TP OpenSSL/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, sécurité pour la couche de transport) ainsi que les standards cryptographiques liés dont ils ont besoin.

Le programme **openssl** est un outil en ligne de commande qui permet d'utiliser les différentes fonctions cryptographiques de la bibliothèque **crypto** d'OpenSSL à partir du shell. Il peut être utilisé pour

- La création de paramètres des clefs RSA, DH et DSA
- La création de certificats X.509, CSRs et CRLs
- Le calcul de condensés de messages
- Le chiffrement et le déchiffrement
- Le test de clients et serveurs SSL/TLS
- La gestion de courriers S/MIME signés ou chiffrés

Installer OpenSSl sur un poste windows

Pour effectuer certaines opérations de cryptographie (création d'une clef privée, génération d'un CSR, conversion d'un certificat...) sur un poste windows nous pouvons utiliser l'outil OpenSSL.

- Accéder au site officiel : http://www.openssl.org/
  Puis télécharger le programme "binaire" pour Windows : > related > Binaries : http://www.openssl.org/related/binaries.html
- https://slproweb.com/products/Win32OpenSSL.html
-

Utiliser OpenSSL sur un poste Windows

L'installation standard d'OpenSSL sur un poste Windows est effectuée sur "**C:\OpenSSL-Win32**" et l'exécutable est situé dans le sous répertoire "**bin**". Ainsi pour exécuter le programme dans "l'invite de commandes" Windows il vous faudra donner le chemin complet : **>C:\OpenSSL-Win32\bin\openssl ...**

- La version 1.0 d'openSSL en téléchargement nécessite la présence d'un fichier de configuration "openssl.cnf".

  Le fichier openssl.cnf présent dans le dossier C:\Program Files\OpenSSL\apps doit être copié dans le dossier c:\OpenSSL-Win32\bin\openssl.cnf

Le travail sur les certificats nécessitera le paramètrage du fichier openssl.cnf.
Ce fichier adapté à vos besoins devra être pointé par la variable d'environnement

OPENSSL_CONF.

```
set OPENSSL_CONF=c:\OpenSSL-Win32\bin\openssl.cnf
```

- **PEM**, Privacy Enhanced Mail (courrier personnel amélioré): c'est un format ASCII, codé en Base64, le fichier est délimité par <BEGIN CERTIFICAT >…. <END CERTIFICAT>

- **PKSC#7**, Public-Key Cryptography Standards: est un standard de format de message cryptographique(utilisé pour les certificat, et les requêtes …)

- **PKSC#10:** le format d'une requête (demande) de certificat. Il impose un mot de passe(challenge password) utilisé pour authentifier le détenteur en cas de révocation

- **PKSC#12**: est le standard utilisé pour les certificats utilisateur (ce fichier contient en plus du certificat utilisateur, sa clé privée protégée par mot de passe)

openssl *command* [*command_opts*] [*command_args*]

openssl [**list-standard-commands** | **list-message-digest-commands** | **list-cipher-commands** | **list-cipher-algorithms** | **list-message-digest-algorithms** | **list-public-key-algorithms**]

openssl no-*XXX* [*arbitrary options*]

COMMANDES STANDARDS

asn1parse Traitement d'une séquence ASN.1.

ca Gestion des Autorité de Certification (CA)

ciphers Détermination de la description des suites de chiffrements.

crl Gestion des listes de certificats révoqués (CRL)

crl2pkcs7 Conversion CRL vers PKCS#7.

dgst Calcul de condensés de messages.

dh Gestion des paramètres Diffie-Hellman. Rendu obsolète par **dhparam**.

dsa Gestion de données DSA.

dsaparam Production de paramètres DSA.

enc Chiffrement.

errstr Conversion numéro d'erreur vers descriptif textuel.

dhparam Production et gestion de paramètres Diffie-Hellman.

gendh Production de paramètres Diffie-Hellman. Rendu obsolète par **dhparam**.

gendsa Production de paramètres DSA.

genrsa Production de paramètres RSA.

ocsp Utilitaire pour le protocole de vérification en ligne de certificats.

passwd Production de mots de passe hashés.

pkcs12 Gestion de données PKCS#12.

pkcs7 Gestion de données PKCS#7.

rand Production d'octets pseudo-aléatoires.

req Gestion des demande de chiffrement de certificats X.509 (CSR).

rsa Gestion de données RSA.

rsautl Utilitaire RSA pour signer, vérifier, chiffrer, et déchiffrer.

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

s_server Ceci fournit un serveur SSL/TLS générique qui accepte des connexions provenant de clients qui parlent SSL/TLS. Étant seulement prévu pour du test, il n'offre qu'une interface fonctionnelle rudimentaire tout en utilisant en interne la quasi-totalité des fonctionnalités de la bibliothèque **ssl** d'OpenSSL. Il fournit à la fois son propre protocole orienté commandes en ligne pour le test de fonctions SSL et la possibilité de répondre simplement à des demandes HTTP pour muler un serveur internet qui gère SSL/TLS.

s_time Minuterie de connexion SSL

sess_id Gestion des données de session SSL.

smime Traitement de courriers S/MIME.

speed Mesure la vitesse de l'algorithme.

verify Vérification de certificats X.509.

version Information sur la version d'OpenSSL.

x509 Gestion de données pour les certificats X.509.

voir la documentation de openssl sur le lien suivant:

https://www.openssl.org/docs/man1.1.1/man1/

1. Création d'une autorité racine (*Certificat Autority*)

Cette autorité personnelle devra être diffusée auprès des futurs utilisateurs des certificats pour éviter les messages d'alerte concernant une autorité de certification inconnue lors de l'utilisation des certificats générés.

1.1. Création de la clef privée de la CA

openssl genrsa -aes256 -out CA_pvk.pem 1024

Loading 'screen' into random state - done
Generating RSA private key, 1024 bit long modulus
.....++++++
...++++++
e is 65537 (0x10001)
Enter pass phrase for CA_pvk.pem:*mot de passe*↵
Verifying - Enter pass phrase for CA_pvk.pem:*mot de passe*↵

1.2. Autogénération du certificat de la nouvelle autorité racine

openssl req -new -x509 -days 365 -key CA_pvk.pem -out CA_crt.pem

req -new -newkey rsa:2048 -nodes -out www.monsite.fr.csr -keyout www.monsite.fr.key -subj "/C=FR/ST=Calvados/L=CAEN/O=Mon organisation/CN=www.mon-site.fr"

Enter pass phrase for CA_pvk.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]: (le plus souvent MR)
State or Province Name (full name) [Some-State]: (le nom de votre pays en toutes lettres)
Locality Name (eg, city) []:(le nom de votre ville)
Organization Name (eg, company) [Internet Widgits Pty Ltd]: (le nom de votre organisation)
Organizational Unit Name (eg, section) []:(laisser vide -recommandé - ou entrer un terme générique comme "Service Informatique")
Common Name (eg, YOUR name) []:(le nom du site a sécuriser)
Email Address []:(ne rien mettre, laisser vide)

- CA_pvk.pem est le fichier contenant la clef privée, il sera nécessaire à la signature des

certificats des clients.

- CA_crt.pem est le fichier contenant le certificat, il doit être fournit aux clients avec leurs certificats.

Ces deux fichiers doivent être renseignés dans les variables ***certificate*** et ***private_key*** du fichier de configuration openssl.cnf.

Pour voir le certificat en format texte, appliquer la commande suivante:

Openssl x509 –in CA_crt.pem -text

Transformation au format Microsoft

```
openssl x509 -in CA_crt.pem -outform DER -out CA_crt.der
```

Il s'agit d'une simple conversion d'un format ascii Base64 en son équivalent binaire

2. Création d'un certificat client

2.1. du Certificat Signing Request (CSR) client

on commence par créer une clef privée

```
openssl genrsa -aes256 -out client_pvk.pem 1024

Loading 'screen' into random state - done
Generating RSA private key, 1024 bit long modulus
....++++++
.....................................................................++++++
e is 65537 (0x10001)
Enter pass phrase for client_pvk.pem:↲
Verifying - Enter pass phrase for client_pvk.pem:↲
```

puis à partir de cette clef privée nous générons la CSR

```
req -new -key client_pvk.pem -out client_csr.csr
```

```
OpenSSL> req -new -key client_pvk.pem -out client_csr.csr
```

Enter pass phrase for client_pvk.pem:

You are about to be asked to enter information that will be incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:MR

State or Province Name (full name) [Some-State]:rabat

Locality Name (eg, city) []:Rabat

Organization Name (eg, company) [Internet Widgits Pty Ltd]:Berbiche

Organizational Unit Name (eg, section) []:www.berbiche.com

Common Name (e.g. server FQDN or YOUR name) []:www.Berbiche.com

Email Address []:

Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []:

An optional company name []:

OpenSSL>

openssl req -new -key client_pvk.pem -out client_csr.csr

Enter pass phrase for client_pvk.pem:↲
You are about to be asked to enter information that will be incorporated
into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:↲
State or Province Name (full name) [Some-State]:↲
Locality Name (eg, city) []:↲
Organization Name (eg, company) [Internet Widgits Pty Ltd]:↲
Organizational Unit Name (eg, section) []:↲
Common Name (eg, YOUR name) []:↲
Email Address []:↲

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:↲
An optional company name []:↲

Contenu de CSR

Si vous souhaitez contrôler les données que vous avez entrées dans votre CSR, cela ce fait avec la commande suivante:

```
req -in client_csr.csr -noout -text
```

OpenSSL> req -in client_csr.csr -noout -text

Certificate Request:

Data:

Version: 1 (0x0)

Subject: C = MR, ST = rabat, L = Rabat, O = Berbiche, OU = www.berbiche.com, CN = www.Berbiche.com

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

Public-Key: (1024 bit)

Modulus:

00:b5:19:bb:aa:f1:d5:d9:3f:ec:d7:f7:92:85:8f:

78:b7:12:5a:64:14:1a:3e:a1:b6:e5:f7:6f:e4:7b:

```
            15:2f:b3:f5:12:1d:1f:e1:f0:b5:32:88:0b:81:15:
            8d:d6:e4:4c:6b:95:98:46:d6:7d:a2:ea:52:8a:ce:
            3c:10:51:e8:ac:44:69:89:a7:19:ee:ee:53:9f:77:
            d3:f0:58:e1:3f:6a:10:9a:26:c1:0b:fd:08:47:0c:
            6b:c6:df:3e:88:f6:73:be:8f:da:53:7b:c2:2d:d9:
            c7:ea:b6:3d:01:7c:9d:5f:39:9e:a9:0c:bd:6a:7a:
            7b:c5:04:25:c8:95:b8:dd:41
        Exponent: 65537 (0x10001)
    Attributes:
        a0:00
Signature Algorithm: sha256WithRSAEncryption
    a3:f0:65:a8:99:b7:ad:65:bd:57:bd:85:66:18:13:10:4d:6b:
    45:44:f2:aa:ea:dd:f0:4f:bc:87:a2:5f:48:22:00:25:d7:9b:
    59:15:ef:8c:85:2d:99:87:94:7c:56:9b:a0:ed:23:94:ba:78:
    b5:46:cf:7f:66:7b:e8:49:c1:a6:7d:11:01:19:05:09:45:7e:
    85:63:8c:52:5e:9d:22:72:9a:a3:ab:0f:1a:d6:52:4c:59:f0:
    ff:7a:13:52:47:ad:52:0c:74:18:1f:32:02:0f:eb:64:d7:67:
    27:80:d8:fd:26:b4:d3:d3:20:e4:20:10:5d:ff:76:27:4c:d0:
    99:7f
```

2.2. Signature de la requête CSR

Maintenant, nous agissons en temps qu'autorité de certification pour valider les informations contenues dans la CSR

```
openssl ca -out client_crt.pem -in client_csr.csr -cert CA_crt.pem -keyfile CA_pvk.pem
x509 -req -days 730 -in ia.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out ia.crt
x509 -req -days 730 -in client_csr.pem -CA CA_crt.pem -CAkey CA_pvk.pem -
```

set_serial 01 -out client_crt.pem

Using configuration from D:\dev\openssl-0.9.8e\apps\openssl.cnf
Loading 'screen' into random state - done
Enter pass phrase for CA_pvk.pem:↲
Check that the request matches the signature
Signature ok
Certificate Details:
Serial Number: 286 (0x11e)

Validity
Not Before: Aug 26 14:56:09 2007 GMT
Not After : Aug 25 14:56:09 2008 GMT
Subject:
countryName = FR
stateOrProvinceName = Some-State
organizationName = Internet Widgits Pty Ltd
commonName = TEST
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
X509v3 Subject Key Identifier:
26:A6:C6:16:CF:2A:5B:D5:9D:FF:2C:48:60:B6:E0:E3:A6:3B:28:88
X509v3 Authority Key Identifier:
keyid:68:0D:3F:8A:D3:D7:11:35:EA:AC:C3:28:6C:78:92:04:36:EA:A3:2F

Certificate is to be certified until Aug 25 14:56:09 2008 GMT (365 days)
Sign the certificate? [y/n]:y↲

1 out of 1 certificate requests certified, commit? [y/n]y↲
Write out database with 1 new entries
Data Base Updated

Si vous obtenez ce message :
failed to update database
TXT_DB error number 2
error in ca
Le numéro de série du certificat est déjà utilisé, vous devrez révoquer le certificat l'utilisant.

unable to load certificate
260:error:0906D06C:PEM routines:PEM_read_bio:no start
line:.\crypto\pem\pem_lib.c:642:Expecting: TRUSTED CERTIFICATE
error in ca
La CA passée en paramètre n'est pas au format PEM.

Le certificat est créé.

3. Convertion de la clef privée au format pour IIS

```
openssl rsa –inform pem –outform net –in client_pvk.pem -out client_pvk.net
```

```
rsa –inform pem –outform DER –in client_pvk.pem –out client_pvk.der

Enter pass phrase for client_pvk.pem:
writing RSA key
Enter Private Key password:
Verifying - Enter Private Key password:
```

La transformation au format binaire de la clef privée est nécessaire pour l'insérer dans le gestionnaire de clefs de IIS.
Il reste à supprimer l'entête du certificat client (client_crt.pem) et à importer la clef privée et le certificat dans le gestionnaire de clef de IIS.

4. Convertion du certificat pour importation sous IIS6

```
openssl pkcs12 -export -in client_crt.pem -inkey client_pvk.pem -out mycert.pfx -name "Mon certificat SSL personnel"
```

```
Loading 'screen' into random state - done
Enter pass phrase for client_pvk.pem:
Enter Export Password:
Verifying - Enter Export Password:
```

Il est à noter pour les utilisateurs sous Windows que le Platform SDK Redistributable: CAPICOM contient un script (CStore.vbs) permettant d'automatiser les actions sur les certificats stockés dans le gestionnaire de clefs de Windows.

5. Commandes utiles

Révocation d'un certificat

```
openssl ca -revoke client_crt.pem
```

Retirer la passphrase de la clef privée

```
openssl rsa -in client_pvk.pem -out client_unpvk.pem
```

Changer la passphrase de la clef privée

```
openssl rsa -der -in client_pvk.pem -out client_unpvk.pem
Enter pass phrase for client_pvk.pem:
writing RSA key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

Extraction des informations d'un certificate

Il faut que le certificate google.crt soit present dans le dossier courant

```
openssl x509 -text -noout -in google.crt
Certificate:

   Data:
      Version: 3 (0x2)
      Serial Number:
         4b:a5:ae:59:de:dd:1c:c7:80:7c:89:22:91:f0:e2:43
      Signature Algorithm: md5WithRSAEncryption
      Issuer: C=ZA, O=Thawte Consulting (Pty) Ltd., CN=Thawte SGC CA
      Validity
         Not Before: May 15 23:18:11 2006 GMT
         Not After : May 15 23:18:11 2007 GMT
      Subject: C=US, ST=California, L=Mountain View, O=Google Inc,
CN=www.google.com
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
         RSA Public Key: (1024 bit)
            Modulus (1024 bit):
               00:e6:c5:c6:8d:cd:0b:a3:03:04:dc:ae:cc:c9:46:
               be:bd:cc:9d:bc:73:34:48:fe:d3:75:64:d0:c9:c9:
               76:27:72:0f:a9:96:1a:3b:81:f3:14:f6:ae:90:56:
               e7:19:d2:73:68:a7:85:a4:ae:ca:24:14:30:00:ba:
               e8:36:5d:81:73:3a:71:05:8f:b1:af:11:87:da:5c:
               f1:3e:bf:53:51:84:6f:44:0e:b7:e8:26:d7:2f:b2:
               6f:f2:f2:5d:df:a7:cf:8c:a5:e9:1e:6f:30:48:94:
               21:0b:01:ad:ba:0e:71:01:0d:10:ef:bf:ee:2c:d3:
               8d:fe:54:a8:fe:d3:97:8f:cb
            Exponent: 65537 (0x10001)
      X509v3 extensions:
         X509v3 Extended Key Usage:
            TLS Web Server Authentication, TLS Web Client Authentication, Netscape Server
Gated Crypto
         X509v3 CRL Distribution Points:
            URI:http://crl.thawte.com/ThawteSGCCA.crl
```

```
        Authority Information Access:
            OCSP - URI:http://ocsp.thawte.com
            CA Issuers - URI:http://www.thawte.com/repository/Thawte_SGC_CA.crt

        X509v3 Basic Constraints: critical
            CA:FALSE
    Signature Algorithm: md5WithRSAEncryption
        57:4b:bc:a4:43:e7:e0:01:92:a0:96:35:f9:18:08:88:1d:7b:
        70:19:8f:f9:36:b2:05:3a:05:ca:14:59:4d:24:0e:e5:8a:af:
        4e:87:5a:f7:1c:2a:96:8f:cb:61:40:9e:d2:b4:38:40:21:24:
        c1:4f:1f:cb:13:4a:8f:95:02:df:91:3d:d6:40:eb:11:6f:9b:
        10:a1:6f:ce:91:5e:30:f6:6d:13:5e:15:a4:2e:c2:18:9e:00:
        c3:d8:32:67:47:fc:b8:1e:9a:d9:9a:8e:cc:ff:7c:12:b7:03:
        bf:52:20:cf:21:f4:f3:77:dd:12:15:f0:94:fa:90:d5:e3:59:
        68:81
```

Identité du propriétaire

```
openssl x509 -subject -noout -in google.crt


subject= /C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
```

Numéro de série du certificat

Nécessaire à la CRL (*'certification revocation list*)

```
openssl x509 -serial -noout -in google.crt


serial=4BA5AE59DEDD1CC7807C892291F0E243
```

Identification de l'émetteur du certificat

```
openssl x509 -issuer -noout -in google.crt
issuer= /C=ZA/O=Thawte Consulting (Pty) Ltd./CN=Thawte SGC CA
```

Date de début de validité du certificat

```
openssl x509 -startdate -noout -in google.crt
notBefore=May 15 23:18:11 2006 GMT
```

La période de validité

```
openssl x509 -noout -in google.crt -dates
notBefore=Apr  9 15:28:28 2013 GMT
notAfter=Apr  9 15:28:28 2014 GMT
```

La valeur de hachage ?

```
openssl x509 -noout -in google.crt -hash
bf163efd
```

L'empreinte MD5 ?

```
openssl x509 -noout -in google.crt -fingerprint
SHA1 Fingerprint=C1:CD:DD:29:D1:8D:23:63:6D:3F:71:AD:7E:29:DE:26:FF:D4:11:17
```

Validation de la CA émettrice du certificat

```
openssl verify -CAfile CA_crt.pem client_crt.pem
client_crt.pem: OK
```

Validation du rôle du certificat

```
openssl verify -CAfile CA_crt.pem -purpose sslclient client_crt.pem
client_crt.pem: OK
```

Le certificat est-il autosigné ?

```
openssl verify -CAfile google.crt google.crt
google.crt: /C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
error 20 at 0 depth lookup:unable to get local issuer certificate

openssl verify -CAfile "Verisignroot.cer" "Verisign root.cer"
Verisign root.cer: OK
```

6. Divers

Connexion manuelle à un serveur HTTPS

Si pour se connecter à une serveur HTTP, une connexion telnet sur le port 80 suffit, se connecter en ligne de commande à un serveur HTTPS reste possible grâce à OpenSSL. Ci-dessous, un exemple de GET aveugle sur https://www.google.fr :

```
openssl s_client -connect www.google.fr:443


Loading 'screen' into random state - done
CONNECTED(00000788)
depth=1 /C=ZA/O=Thawte Consulting (Pty) Ltd./CN=Thawte SGC CA
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
```

```
 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
   i:/C=ZA/O=Thawte Consulting (Pty) Ltd./CN=Thawte SGC CA
 1 s:/C=ZA/O=Thawte Consulting (Pty) Ltd./CN=Thawte SGC CA
   i:/C=US/O=VeriSign, Inc./OU=Class 3 Public Primary Certification Authority

---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDITCCAoqgAwIBAgIQS6WuWd7dHMeAfIkikfDiQzANBgkqhkiG9w0BAQQFADBM
MQswCQYDVQQGEwJaQTElMCMGA1UEChMcVGhhd3RlIENvbnN1bHRpbmcgKFB0eSkg
THRkLjEWMBQGA1UEAxMNVGhhd3RlIFNHQyBDQTAeFw0wNjA1MTUyMzE4MTFaFw0w
NzA1MTUyMzE4MTFaMGgxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlh
MRYwFAYDVQQHEw1Nb3VudGFpbiBWaWV3MRMwEQYDVQQKEwpHb29nbGUgSW5jMRcw
FQYDVQQDEw53d3cuZ29vZ2xlLmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkC
gYEA5sXGjc0LowME3K7MyUa+vcydvHM0SP7TdWTQycl2J3IPqZYaO4HzFPaukFbn
GdJzaKeFpK7KJBQwALroNl2BczpxBY+xrxGH2lzxPr9TUYRvRA636CbXL7Jv8vJd
36fPjKXpHm8wSJQhCwGtug5xAQ0Q77/uLNON/lSo/tOXj8sCAwEAAaOB5zCB5DAo
BgNVHSUEITAfBggrBgEFBQcDAQYIKwYBBQUHAwIGCWCGSAGG+EIEATA2BgNVHR8E
LzAtMCugKaAnhiVodHRwOi8vY3JsLnRoYXd0ZS5jb20vVGhhd3RlU0dDQ0EuY3Js
MHIGCCsGAQUFBwEBBGYwZDAiBggrBgEFBQcwAYYWaHR0cDovL29jc3AudGhhd3Rl
LmNvbTA+BggrBgEFBQcwAoYyaHR0cDovL3d3dy50aGF3dGUuY29tL3JlcG9zaXRv
cnkvVGhhd3RlX1NHQ19DQS5jcnQwDAYDVR0TAQH/BAIwADANBgkqhkiG9w0BAQQF
AAOBgQBXS7ykQ+fgAZKgljX5GAiIHXtwGY/5NrIFOgXKFFlNJA7liq9Oh1r3HCqW
j8thQJ7StDhAISTBTx/LE0qPlQLfkT3WQOsRb5sQoW/OkV4w9m0TXhWkLsIYngDD
2DJnR/y4HprZmo7M/3wStwO/UiDPIfTzd90SFfCU+pDV41logQ==
-----END CERTIFICATE-----
subject=/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
issuer=/C=ZA/O=Thawte Consulting (Pty) Ltd./CN=Thawte SGC CA
---
No client certificate CA names sent
---
SSL handshake has read 1777 bytes and written 322 bytes
---
New, TLSv1/SSLv3, Cipher is AES256-SHA
Server public key is 1024 bit
Compression: NONE
Expansion: NONE
SSL-Session:

    Protocol  : TLSv1
    Cipher    : AES256-SHA
    Session-ID:
```

```
D596685E782544409A2AAA0209CB1DC93C9136FA7BD704E9F6E2CF9034F5ED3A
    Session-ID-ctx:
    Master-Key:
DD3B0CDF8BCC32ECFB9AF7B97AB46CCF2D731E929B0048444D62EA45486774E74F
23FF950F3BBBE0992AFED425413FBA
    Key-Arg   : None
    Start Time: 1169126487
    Timeout   : 300 (sec)
    Verify return code: 20 (unable to get local issuer certificate)

---
GET / HTTP/1.0

HTTP/1.0 302 Found
Location: http://www.google.com
Date: Thu, 18 Jan 2007 13:21:21 GMT
Content-Type: text/html
Server: GFE/1.3
Connection: Close
Content-Length: 218

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.com">here</A>.
</BODY></HTML>
read:errno=0
```

On remarquera que le CN du certificat retourné par le serveur Web de Google (https://www.google.com) ne correspond pas à l'URL accédée ce qui générerait une alerte dans un navigateur web

Génération d'un mot de passe aléatoire

```
openssl rand 15 -base64
Loading 'screen' into random state - done
6TMCLEgKG4JFot/XFETj
```

7. Signature de fichiers

Signature au format hexadécimale d'un fichier avec un certificat

```
openssl dgst -hex -sha1 -sign client_pvk.pem -out test.sha1 C:\archive.zip
```

Enter pass phrase for client_pvk.pem:

Nous obtenons alors un fichier contenant ceci :

SHA1(C:\archive.zip)=
3a5bf992899216c320835e98b700a1e669c1716e9145c4e2fec91eb3ee473d3d3d1dc264
e139f6bffc76d4d00e2cac6c6687e7d53613c94f0ca866334f9d6d43bfeb23a0d1bd51af8f500d1
29146d2a83d58c
b4051f112805beb7519489abd79f75d4878f834005631f5c59b5fbbcd0ecb391c5a9f1dbdcef1f9
d2f287cd3e41

A noter que le format hex n'est pas supporté par la méthode verify, la signature devra être au format binaire.

Signature au format binaire d'un fichier avec un certificat

openssl dgst -sha1 -sign client_pvk.pem -out test.sha1 C:\archive.zip

Extraction de la clef publique contenue dans le certificat

openssl x509 -in client_crt.pem -pubkey -noout > client_pub.pem

La clef publique étant extraite du certificat, il ne sera pas nécessaire de la fournir avec le fichier de signature

Vérification de la concordance fichier / clef publique

openssl dgst -sha1 -verify client_pub.pem -signature test.sha1 C:\archive.zip

Verified OK *ou* Verification Failure

vous obtenez un *unable to load key file* si vous utilisez un certificat au lieu d'une clef publique

Nous en concluons que le fichier a été signé par le propriétaire du certificat et qu'il n'a pas été modifié depuis.
Nous devons maintenant vérifier le contenu du certificat.

L'identité de son propriétaire

openssl x509 -in pubcert2.pem -noout -subject
subject= /C=FR/ST=FRANCE/O=Le nom du

client/OU=Service/CN=www.test.com/emailAddress=monemail@test.com

L'identité de l'émetteur du certificat

openssl x509 -in pubcert2.pem -noout -issuer
issuer= /C=FR/ST=France/L=Lille/O=Domain Public Primary Certification autority/OU=www.grandville.net/CN=cdn/emailAddress=nospam_autority@remove_grandville.net

Chiffrement par clef symétrique

Le chiffrement par clef symétrique est aujourd'hui le seul moyen performant pour la protection des données. [**la suite**]

Chiffrement par clef asymétrique

Le but est ici de rendre illisible le contenu d'un fichier par un algorythme à sens unique dont la spécificité est de permettre à tout le monde le chiffrement de ces données tout en ne permettant le déchiffrement qu'au seul destinatataire.

S'agissant d'un chiffrement par bloc, la quantité de données chiffrables par la clef est limitée à la taille de la clef moins la taille nécessaire à la déclaration du padding.
La taille de la clef RSA est de 1024 bits, petite vérification :

openssl rsa -in client_pvk.pem -inform PEM -text
Enter pass phrase for client_pvk.pem:
Private-Key: (**1024 bits**)
modulus:
00:b0:62:be:04:d9:04:6b:67:01:18:d4:4f:97:11:
f4:1d:84:14:99:70:b9:a7:01:fd:c5:41:95:c3:62:
...

1024 bits, est-ce une taille acceptable pour le chiffrement de mes données ?
Oui car 2^1024 est un chiffre absolument gigantesque, un exemple pour en caresser l'idée.
Par an, en France, entre 10 et 30 personnes meurent foudroyées ce qui correspond à une probabilité inverse de 2^21
La probabilité qu'une de ces mêmes personnes gagne à l'euromillion LE MEME JOUR n'est que de 2^46
Nous sommes encore bien loin du compte car 46 est largement plus petit que 1024.
Remarquons au passage que les dernières estimations de l'âge de l'univers sont de 2^59 secondes (13.7x10^9 années)...
Maintenant que nous sommes rassurés sur la taille et donc la complexité de notre clef de chiffrement, voyons comment l'utiliser.

Rappellons que des clefs de 1024 bits ne peuvent chiffrer qu'un bloc de données au maximum égal à 128 (1024/8 octets) caractères moins le padding.

Chiffrer 128 octets de données avec un certificat 1024 bits

Le chiffrement d'un bloc de données d'exactement 128 octets avec une clef 1024 bits ne pose pas de problème leurs tailles sont les mêmes.

```
openssl rsautl -certin -inkey client_crt.pem -in test.txt -encrypt -out test.enc -raw
Loading 'screen' into random state - done
```

On obtient une erreur
RSA operation error
816:error:0406B07A:rsa routines:RSA_padding_add_none:data too small for key size:.\crypto\rsa\rsa_none.c:76:
error in rsautl
si le fichier à chiffrer fait moins de 128 caractères car nous avons précisé qu'il ne faut pas utiliser de padding (-raw) et
816:error:0406B06E:rsa routines:RSA_padding_add_none:data too large for key size:.\crypto\rsa\rsa_none.c:70:
si le fichier à crypter est d'une taille supérieure à la clef

Chiffrer 128 octets de données (ou moins) avec un certificat

Pour un bloc de données de taille inférieure à la taille de la clef, la commande à utiliser sera

```
openssl rsautl -certin -inkey client_crt.pem -in test.txt -encrypt -out test.enc
```

Sans paramètre, le padding sera PKCS#1 v1.5, les tailles de données cryptables en fonction des padding sont les suivantes :
http://www.openssl.org/docs/crypto/RSA_public_encrypt.html
- exactement 128 octets si pas de padding (-raw)
- moins de 117 octets pour un padding PKCS #1 v1.5 ou SSL (-pkcs, -ssl)
- moins de 87 octets pour un padding PKCS#1 OAEP (-oaep)

Déchiffrer un bloc de données cryptées

La déchiffrement se fait uniquement avec la clef privée du certificat, le type de padding doit être indiqué et biensûr ne peut être deviné !

```
openssl rsautl -inkey client_pvk.pem -in test.enc -decrypt -out test.txt
Loading 'screen' into random state - done
Enter pass phrase for client_pvk.pem:
```

Utilisation du chiffrement asymétrique

Dans la pratique, les données à transférer sont cryptées avec une clef symétrique pour passer outre la limitation des 128 octets et c'est cette clef symétrique protégée par le chiffrement asymétrique qui est envoyée avec les données sécurisées au destinataire. Cette méthode accélére grandement le processus de chiffrage car ce travail sur les clefs asymétriques est environ 500 fois plus couteux que le même travail sur des clefs symétriques.

Voici un test de performance :

openssl speed aes-128-cbc rsa1024
To get the most accurate results, try to run this
program when this computer is idle.
First we calculate the approximate speed ...
Doing aes-128 cbc 20971520 times on 16 size blocks: 20971520 aes-128 cbc's in 7.01s
Doing aes-128 cbc 5242880 times on 64 size blocks: 5242880 aes-128 cbc's in 6.96s
Doing aes-128 cbc 1310720 times on 256 size blocks: 1310720 aes-128 cbc's in 6.86s
Doing aes-128 cbc 327680 times on 1024 size blocks: 327680 aes-128 cbc's in 6.82s
Doing aes-128 cbc 40960 times on 8192 size blocks: 40960 aes-128 cbc's in 6.93s
Doing 1310 1024 bit private rsa's: 1310 1024 bit private RSA's in 14.96s
Doing 13107 1024 bit public rsa's: 13107 1024 bit public RSA's in 7.13s
OpenSSL 0.9.8h 11 Oct 2005
built on: Fri Feb 3 15:41:35 2006
options:bn(64,32) md2(int) rc4(idx,int) des(idx,cisc,4,long) aes(partial) idea(int) blowfish(idx)
compiler: cl /MD /Ox /O2 /Ob2 /W3 /WX /Gs0 /GF /Gy /nologo -DOPENSSL_SYSNAME_WIN32 -DWIN32_LEAN_AND_MEAN -DL_ENDIAN -DDSO_WIN32 -D_CRT_SECURE_NO_DEPRECATE -DOPENSSL_USE_APPLINK -I. /Fdout32dll -DOPENSSL_NO_RC5 -DOPENSSL_NO_MDC2 -DOPENSSL_N O_KRB5
available timing options: TIMEB HZ=1000
timing function used: ftime

The 'numbers' are in 1000s of bytes per second processed.

type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
aes-128 cbc	47866.52k	48210.39k	48913.17k	**49207.26k**	48419.09k

	sign	verify	sign/s	verify/s
rsa 1024 bits	0.011421s	0.000544s	**87.6**	1838.3

Le chiffrement avec une clef asymétrique est **49207/87 = 565** fois plus couteux en temps processeur qu'un chiffrement symétrique AES-CBC.

Exemple de chiffrement d'un fichier

Commençons par chiffrer le fichier avec une clef symétrique et protégeons cette clef avec le ou les certificats des destinataires.
Après avoir généré les certificats de chaque destinataire, générons une clef symétrique

openssl rand -out pass.txt -base64 48
Loading 'screen' into random state - done

Le fichier pass.txt contient maintenant une chaine de caractères au format Base64 qui sera la clef de chiffrement symétrique.

Chiffrons le fichier archive.rar.

openssl enc -kfile pass.txt -e -in archive.rar -out archive.rar.enc -aes-128-cbc -p

```
salt=F384A365EC854856
key=14F7D9F96DC2C17CA02EF065A45A76E0
iv =5709622E05C1EE453DBD031975F4A96D
```

Chiffrons la clef de déchiffrement. Il faut d'abord créer le certificat du destinataire client1

```
openssl rsautl -certin -inkey client1_crt.pem -in pass.txt -encrypt -out pass.client1.enc
Loading 'screen' into random state - done
```

Précisons qu'il n'est pas nécessaire de connaitre la clef privée pour protéger les données à envoyer.
La clef de déchiffrement est maintenant irrémédiablement cryptée avec la clef publique du certificat du destinataire (client1_crt.pem).

Le destinataire recevra ces fichiers :
- archive.rar.enc
- pass.client1.enc

Procédure de déchiffrement

```
openssl rsautl -inkey client1_pvk.pem -in pass.client1.enc -decrypt -out pass.client1.txt
Loading 'screen' into random state - done
Enter pass phrase for client1_pvk.pem:↵

openssl enc -kfile pass.client1.txt -d -in archive.rar.enc -out archive.rar -aes-128-cbc
```

Le fichier archive.rar est maintenant lisible.

Format S/MIME

OpenSSL/SMIME

L'email est un moyen de communication courant mais qui a le défaut dans sa version classique de ne pas identifier les différents acteurs de cet échange.
La sécurisation du message comporte deux chapitres, assurer que le contenu a bien été expédié par la personne attendue avec un contenu non modifié et un deuxième point qui concerne la confidentialité des messages échangés.

L'identification forte de l'expéditeur et la non falsification du message.

L'identification s'appuie sur l'utilisation de la clef privée du certificat personnel de l'émetteur, le message d'origine est véhiculé avec un code de contrôle supplémentaire (Secure / Multipurpose Internet Mail Extensions). Ce hash (qui est le résultat du croisement d'une compression du message d'origine et de la clef privée) est ensuite vérifié par le destinataire par une opération inverse faisant entrer en jeu, la partie publique de la clef du certificat et le message reçu, si la vérification est correcte alors le message n'a pas été modifié et a bien été écrit par l'expéditeur, voici pour le principe.

Exemple de génération d'un message S/MIME

Le contenu de message.txt

Content-Type: text/plain;

charset="us-ascii"

Content-Transfer-Encoding: 7bit

le message

La commande OpenSSL

```
openssl smime -sign -in message.txt -inkey user.pvk.pem -out smimeout.txt -from "<contact@xxxxxxxx.net>" -to "<contact@xxxxxxxx.net>" -signer user_crt.pem -subject "My secure email"↲
```

Le fichier smimeout.txt obtenu a ce format :

```
To: <contact@xxxxxxxx.net>
From: <contact@xxxxxxxx.net>
Subject: My secure email
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/x-pkcs7-signature";

micalg=sha1; boundary="----ADA02FC5AFFC8ED8EF7E5E458A51E637"


This is an S/MIME signed message

------ADA02FC5AFFC8ED8EF7E5E458A51E637
Content-Type: text/plain;
charset="us-ascii"
Content-Transfer-Encoding: 7bit

le message

------ADA02FC5AFFC8ED8EF7E5E458A51E637
Content-Type: application/x-pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIIEXAYJKoZIhvcNAQcCoIIETTCCBEkCAQExCzAJBgUrDgMCGgUAMAsGCSqGSIb
3
DQEHAaCCAkcwggJDMIIBrKADAgECAgEIMA0GCSqGSIb3DQEBBQUAMH0xFzAVB
gNV
BAMTDkVtYWlsIGF1dG9yaXR5MSUwIwYDVQQKExxBcm5hdWQgR3JhbmR2aWxsZS
BD
b25zdWx0aW5nMR4wHAYDVQQLExVGb3IgaW50ZXJuYWwgdXNlIG9ubHkxCzAJBgN
```

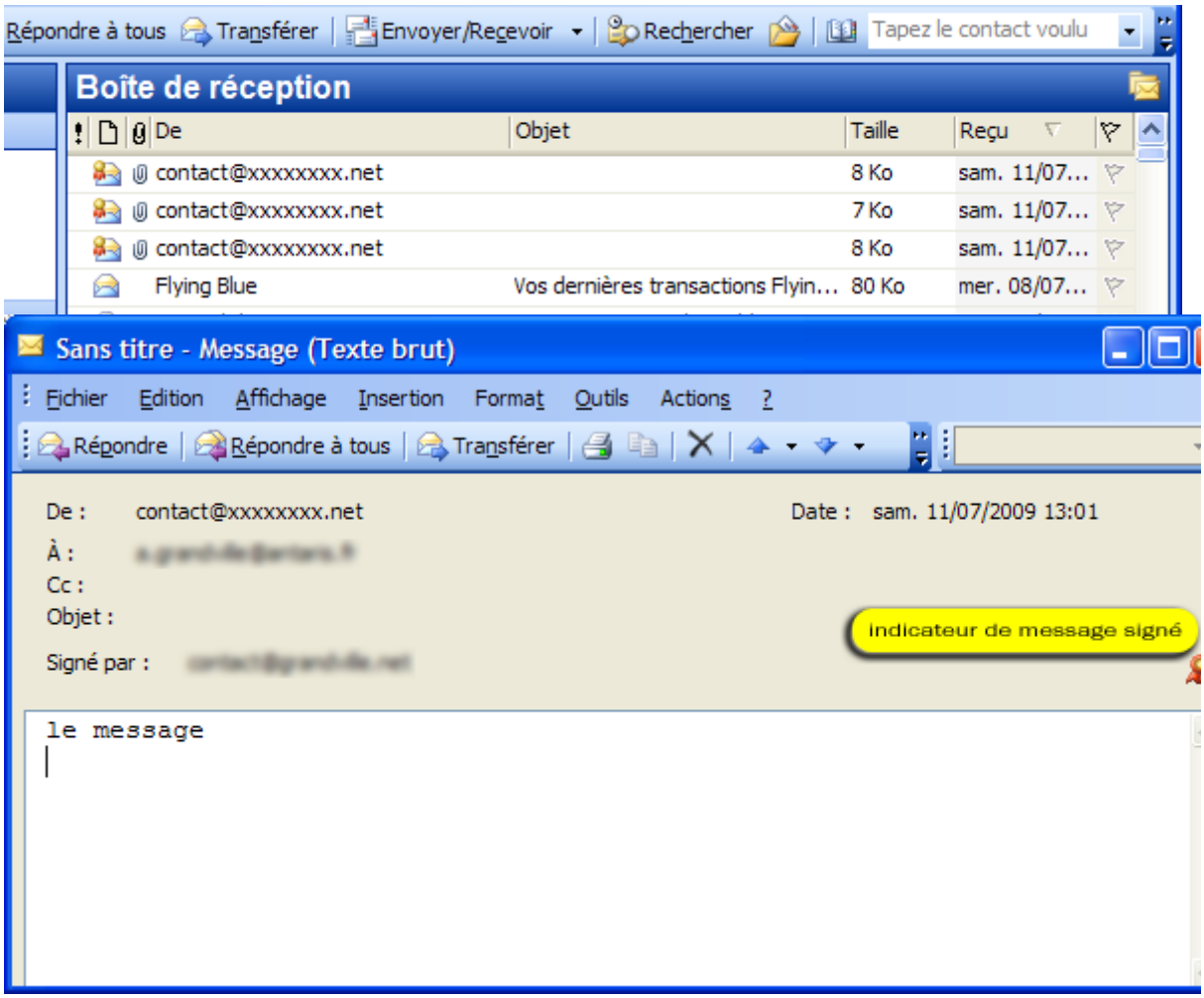
```
V
BAYTAkZSMQ4wDAYDVQQHEwVMSUxMRTAeFw0wOTA3MTEwNzI1NTBaFw0xM
DA3MTEw
NzI1NTBaMEMxGjAYBgNVBAMTEUFybmF1ZCBHcmFuZHZpbGxlMSUwIwYJKoZIhv
cN
AQkBFhZjb250YWN0QGdyYW5kdmlsbGUubmV0MIGfMA0GCSqGSIb3DQEBAQUAA4
GN
ADCBiQKBgQC98RJdPUyTUu4Rj5O0u9/iy/rchp8a8OyIjNNjZjWEghmUIB+JHJQG
zUoEP6MhuW/C7r9CLhuJpZguvb42MHOWhuqyasaHMG2NzhLl1w94Tk7SjFTXugn7
bSvlTTdM+xb76uo9M8G7O3UELIr+12Ua+z8ey44Gze3yPDH6JhNc0wIDAQABow0w
CzAJBgNVHRMEAjAAMA0GCSqGSIb3DQEBBQUAA4GBAFfmFI7a2VWpJRzNqJDBqj
AH
81DZ1ezdTfyP3kKuwVrrg7QZfrxAbor2U4HwlQu61/RPVVZArzCmumrSOfXJxDzp
ABvtFFUkYhGbOb5VSCu2deho43ARWrxhtedI9uYXlOCbn2CkWbU7axE6ggadXYWS
6jXN2g1eocQXEfWCAMlPMYIB3TCCAdkCAQEwgYIwfTEXMBUGA1UEAxMORW1haWwg
YXV0b3JpdHkxJTAjBgNVBAoTHEFybmF1ZCBHcmFuZHZpbGxlIENvbnN1bHRpbmcx
HjAcBgNVBAsTFUZvciBpbnRlcm5hbCB1c2Ugb25seTELMAkGA1UEBhMCRlIxDjAM
BgNVBAcTBUxJTExFAgEIMAkGBSsOAwIaBQCggbEwGAYJKoZIhvcNAQkDMQsGCS
qG
SIb3DQEHATAcBgkqhkiG9w0BCQUxDxcNMDkwNzExMTAzNjQ1WjAjBgkqhkiG9w0B
CQQxFgQUyEMwqN2DDt/48qdTnpfmtXOHICkwUgYJKoZIhvcNAQkPMUUwQzAKBgg
q
hkiG9w0DBzAOBggqhkiG9w0DAgICAIAwDQYIKoZIhvcNAwICAUAwBwYFKw4DAgc
w
DQYIKoZIhvcNAwICASgwDQYJKoZIhvcNAQEBBQAEgYA/8fsPurdfDdDmhSokuB0J
LLiZRQP07SgVJGz+EQqHema4VplhvmEtwaSwlQZ2w85gRyEkX3KSUg5QuE2XySc7
vW4F/EjPlTVzZpogeqCZJVXS017+JiZurlK/8BaSV+w2JudpyX3HVvWk1NJbTuTi
/azLmum3ZP5BGsQC7fYuTA==

------ADA02FC5AFFC8ED8EF7E5E458A51E637--
```

L'envoi du message en telnet

```
telnet smtp.free.fr 25↵
mail from: contact@xxxxxxxx.net↵
rcpt to: contact@xxxxxxxx.net↵
data↵
le contenu de smimeout.txt
.↵
quit↵
```

L'apparence des emails signés sous Outlook



Crypter, décrypter, signer, vérifier un message email

•#openssl smime –encrypt –in msg.clair –des3 –out msg.crypte guest.pem

•#openssl smime –decrypt –in msg.crypte –recip guest.pem

•#openssl smime –sign –in msg.clair –signer guest.pem –out msg.sgn

•#openssl smime –verify –in msg.sgn –CAfile Cacert.crt

Structure binaire des certificats

Un certificat X509 v3 est un fichier binaire dont la structure est définie par la RFC2459...

La structure binaire d'un certificat

Un certificat X509v3 est un fichier binaire dont la structure est définie par la RFC2459 et le cadre d'utilisation par l'International Telecommunication Union (ITU).
Contrairement au fichier XML pour lequel les données sont transportées en même temps que leurs descriptions, un fichier binaire nécessite pour son exploitation la fourniture d'un **lexique** permettant la compréhension de son contenu. Le rôle du lexique est joué par un fichier au format Abstract Syntax Notation One (ASN.1). On retrouve ce même langage de définition dans le protocole de management des équipements réseaux SNMP.

Voici le début de cette description pour les certificats

```
Certificate ::= SEQUENCE {

        tbsCertificate       TBSCertificate,
        signatureAlgorithm   AlgorithmIdentifier,
        signatureValue       BIT STRING  }

   TBSCertificate  ::=  SEQUENCE  {
        version         [0]  EXPLICIT Version DEFAULT v1,
        serialNumber         CertificateSerialNumber,
        signature            AlgorithmIdentifier,
        issuer               Name,
        validity             Validity,
        subject              Name,
        subjectPublicKeyInfo SubjectPublicKeyInfo,
        issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                             -- If present, version shall be v2 or v3
        subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                             -- If present, version shall be v2 or v3
        extensions      [3]  EXPLICIT Extensions OPTIONAL
                             -- If present, version shall be v3
        }

   Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }
   CertificateSerialNumber  ::=  INTEGER


.../...
```

On en comprend qu'un Certificat est composé de trois éléments :
- Une structure (tbsCertificate)
- Un algorithme de hashage (AlgorithmIdentifier)
- Une signature (signatureValue)

Un certificat n'est donc ni plus ni moins qu'une simple structure binaire dont la valeur est garantie par l'apposition d'une signature (Hash produit par une autorité de confiance). Au passage, remarquons qu'il n'y aucune différence technique entre un certificat, une CA, une CRL, etc ... tous ces fichiers sont des structures binaires associées à une signature.

Modification de la version du certificat

La version peut prendre trois valeurs : 0,1,2 chacune correspond respectivement à la version v1,v2 et v3 des certicats x509.
Le certificat de https://www.google.com contient ces informations :

```
C:\openssl-0.9.8g\out32dll>openssl x509 -text -noout -inform
DER -in google.der
Certificate:

    Data:
        Version: 3 (0x2)
        Serial Number:
            68:76:64:38:3d:49:6e:2e:f5:e3:19:98:42:e0:7c:ee
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=ZA, O=Thawte Consulting (Pty) Ltd., CN=Thawte SGC CA
        Validity
            Not Before: May  3 15:34:58 2007 GMT
            Not After : May 14 23:18:11 2008 GMT

.../...
```

Le certificat est en v3 (valeur 2)

OpenSSL permet l'analyse binaire du certificat via l'option asn1parse

```
C:\openssl-0.9.8g\out32dll>openssl asn1parse -i -inform DER -
in google.der

    0:d=0  hl=4 l= 801 cons: SEQUENCE
    4:d=1  hl=4 l= 650 cons:  SEQUENCE
    8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
   10:d=3  hl=2 l=   1 prim:    INTEGER           :02
   13:d=2  hl=2 l=  16 prim:   INTEGER
:687664383D496E2EF5E3199842E07CEE
   31:d=2  hl=2 l=  13 cons:   SEQUENCE
   33:d=3  hl=2 l=   9 prim:    OBJECT            :sha1WithRSAEncryption
   44:d=3  hl=2 l=   0 prim:    NULL
   46:d=2  hl=2 l=  76 cons:   SEQUENCE
   48:d=3  hl=2 l=  11 cons:    SET
   50:d=4  hl=2 l=   9 cons:     SEQUENCE
   52:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
   57:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :ZA
   61:d=3  hl=2 l=  37 cons:    SET
   63:d=4  hl=2 l=  35 cons:     SEQUENCE
   65:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
   70:d=5  hl=2 l=  28 prim:      PRINTABLESTRING   :Thawte Consulting
(Pty) Ltd.

.../...
```

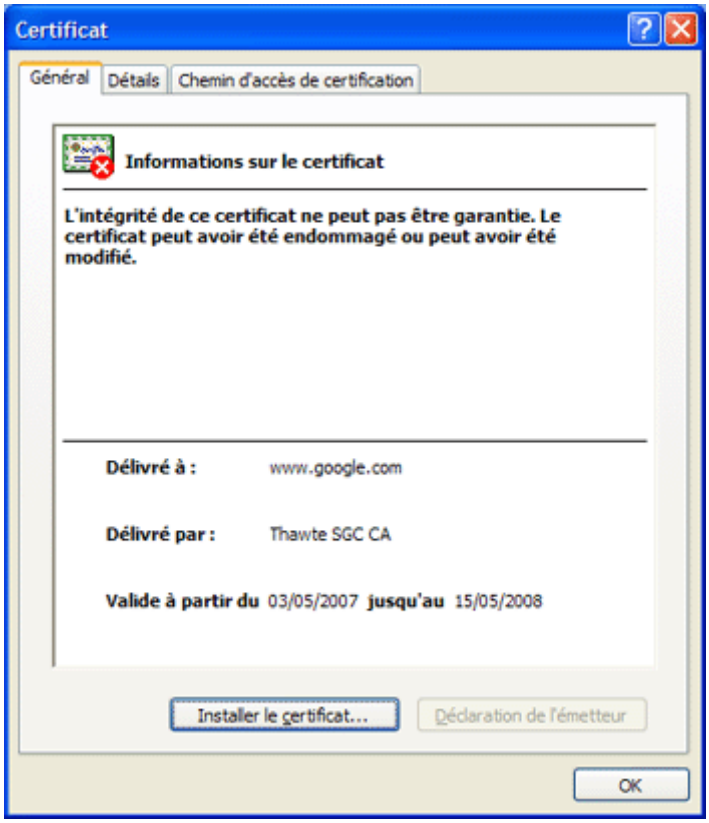
Nous apprenons que dans le fichier google.der à l'offset 10 se trouve trois octets (hl+l), le troisième octet est un entier dont la valeur est deux.
Avec un éditeur hexa fixons ce 12ème octet du fichier à 1 et voici le résultat

```
C:\openssl-0.9.8g\out32dll>openssl x509 -text -noout -inform
DER -in google.der
Certificate:

    Data:
        Version: 2 (0x1)
        Serial Number:
            68:76:64:38:3d:49:6e:2e:f5:e3:19:98:42:e0:7c:ee
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=ZA, O=Thawte Consulting (Pty) Ltd., CN=Thawte SGC CA

.../...
```

Le certificat est bien modifié, le code de hashage pourrait être recalculé mais ne pouvant être signé l'ouverture du certificat donne ceci :



Liens et outils

La syntaxe ASN1 : http://www.oss.com/asn1/booksintro.html
Deux librairies opensource ASN.1 : http://lionet.info/asn1c/basics.html , http://josefsson.org/libtasn1/

Un exemple obtenu par libtasn1

```
c:\libtasn1-0.3.9\tests>..\src\asn1Decoding -c pkix.asn
google.der PKIX1.Certificate
Parse: done.

Decoding: SUCCESS

DECODING RESULT: name:NULL type:SEQUENCE

  name:tbsCertificate  type:SEQUENCE
    name:version  type:INTEGER  value:0x02
      name:NULL  type:DEFAULT  value:v1
    name:serialNumber  type:INTEGER
value:0x687664383d496e2ef5e3199842e07cee
    name:signature  type:SEQUENCE
      name:algorithm  type:OBJ_ID  value:1.2.840.113549.1.1.5
      name:parameters  type:ANY  value:0500
    name:issuer  type:CHOICE
      name:rdnSequence  type:SEQ_OF
        name:NULL  type:SET_OF
          name:NULL  type:SEQUENCE
            name:type  type:OBJ_ID
            name:value  type:ANY
        name:?1  type:SET_OF
          name:NULL  type:SEQUENCE
            name:type  type:OBJ_ID
            name:value  type:ANY
          name:?1  type:SEQUENCE
            name:type  type:OBJ_ID  value:2.5.4.6
            name:value  type:ANY  value:13025a41
        name:?2  type:SET_OF
          name:NULL  type:SEQUENCE
            name:type  type:OBJ_ID
            name:value  type:ANY
          name:?1  type:SEQUENCE
            name:type  type:OBJ_ID  value:2.5.4.10
            name:value  type:ANY
value:131c54686177746520436f6e73756c74696e6720285074792920c74642e
        name:?3  type:SET_OF
          name:NULL  type:SEQUENCE
            name:type  type:OBJ_ID
            name:value  type:ANY
          name:?1  type:SEQUENCE
            name:type  type:OBJ_ID  value:2.5.4.3

.../...
```