales
- hachage cryptographique
- création de certificats
- tests SSL/TLS client et serveur
- gestion de mails S/MIME signés ou chiffrés

Les paramètres de la commande en ligne OpenSSL sont très nombreux et permettent notamment d'indiquer le type de chiffrement (pour ne citer que ceux à clef secrète : Blowfish, AES, DES ou Triple DES, Camellia, RC2, RC4, Aria, ...), d'encodage (base64, ...) et de hachage (MD5, SHA-1, ...). Enfin, cet utilitaire et les bibliothèques associées sont disponibles pour la plupart des Unix (dont Linux et Mac OS X), mais aussi pour Microsoft Windows.
Pour une étude plus approfondie, visitez le site officiel d'OpenSSL :

`http://www.openssl.org`

Préliminaires

Exercice 1)

1. Recueillez le fichier `tpOpenSSL.zip` sur le *Moodle* du cours. La page de `man` est accessible par la commande habituelle (`man openssl`).
2. Pour jeter un coup d'œil aux informations relatives à la version courante d'OpenSSL, commencez par taper la commande suivante, avec ou sans l'option :
   ```
   $ openssl version -a
   ```
3. Le moyen le plus simple de parvenir à la liste des sous-commandes principales (appelées *standard commands*) de la commande en ligne `openssl` est de saisir dans un terminal :
   ```
   $ openssl help
   ```

1. A noter qu'en 2015, Google a aussi lancé son propre utilitaire appelé `BoringSSL`.

Chiffrement à clef secrète

Exercice 2)

1. L'aide d'une sous-commande, comme par exemple `enc`, s'obtient en tapant la ligne ci-dessous ou en regardant la page du `man` (de `enc`) :
   `$ openssl enc -help`
2. Consultez les *chiffres* à votre disposition par l'option `-ciphers` de la commande `openssl enc`. L'option `-list` peut convenir aussi.
   Vous pouvez aussi vous référer à la page de manuel suivante :
   `https://www.openssl.org/docs/man3.0/man1/enc.html`
3. AES, le remplaçant de DES, a été évoqué en cours, pour plus d'informations voir :
   `http://fr.wikipedia.org/wiki/Advanced_Encryption_Standard`
   Le fichier binaire `mystere.2.2.enc` a été chiffré par AES avec une clef de 256 bits en mode `cbc`. Différents modes de chaînage du chiffrement par blocs sont en effet disponibles (*Cipher Block Chaining* `cbc`, *Electronic Code Book* `ecb`, *Cipher Feedback* `cfb` *etc*).
   - Commencez par décoder le mot de passe ayant servi au chiffrement. Il se trouve codé en `base64` dans le fichier `motDePasse` fourni (remarquez qu'il est juste codé, pas chiffré).
   - Procédez ensuite au déchiffrement du texte chiffré `mystere.2.2.enc` et visualisez-le en clair.
4. BLOWFISH est un autre algorithme de chiffrement « à clef secrète » conçu par B. Schneier en 1993 et qui repose sur un schéma de Feistel :
   `http://fr.wikipedia.org/wiki/Blowfish`
   La version complète avec 16 tours est à ce jour entièrement fiable.
   - Chiffrez un message clair en utilisant l'algorithme BLOWFISH et le mot de passe de votre choix. Par défaut, le chiffré apparaît sur la sortie standard en binaire.
     Recommencez en le redirigeant dans un fichier puis déchiffrez-le.
   - Pour inclure le chiffré dans un mail par exemple, il aurait fallu le coder en `base64` en utilisant l'option `-a`. Recommencez le chiffrement et le déchiffrement avec cette option.
   - Vous pouvez imaginer à quoi sert l'option `-salt`. Elle a été utilisée lors du chiffrement du fichier `mystere.2.3.enc` (en plus du codage en `base64`) avec BLOWFISH. Avec notre *clef secrète* commune, procédez au déchiffrement du fichier `mystere.2.3.enc`.
5. Si vous êtes soucieux du message `*** WARNING : deprecated key derivation used. Using -iter or -pbkdf2 would be better`, vous pouvez y remédier en ajoutant une des 2 options suggérées dès le chiffrement. Un *algorithme de dérivation de clefs* sera alors appliqué à votre mot de passe pour plus de sûreté.
6. Le mot de passe peut être introduit soit directement avec l'option `-pass` suivie de la précision `pass:motDePasse`, soit stocké dans un fichier avec `file:/chemin/vers/monFichier`. Testez ces deux options en variant les algorithmes de chiffrement symétrique (`camellia`, `rc4`, `des`, `aria`, `sm4` *etc*). Il y a le choix !
7. Observez enfin qu'un fichier chiffré de la même manière avec l'option `-salt` ou pas donne bien lieu à des chiffrés différents.

Chiffrement à clef publique

Exercice 3)

1. Prenez connaissance de la syntaxe et des différentes options possibles de la commande `genrsa`. A quoi sert-elle ?
2. Générez une clef privée[2] RSA de 1024 bits directement dans le fichier `clefPrivee.pem` (format `.pem` en base64 pour *Privacy Enhanced Mail*).
   Souvenez-vous bien du cours ! La clef privée d est l'inverse $\bmod \varphi(n)$ de l'exposant e de la clef publique (n, e). C'est justement cet exposant public e que OPENSSL affiche.
3. On ne conserve pas une clef privée en clair, ni seulement codée en `base64`. Le mieux aurait été de la chiffrer d'emblée avec un algorithme à clef secrète. Vu qu'elle existe déjà, chiffrez-la *a posteriori* et comparez le contenu des deux fichiers `.pem` obtenus.
   Assurez-vous aussi que vous auriez su la chiffrer au moment même de sa génération. Effacez de suite la version non chiffrée de votre clef privée.
4. Visualisez tout le cryptosystème construit au moyen de la commande suivante puis scrutez attentivement son contenu : `openssl rsa -in clefPriveeChiffree.pem -text`
5. La clef publique a vocation à être publiée : il faut l'extraire de la clef privée au moyen de l'option `-pubout`. Redirigez la sortie afin de consigner votre clef publique dans un fichier avec un nom qui permette de vous identifier.
   Visualisez le contenu de votre clef publique avec l'option `-pubin`. L'option `-text` vous permet d'en savoir plus sur son contenu. Envoyez votre clef publique par mail à votre binôme. Attention, n'envoyez surtout pas votre clef privée !
6. La commande `rsautl` permet de chiffrer/déchiffrer des données (`-encrypt`/`-decrypt`). Avec une clef dont le module est de 1024 bits, la taille du fichier à chiffrer ne doit pas excéder 116 octets. Chiffrez un petit texte clair avec la clef publique de votre binôme puis envoyez-lui le chiffré obtenu par mail.
7. Réciproquement, déchiffrez le message chiffré reçu de votre binôme. Comme il l'a chiffré avec votre clef publique, vous êtes absolument le seul à pouvoir le déchiffrer !

Nombres premiers

OPENSSL contient opportunément plusieurs procédures concernant les nombres premiers.

Exercice 4)

1. Visitez les possibilités de la commande `prime` d'OPENSSL.
2. Testez la primalité d'un nombre donné : en plus de la réponse, constatez que l'écho de ce nombre a lieu en hexadécimal.
3. Testez la primalité d'un nombre passé directement en hexadécimal.
4. *(facultatif)* Ecrivez un petit shell-script UNIX de façon à faire afficher les nombres premiers d'un intervalle donné (utilisez `seq` ou autre).

2. Il faut entendre le terme *clef privée* au sens large car ici, elle va aussi contenir la clef publique.

Nombres aléatoires et pseudo-aléatoires

OPENSSL contient aussi de quoi engendrer des nombres aléatoires, comme ceux fournis par UNIX dans les fichiers binaires `/dev/random` ou `/dev/urandom`.

Exercice 5)

1. La commande `rand` d'OPENSSL permet de générer des nombres aléatoires encodés en différentes bases. Engendrez un nombre aléatoire de 128 octets en base 64 puis en hexadécimal.
2. Générez un nombre aléatoire binaire de 1024 octets directement dans un fichier `random-data.bin` par exemple et visualisez-le avec la commande UNIX `xxd`.
3. On peut arriver au même résultat sous UNIX avec `/dev/random` et souvent avec une meilleure entropie. Extrayez les 32 premiers octets du fichier `/dev/random` et encodez-les en base 64 avec OPENSSL.
4. Consultez les pages de `man` pour connaître les différences entre `/dev/random` et `/dev/urandom`.