Crypto, Sécurité, Réseaux

TP 5+6

OpenSSL est une librairie qui implémente un des protocoles les plus utilisés aujourd'hui, notamment TLS. En conséquence openSSL permet l'utilisation de plusieurs algorithmes de chiffrement, signatures, hachage, etc. Cette librairie sert également à la programmation (en C) des applications s'appuyant sur le protocole TLS/SSL.

On peut utiliser openSSL pour des divers propos, y compris :

- Générer des clés de DSA et RSA (chiffrement/signatures à clé publique)
- La création des certificats X509
- Hacher des messages (avec par exemple SHA1)
- Utiliser des méthodes de chiffrement à clé symétrique (comme par exemple DES, 3DES, AES, etc.)
- La signature et chiffrement des courriers avec S/MIME.

Nous allons utiliser openSSL pour mettre en pratique ce que nous avons déjà appris en termes de primitives cryptographiques.

Ouvrez un terminal et tapez `openssl --help` . Ceci vous donnera :

- Une liste de commandes
- Une liste d'algorithmes de chiffrement symétrique (cipher commands)
- Une liste de fonctions de hachage

Ceci n'est pas une liste exhaustive d'algorithmes disponibles. Par exemple, avec la commande `rsa` nous pouvons accéder aux commandes de chiffrement et déchiffrement et aux signatures RSA.

RSA : clés, chiffrement, déchiffrement, signatures

Attention : dans ce TP vous allez devoir générer beaucoup de fichiers et les utiliser. Ne jamais superimposer des infos dans un fichier déjà créé sans être sûr que c'est ce que vous voulez !

Le chiffrement et les signatures RSA demandent l'utilisation d'une paire de clés : une clé publique et une clé privée. Il faut les générer tout d'abord.

1. Les clés RSA sont de divers tailles (en fonction de la taille du module RSA généré). Deux tailles typiques sont 1024 bits (maintenant en cours de déprécation) et 2048 bits (de plus en plus utilisée). En openSSL, les clés RSA générées sont stockés dans un fichier avec l'extension `.pem`. L'instruction à utiliser est : `openssl genrsa -out <fichier> <taille>`, où <fichier> indique le nom du fichier dans lequel on veut stocker les clés et <taille> indique la taille du module RSA.

Utilisez cette instruction pour générer une paire de clés de 1024 de bits et une paire de clés à 2048 bits, et stockez-les dans deux fichiers distincts.

2. Nous allons visualiser les clés. Premièrement lisez les deux fichiers en utilisant la commande `cat.`

   cat <fichier>

   Le format de la clé est par défaut en base 64. Lisez les deux fichiers

3. Une façon de visualiser les clés en format complet est d'utiliser la commande `rsa`.

   ```
   openssl rsa -in <fichier> -text -noout
   ```

   Les options -text et -noout vont afficher les clés en format hexadécimal, en supprimant la sortie normalement produite par l'instruction `rsa`.

   Quelles sont les différences entre les deux paires de clés générées ? Quelles informations sont contenus dans les deux fichiers .pem ?

4. Une paire de clés RSA contient une clé publique et une clé privée. La clé publique n'est pas une information sensible et elle peut être envoyée à d'autres utilisateurs. Par contre, la clé privée doit être protégée.

   Nous allons premièrement séparer la clé publique de la clé privée et nous allons la sauvegarder dans un autre fichier. Ceci se fait avec l'instruction :

   ```
   openssl rsa -in <fichier entree> -pubout -out <fichier sortie>
   ```

   Ici, le fichier en entrée c'est celui dans lequel vous avez stocké les clés générées, tandis que le fichier de sortie sera celui dans lequel vous voulez stocker la clé publique correspondante. Les deux fichiers auront l'extension `.pem`. Finalement, l'instruction `-pubout` assure que seulement la partie publique est exporté dans le fichier de sortie.

   Sauvegardez les clés publiques des deux paires de clés dans deux fichiers différents. Pour vérifier si la clé à bien été exporté il va falloir qu'on lise la clé publique dans le fichier qui stocke la clé entière (voire l'instruction utilisée dans l'exercice précédent) et aussi la clé dans le fichier qui stocke seulement la clé privée. Pour faire cette dernière opération, on peut soit lire le fichier avec la commande `cat` (mais le résultat sera en base 64), soit utiliser la commande suivante :

   ```
   openssl rsa -in <fichier> -pubin -text -noout
   ```

   L'option `-pubin` indique à openSSL qu'on doit s'attendre à avoir une clé publique dans le fichier indiqué.

5. Nous devons maintenant chiffrer la clé privée. Pour voir quels algorithmes on peut utiliser pour le chiffrement des clés privées on peut utiliser l'option --help à nouveau.

   ```
   openssl rsa --help
   ```

   Quelles sont les options pour chiffrer le fichier `.pem` ?

6. Nous allons utiliser un chiffré AES256. Ceci est un chiffré à clé symétrique, alors il va falloir indiquer une clé avec laquelle on chiffre. openSSL nous donne la possibilité de donner un mot de passe en entrée : étant donné le mot de passe, openSSL dérive une clé de chiffrement.

   ```
   openssl rsa -in <fichier PEM avec cle privee> -aes256 -out <meme fichier>
   ```

   Attention : le fichier qu'on veut chiffrer n'est pas celui qui contient juste la clé publique, mais celui qui contient la clé privée ! Ceci fera en sort qu'à chaque fois qu'on veut utiliser l'information stockée dans le fichier de clés, on devra mettre notre mot de passe.

7. Dans un fichier Plaintext.txt, écrivez un seul mot (à votre choix, sans accents).

   Nous allons chiffrer le fichier Plaintext.txt en utilisant la clé RSA à 1024 bits. Rappel : si on veut chiffrer à clé publique (RSA) un message à choix, on utilise la clé publique ou la clé privée ? La commande pour chiffrer le fichier Plaintext.txt est :

   ```
   openssl rsautl -encrypt -in Plaintext.txt -inkey <fichier contenant la cle> -pubin -out <fichier de sortie>
   ```

   Essayez lire le fichier qui stocke le chiffré avec la commande `cat`. Qu'observez-vous ?
   Pour lire le fichier en hexadécimal on peut utiliser la commande :

   ```
   od -Ad -tx1 <fichier chiffre>
   ```

   Le premier bloc de chaque colonne de l'affichage compte le nombre d'octets du chiffré. Qu'observez-vous concernant la taille du chiffré ? Pourquoi le chiffré a-t-il cette taille ?

8. Répétez l'exercice 7 en augmentant la taille du texte dans Plaintext.txt un mot à la fois, jusqu'au moment où l'instruction de chiffrement vous donne une erreur. Utilisez la commande `od -Ad -tx Plaintext.txt` pour déterminer la taille du texte clair. Essayez chiffrer le texte dans Plaintext.txt avec la clé RSA à 2048 bits. Qu'observez-vous ? Quelle est votre explication de ce phénomène ?

9. Demandez à un autre collègue (ou binôme) qu'il vous envoie le fichier contenant sa clé publique à 1024 bits. Utilisez la clé pour chiffrer un message à choix et envoyez à votre collègue le chiffré. L'instruction pour déchiffrer est :

```
openssl rsautl -decrypt -in <fichier du chiffrement> -inkey <fichier avec la cle de dechiffrement> -out <fichier de sortie>
```

Envoyez le chiffré à un autre collègue et recevez le chiffré d'un de vos collègues. Essayez de déchiffrer son chiffré (qui n'est pas chiffré avec votre clé publique) avec la clé privée.

10. Les clés RSA ne sont pas juste utiles pour chiffrer des messages, sinon également pour calculer des signatures sur des messages. Rappel : pour signer un message avec un schéma à clé publique (comme RSA) on utilise la clé publique ou la clé privée ?

    Choisissez un texte clair de taille petite. Calculez une signature RSA pour ce message en utilisant l'instruction :

    ```
    openssl rsautl -sign -in <fichier en entree> -inkey <fichier de la clé> -out <fichier de sortie>
    ```

    Envoyez à un de vos collègues les données suivantes : le fichier contenant la clé utilisée pour la vérification des signatures, le message que vous avez signé et le fichier contenant la signature. Vous allez recevoir les mêmes donnés. Vérifiez la validité de la signature en utilisant l'instruction :

    ```
    openssl rsautl -verify -in <signature> -pubin -inkey <fichier de la cle>
    ```

    Si la vérification s'est bien passée, on affichera le texte clair auquel corresponde la signature. Sinon, le résultat sera une erreur.

11. Créez un autre fichier plus long (par exemple prenez la première section de l'entrée Wikipedia sur Hash Functions), qui sera sauvegardé dans un fichier LongPlaintext.txt. Essayez de signer ce fichier avec la clé RSA à 2048 bits. Qu'observez vous ? Pourquoi ?

12. Nous voulons réduire la taille du message en entrée. Au lieu de signer le message lui-même, qui a une taille arbitraire (et dans notre cas, trop large), nous voulons plutôt signer une fonction du message qui nous donnera pourtant la sûreté que le message est authentique. Alors on va utiliser une fonction de hachage. Pour regarder quelles fonctions de hachage on peut utiliser, vous pouvez utiliser l'instruction :

    ```
    openssl dgst --help
    ```

    Nous allons utiliser l'algorithme `sha1`, qui calcule une empreinte de 160 bits. Ceci nous permettra de signer l'empreinte avec la clé RSA à 2048 bits. Quelle propriété de la fonction de hachage est nécessaire pour assurer l'authenticité du message étant donné que la signature vérifie ?

    Utilisez l'instruction :

    ```
    openssl dgst -sha1 -out <fichier sortie> LongPlaintext.txt
    ```

Puis, signez le texte dans votre fichier de sortie avec la clé RSA à 2048 bits. Puis, pour la vérification, utilisez l'instruction suivante :

```
openssl rsautl -verify -in <signature> -pubin -inkey <fichier de la cle> -out <un autre fichier de sortie>
```

Comparez le fichier où vous avez stocké l'empreinte SHA1 du fichier LongPlaintext.txt et le fichier où vous avez stocké la sortie de la dernière commande en utilisant la commande `cmp <premier fichier> <deuxieme fichier>` . Quel est le résultat ? Qu'est-ce que cela indique ?

13. Vous voulez maintenant envoyer un message à un collègue de votre choix, tel que ce message soit et authentifié et confidentiel. Il faut alors le chiffrer et l'authentifier. Rappelez-vous : dans quel ordre faut-il utiliser la signature et le schéma de chiffrement ?

    Choisissez un message. Vous voulez l'envoyer à un collègue tel que le message soit confidentiel et tel que votre collègue puisse vérifier que ce message vient de vous. À son tour il fera la même chose.

    Comment allez-vous faire cela ? Quelle(s) clé(s) devez vous envoyer à votre collègue ? Quelle(s) clé(s) devez-vous recevoir de son côté ?

Le chiffrement à clé symétrique ; les MACs

Le chiffrement à clé publique a plusieurs avantages : un nombre réduit de clés, ainsi que la possibilité d'envoyer des messages chiffrés ou signés aux gens qu'on connaît pas. D'un autre côté nous avons déjà pu observer que ces algorithmes ont également des inconvénients. (Lesquels ?) Alors on va voir également comment chiffrer et authentifier des messages à clé symétrique.

1. Reprenez le fichier LongPlaintext.txt . Pour le chiffrer avec un chiffré à clé symétrique vous pouvez utiliser l'instruction suivante :

   ```
   openssl -enc -<algorithme de votre choix> -in LongPlaintext.txt -out <fichier de sortie> -pass pass:<mot de passe>
   ```

   Pour voir les algorithmes qu'on peut utiliser pour chiffrer à clé symétrique vous pouvez utiliser l'instruction :

   ```
   openssl -enc --help
   ```

   Choisissez un algorithme de votre choix et faites chiffrer votre texte clair.

2. Pour déchiffrer le message, il faut utiliser l'instruction :

   ```
   openssl enc -d -<algoirthme de votre choix>-in <fichier chiffre> -out
   <fichier de sortie> -pass pass:<mot de passe>
   ```

   Pour vérifier si le chiffrement s'est bien passé utilisez cette instruction avec les mêmes paramètres que vous avez utilisé pour le chiffrement et sauvegardez le résultat dans un fichier différent de celui qui stocke le texte clair.

   Vérifiez si les deux fichiers sont identiques en utilisant la commande cmp indiquée dans la partie antérieure.

3. Dans l'exercice précédent vous avez donné votre mot de passe en clair. Toutefois, les données que vous donnez en entrée seront retenues dans le terminal, et donc elles pourraient être accédées par une autre personne. Une meilleure idée serait d'utiliser un fichier qui contiendrait le mot de passe.

   Pour cela on change l'instruction ci-dessus à :

   ```
   openssl -enc -<algorithme de votre choix> -in LongPlaintext.txt -out
   <fichier de sortie> -kfile <fichier dont la premiere ligne est le mot
   de passe>
   ```

4. Dans un autre fichier texte écrivez une phrase assez courte (sans accents). Chiffrez ce fichier avec l'algorithme aes256 avec un certain mot de passe. Sauvegardez le résultat dans un fichier SymCiphertext.txt.

   Changez le texte en entrée par un seul caractère. Chiffrez le résultat avec aes256 et le même mot de passe, dans un autre fichier, Sym2.txt.

   Pour observer les différences entre les deux fichiers on peut utiliser l'instruction :

   ```
   diff -y <(od -Ad -tx1 SymCiphertext.txt) <(od -Ad -tx1 Sym2.txt)
   ```

   Avec cette commande vous allez voir les deux fichiers l'un à côté de l'autre. Les 8 premiers octets seront les mêmes à cause du fait qu'on a dérivé la clé à partir du même mot de passe, sans utiliser un «sel» pseudo-aléatoire. Pour le reste des deux chiffrés, pouvez vous compter combien d'octets sont différents entre les deux fichiers ?

5. Voyons maintenant comment calculer le HMAC d'un fichier. L'instruction qui nous permettra d'obtenir la valeur de l'HMAC est :

   ```
   openssl dgst -sha1 -hex -hmac '<mot de passe>' -out <fichier de sortie>
   <fichier en entree>
   ```

Cette opération génère le HMAC (en hexadécimal) du fichier en entrée. La sortie est stockée dans le fichier de sortie.

6. Partagez votre clé d'HMAC et votre clé de chiffrement à clé symétrique avec un collègue. Vous voulez échanger un message tel que la confidentialité et l'authenticité du message soit respectée. Comment allez-vous chiffrer et authentifier l'information ?

   Envoyez le chiffrement et le MAC à votre collègue. Vous devriez avoir déjà vérifié que le chiffré est authentique avant de déchiffrer le message !

L'échange de clé Diffie-Hellman

Pour l'échange de clé Diffie-Hellman il faut premièrement générer la structure de groupe, y compris l'ordre et générateur de groupe. Puis, les deux partenaires vont générer des éléments privés et publiques qu'ils vont échanger.

1. Pour générer des paramètres Diffie Hellman utilisez l'instruction :

   ```
   openssl genpkey -genparam -algorithm DH -out <fichier .pem des parametres>
   ```

2. Lisez le fichier obtenu avec l'instruction `cat`. Quel est l'en tête des paramètres et dans quelle base sont-ils affichés ?
   Pour voir les paramètres en hexadécimal, vous pouvez utiliser une autre instruction, notamment :

   ```
   openssl pkeyparam -in <fichier des parametres> -text
   ```

   Quelle est la taille du nombre premier choisi ? Quel est le générateur du groupe ?

3. Dans la terminologie openSSL les tuples de valeurs de type (x, g^x) s'appellent « clés » : x est la clé privée, tandis que g^x est la clé publique. Il est important de réaliser que souvent ce dont on a besoin dans des divers protocoles ce sont des tuples (x, g^x) *éphémères* : c.a.d. elles ne sont pas des valeurs qui caractérisent un utilisateur à long terme, sinon juste des valeurs aléatoires, qui assurent la fraîcheur d'une valeur obtenue. Ceci est le cas lors des échanges Diffie-Hellman, où chaque partie choisit aléatoirement une valeur x et calcule g^x.

   Pour choisir un tuple (x, g^x) il nous faut le fichier de paramètres généré avant. L'instruction sera :

```
openssl genpkey -paramfile <fichier des parametres> -out <fichier
sortie, extension .pem>
```

Le fichier de sortie sera celui qui nous stockera les valeurs x, g^x, ainsi qu'une description des paramètres publiques.

4. Voyons les valeurs générées. Utilisez l'instruction :

   ```
   openssl pkey -in <fichier des valeurs generees> -text -noout
   ```

   Quelles valeurs sont incluses dans votre fichier ? On se rappelle que les valeurs seront appelées « clés », même si ce nom n'est pas toujours correct.

5. Répétez les exercices 3 et 4 en sauvegardant vos résultats dans des fichiers distincts que la première fois. Regardez les valeurs générées la deuxième fois. Est-ce qu'elles différent de vos premiers résultats ?

6. Nous allons séparer la partie privée de la partie publique. Pour sauvegarder la partie publique dans un nouveau fichier, il faut utiliser l'instruction

   ```
   openssl pkey -in <fichier des valeurs generees en exo 3> -pubout -out
   <fichier de sortie, extension .pem>
   ```

   Pour visualiser la clé publique on peut utiliser l'instruction :

   ```
   openssl pkey -pubin -in <fichier valeur publique>
   ```

   Faites cela deux fois pour les deux tuples de valeurs que vous avez générés.

7. Nous allons simuler l'échange de clés Diffie Hellman. Maintenant vous avez deux pairs de valeurs (x, g^x), (y, g^y), chacune dans un autre fichier. Rappelez-vous : comment est-ce qu'on utilise ces valeurs pour obtenir un secret commun ?

8. Utilisez la commande :

   ```
   openssl pkeyutl -derive -inkey <fichier contenant x, g^x> -peerkey
   <fichier contenant g^y> -out <fichier de sortie, extension .bin>
   ```

   Selon vous, quelle valeur est stocké dans votre nouveau fichier ?
   Répétez cette instruction pour l'autre participant au protocole d'échange de clé.

9. Une fois les secrets calculés, comparez les deux fichiers en utilisant l'instruction `cmp`.