Openssl

> Made By: Haroun Mohamed Akli

OpenSSl est une boite à outils utilisée pour réaliser toute sortes de fonctions cryptographiques

Il permet entre autres :

- Le cryptage et le décryptage de documents en utilisant des algorithmes symétriques et
  asymétriques
- La création des clés RSA, DES, ...etc
- La création de certificats X509
- La signature numérique
- Le calcul d'empreinte
- Le test des applications SSL/TLS

Chiffrement symétrique

Le chiffrement symétrique est une méthode de cryptographie où une

même clé est utilisée à la fois pour **chiffrer** (rendre illisible) et **déchiffrer** (rendre lisible) les données. Il est appelé "symétrique" parce que la même clé est partagée entre l'expéditeur et le destinataire.

Fonctionnement de base :

1. **Chiffrement** :
    - L'expéditeur utilise une clé secrète pour transformer le message clair en un message chiffré (appelé *texte chiffré*).
2. **Transmission** :
    - Le texte chiffré est envoyé au destinataire via un canal de communication (par exemple, un réseau).
3. **Déchiffrement** :
    - Le destinataire utilise la même clé secrète pour retransformer le texte chiffré en texte clair.

Avantages :

- **Rapidité** : Les algorithmes de chiffrement symétrique sont généralement plus rapides que les algorithmes asymétriques.
- **Simplicité** : L'utilisation d'une seule clé rend son implémentation plus directe.

Inconvénients :

- **Gestion des clés** : La clé doit être partagée de manière sécurisée entre les parties. Si elle est interceptée ou compromise, tout le système est vulnérable.
- **Nombre de clés** : Dans un réseau avec plusieurs participants, il faut gérer une clé unique pour chaque paire d'individus.

Exemples d'algorithmes :

- **AES (Advanced Encryption Standard)** : Très utilisé pour sa sécurité et sa rapidité.
- **DES (Data Encryption Standard)** : Plus ancien, mais moins sécurisé aujourd'hui.
- **Blowfish** et **RC4** : Autres exemples populaires.

Cryptage

```
openssl enc -in fichier.txt -out fichier.enc -e -des3
```

- Utilise le triple DES (dans cet exemple) pour crypter le fichier.txt et générer un fichier crypté fichier.enc
- La clé de cryptage est générée automatiquement à partir d'un mot de passe demandé par openssl
- Pour savoir quels sont les algorithmes symétriques implémentés, on tape:

```
openssl enc -ciphers
```

Décryptage

```
openssl enc -in fichier.enc -out fichier.txt -d -des3
```

- -d pour décrypter

Génération d'une clé RSA

La génération d'une clé RSA (Rivest-Shamir-Adleman) implique la création d'une paire de clés : une **clé publique** et une **clé privée**. Cette paire est utilisée dans la cryptographie asymétrique, où la clé publique chiffre les données, et la clé privée les déchiffre.

Génération d'un couple de clés RSA

Syntaxe :

```
openssl genrsa -out cle.rsa <size>
```

Options :

- `genrsa` : Génère une clé privée RSA.

- `out cle.rsa` : Spécifie le fichier de sortie où la clé privée sera enregistrée (dans cet exemple, `cle.rsa`).
- `<size>` : Indique la taille de la clé RSA en bits (par exemple, 2048, 3072, 4096).

Exemple :

```
openssl genrsa -out cle.rsa 2048
```

- Cela génère une clé RSA privée de 2048 bits et l'enregistre dans un fichier nommé `cle.rsa`.

Exportation de la clé publique

Commande :

```
openssl rsa -in cle.rsa -pubout -out cle.pub
```

Options expliquées :

- `openssl rsa` : Utilitaire OpenSSL pour manipuler les clés RSA.
- `in cle.rsa` : Spécifie la clé privée RSA en entrée (ici, le fichier `cle.rsa`).
- `pubout` : Indique que la clé publique doit être extraite à partir de la clé privée.
- `out cle.pub` : Définit le fichier de sortie où la clé publique sera sauvegardée (dans cet exemple, `cle.pub`).

Exemple complet :

1. **Générer une clé privée :**

   ```
   openssl genrsa -out cle.rsa 2048
   ```

2. **Extraire la clé publique :**

   ```
   openssl rsa -in cle.rsa -pubout -out cle.pub
   ```

3. **Vérifier le contenu de la clé publique :**

```
cat cle.pub
```

Résultat :

- Le fichier `cle.rsa` contiendra la clé privée RSA.
- Le fichier `cle.pub` contiendra la clé publique RSA au format PEM, utilisable pour le chiffrement ou la vérification de signatures numériques.

Affichage des détails de la clé publique :

Si vous souhaitez voir les informations détaillées sur la clé publique, utilisez cette commande :

```
openssl rsa -pubin -in cle.pub -text -noout
```

Format de la clé publique :

Le fichier `cle.pub` sera au format PEM (Base64 encodée) et ressemblera à ceci :

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA...
...contenu encodé...
-----END PUBLIC KEY-----
```

Affichage des détails de la clé privée :

```
openssl rsa -in cle.rsa -text -noout
```

Format de la clé privée :

Le fichier `cle.rsa` ressemblera à ceci :

```
Private-Key: (1024 bit) modulus:
     00:af:79:58:cb:96:d7:af:4c:2e:64:48:08:93:62:
     31:cc:56:e0:11:f3:40:c7:30:b5:82:a7:70:4e:55:
     9e:3d:79:7c:2b:69:7c:4e:ec:07:ca:5a:90:39:83:
```

```
    4c:05:66:06:4d:11:12:1f:15:86:82:9e:f6:90:0d:
    00:3e:f4:14:48:7e:c4:92:af:7a:12:c3:43:32:e5:
    20:fa:7a:0d:79:bf:45:66:26:6b:cf:77:c2:e0:07:
    2a:49:1d:ba:fa:7f:93:17:5a:a9:ed:bf:3a:74:42:
    f8:3a:75:d7:8d:a5:42:2b:aa:49:21:e2:e0:df:1c:
    50:d6:ab:2a:e4:41:40:af:2b
publicExponent: 65537 (0x10001)

privateExponent:
    35:c8:54:ad:f9:ea:db:c0:d6:cb:47:c4:d1:1f:9c:
    b1:cb:c2:db:dd:99:f2:33:7c:be:b2:01:5b:11:24:
    f2:24:a5:29:4d:28:9b:ab:fe:6b:48:3c:c2:53:fa:
    de:00:ba:57:ae:ae:c6:36:3b:c7:17:5f:ed:20:fe:
    fd:4c:a4:56:5e:0f:18:5c:a6:84:bb:72:c1:27:46:
    96:07:9c:de:d2:e0:06:d5:77:ca:d2:45:8a:50:15:
    0c:18:a3:2f:34:30:51:e8:02:3b:8c:ed:d4:95:98:
    73:ab:ef:69:57:4d:c9:04:9a:18:82:1e:60:6b:0d:
    0d:61:18:94:eb:43:4a:59

prime1:
    00:d5:46:7f:49:4b:5e:46:f2:9f:87:e1:9d:0f:6d:
    4f:59:42:0d:8b:d6:ef:7f:48:3f:e4:10:b5:5e:dc:
    ac:75:7d:d1:ee:97:11:f4:bf:c5:76:02:03:dd:5c:
    0d:bd:36:18:be:4f:15:2d:0b:e5:7c:08:c7:36:29:
    79:80:bc:5b:07

prime2:
    00:d2:a0:43:9a:31:cc:0c:fc:1c:19:aa:9b:43:69:
    72:99:84:08:1a:56:79:4c:b4:05:03:df:03:7b:2d:
    ae:13:e0:81:ea:99:92:fd:ef:d9:8a:5b:b5:21:a6:
    ac:b8:4d:ef:07:35:df:07:f2:54:ec:35:24:57:ef:
    89:43:b4:ed:bd

exponent1:
    00:bd:ac:e5:d5:1c:87:6b:17:aa:53:a1:8e:1a:43:
    3f:f7:84:ec:21:3a:f5:62:c0:b1:b9:b6:36:67:78:
    60:94:59:62:d4:0b:5c:f7:cb:79:e4:9a:a4:2f:41:
    08:23:07:b2:77:c6:43:71:fd:8b:89:85:11:0e:95:
```

```
    52:2e:f0:d5:0f

exponent2:
    04:1a:a5:56:92:d3:d4:08:f1:8f:3a:78:ce:06:76:
    fa:30:cd:6b:9d:f5:bd:1d:e0:df:23:70:50:ed:21:
    f0:37:36:b0:d8:8f:39:ad:7b:c2:ab:68:cb:20:11:
    4b:82:11:3f:45:b8:73:d2:2f:ff:6e:45:a8:04:fd:
    da:b8:e2:cd

coefficient:
    23:10:53:83:cc:aa:43:2d:c3:30:85:b1:5f:19:a8:
    b9:a4:0c:f9:f5:6e:29:c8:03:04:4b:60:57:2c:41:
    10:ed:81:38:ba:af:27:33:dc:f9:35:84:25:73:05:
    fc:8c:77:cc:f0:aa:9c:0a:99:1e:45:a0:e5:ee:24:
    4b:fe:99:58
```

Cryptage/ décryptage RSA

Chiffrement avec la clé publique

```
openssl rsautl -encrypt -in fichier.txt -pubin -inkey cle.pub -out chiffre.enc
```

Options expliquées :

- `rsautl` : Utilitaire OpenSSL pour effectuer des opérations RSA sur des fichiers ou données.
- `encrypt` : Indique que vous voulez chiffrer des données.
- `in fichier.txt` : Spécifie le fichier d'entrée (le fichier en texte clair à chiffrer).
- `pubin` : Indique que la clé fournie (avec `inkey`) est une clé publique.
- `inkey cle.pub` : Spécifie le fichier contenant la clé publique RSA à utiliser pour le chiffrement.
- `out chiffre.enc` : Définit le fichier de sortie, qui contiendra les données chiffrées.

Principe :

- Le contenu du fichier `fichier.txt` est chiffré avec la clé publique `cle.pub`, produisant un fichier chiffré `chiffre.enc`.
- Seule la clé privée associée (dans `cle.rsa`) pourra déchiffrer le fichier.

Déchiffrement avec la clé privée

```
openssl rsautl -decrypt -inkey cle.rsa -in chiffre.enc -out
fichier.txt
```

Options expliquées :

- `rsautl` : Utilitaire OpenSSL pour effectuer des opérations RSA.
- `decrypt` : Indique que vous voulez déchiffrer des données.
- `inkey cle.rsa` : Spécifie le fichier contenant la clé privée RSA à utiliser pour le déchiffrement.
- `in chiffre.enc` : Spécifie le fichier chiffré (à déchiffrer).
- `out fichier.txt` : Définit le fichier de sortie, qui contiendra les données déchiffrées (en clair).

Principe :

- Le fichier chiffré `chiffre.enc` est déchiffré à l'aide de la clé privée `cle.rsa` pour produire le fichier d'origine `fichier.txt`.

Résumé du processus :

1. L'expéditeur chiffre un fichier avec la **clé publique** du destinataire (`cle.pub`).
2. Le destinataire déchiffre le fichier avec sa **clé privée** (`cle.rsa`).

Signature numérique

La signature numérique permet de garantir l'intégrité et l'authenticité d'un document.

1. Signature d’un petit document avec la clé privée

Commande :

```
openssl rsautl -sign -inkey cle.rsa -in fichier.txt -out fi
chier.sig
```

Options expliquées :

- `rsautl` : Utilitaire pour opérer avec RSA.
- `sign` : Utilise la clé privée spécifiée pour signer les données.
- `inkey cle.rsa` : Spécifie le fichier contenant la clé privée RSA utilisée pour signer.
- `in fichier.txt` : Fichier d'entrée à signer (ici, `fichier.txt`).
- `out fichier.sig` : Produit le fichier signé (la signature numérique, ici `fichier.sig`).

Principe :

1. La clé privée est utilisée pour chiffrer le contenu entier de `fichier.txt`.
2. Le fichier `fichier.sig` est la signature qui peut être vérifiée avec la clé publique correspondante.

Vérification de la signature :

Pour vérifier le fichier signé :

```
openssl rsautl -verify -inkey cle.pub -pubin -in fichier.si
g -out fichier_verifie.txt
```

- Si `fichier_verifie.txt` est identique à l'original (`fichier.txt`), la signature est valide.

---

2. Signature d'un document volumineux à l'aide du hash (MAC)

Étape 1 : Générer l'empreinte (hash)

```
openssl dgst -md5 -out fichier.hash fichier.txt
```

Options expliquées :

- `dgst` : Utilitaire pour générer des fonctions de hachage.
- `md5` : Utilise l'algorithme de hachage MD5 (remplaçable par `sha256` pour plus de sécurité).
- `out fichier.hash` : Spécifie le fichier où sera sauvegardé le hash.
- `fichier.txt` : Document source dont le hash sera calculé.

Étape 2 : Signer le hash avec la clé privée

```
openssl rsautl -sign -inkey cle.rsa -in fichier.hash -out f
ichier.sig
```

Principe :

1. Le fichier volumineux n'est pas signé directement. À la place, une empreinte cryptographique (hash) du fichier est calculée.
2. Cette empreinte est signée avec la clé privée pour générer le fichier `fichier.sig`.

Vérification de la signature :

1. Recalculer l'empreinte du fichier original :

   ```
   openssl dgst -md5 -out fichier_verif.hash fichier.txt
   ```

2. Vérifier la signature avec la clé publique :

   ```
   openssl rsautl -verify -inkey cle.pub -pubin -in fichie
   r.sig -out fichier_verifie.hash
   ```

3. Comparer les deux empreintes (`fichier_verif.hash` et `fichier.hash`). Si elles sont identiques, la signature est valide.

   #### Commande :

   ```
   diff -q -s fichier.hash fichier.ver
   ```

   #### Options expliquées :

- `q` : Mode silencieux. Affiche uniquement si les fichiers sont différents ou identiques, sans afficher les différences en détail.
- `s` : Affiche un message pour indiquer explicitement si les fichiers sont identiques.

Exemple de sortie :

- Si les fichiers sont identiques :

```
Files fichier.hash and fichier.ver are identical
```

- Si les fichiers sont différents :

```
Files fichier.hash and fichier.ver differ
```

Gestion de certificats numériques

La gestion des certificats numériques commence par la création d'une **Autorité de Certification (CA)**, qui permet de signer des certificats pour les clients ou les serveurs. Voici les étapes détaillées pour créer une CA avec OpenSSL :

1. Génération d'un couple de clés pour l'Autorité de Certification (CA)

Commande :

```
openssl genrsa -out ca.key 2048
```

Explications :

- `genrsa` : Génère une clé privée RSA.
- `out ca.key` : Enregistre la clé privée dans le fichier `ca.key`.
- `2048` : Spécifie la taille de la clé RSA en bits (2048 bits est une taille minimale recommandée).

Clé privée de la CA :

Cette clé est essentielle pour signer les certificats et doit rester secrète.

---

2. Génération d’un certificat auto-signé pour la CA

Commande :

```
openssl req -new -x509 -days 356 -key ca.key -out ca.crt
```

Explications :

- `req` : Utilitaire pour gérer les requêtes et certificats X.509.
- `new` : Génère une nouvelle requête ou un certificat.
- `x509` : Indique que nous générons un certificat auto-signé.
- `days 356` : Spécifie la durée de validité du certificat (ici, 356 jours, soit ~1 an).
- `key ca.key` : Utilise la clé privée générée précédemment pour signer le certificat.
- `out ca.crt` : Enregistre le certificat auto-signé dans le fichier `ca.crt`.

Lors de l'exécution, des informations seront demandées :

- **Country Name (2 letter code)** : Code pays (ex. `FR` pour la France).
- **State or Province Name** : Nom de l’état ou région.
- **Locality Name** : Ville.
- **Organization Name** : Nom de l’organisation.
- **Organizational Unit Name** : Département ou service.
- **Common Name** : Nom commun (par exemple, le domaine ou le nom de la CA).
- **Email Address** : Adresse email associée à la CA.

---

3. Utilisation du certificat CA

- **Certificat auto-signé (`ca.crt`)** : Ce certificat représente votre CA et doit être distribué aux clients pour qu'ils lui fassent confiance.
- **Navigateurs** : Intégrez `ca.crt` dans les navigateurs pour que les certificats signés par votre CA soient reconnus comme valides.

- **Applications** : Importez `ca.crt` dans les systèmes (mail, FTP, etc.) qui doivent utiliser vos certificats.

Important :

- **Sécurité de la clé privée :** La clé privée `ca.key` doit être protégée rigoureusement. Si elle est compromise, toute la chaîne de certificats sera vulnérable.
- **Validité :** Une durée de validité trop longue augmente le risque de sécurité, mais est courante pour les autorités racines (5-10 ans).
- **Format standard :** Le certificat (`ca.crt`) suit le format **X.509** utilisé par la plupart des systèmes.

Générer une demande de certificat pour l'autorité de certification (certificate Signing Request)

Voici les étapes pour générer une demande de certificat et le signer avec l'Autorité de Certification (CA) :

---

1. Générer une clé privée pour le serveur

Commande :

```
openssl genrsa -out serveur.key 2048
```

Explications :

- `genrsa` : Génère une clé privée RSA.
- `out serveur.key` : Enregistre la clé privée dans le fichier `serveur.key`.
- `2048` : Spécifie la taille de la clé (2048 bits).

---

2. Générer une requête de signature de certificat (CSR)

Commande :

```
openssl req -new -key serveur.key -out serveur.csr
```

Explications :

- `req` : Utilitaire pour gérer les requêtes de certificats.
- `new` : Indique qu'il s'agit d'une nouvelle CSR.
- `key serveur.key` : Utilise la clé privée du serveur pour signer la CSR.
- `out serveur.csr` : Produit la CSR et la sauvegarde dans `serveur.csr`.

Informations demandées :

Vous devrez remplir les champs suivants lors de la génération :

- **Country Name (2 letter code)** : Code du pays (ex. `DZ`).
- **State or Province Name** : État ou région.
- **Locality Name** : Ville.
- **Organization Name** : Nom de l'entreprise.
- **Organizational Unit Name** : Département ou division.
- **Common Name** : Nom du domaine ou adresse IP du serveur.
- **Email Address** : Adresse email.

---

3. Signer le certificat avec la CA

Commande :

```
openssl x509 -req -in serveur.csr -out serveur.crt -CA ca.c
rt -CAkey ca.key -CAcreateserial -CAserial ca.srl
```

Explications :

- `x509` : Utilitaire pour gérer les certificats X.509.
- `req` : Indique que la commande traite une CSR pour générer un certificat.
- `in serveur.csr` : Spécifie la CSR comme entrée.
- `out serveur.crt` : Produit le certificat signé et le sauvegarde dans `serveur.crt`.

- `CA ca.crt` : Spécifie le certificat auto-signé de la CA.
- `CAkey ca.key` : Spécifie la clé privée de la CA pour signer le certificat.
- `CAcreateserial` : Crée un fichier de numéro de série (`ca.srl`) pour suivre les numéros de certificats émis. Nécessaire pour le premier certificat.
- `CAserial ca.srl` : Utilise le fichier `ca.srl` pour le numéro de série du certificat.

Résultat :

- `serveur.crt` : Certificat signé pour le serveur.
- `ca.srl` : Fichier contenant le numéro de série utilisé pour les certificats futurs.

Vérification :

1. **Afficher les détails du certificat serveur :**

   ```
   openssl x509 -in serveur.crt -text -noout
   ```

   Cela affiche les informations du certificat, y compris la validité, le Common Name, et les extensions.

2. **Vérifier que le certificat est signé par la CA :**

   ```
   openssl verify -CAfile ca.crt serveur.crt
   ```

   - Si le certificat est valide, vous verrez :

     ```
     serveur.crt: OK
     ```


> Made By : Aklidevlop
>
> Haroun Mohamed AKli