Travaux pratiques –TP
Mise en œuvre d'une PKI avec Openssl

Partie 1 : OpenSSL

OpenSSL est une boîte à outils cryptographiques implémentant les protocoles SSL et TLS. Il offre :

- Une bibliothèque de programmation en C permettant de réaliser des applications client/serveur sécurisées s'appuyant sur SSL/TLS
- Une commande en ligne (OpenSSL) permettant
  - la création de clés RSA, DSA (signature)
  - la création de certificats X509
  - le calcul d'empreintes (MD5, SHA, RIPEMD160, ...)
  - le chiffrement et déchiffrement (RSA, DES, IDEA, RC2, RC4, Blowfish, ...)
  - la réalisation de tests de clients et serveurs SSL/TLS
  - la signature et le chiffrement de courriers (S/MIME)

La syntaxe générale de la commande openssl est `$ Openssl <commande> <option>`
On trouvera toutes les informations la concernant à l'adresse : http://www.openssl.org

Définition des AC dans le navigateur

a) Ouvrez le navigateur Firefox sur windows et affichez la liste des Autorités de Certification par défaut
b) Affichez le certificat du serveur https://www.lcl.fr/
   1. Qui l'a certifié ?
   2. Quelle est sa date de validité ?
   3. Quel est l'algorithme de la clé publique utilisée ? la taille de la clé ?
   4. Sauvegarder le certificat dans un fichier appelé Certificat_lcl.crt
   5. Pour vérifier le nombre de certificat révoqué par l'autorité de confiance qui a signé le certificat :
      i. Copiez du certificat le lien qui pointe vers la liste CRL
      ii. Pour afficher la liste des certificats révoqués par l'autorité de confiance, tapez sur windows *exécuter -> cmd* et ensuite la commande suivante:
      `certutil –split –URL COLLER_LE_LIEN_ICI`
      iii. Vérifiez l'état du certificat avec la commande :
      `certutil –verify Certificat_nasa.crt`
      iv. Afficher la liste des certificats dans le magasin de certificats de Windows
      `certutil –viewstore`
      v. Afficher le cache CRL dans votre machine
      `certutil –urlcache CRL`
      vi. Pour afficher les options de la commande certutil tapez : `certutil -?`

Partie 2 : Mise en œuvre d'une PKI

L'objectif de ce TP est de mettre en œuvre une PKI (*Public Key Infrastructure*). Vous jouerez donc le rôle de l'autorité de certification *TrustSign*. Pour cela il vous faut un certificat d'autorité de certification, ainsi qu'une paire de clés. La figure 1, représente ce scénario. Il faut mettre en œuvre ce scenario afin que l'utilisateur puisse accéder en https au serveur web. Les fichiers à générer pour mettre en œuvre cette PKI sont :

Coté Serveur web :

ServerPrivatekey.key : Fichier qui contient les paramètres des clés publique/privée du serveur
ServerPublicKey.key : Fichier qui contient uniquement la clé publique du serveur
ServerCertificatRequest.csr : Fichier généré par le serveur, il contient la requête de la demande de signature par l'AC
ServerCertificat.crt : certificat du serveur (à envoyer aux clients qui se connectent en https au serveur)

Coté autorité de certification :

CaPrivateKey.key : Fichier qui contient les paramètres des clés publique/privée de l'autorité de confiance
CertCA.crt : le certificat de l'autorité de confiance

Coté client :

ClientPrivatekey.key : Fichier qui contient les paramètres des clés publique/privée du client
ClientPublicKey.key : Fichier qui contient uniquement la clé publique du client
ClientCertificatRequest.csr : Fichier généré par le client, il contient la requête de la demande de signature par l'AC
ClientCertificat.crt : certificat du client (à envoyer au serveur dans le cas d'une authentification mutuelle avec le serveur)

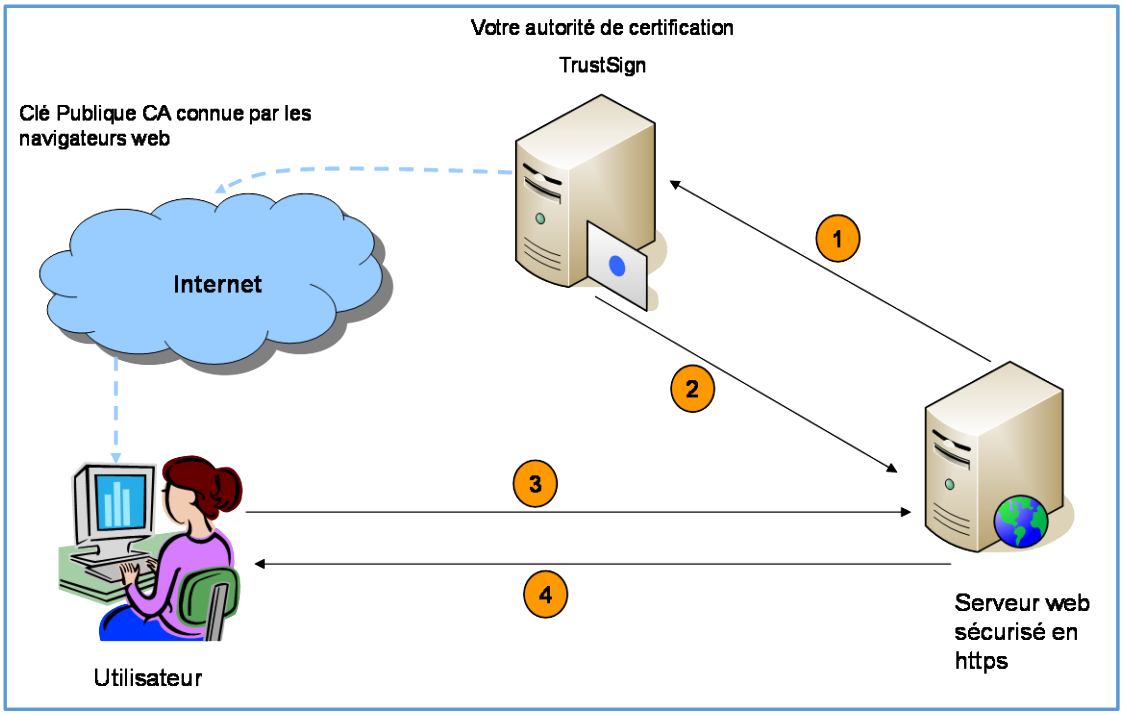

Figure 1. Mise en oeuvre d'une PKI

a) Génération de la paire de clés du serveur

Engendrez une paire de clés RSA de 1024 bits protégée par mot de passe. On supposera ensuite que le fichier ServerPrivateKey.*key* contient votre paire de clés, sécurisé par un mot de passe. Commande à utiliser *openssl genrsa [options]*

Créez le fichier *ServerPublicKey.key* ne contenant que la clé publique.

b) Création d'une requête de certificats

Disposant de votre clé publique, vous pouvez établir une requête *ServerCertificatRequest.csr* pour obtenir un certificat. Lors de la création du fichier de requête, il vous sera demandé les informations relatives à la norme X509 qu'il vous faudra saisir. Voir annexe 4.

Commande à utiliser *openssl req [options]*

Remarque : dans la pratique, on utilise l'outil fourni par la CA qu'on a choisi pour générer la requête csr. Exemple :

- Symantec propose l'outil : *Symantec SSL Assistant, à télécharger site de Symantec*
  - *https://www.symantec.com*

- *DigiCert propose l'outil : DigiCert Certificate Utility for Windows, à télécharger du site DigiCert.*
  - *https://www.digicert.com/util/DigiCertUtil.zip*

Les objets cryptographiques (clés, requêtes, signature,) doivent être générés par des outils certifiés appelées *Cryptographic Service Providers (CSP) comme*

- *Microsoft Strong Cryptographic Provider*
- *Microsoft RSA SChannel Cryptographic Provider*
- *Microsoft Base Smart Card Cryptographic Service Provider*

c) L'autorité de certification *TrustSign*

Vous jouerez le rôle de l'autorité de certification. Pour cela il vous faut un certificat d'autorité de certification, ainsi qu'une paire de clés.

Création du certificat du serveur par l'autorité de confiance

Engendrez une paire de clés RSA de 2048 bits protégée par mot de passe. On supposera ensuite que le fichier *CaPrivateKey.key* contient la paire de clés, sécurisé par un mot de passe.

Créez un certificat pour *TrustSign* avec la commande openssl req –x509 [options], voir l'annexe 4.

Pour créer et signer un certificat à partir d'une requête *ServerCertificatRequest.csr*, l'autorité invoque la commande :

openssl x509 [options], voir l'annexe 4.

Le certificat du serveur est le fichier : *ServerCertificat.crt*

Vérification de certificats

On peut vérifier la validité d'un certificat avec la commande *verify*. Pour vérifier la validité d'un certificat, il est nécessaire de disposer du certificat de l'autorité qui l'a émis.

```
openssl verify -CAfile CertCA.crt  ServerCertificat.crt
```

Partie 3 : Installation et configuration du serveur web

Installation du serveur web (s'il n'est pas installé sur votre machine)

Faire une mise à jour avec la commande

```
# apt-get update
```

Installer le serveur apache2 avec la commande

```
# apt-get install apache2
```

Démarrer le serveur apapche2 avec la commande :

```
# service apache2 start
```

Lancer un web browser (Mozilla Firefox, par exemple) et taper http://192.168.1.1

NB : Remplacer 192.168.1.1 par l'adresse de votre machine.

Authentification du client par un mot de passe

L'objectif est de mettre en place une authentification classique du client par mot de passe sur un dossier donné. Ouvrez le fichier de configuration /etc/apache2/sites-available/default et y ajouter les directives suivantes qui sont en *bold*.

```
<VirtualHost *:80>
DocumentRoot /var/www/
<Directory /var/www/>
        Order allow,deny
        Allow from all
        Authname "Authentifiez-vous SVP"
        Authtype basic
        AuthUserFile /var/www/htpasswd
        Require valid-user
</Directory>
</VirtualHost>
```

AuthName : c'est est le message qui sera affiché dans la boite de dialogue de saisie de l'identifiant et du mot de passe.
AuthType : indique à Apache d'utiliser le protocole *Basic* pour authentifier l'utilisateur, c'est le protocole le plus simple et le plus compatible avec tous les navigateurs.
AuthUserFile : indique le chemin vers le fichier contenant les noms des utilisateurs et leurs mots de passe.
require valid-user : indique à Apache que tout utilisateur ayant réussi à rentrer son mot de passe est autorisé à rentrer dans le dossier

Il faut maintenant fournir une liste d'utilisateurs avec leurs mots de passe. Le simple est d'utiliser l'infrastructure de mots de passe d'Apache lui-même avec la commande Apache *htpasswd*. *htpasswd* permet de créer et de maintenir les fichiers textes où sont stockés les noms d'utilisateurs et mots de passe pour l'authentification de base des utilisateurs HTTP

Syntaxe : *htpasswd -cm htpasswd toto*

-c : pour créer le fichier qui va contenir les mots de passe
-m : pour hacher les mots de passe avec md5

Redémarrage du service : service apache2 restart

Authentification du serveur par un certificat

Pour authentifier le serveur par un certificat, il faut configurer et activer le module *ssl* (c'est le module qui va implémenter le protocole SSL au-dessous de http). Le module ssl (les fichiers ssl.conf et ssl.load) se trouve déjà dans */etc/apache2/mods-available/.*

Généralement pour activer ou désactiver des modules Apaches, il suffit d'utiliser les commandes suivantes :

a2enmod {nom_du_module} pour l'activation

a2dismod {nom_du_module} pour la désactivation

Pour activer, dnc, le module ssl, taper les commandes suivantes :

a2enmod ssl et a2ensite default-ssl

Redémarrez le serveur : service apache2 restart

Vérifiez que le module ssl est activé en tapant sur le browser : ***https://192.168.1.1***

Pour utiliser le certificat serveur que vous avez signé dans la partie précédente, ouvrez le fichier /etc/apache2/sites-available/default-ssl, et y ajouter le *path* vers le certificat du serveur (ServerCertificat.crt) et la clé privée du serveur (ServerPrivateKey.key) :

SSLEngine on
SSLCertificateFile /etc/ssl/certs/ ServerCertificat.crt
SSLCertificateKeyFile /etc/ssl/private/ ServerPrivateKey.key
Redémarrez le serveur avec la commande : service apache2 restart

Connexion à https:// 192.168.1.1:443

NB : Remplacer 192.168.1.1 par l'adresse IP de votre machine

Par défaut, Apache2 est configuré pour écouter en *http* sur le port 80 et en *https* sur le 443. Vous pouvez vérifier les ports actifs sur votre machine à l'aide de la commande : netstat -nlt

Authentification du client par un certificat : (Authentification mutuelle)

Pour certaines applications sensibles, on préfère utiliser un système d'authentification forte par certificats, plutôt que des couples login/mot de passe. Le principe de l'authentification forte est un principe de vérification mutuelle, le client à la connexion va vérifier le certificat présenté par le serveur et inversement le serveur va vérifier que le certificat du client est valide et autorisé.

Il faut d'abord générer la paire de clé du client, faire une requête de certification et ensuite signer le certificat du client par la clé privée de l'autorité de confiance.

```
# Génération de la paire de clé du client
openssl genrsa  -des3 –out ClientPrivateKey.key 1024

# Création d'un certificate request pour le client
openssl  req  -new  -key ClientPrivateKey.key -out RequestClient.csr

# Signature du certificat client avec la CA
openssl  x509 -req -CA CertCa.crt  -CAkey CaPrivateKey.key  -in RequestClient.csr -out
ClientCertificat.pem  -days 365 –set_serial 123

# Export du certificat client (clef prive+publique) au format PKCS12
openssl  pkcs12  -export  -in ClientCertificat.pem  -inkey  ClientPrivate.key  -out ClientCertificat.p12
```

openssl va vous demander de taper un *name* et *password* pour protéger l'exportation.

Ouvrez le fichier */etc/apache2/sites-available/default-ssl*, et y ajouter le *path* vers le certificat du de l'autorité (CertCA.crt) :

Rajouter dans virtualHost

SSLVerifyClient require
SSLVerifyDepth 1
SSLCACertificateFile /etc/apache2/ssl/CertCA.crt

Import du certificat dans le navigateur sinon erreur

Preferences > Advanced > Encryption > View certificates > Your certificates

Vérification de la validité d'un certificat par OCSP

Le protocole OCSP permet de vérifier l'état de révocation actuel d'un certificat sans avoir recourt aux CRL. Un client OCSP émet une requête d'état de certificat (en HTTP) au serveur OCSP et attend la réponse du serveur. Une requête OCSP contient les données suivantes: la version du protocole, le service requis et l'identifiant du certificat cible. Une réponse est composée du nom du répondeur, une valeur de statut pour chacun des certificats de la requête et une signature effectuée par la clé privée du certificat OCSP. Il existe trois valeurs pour le statut des certificats: **good**, **revoked**, **unknown**.

- **Good** : signifie que le certificat n'est pas révoqué.
- **Revoked** : indique que le certificat a été révoqué (soit de façon permanente soit temporaire).
- **Unknown** : indique que le répondeur ne dispose pas d'informations concernant le certificat faisant l'objet de la requête.

Vérification par le client ocsp d'openssl

- Sauvegarder le certificat du serveur (taper la commande ci-dessous) dans un fichier
  - Openssl s_client –connect www.lcl.fr:443
- Afficher et sauvegarder la chaine de certificats avec l'option **showcerts**
  - Openssl s_client -connect www.lcl.fr:443 -showcerts
- Chercher l'url du serveur ocsp dans le certificat du serveur et mettre le dans la commande suivante :
  - openssl ocsp -issuer chain.pem -cert Certificat_lcl.crt -url http://COLLER_URL_ICI -CAfile chain.pem -no_nonce -text
- Vérifier le résultat

Partie 4 : Analyse par Wireshark

a) Lancez Wireshark et analysez les messages échangés (*ClientHello* et *ServerHello*) entre le client et le serveur.
b) Quel est le nombre de *suites* (propositions) proposé par le client
c) Forcez votre navigateur à utiliser une seule *suite* que vous choisissez de la liste envoyée par le client dans le *ClientHello*.
d) Forcez votre navigateur à utiliser une *suite* qui ne sera pas acceptée par le serveur. Analysez le résultat.

Annexe1 : Commandes utiles chiffrement, déchiffrement et signature

$ time openssl commande : pour visualiser le temps d'exécution d'une commande
$ diff file1.txt file2.txt –q : pour comparer le contenu de deux fichiers

$ openssl genrsa -out <fichier_rsa.priv> <size> : génére la clé privé RSA de taille size.

$ openssl rsa -in <fichier_rsa.priv> -des3 -out <fichier.pem> : chiffre la clef privé RSA avec l'algorithme DES3. Vous pouvez utiliser DES, 3DES, IDEA, etc.

$ openssl rsa -in <fichier_rsa.priv> -pubout -out <fichier_rsa.pub> : stocke la partie publique dans un fichier à part

$ openssl enc <-algo> -in <claire.txt> -out <chiffre.enc>: pour le chiffrement de claire.txt avec l'algorithme spécifié (openssl enc –help pour avoir la liste des possibilités ou bien openssl list-cipher-commands) dans un fichier chiffre.enc. On peut ajouter –k comme option et saisir la clé comme une passphrase

$ openssl enc <-algo> -in <chiffre> -d -out <claire> : pour le déchiffrement.

$ openssl dgst <-algo> -out <sortie> <entrée> : pour hacher un fichier. L'option <-algo> est le choix de l'algorithme de hachage (sha, sha1, dss1, md2 ,md4, md5, ripemd160).

$ openssl rand -out <clé.key> <nombre_octets> : pour générer un nombre aléatoire de taille nombre_octets (utiliser l'option -base 64 pour la lisibilité).

$ openssl aes-256-cbc -in <claire.txt> -out <chiffre.enc> -e -k <clé.key> : pour chiffrer un fichier avec l'AES.

$ openssl rsautl -encrypt `-pubin` -inkey <rsa.pub> -in <clair.txt> -out <chiffre.enc> : chiffrer fichier.txt avec la RSA en utilisant la clef publique rsa.pub

$ openssl rsautl -decrypt -inkey <rsa.priv> -in <chiffre.enc> -out <fihcier.txt> : pour déchiffrer le fichier fic.dec

$ openssl rsautl -sign -inkey <ras.priv> -in <fichier.txt> -out <fic.sig> : pour générer une signature.

$ openssl rsautl -verify -pubin -inkey <rsa.pub> -in fic fic.sig : pour vérifier une signature.

- *$openssl aes-256-cbc -in <fichier.txt> -out <fichier.enc> -e -k clé.key* : pour chiffrer un fichier avec l'algorithme AES utilisant une clé symétrique.

- *$openssl aes-256-cbc -in <fichier.txt> -out <fichier.enc> -d –k clé.key* : pour déchiffrer un fichier avec l'algorithme AES utilisant une clé symétrique.

Annexe2 : Extraire les informations d'un certificat?

```
# lots of information:
openssl x509 -text -in cert.pem

# issuer
openssl x509 -noout -in cert.pem -issuer

# to whom was it issued (subject)
openssl x509 -noout -in cert.pem -subject

# for what dates is it valid?
openssl x509 -noout -in cert.pem -dates

# what is the hash value of the certificate?
openssl x509 -noout -in cert.pem -hash

# what is the MD5 fingerprint?
openssl x509 -noout -in cert.pem -fingerprint
```

Annexe 3 :

bc est une calculatrice basique sous unix.
`bc` takes the following options from the command line:

```
-h, --help
      Print the usage and exit.
-l, --mathlib
      Define the standard math library.
-w, --warn
      Give warnings for extensions to POSIX bc.
-s, --standard
      Process exactly the POSIX bc language.
-q, --quiet
      Do not print the normal GNU bc welcome.
-v, --version
      Print the version number and copyright and quit.
```

convert from hexadecimal to decimal :
$ echo 'ibase=16;obase=A;FF' | bc
255

wc (en référence aux termes anglais word count, « décompte des mots ») est une commande Unix qui permet d'obtenir plusieurs informations au sujet de l'entrée standard ou d'une liste de fichiers : le nombre d'octets, le nombre de mots, le nombre de lignes (plus précisément le nombre de retour à la ligne).

Options

```
wc -l <nom_du_fichier> # affiche le nombre de lignes
wc -c <nom_du_fichier> # affiche le nombre d'octets
wc -m <nom_du_fichier> # affiche le nombre de caractères
wc -L <nom_du_fichier> # indique la longueur de la plus longue ligne
wc -w <nom_du_fichier> # affiche le nombre de mots
```

tr est une commande Unix qui permet de transposer ou d'éliminer des caractères dans un fichier ou un flux de données.

```
Exemples :

    Pour remplacer les minuscules par les majuscules correspondantes
    tr '[a-z]' '[A-Z]' < fichier

    Pour remplacer n blancs continus par un seul
    tr -s ' ' < fichier

    Pour remplacer le caractère ':' par un saut de ligne
    tr ':' '\012' < fichier
```

Annexe 4 :

Créer un certificat CSR en utilisant OpenSSL :

openssl req -newkey rsa:2048 -nodes -keyout myserver.key -out server.csr

- Country Name : Renseignez votre pays au format ISO 3166 (2 lettres).
  Exemple : "FR" pour la France
- State or Province Name : Renseignez le nom de votre département ou région.
  Exemple : Ile de France
- Locality Name : Renseignez le nom de votre ville.
  Exemple : PARIS
- Organization Name : Renseignez le nom de votre organisation.
  Exemple : google SARL
- Organizational Unit Name : Renseignez le nom du service en charge du certificat SSL.
  Exemple : Informatique
- Common Name : Renseignez l'adresse du site à sécuriser.
  Exemples : www.google.fr OU secure.google.fr
- Email Address : Eventuellement, renseignez votre adresse e-mail.
  Exemple : support@google.fr
- A challenge password : Laisser vide et valider en appuyant sur "ENTRER"
- An optional company name : Laisser vide et valider en appuyant sur "ENTRER"

req - PKCS#10 certificate request and certificate generating utility.

openssl req [**-inform PEM|DER**] [**-outform PEM|DER**] [**-in filename**] [**-passin arg**] [**-out filename**] [**-passout arg**] [**-text**] [**-pubkey**] [**-noout**] [**-verify**] [**-modulus**] [**-new**] [**-rand file(s)**] [**-newkey rsa:bits**] [**-newkey alg:file**] [**-nodes**] [**-key filename**] [**-keyform PEM|DER**] [**-keyout filename**] [**-keygen_engine id**] [**-[digest]**] [**-config filename**] [**-subj arg**] [**-multivalue-rdn**] [**-x509**] [**-days n**] [**-set_serial n**] [**-asn1-kludge**] [**-no-asn1-kludge**] [**-newhdr**] [**-extensions section**] [**-reqexts section**] [**-utf8**] [**-nameopt**] [**-reqopt**] [**-subject**] [**-subj arg**] [**-batch**] [**-verbose**] [**-engine id**]

-new : this option generates a new certificate request. It will prompt the user for the relevant field values. The actual fields prompted for and their maximum and minimum sizes are specified in the configuration file and any requested extensions. If the **-key** option is not used it will generate a new RSA private key using information specified in the configuration file.

-x509 : this option outputs a self signed certificate instead of a certificate request. This is typically used to generate a test certificate or a self signed root CA. The extensions added to

the certificate (if any) are specified in the configuration file. Unless specified using the **set_serial** option, a large random number will be used for the serial number.

-days n : when the **-x509** option is being used this specifies the number of days to certify the certificate for. The default is 30 days.

-key filename : This specifies the file to read the private key from. It also accepts PKCS#8 format private keys for PEM format files.

-in filename : This specifies the input filename to read a request from or standard input if this option is not specified. A request is only read if the creation options (**-new** and **-newkey**) are not specified.

-out filename : This specifies the output filename to write to or standard output by default.

x509 - Certificate display and signing utility

openssl x509 [**-inform DER|PEM|NET**] [**-outform DER|PEM|NET**] [**-keyform DER|PEM**] [**-CAform DER|PEM**] [**-CAkeyform DER|PEM**] [**-in filename**] [**-out filename**] [**-serial**] [**-hash**] [**-subject_hash**] [**-issuer_hash**] [**-ocspid**] [**-subject**] [**-issuer**] [**-nameopt option**] [**-email**] [**-ocsp_uri**] [**-startdate**] [**-enddate**] [**-purpose**] [**-dates**] [**-checkend num**] [**-modulus**] [**-pubkey**] [**-fingerprint**] [**-alias**] [**-noout**] [**-trustout**] [**-clrtrust**] [**-clrreject**] [**-addtrust arg**] [**-addreject arg**] [**-setalias arg**] [**-days arg**] [**-set_serial n**] [**-signkey filename**] [**-passin arg**] [**-x509toreq**] [**-req**] [**-CA filename**] [**-CAkey filename**] [**-CAcreateserial**] [**-CAserial filename**] [**-force_pubkey key**] [**-text**] [**-certopt option**] [**-C**] [**-md2|-md5|-sha1|-mdc2**] [**-clrext**] [**-extfile filename**] [**-extensions section**] [**-engine id**]

-req : by default a certificate is expected on input. With this option a certificate request is expected instead.

-in filename : This specifies the input filename to read a certificate from or standard input if this option is not specified.

-out filename : This specifies the output filename to write to or standard output by default.

-CA filename : specifies the CA certificate to be used for signing. When this option is present **x509** behaves like a ``mini CA". The input file is signed by this CA using this option: that is its issuer name is set to the subject name of the CA and it is digitally signed using the CAs private key. This option is normally combined with the **-req** option. Without the **-req** option the input is a certificate which must be self signed.

-CAkey filename : sets the CA private key to sign a certificate with. If this option is not specified then it is assumed that the CA private key is present in the CA certificate file.

-CAcreateserial : with this option the CA serial number file is created if it does not exist: it will contain the serial number ``02" and the certificate being signed will have the 1 as its serial number. Normally if the **-CA** option is specified and the serial number file does not exist it is an error.

-CAserial filename : sets the CA serial number file to use. When the **-CA** option is used to sign a certificate it uses a serial number specified in a file. This file consist of one line containing an even number of hex digits with the serial number to use. After each use the serial number is incremented and written out to the file again. The default filename consists of the CA certificate file base name with ``.srl" appended. For example if the CA certificate file is called ``mycacert.pem" it expects to find a serial number file called ``mycacert.srl".

Annexe 5 :

s_client

Ceci fournit un client SSL/TLS générique qui sait établir une connexion transparente avec un serveur distant parlant SSL/TLS. Étant seulement prévu pour des propos de test, il n'offre qu'une interface fonctionnelle rudimentaire tout en utilisant en interne la quasi-totalité des fonctionnalités de la librairie **ssl** d'OpenSSL.

La commande suivante permet de récupérer un certificat depuis un serveur SSL :

```
echo |\
openssl s_client -connect <host>:443 2>&1 |\
sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' >  CertificatGoogle.txt
```

```
openssl s_client -connect www.google.fr:443 > CertificatGoogle.txt
```