OpenSSL, certificats, CA

Crypto à clé secrète

Crypter un fichier :

```
openssl enc -aes-256-cbc -in fichier_entrée -out fichier_sortie
```

Remplacer -aes-256-cbc par l'algorithme de cryptage voulu. Le programme demande le mot de passe, pour le préciser, utiliser `-pass pass:XXXX` où XXXX est le mot de passe

Decrypter un fichier

```
openssl enc -d -aes-256-cbc -in fichier_entrée -out fichier_sortie
```

RSA

Pour générer une paire clé privée/clé publique

```
openssl genrsa -out fichier.key 1024
```

Remplacer 1024 par la taille de clé désirée. Le fichier contiendra les clés publiques et privées

Pour extraire la clé privée

```
openssl rsa -in  fichier.key -pubout -out fichier-pub.key
```

Pour afficher les détails de la clé

```
openssl rsa -in  fichier.key -text -noout
```

pour crypter

```
openssl pkeyutl -encrypt -in fichier_entrée -inkey fichier.key -out fichier_sortie
openssl pkeyutl -encrypt -in fichier_entrée -pubin -inkey fichier-pub.key -out fichier_sortie
```

Attention, taille fichier à crypter TRES limitée (une centaine d'octets)

Pour décrypter

```
openssl pkeyutl -decrypt -in fichier_entrée -inkey fichier.key -out fichier_sortie
```

Pour mettre un mot de passe sur la clé

```
openssl rsa -in fichier.key -des3 -out fichier-enc.key
```

On peut utiliser un autre algorithme que des3. Si la clé est cryptée, le mot de passe sera demandé pour chaque utilisation de la partie privée. Pour retirer le mot de passe, utiliser la même commande sans le `-des3`

Utilitaires

Tester si un nombre est premier

```
openssl prime val
```

Générer un nombre premier

```
openssl prime -generate -bits 512
```

Remplacer 512 par la taille. Attention, une taille trop longue mettra très longtemps à être générée

Générer un nombre aléatoire

```
openssl rand 64
```

remplacer 64 par la taille du nombre généré. On peut ajouter les options -hex ou -base64 pour changer le format de sortie

Digest et signature

Pour calculer un digest

```
openssl dgst -sha256 -out  fichier_sortie fichier_entrée
```

On peut remplacer le `-sha256` par un autre algorithme

Pour signer avec une clé privée

```
openssl pkeyutl -sign -in fichier_entrée -out fichier_signature -inkey fichier.key
```

Pour vérifier la signature

```
openssl pkeyutl -verify -in fichier_entrée -sigfile fichier_signature -pubin -inkey fichier-pub.key
```

Certificat (base)

Créer une demande de certificat à partir d'une clé RSA

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

Créer une clé RSA et une demande de certificat

```
openssl req -newkey rsa:2048 -noenc -keyout fichier.key -out fichier.csr
```

on peut remplacer `2048` par une autre taille de clé, `-noenc` permet de ne pas crypter la clé mais pourrait avoir à être remplacé par -nodes sur les anciennes versions de openssl.

Créer à partir d'un ancien certificats

```
openssl x509 -in fichier.crt -signkey fichier.key -x509toreq -out fichier.csr
```

Pour créer un certificat "auto-signé", ajouter

```
-x509 -days 365
```

et remplacer le fichier de sortie en `fichier.crt`. Le nombre de jour indique la validité du certificat.

Pour afficher le contenu d'une demande de certificat, utiliser

```
openssl req -text -noout -in fichier.csr
```

Pour afficher le contenu d'un certificat utiliser

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

Certificat (avancé)

Lors de la génération d'une demande de certificat, on peut passer les informations (pays, état, localité, …) sur la ligne de commande à l'aide de

```
-subj "/C=pays/ST=etat/L=localité/O=organisation/CN=name"
```

OpenSSL peut utiliser un fichier de configuration. Ce fichier permettra de définir une série de valeurs (par exemple les "extensions x509" utilisées).

Ce fichier sera composés de plusieurs sections et aura la forme suivante :

```
[section1]
 param = valeur
 param = valeur
 …
 [section2]
 …
```

On peut préciser le fichier de configuration à utiliser à l'aide du paramètre

```
-config fichier.cnf
```

Selon l'opération effectuée, openssl cherchera une section différente.

Création d'une demande de certificat (fichier CSR) à l'aide de req	[req]
Création d'un certificat auto-signé (req -x509)	[req]
Signature de certificats par un CA	[ca]
Si la section nécessaire n'est pas trouvée (absente)	[default]

Cette section pourra contenir plusieurs paramètres. Le paramètre `distinguished_name` aura une signification différente selon que l'on est dans un [ca] ou dans un [req]. De plus, dans un [req], le paramètre `prompt=no` changera également son comportement.

Pour les requêtes, distinguished_name pointera sur une section qui contiendra

Nom	`prompt=no`	Pas de prompt=no
field	La valeur du champ C=BE	"prompt" à afficher pour demander la valeur du champ en question countryName=Pays (2 lettres)
field_default		Valeur par défaut pour le champ en question countryName_default=BE
field_min		Taille minimale du champ en question countryName_min=2
field_max		Taille maximale du champ en question country_name_max=2

A noter que pour le cas prompt=no, on utilisera généralement la forme courte du champ.

Certains champs peuvent apparaître plusieurs fois (organizationName par exemple). Dans ce cas, on les fera précéder d'un nombre suivi d'un point (0.organizationName, 1.organizationName,…) Ce texte sera ignoré dans le certificat.

Les valeurs des champs ainsi que leurs abbréviations sont compilées dans OpenSSL (ne peuvent être changées) et sont :
countryName(C), stateOrProvinceName(ST), localityName(L), organizationName(O), organizationalUnitName(OU), commonName(CN), emailAddress, name, surname, givenName, initials.

Le champ CN contiendra l'identité du certificat (adresse DNS, E-Mail, adresse IP, …)

Le paramètre

```
req_extensions = XXX
```

fera référence à une section où des paramètres demandés seront précisés pour la génération d'une demande de certificat.

```
x509_extensions = XXX
```

fera référence à une section où des paramètres seront précisés pour la signature lorsque l'on génère un certificat autosigné (-x509 dans un req)

Les extensions peuvent être soit une série paramètre:valeur séparés par des virgules, soit @XXX où XXX est un nom de section contenant les paramètre=valeurs à raison de une par ligne.

Si un paramètre a plusieurs valeurs ou si une des valeurs doit contenir une virgule, la forme indirecte (@XXX) doit être utilisée.

Pour les paramètres a valeur multiple, on fait suivre le nom du paramètre par .1, .2, …
ex : DNS.1= … DNS.2=… etc...

basicConstraints	CA:FALSE ou CA :TRUE	Est-ce que le certificat peut être utilisé comme CA ?
keyUsage	digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment, cRLSign, encipherOnly, decipherOnly, keyAgreement, keyCertSign	Utilisations possibles de la clé (mode d'utilisation) séparées par des virgules
ExtendedKeyUsage	serverAuth, clientAuth, codeSigning, emailProtection, timeStamping, OCSPSigning, ipsecIKE, ...	Précisions pour l'utilisation de la clé (contexte d'utilisation)
SubjectAltName	email:… ou URI:http://.. ou IP:1.2.3.4 ou DNS:...	Précise noms alternatifs séparés par des virgules
SubjectKeyIdentifier	hash	A mettre tel quel

A noter que si les noms supplémentaires apparaissent dans la ligne SubjectAltName, on utilise un signe : au lieu d'un signe =

Créer un CA

1. Générer une clé RSA pour le CA (ca.key)
   ```
   openssl genrsa -out ca.key 2048
   ```
2. faire un fichier de configuration pour requête ([req]) avec l'option distinguished_name qui pointe vers une section fournissant les intitulés des questions pour le DN
   ```
   countryName             = Pays (2 lettres)
   stateOrProvinceName     = Province
   localityName            = Ville
   0.organizationName      = Organisation
   organizationalUnitName  = Section
   commonName              = Common Name
   name                    = Nom
   emailAddress            = Email
   ```
   et l'option req_extensions qui pointe sur une section qui contient
   ```
   subjectKeyIdentifier = hash
   authorityKeyIdentifier = keyid:always,issuer:always
   basicConstraints = CA:TRUE
   keyUsage =  cRLSign, keyCertSign
   ```
3. générer le certificat autosigné
   ```
   openssl req -new -x509 -days 3650 -key ca.key -out ca.crt
   -config req.cnf
   ```
4. Créer la structure suivante :
   ```
   CAdir/
        index.txt (fichier vide)
        newcerts/
        ca.crt
        serial (fichier contenant un numéro de série, par ex 01)
        private/
             ca.key
        ca.cnf (un nouveau fichier de configuration)
   ```
5. créer le fichier de configuration pour signer comme décrit ci-dessous
6. signer à l'aide de
   ```
   openssl ca -in cert.csr -out cert.crt -config  ca.cnf
   ```

On peut préciser une section d'options de certificats différente de celle signalée dans le fichier de configuration (par x509_extensions) en utilisant l'option `-extension nom_section` pour préciser une section différente.

De même, l'option -subj permet de modifier tout ou partie de la description (C, ST, L, …) de la clé par rapport à la requête.

Pour générer un crl on utilisera
```
openssl ca -gencrl -out crl.pem
```

Le fichier de configuration contiendra au minimum ce qui suit

```
[ca]
default_ca = CA_def

[CA_def]
dir=./CAdir
database = $dir/index.txt
new_certs_dir = $dir/newcerts
certificate =  $dir/ca.crt
serial =  $dir/serial
private_key = $dir/private/ca.key
crl_dir = $dir
crl = $dir/crl.pem

default_days = 365
default_crl_days = 30
default_md = sha1
policy = policy_sign
x509_extensions = usr_ext
copy_extensions = copy
name_opt = ca_default
cert_opt = ca_default

[policy_sign]
countryName = supplied
stateOrProvinceName = optional
localityName = optional
organizationName = optional
organizationalUnitName = optional
commonName = supplied
name = optional
emailAddress = optional

[usr_ext]
basicConstraints = CA:FALSE
keyUsage =  nonRepudiation, digitalSignature, keyEncipherment
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid, issuer:always
extendedKeyUsage  = clientAuth
#crlDistributionPoints = URI:http://myhost.com/myca.crl
```

Les certificats signés seront repris dans le index.txt. Les certificats (texte descriptif + certificat) seront sauvés dans newcerts

Les options précisées dans [usr_ext] peuvent évidemment être adaptées aux besoins.

On peut concaténer le certificat généré et le certificat du CA pour obtenir un "bundle".

Il est maintenant conseillé utiliser les propriétés name_opt et cert_opt pour forcer un affichage plus détaillé du certificat avant signature. Si elles sont absentes, l'ancien format d'affichage est utilisé par défaut.

Pour la policy, on peut mettre match (même valeur que celle du CA), optional (valeur transmise si présente) ou supplied (valeur transmise obligatoire)

Conversions

Les certificats peuvent être dans plusieurs formats

PEM	Format standard pour le X509
DER	Format x509 utilisé entre autres par Java
PKCS7	Utilisé par Java et IIS
PKCS12	Utilisé par IIS et pour les E-Mail

Pour convertir de PEM vers DER on utilisera

```
openssl x509 -in fichier.pem -outform der -out fichier.der
```

Pour convertir de DER vers PEM on utilisera

```
openssl x509 -inform der -in fichier.der -out fichier.pem
```

Pour convertir vers PKCS7 (Windows, Tomcat,...) on utilisera

```
openssl crl2pkcs7 -nocrl -certfile fichier.crt -certfile ca.crt […]
      -out fichier.p7b
```

Pour convertir vers PKCS12 (Mail, Windows,...) on utilisera

```
openssl pkcs12 -inkey fichier.key -in fichier.crt -export -out fichier.p12
```