



Travaux pratiques de confiance électronique

Sébastien Pujadas

Table des matières

Préface	vii
Introduction	ix
1. Cryptographie symétrique	1
1.1. Chiffrement symétrique	1
1.2. Déchiffrement symétrique	2
2. Hachage	3
3. Cryptographie asymétrique	5
3.1. Bi-clé	5
3.2. Chiffrement asymétrique	8
3.3. Signature numérique	10
3.4. Construction d'une clé publique	12
4. Certificats	19
4.1. Génération des bi-clés	19
4.2. Génération des requêtes de signature de certificat	20
4.3. Génération des certificats	25
4.4. Vérification manuelle de la signature d'un certificat	28
4.5. Génération d'un certificat de confidentialité	31
4.6. Importation dans le magasin de certificats de Windows	32
4.7. Construction d'un certificat	36
5. Liste de certificats révoqués	47
5.1. Émission d'une liste de certificats révoqués	47
5.2. Révocation d'un certificat	48
5.3. Construction d'une liste de certificats révoqués	49
6. Confidentialité — PKCS#7 et CMS	55
6.1. Chiffrement d'un fichier	55
6.2. Déchiffrement d'un fichier	55
6.3. Analyse d'une structure PKCS#7/CMS	57
6.4. Déchiffrement manuel d'un fichier	59
6.5. Spécificités de CMS	61
7. Confidentialité — XML Encryption	63
7.1. Chiffrement d'un fichier non XML	64

7.2. Déchiffrement d'un fichier	66
7.3. Déchiffrement manuel d'un fichier	66
7.4. Chiffrement d'un élément XML	69
8. Signature électronique — PKCS#7/CMS	73
8.1. Signature d'un fichier	73
8.2. Vérification d'une signature	75
8.3. Analyse d'une structure PKCS#7/CMS	75
8.4. Vérification manuelle d'une signature	77
8.5. Construction d'une signature électronique PKCS#7/CMS	83
9. Signature électronique — XML Signature	89
9.1. Signature d'un fichier	90
9.2. Vérification d'une signature	94
9.3. Vérification manuelle d'une signature	94
10. Authentification web	103
10.1. Génération des certificats et des listes de certificats révoqués	103
10.2. Compilation et installation du serveur web nginx	111
10.3. Paramétrage du serveur web nginx	113
10.4. Mise en place des serveurs OCSP	115
10.5. Paramétrage de TLSv1.2	118
10.6. Internet Explorer	120
10.7. Firefox	122
11. Horodatage	125
11.1. Horodatage de données	125
11.2. Serveur d'horodatage	129
11.3. Construction d'un jeton d'horodatage	131
12. Signature électronique avancée — CAdES	141
12.1. Constitution de la structure CAdES-BES	143
12.2. Constitution de la structure CAdES-T	147
12.3. Constitution de la structure CAdES-LT	151
13. Signature électronique avancée — XAdES	173
13.1. Constitution de la structure XAdES-BES	174
13.2. Constitution de la structure XAdES-T	177
13.3. Constitution de la structure XAdES-A	180
A. Représentation et codage des données	193
A.1. Base64	193
A.2. PEM	193

A.3. Distinguished Name	193
A.4. Codage des caractères	194
B. ASN.1	197
B.1. Distinguished Encoding Rules	197
B.2. Génération d'un fichier de configuration ASN.1 pour OpenSSL	198
B.3. dumpasn1	214
B.4. Initiation au compilateur asn1c	214
C. Compilation et installation d'OpenSSL	219
C.1. Linux	219
C.2. Windows	221
D. Compilation de xmlsec sous Windows	223
D.1. Compilation de libxml2	223
D.2. Compilation de libxslt	224
D.3. Compilation et installation de xmlsec	224
E. Spécificités de Windows	227
E.1. OpenSSL et CryptoAPI	227
E.2. Analyse du magasin de certificats de Windows	229
F. Code source	237
F.1. Paramétrage de l'algorithme de signature des jetons OCSP produits par OpenSSL	237
F.2. Génération d'un fichier test_ev_roots.txt pour Firefox	239
G. Considérations légales	245
G.1. Conditions d'utilisation de ce document	245
G.2. Droit des marques	246
G.3. Usage de la cryptographie	246
G.4. Attributions	247
H. Acronymes et sigles	249
Colophon	255

Préface

La confiance électronique est un domaine assez méconnu de l'informatique, dont l'usage se limite la plupart du temps à la génération et installation de certificats d'application pour permettre l'accès à des serveurs web en HTTPS. Il s'agit pourtant d'un sujet passionnant et riche dont le potentiel (signature électronique notamment) n'est pas encore pleinement exprimé. J'ai eu le plaisir de travailler sur des projets mettant en œuvre des éléments de confiance électronique pendant une dizaine d'années, explorant souvent un terrain encore en friche, parsemé de normes et implémentations parfois instables ou contradictoires et de formats d'un abord hermétique, le tout sur un fond de cryptographie qui en a certainement découragé plus d'un. Conséquence ou facteur aggravant, il a toujours été difficile de trouver des ressources sur le sujet, et tous ceux qui ont voulu approfondir leurs connaissances ont dû découvrir séparément comment utiliser le peu d'outils disponibles, en en créant de nouveaux au besoin pour aller plus loin. J'avais initialement prévu d'écrire et de publier un aide-mémoire pour éviter aux aventuriers de la confiance électronique de réinventer la roue à leur insu, mais ce projet a pris une toute autre dimension, et le recto-verso que j'avais initialement prévu d'écrire a finalement pris la forme du document que vous avez sous les yeux.

Aux dires des relecteurs – Benjamin Dossat, Cédric Clément et Pierre Lagnier, spécialistes du domaine que je remercie ici – ce livre est exigeant, et écrivant cette préface lors de ma propre relecture plus d'un an après l'écriture initiale, je suis bien obligé de leur donner raison : la notion de « travaux » du titre trouve pleinement son sens ici, mais disons qu'à vaincre sans péril... ;o)

J'espère que ce document vous permettra d'apprécier la confiance électronique en profondeur et vous aidera à contribuer à sa généralisation.

Sébastien Pujadas

Châtenay-Malabry, octobre 2013

Introduction

Ce document a pour but d'illustrer les grands concepts de la confiance électronique (cryptographie, certificats, signature électronique etc.), principalement par la manipulation d'outils en ligne de commande.

Les chapitres 1 à 3 introduisent les primitives cryptographiques utilisées dans le domaine de la confiance électronique : algorithmes de chiffrement symétrique, algorithmes de hachage, et cryptographie asymétrique (bi-clés RSA, chiffrement et signature).

Ensuite, les chapitres 4 et 5 s'intéressent aux structures de données de base d'une infrastructure de gestion de clés (IGC, ou PKI pour *Public Key Infrastructure*, ou encore ICP pour infrastructure à clés publiques) : les certificats électroniques X.509 et les listes de certificats révoqués.

Les usages de la confiance électronique font l'objet des chapitres 6 à 10 : protection de la confidentialité des données à l'aide des formats PKCS#7/CMS et XML Encryption, signature électronique en utilisant les formats PKCS#7/CMS et XML Signature, authentification web par certificat.

Le chapitre 11 sur l'horodatage prépare enfin les chapitres 12 et 13 sur la signature électronique avancée avec les formats CAdES et XAdES.

Pré-requis

Ce document ne se substitue volontairement pas à une présentation académique de la confiance électronique : les concepts fondamentaux ne sont pas rappelés, et il conviendra donc de se tourner vers d'autres ressources¹. Le lecteur est par ailleurs supposé être familiarisé avec les sujets suivants : les bases numériques (binaire et hexadécimal), XML, l'utilisation du shell du système d'exploitation.

Les outils suivants sont supposés être installés sur l'environnement utilisé par le lecteur (lequel peut être notamment Windows, UNIX/Linux, ou Mac OS X) :

- OpenSSL², pour les opérations cryptographiques unitaires et la manipulation des structures binaires usuelles de la confiance électronique (X.509, PKCS, etc.). Une annexe présente la compilation et l'installation d'OpenSSL en environnement GNU/Linux et Windows).
- XML Security Library³, pour la manipulation des formats XML Encryption et XML Signature. La compilation de XML Security Library sous Windows est détaillée en annexe.
- XMLStarlet⁴, pour la manipulation de structures XML.

1. Par exemple *Cryptographie appliquée*, deuxième édition, de Bruce Schneier. Éditions Vuibert, 2001, 846 p.

2. <http://www.openssl.org>

3. <http://www.aleksey.com/xmlsec/>

4. <http://xmlstar.sourceforge.net/>

- Les commandes `od` (ou `hexdump` en adaptant la syntaxe), `grep`, `tr`, `fold`, `head`, `sed`, `dd`, `sort`, `cut`, `xargs` et `cat`. Ces commandes sont toutes standard en environnement GNU/Linux et BSD, et utiliser GnuWin⁵ sous Windows.


La version actuelle de ce document a été conçue pour rentrer le plus rapidement possible dans le vif du sujet. Les éventuelles futures révisions seront moins exigeantes en pré-requis.

Conventions d'écriture

Sauf précision contraire :

- La marque d'invite de commande « `$` » est générique, indiquant que la commande peut-être saisie dans le shell de tout système d'exploitation compatible, tandis que la marque d'invite de commande « `>` » indique que la commande est spécifique à l'environnement Windows. La marque d'invite de commande « `#` » indique une commande à saisir sous le compte `root` sous UNIX/Linux.
- Les lignes de commande saisies par l'utilisateur sont en caractères gras.
- Le caractère « `\` » est employé dans les lignes de commande pour mettre en évidence une commande trop longue pour tenir sur une ligne « physique » de 80 colonnes : en environnement UNIX/Linux, ce caractère peut être saisi littéralement (suivi de la touche Entrée) pour saisir la commande sur plusieurs lignes, mais sous Windows l'intégralité de la commande doit être saisie sur une seule ligne.

Les remarques sont mises en forme comme ceci :

 Ceci est une remarque.

Enfin, des sujets connexes sont traités dans des encadrés.

Ressources complémentaires

Le code source, les fichiers générés et le code source de ce livre sont publiés sur GitHub, à l'URL <https://github.com/spujadas/tp-confiance>.

5. <http://gnuwin32.sourceforge.net/>

Chapitre 1 — Cryptographie symétrique

Ce chapitre traite des deux algorithmes de chiffrement symétrique par bloc les plus couramment utilisés dans le domaine de la confiance électronique : DES et AES.

La terminologie courante mais non officielle « 3DES » est employée ci-après pour désigner Triple DES à trois clés indépendantes, officiellement connu sous le nom de 3TDEA (*Triple Data Encryption Algorithm, keying option 1*).

Les algorithmes de chiffrement symétrique par bloc chiffrent successivement des blocs de données de longueur fixe, par exemple 64 bits pour DES/3DES et 128 bits pour AES.

L'utilisation des algorithmes de chiffrement symétriques nécessite :

- une *clé de chiffrement* symétrique (qui peut par exemple être dérivée d'une clé partagée), dont la longueur dépend de la taille du bloc de chiffrement et du mode d'application de l'algorithme, par exemple 256 bits pour AES-256, 56 bits pour DES, et 168 bits (3×56) pour 3DES,
- dans le cadre de l'utilisation du mode opératoire de chiffrement CBC (*cipher-block chaining*), qui est le mode le plus courant dans le domaine de la confiance électronique, un *vecteur d'initialisation* (qui doit être initialisé aléatoirement), de longueur égale à la taille du bloc de chiffrement de l'algorithme.

Les exemples ci-après s'appuient sur l'algorithme AES-256 en mode CBC. Ils peuvent facilement être adaptés à d'autres algorithmes (par exemple en utilisant la commande `des3` ou `des-ede3-cbc` pour 3DES).

1.1. Chiffrement symétrique

Utiliser la commande `openssl` pour chiffrer un fichier avec une clé AES-256 (commande `aes-256-cbc` ou `enc aes-256-cbc`) dérivée d'un mot de passe arbitraire (saisir et confirmer le mot de passe au moment demandé) sans salage (option `-nosalt`), et afficher la clé et le vecteur d'initialisation.

```
$ openssl aes-256-cbc -nosalt -p -in data.txt
enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
key=59CC30AD02C25BB7A8757E20D03BD6219DF9C4E87FDFA644CADD42DEB1DCB6F
iv =A2439E6F27D61066B688122C5B44A242
îkRL_kÅ+f,ŕu†Ū46%
```

La clé (`key`) et le vecteur d'initialisation (`iv`) sont affichés sous la forme d'une chaîne hexadécimale. Les données chiffrées sont affichées en brut.

L'option `-nosalt` permet pour les besoins de l'exemple de dériver la même clé et le même vecteur d'initialisation à partir d'un mot de passe donné. En pratique, pour ne pas dégrader le niveau de sécurité,

1. Cryptographie symétrique

il conviendra soit de générer la clé et le vecteur d'initialisation aléatoirement, soit de dériver ces éléments en utilisant l'option de salage PKCS#5 (`-salt`, ou en omettant `-nosalt`) pour éviter une attaque par dictionnaire sur le mot de passe d'entrée.

Réutiliser la commande précédente avec l'option `-out data.enc` pour écrire les données chiffrées dans un fichier.

1.2. Déchiffrement symétrique

Déchiffrer le fichier `data.enc` en utilisant la clé et le vecteur d'initialisation généré précédemment.

```
$ openssl aes-256-cbc -d -in data.enc \
-K 59CC30AD02C25BB7A8757E20D03BD6219DF9C4E87FDFA644CADD42DEB1DCB6F \
-iv A2439E6F27D61066B688122C5B44A242
texte en clair
```

Exercice — Chiffrer un texte clair d'une longueur au moins égale à trois blocs, modifier un bit dans un des premiers blocs, et déchiffrer le fichier résultant. Observer que le bloc dans lequel a eu lieu la modification est inintelligible, et que le bloc suivant est légèrement affecté (le bit correspondant au bit modifié du bloc précédent est inversé). Cet exercice met en évidence l'absence de contrôle d'intégrité global, et le mode d'opération du déchiffrement par CBC.

Padding des blocs

Lorsque, après découpage des données à chiffrer en blocs, le dernier bloc à chiffrer est de longueur inférieure à la taille de bloc attendue par l'algorithme de chiffrement, des octets de remplissage (*padding*) doivent être utilisés pour compléter le bloc. OpenSSL propose par défaut le mécanisme de *padding* dit standard (issu de la RFC 1423 et popularisé par PKCS#5) : s'il manque n octets pour compléter le bloc, alors celui-ci est complété avec n octets contenant la valeur n (ou, si le bloc était complet, le mécanisme ajoute un bloc dont chaque octet contient le nombre d'octets du bloc).

Pour visualiser l'effet du mécanisme de *padding* sur le texte en clair, déchiffrer le texte chiffré avec l'option `-nopad` et analyser la représentation hexadécimale du résultat.

```
$ openssl aes-256-cbc -d -in data.enc \
-K 59CC30AD02C25BB7A8757E20D03BD6219DF9C4E87FDFA644CADD42DEB1DCB6F \
-iv A2439E6F27D61066B688122C5B44A242 -nopad | od -tx1z
00000000 74 65 78 74 65 20 65 6e 20 63 6c 61 69 72 02 02  >texte en clair..<
00000020
```

Le texte en clair constituant le fichier a une longueur de 14 octets, soit deux octets de moins que la taille d'un bloc AES-256 de 16 octets. Les deux octets restants sont donc complétés avec la valeur `0x02`.

Chapitre 2 — Hachage

Générer un fichier arbitraire (`C^Z` désigne la combinaison de touches `Ctrl` et `Z`).

```
$ cat > data.txt
texte en clairC^Z
C^Z
```

Utiliser OpenSSL pour calculer l'empreinte SHA-256 du fichier.

```
$ openssl sha256 data.txt
SHA256(data.txt)= 89bd92286d6c8014c06030b25f8b40cc1d5656d4b3b7b4831874f50d6f5557
f3
```

La valeur affichée est la représentation hexadécimale de l'empreinte. La valeur brute (binaire) de l'empreinte peut être produite en utilisant l'option `-binary` :

```
$ openssl sha256 -binary data.txt
oUW%mlÇ¶L`0_ï@|←VVE|À-â↑t$
```

Pour observer le phénomène de dispersion des empreintes, modifier un bit du fichier d'origine, par exemple en remplaçant « texte » par « uexte » (ce qui transforme le premier octet `0b01110100` en `0b01110101`), puis calculer l'empreinte du nouveau fichier et noter que l'empreinte obtenue est totalement différente de la première.

```
$ openssl sha256 data2.txt
SHA256(data2.txt)= cf028a03deccb1928ac3ec19a64e61f557a0a1b2d9c5352a9636fe9f6f1e1
c16
```

2. Hachage

Chapitre 3 — Cryptographie asymétrique

3.1. Bi-clé

Génération d'un bi-clé RSA

Les deux genres du mot « bi-clé » coexistent pacifiquement dans le monde de la confiance électronique : les uns parlent d'une bi-clé (une clé double) et les autres (dont l'auteur) d'un bi-clé (un objet à deux clés).

Générer un bi-clé RSA de 2048 bits.

```
$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048
.....
.....
.....+++
.....+++
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQCnVZCIiymfCGJK
E9zMVmnVBt5HG4IG7a7jv50fFjYi0oHDVIOacTfbu6tXFQXe90ENm140/B+SVbyI
t/vG3DkRfHGQe8hCqmSuaEoyNe/6okwac11/ZsVXILw+rh4szbeNC0QqvK+xtxY6
Z7VRzh+7DQL+AE8YLKV4yPgPfcUyGFQBiWtw55VdKQKiFQRxD+jmist7fBHda1X
CmY+oRemIIoH8Fu9FIcyl++j43pppgnYP8n+B7k09ct2Qq/dlvjN3S4UNgJ6vMuH
Yuq8prXUVM+zdmy5nsdxYdMBzaIKwhQKZ84rplYNd9eLPYzyQWFZrKUJCYLBVQ6
Po2RaadZAgMBAAECggEBAJv16exsyvAxnXEqpDFz6NHhvkE9bZOK4D9Knmw/A1za
Bq93E81V4hbD99P/8CJU9bS5pcFzi+6IFkjCML6K+lhC0oPD70W8/mRHIak5+OTk
5EZYaSmDFCQiiX9UNSTE3Fd5wZ6XgJDv5L0xX4rBoWphqW+Q7JgFiI8CnBUX+LQ
c5ah60uHc5CyuJCD5nwPr2ojlCRHhNr5iI97v06HEABB3Q5QQVQBZF324JinkJq1
tN9ZcBDKd+y9/G7TxqGZliUGVDcGJkHxo4KsUSyE7Zk3RXqciZ1lkJxF8LfZi2gX
nUIXV/tDH/8UMXK+01ps6KJPyc60IQ+ecVaeBM01WkCgYEA1RLrSGQeBQ7FWa/w
IRFDX9mOPLpG12n0yAxitZOPhAGEpDcKM4YpCCMxuQPbvLyLd2CbAWvdCr9e4UPF
KTyjL/+TnwFR5EALoVu06X74k5fW5BEyDQTAmYi9zM2Afa2Ydw6BZlo9u19uIMpu
gDAnH/klz8cuNxbdmHHSaOi+V58CgYEAyQuql2wuwVmCR8oXFX5ggipnAf+3tfeQ
WETPH567mjyhNWbZ1PqxRBztnp2DkFoAozve6fxxIhkoJ5V2kXciDnYaj79R6Xv5
LnJM4/Jtt1AEmt2RRaKjR0Iw43u5cuumQlvb86k9I+W6SJbyMuzZFdwadcb64cUT
NLGW4bbLfgcCgYBe4TjSGHrhp60rfdPA9c35nZjLver306BbgGBoHNMu5fopobSy
MtiMnhdjGpu7lH3KH80GQ4C/a24VgzcfYNNIN6pHSqSTNLd0/+2RNB1QrbN6s2heM
Ohvtgl16GDxS0n387gGjESYSDLGeemXwQQD7FNfIiRpEP7NXUb/A1eMEYwKBgQC7
MSakmPERzEjWyr8XPzVi3VJN9Si7wIdg+KiV+3kYEk3T6DD48nbtQbYqEuV2Gacw
VcWnvqokwqG1wZ/Fr9RA/MycfXoqW0lZysk3EPoBTfsLqzPhT56Pu7/jf8bbbvi+
Hs08qx6ndvYtMob0zeMPTIUCgWgFoTbRwG0J0udcLwKBgC4mWdKGAS3KrfvcT+z5
dsbu0NuN3X8qDcb0J1iCTQpV4eQUi+JwB3MaKM1ThlY5AQXmev0UgwbQrX3tp4bc
ts+8iEJktZGI7TN6LBqW679VCqRkEMEqJ5w8xj5NuLimXzDbhX1Dysn7pe54ZArL
BOIcc+eDX6AKmoQp80q698Jj
-----END PRIVATE KEY-----
```

3. Cryptographie asymétrique

La syntaxe ci-dessus remplace l'ancienne syntaxe `openssl genrsa 2048`, déclarée *superceded* par la documentation d'OpenSSL, mais qui est toujours utilisable à la date d'écriture (en version 1.0.1 d'OpenSSL).

Copier la clé ci-dessus (de `-----BEGIN PRIVATE KEY-----` à `-----END PRIVATE KEY-----`, ces deux balises PEM étant incluses dans la clé) dans un fichier nommé `rsakey.pem`, ou utiliser la commande ci-dessus avec l'option `-out rsakey.pem` pour poursuivre avec une autre clé.

Analyse d'un bi-clé

Afficher le contenu du bi-clé.

```
$ openssl rsa -in rsakey.pem -text -noout
Private-Key: (2048 bit)
modulus:
  00:b8:6f:48:f9:99:f1:99:71:c6:6f:80:64:d1:ca:
  ...
publicExponent: 65537 (0x10001)
privateExponent:
  4b:10:f0:50:b6:f9:80:8c:b0:49:5d:a6:ab:44:63:
  ...
prime1:
  00:e7:36:d7:37:13:f4:9a:5f:36:60:e7:ac:85:a0:
  ...
prime2:
  00:cc:34:af:4f:2c:14:14:d9:b0:d7:52:a2:58:c8:
  ...
exponent1:
  65:7f:34:70:70:29:23:0f:02:ce:fd:44:45:90:6c:
  ...
exponent2:
  00:96:14:6a:7b:bf:a3:8a:a7:6b:86:f5:1e:88:2d:
  ...
coefficient:
  39:49:5e:a9:d9:4b:8e:a2:24:58:12:0d:3c:ea:c2:
  ...
```

Ce contenu reflète la représentation d'un bi-clé RSA selon la structure ASN.1 suivante, définie par PKCS#1 (ou, de manière équivalente à PKCS#1 version 1.5, dans la RFC 2313) :

```
RSAPrivateKey ::= SEQUENCE {
    version          Version,
    modulus           INTEGER,  -- n
    publicExponent    INTEGER,  -- e
    privateExponent   INTEGER,  -- d
    prime1            INTEGER,  -- p
    prime2            INTEGER,  -- q
    exponent1         INTEGER,  -- d mod (p-1)
    exponent2         INTEGER,  -- d mod (q-1)
    coefficient        INTEGER,  -- (inverse of q) mod p
    otherPrimeInfos    OtherPrimeInfos OPTIONAL
}
```


Contrairement à ce que son nom indique, la structure `RSAPrivateKey` contient le bi-clé RSA et non simplement la clé privée, laquelle peut d'ailleurs être représentée mathématiquement par le couple (n, d) ou, de manière équivalente et permettant d'appliquer plus rapidement les algorithmes de chiffrement et déchiffrement RSA, par le quintuplet $(p, q, d \bmod (p-1), d \bmod (q-1), (\text{inverse of } q) \bmod p)$.

La clé publique correspond au couple (n, e) . Elle peut être extraite du bi-clé RSA par la commande suivante :

```
$ openssl rsa -in rsakey.pem -pubout
writing RSA key
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuG9I+ZnxmXHGb4
Gm7Iiv050f0HCI2Xsr4eJ5W9HoaI/w5hcrc9N6W4GTXHw65Xon5dRvODgw
qV0e+pnEkxWGzldJAGAAOQIDSjWOB/kP8tNHNC5r+VE54G5j88mXh09LNe
9RiKhNTBsRug8yn8XC62zSb0ZXUcN4nGuX45YzaKoYgZECYrQdj+5500N0
90gt0Xk1W2JUODKjpSwxf6XSmEA2k6bDoWZfP/b87UVKBkWX9e9Bfmi7f9
BM01nysBM0YAer57UoLbI6rSP/s3ZA9lX5jzgG8Q2bS/dS9Bp+W9Fzq4NF
OQIDAQAB
-----END PUBLIC KEY-----
```

Ajouter l'option `-out rsakey.pub.pem` pour écrire la clé publique dans un fichier, et analyser celui-ci.

```
$ openssl rsa -in rsakey.pub.pem -noout -text -pubin
Public-Key: (2048 bit)
Modulus:
    00:b8:6f:48:f9:99:f1:99:71:c6:6f:80:64:d1:ca:
...
    8d:d1
Exponent: 65537 (0x10001)
```

Ce contenu représente bien le couple (n, e) , conformément à la représentation ASN.1 normalisée suivante :

```
RSAPublicKey ::= SEQUENCE {
    modulus          INTEGER,  -- n
    publicExponent   INTEGER  -- e
}
```

L'option `-pubout` d'`openssl rsa` produit une clé publique conforme à la structure `SubjectPublicKeyInfo` de X.509, qui encapsule le type de clé et la valeur de la clé (elle-même de type `RSAPublicKey`), et qui est notamment utilisée pour représenter une clé publique dans un certificat électronique. Pour obtenir une clé publique de type `RSAPublicKey`, l'option `-pubout` doit être remplacée par la commande non documentée `-RSAPublicKey_out`.

La structure ASN.1 brute de la clé publique (ou privée) peut être visualisée en convertissant la clé au format DER et en utilisant la commande `openssl asn1parse`. Par exemple :

```
$ openssl rsa -in rsakey.pem -outform DER | openssl asn1parse -inform DER -i
writing RSA key
    0:d=0  hl=4  l=1187 cons: SEQUENCE
    4:d=1  hl=2  l=   1 prim:  INTEGER           :00
    7:d=1  hl=4  l= 257 prim:  INTEGER           :B86F...8DD1
```

3. Cryptographie asymétrique

```
268:d=1  hl=2 l= 3 prim:  INTEGER          :010001
273:d=1  hl=4 l= 256 prim:  INTEGER          :4B10...B4E1
533:d=1  hl=3 l= 129 prim:  INTEGER          :E736...C515
665:d=1  hl=3 l= 129 prim:  INTEGER          :CC34...CCCD
797:d=1  hl=3 l= 128 prim:  INTEGER          :657F...6C95
928:d=1  hl=3 l= 129 prim:  INTEGER          :9614...3649
1060:d=1 hl=3 l= 128 prim:  INTEGER          :3949...2DBF
```

Exercice — Utiliser les options `-offset` et `-out` de la commande `openssl asn1parse` pour extraire la clé publique de type `RSAPublicKey` à partir du BIT STRING de la structure `SubjectPublicKeyInfo` (au besoin se reporter à la section 4.1 de la RFC 5280 pour la définition de la structure `SubjectPublicKeyInfo`).

3.2. Chiffrement asymétrique

Chiffrement par une clé publique

Chiffrer un fichier avec la clé publique RSA précédente.

```
$ openssl pkeyutl -encrypt -pubin -inkey rsakey.pub.pem -in data.txt \
-out data.rsa
```

Par défaut, OpenSSL utilise pour le chiffrement l'algorithme de remplissage (*padding*) défini dans le mode de codage EME-PKCS1-v1_5 de PKCS#1 version 1.5 ou RFC 2313 (par opposition à EME-OAEP introduit dans PKCS#1 version 2.0 ou RFC 2437, plus sécurisé car non déterministe, mais beaucoup moins courant dans le domaine de la confiance électronique en pratique), correspondant à l'option implicite `-pkeyopt rsa_padding_mode:pkcs1` (versus `-pkeyopt rsa_padding_mode:oaep` — la coquille `oaep` pour OAEP étant volontaire).

Déchiffrement par une clé privée

Déchiffrer le fichier obtenu avec la clé privée RSA.

```
$ openssl pkeyutl -decrypt -inkey rsakey.pem -in data.rsa
Loading 'screen' into random state - done
texte en clair
```

Le *padding* défini par EME-PKCS1-v1_5 a la forme `0x02 0x00 données aléatoires 0x00 données`, avec autant de données aléatoires non nulles que nécessaire pour obtenir une taille de bloc finale égale à la longueur de la clé RSA.

Déchiffrer le fichier en utilisant le *padding* `none` pour visualiser le *padding* réalisé précédemment par EME-PKCS1-v1_5 :

```
$ openssl pkeyutl -decrypt -in data.rsa -inkey rsakey.pem \
-pkeyopt rsa_padding_mode:none -hexdump
0000 - 00 02 ac 39 c7 54 7f 7b-50 65 61 46 0c 69 61 c8    ...9.T.{PeaF.ia.
```

... octets aléatoires ...

00f0 - 42 00 74 65 78 74 65 20-65 6e 20 63 6c 61 69 72 B.texte en clair

Observer que la longueur totale du bloc est de 256 octets, soit 2048 bits, c'est-à-dire la taille de la clé RSA utilisée.

Performances des algorithmes de chiffrement

En pratique (cf. chapitre sur la confidentialité plus loin), les données sont chiffrées à l'aide d'une clé symétrique, puis la clé symétrique est chiffrée avec une clé publique, et ce pour des raisons de performances, les algorithmes symétriques étant beaucoup plus rapides que les algorithmes asymétriques, comme le montre par exemple la comparaison des performances entre RSA et AES-256-CBC pour chiffrer des blocs de 1024 bits :

```
$ openssl speed rsa aes-256-cbc
```

```
...
Doing aes-256 cbc for 3s on 1024 size blocks: 93499 aes-256 cbc's in 3.72s
...
Doing 1024 bit private rsa's for 10s: 2087 1024 bit private RSA's in 12.80s
Doing 1024 bit public rsa's for 10s: 41888 1024 bit public RSA's in 12.95s
...
OpenSSL 1.0.1a 19 Apr 2012
built on: Fri Apr 20 06:01:45 2012
options:bn(64,32) rc4(8x,mmx) des(idx,cisc,2,long) aes(partial) idea(int) blowfi
sh(idx)
...
The 'numbers' are in 1000s of bytes per second processed.
type           16 bytes      64 bytes      256 bytes      1024 bytes      8192 bytes
aes-256 cbc      9200.68k      9426.32k      9677.20k      25746.01k      23797.85k
              sign    verify    sign/s    verify/s
rsa  512 bits 0.001050s 0.000102s   952.4    9828.7
rsa 1024 bits 0.006132s 0.000309s   163.1    3233.8
rsa 2048 bits 0.037914s 0.001097s    26.4     911.7
rsa 4096 bits 0.259566s 0.004098s     3.9     244.0
```

Les chiffres varient bien entendu d'une machine à l'autre, seuls les ordres de grandeur sont pris en compte.

Lors de ce test, AES-256-CBC a chiffré des blocs de 1024 bits à une vitesse moyenne de 25 746,01 kB par seconde (25,1 MB/s), tandis que RSA a chiffré par clé publique des blocs de 1024 bits à la vitesse moyenne de 3 233,8 blocs de 1024 bits par seconde, soit 3 233,8 kB par seconde (3,2 MB/s), soit un rapport de presque 8 entre les deux algorithmes. Noter également que doubler la taille de la clé RSA engendre un facteur de pénalité d'environ 3,5 sur le temps de chiffrement.

3.3. Signature numérique

Signature par une clé privée

Signer un fichier avec la clé privée RSA précédente, en chiffrant avec la clé privée l'empreinte SHA-256 du fichier choisi.

```
$ openssl sha256 -binary data.txt | openssl pkeyutl -sign -inkey rsakey.pem
```

Par analogie avec le chiffrement, par défaut, OpenSSL utilise pour la signature RSA l'algorithme de *padding* (remplissage) défini dans le mode de codage EMSA-PKCS1-v1_5 de PKCS#1 ou RFC 3447, correspondant à l'option implicite `-pkeyopt rsa_padding_mode:pkcs1`. La version sécurisée de l'algorithme de *padding* est EMSA-PSS (`-pkeyopt rsa_padding_mode:pss`), mais son utilisation est actuellement rare dans le domaine de la confiance électronique.

Vérification par une clé publique

Le déchiffrement de la signature à l'aide de la clé publique produit l'empreinte du fichier, comme attendu, en injectant le résultat de la commande ci-dessus dans `openssl pkeyutl -verifyrecover`:

```
... | openssl pkeyutl -verifyrecover -pubin -inkey rsakey.pub.pem
Loading 'screen' into random state -Loading 'screen' into random state - done
done
oUW%mlQ¶L`O_ï@|kVVÈ|À|â↑t$
$ openssl sha256 -binary data.txt
oUW%mlQ¶L`O_ï@|kVVÈ|À|â↑t$
```

La commande `openssl` est parfois un peu capricieuse quant à l'ordre des arguments. Par exemple, dans la ligne de commande précédente, placer l'option `-pubin` (qui indique que la clé référencée par l'option `-inkey` est une clé publique et non privée) *après* l'option `-inkey` provoque l'erreur `unable to load Private Key` survient. En cas d'erreur inattendue, il vaut parfois la peine d'essayer d'utiliser l'ordre des options tel qu'il figure dans la documentation d'OpenSSL.

Pour vérifier simplement une signature, utiliser l'option `-verify` d'`openssl pkeyutl`, avec les options `-in` pour définir le fichier d'empreinte des données et `-sigfile` pour le fichier contenant la signature.

```
$ openssl sha256 -binary data.txt > data.sha256
$ openssl pkeyutl -sign -in data.sha256 -inkey rsakey.pem -out data.sha256.sig
$ openssl pkeyutl -verify -pubin -inkey rsakey.pub.pem -in data.sha256 \
  -sigfile data.sha256.sig
Signature Verified Successfully
```

Variante de la dernière commande pour éviter le stockage de l'empreinte dans un fichier intermédiaire :

```
$ openssl sha256 -binary data.txt | openssl pkeyutl -verify -pubin \
  -inkey rsakey.pub.pem -sigfile data.sha256.sig
Signature Verified Successfully
```

Analyse d'une signature

En mode EMSA-PKCS1-v1_5, l'empreinte chiffrée est incluse dans une structure de données DigestInfo, dont la définition ASN.1 est :

```
DigestInfo ::= SEQUENCE {
    digestAlgorithm AlgorithmIdentifier,
    digest OCTET STRING
}
```

Pour inclure l'empreinte dans une structure DigestInfo avant *padding* et chiffrement, préciser l'algorithme de hachage devant faire l'objet du premier élément de la structure, via l'option `-pkeyopt digest::`

```
$ openssl sha256 -binary data.txt | openssl pkeyutl -sign -inkey rsakey.pem \
  -pkeyopt rsa_padding_mode:pkcs1 -pkeyopt digest:sha256
```

L'option `-pkeyopt rsa_padding_mode:pkcs1` a été volontairement explicitée, mais elle peut être omise.

Le déchiffrement et le décodage ASN.1 de cette signature peuvent être obtenus en injectant le résultat de la commande ci-dessus dans la commande suivante :

```
... | openssl pkeyutl -verifyrecover -pubin -inkey rsakey.pub.pem -asn1parse
Loading 'screen' into random state -Loading 'screen' into random state - done
done
 0:d=0 hl=2 l= 49 cons: SEQUENCE
 2:d=1 hl=2 l= 13 cons: SEQUENCE
 4:d=2 hl=2 l=  9 prim: OBJECT                :sha256
15:d=2 hl=2 l=  0 prim: NULL
17:d=1 hl=2 l= 32 prim: OCTET STRING
      0000 - 89 bd 92 28 6d 6c 80 14-c0 60 30 b2 5f 8b 40 cc  ... (ml...`0._.@.
      0010 - 1d 56 56 d4 b3 b7 b4 83-18 74 f5 0d 6f 55 57 f3  .VV.....t..oUW.
```

L'OCTET STRING final contient bien l'empreinte du fichier d'entrée (en hexadécimal : 89bd92286d6c8014c06030b25f8b40cc1d5656d4b3b7b4831874f50d6f5557f3).

Exercice — Vérifier, par exemple à l'aide de la commande `od -tx1`, que le résultat obtenu est conforme à la note 1 de la section 9.2 de la RFC 3447.

Padding d'une signature

Le *padding* défini par EMSA-PKCS1-v1_5 a pour forme 0x01 0x00 0xff ... 0xff *données* , où les 0xff sont répétés autant de fois que nécessaire pour obtenir une taille de bloc finale égale à la longueur de la clé RSA.

Déchiffrer la signature en utilisant le *padding* none pour visualiser le *padding* réalisé précédemment par EMSA-PKCS1-v1_5 :

```
$ openssl sha256 -binary data.txt | openssl pkeyutl -sign -pubin \
-inkey rsakey.pub.pem | openssl pkeyutl -verifyrecover -inkey rsakey.pem \
-pkeyopt rsa_padding_mode:none -hexdump
```

Loading 'screen' into random state -

Loading 'screen' into random state - done

done

```
0000 - 00 01 ff ff ff ff ff ff-ff ff ff ff ff ff ff ff .....
```

... octets 0xff ...

```
00d0 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff 00      .....
```

```
00e0 - 89 bd 92 28 6d 6c 80 14-c0 60 30 b2 5f 8b 40 cc    ... (ml...`0._.@.
```

```
00f0 - 1d 56 56 d4 b3 b7 b4 83-18 74 f5 0d 6f 55 57 f3 .VV.....t..oUW.
```

Observer que la longueur totale du bloc est de 256 octets, soit 2048 bits, c'est-à-dire la taille de la clé RSA de signature.

3.4. Construction d'une clé publique

Pour construire des structures de données associées à des modules ASN.1 non supportés nativement par OpenSSL (via l'outil en ligne de commande `openssl` ou la bibliothèque `libcrypto`), à l'exemple du format de signature électronique avancée CAdES, il est possible de recourir à la commande `openssl asn1parse`. Celle-ci permet de constituer une structure ASN.1 à partir d'une description fournie soit en ligne de commande pour les structures élémentaires (via l'option `-genstr`), soit dans un fichier de configuration pour les structures imbriquées (via l'option `-genconf`).

Les principales alternatives sont d'écrire du code invoquant les fonctions ASN.1 de la bibliothèque lib-crypto d'OpenSSL, et d'adapter et compiler les modules ASN.1 issus des normes souhaitées à l'aide d'un compilateur ASN.1 (tel que `asn1c`, présenté en annexe).

Une méthode (un peu artisanale mais pratique) est proposée dans l'annexe B.2 pour générer un fichier de configuration ASN.1 interprétable par OpenSSL à partir d'un fichier codé en DER.

Dans chacun des chapitres impliquant une structure de données intervenant dans le format CAdES, il est décrit comment construire cette structure à l'aide de la commande `openssl asn1parse`, à commencer par ce chapitre pour la structure `SubjectPublicKeyInfo`, utilisée pour référencer la clé publique dans le champ `subjectPublicKeyInfo` d'un certificat.

La structure à générer a la syntaxe ASN.1 suivante :

```
SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm             AlgorithmIdentifier,
```

```

    subjectPublicKey      BIT STRING
}

```

Création du champ *algorithm*

Créer un fichier `ee-SubjectPublicKeyInfo.asn.cnf`, avec le contenu suivant, à enrichir au fur et à mesure :

```
asn1=SEQUENCE:subjectPublicKeyInfo
```

```

[subjectPublicKeyInfo]
algorithm = ...
subjectPublicKey = ...

```

Dans cette structure, le champ `algorithm` est de type `AlgorithmIdentifier`, dont la syntaxe est la suivante :

```

AlgorithmIdentifier ::= SEQUENCE {
    algorithm          OBJECT IDENTIFIER,
    parameters        ANY DEFINED BY algorithm OPTIONAL
}

```

La norme X.509 emploie la syntaxe objet d'ASN.1 (cf. X.681) pour contraindre le rapport entre les champs `algorithm` et `parameters` :

```

AlgorithmIdentifier{ALGORITHM:SupportedAlgorithms} ::= SEQUENCE {
    algorithm ALGORITHM.&id ({SupportedAlgorithms}),
    parameters ALGORITHM.&Type ({SupportedAlgorithms}{ @algorithm}) OPTIONAL
}

```

où la définition de la classe `ALGORITHM` est :

```

ALGORITHM ::= CLASS {
    &Type          OPTIONAL,
    &id            OBJECT IDENTIFIER  UNIQUE
}
WITH SYNTAX {
    [&Type]
    IDENTIFIED BY  &id
}

```

Avec cette syntaxe, l'algorithme RSA aurait une définition comparable à celle-ci :

```

pkcs-1 OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) 1
}

rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1 }

rsa ALGORITHM ::= {
    NULL
    IDENTIFIED BY rsaEncryption
}

```

où `NULL` indique que si le champ `parameters` est présent (ce qui est imposé par PKCS#1, même si son éventuelle absence est acceptée par OpenSSL), il doit être de type `NULL`.

3. Cryptographie asymétrique

La liste des algorithmes supportés (dont la valeur est { ... } dans X.509) serait alors définie ainsi :

```
SupportedAlgorithms ALGORITHM ::= { rsa }
```

La SEQUENCE de la structure `AlgorithmIdentifier` correspondant à l'algorithme RSA inclut les champs suivants :

- `algorithm`, qui contient l'OID dont l'alias court (connu d'OpenSSL) est `rsaEncryption`.
- `parameters`, qui contient `NULL`.

Cela se transpose dans le fichier `ee-SubjectPublicKeyInfo.asn.cnf` par la section suivante :

```
[rsa_AlgorithmIdentifier]
algorithm = OID:rsaEncryption
parameters = NULL
```

Les noms de section sont arbitraires : ici le nom `rsa_AlgorithmIdentifier` a été choisi pour expliciter la structure définie.

Mettre à jour la définition du champ `algorithm` de la structure `SubjectPublicKeyInfo`

```
[subjectPublicKeyInfo]
algorithm = SEQUENCE:rsa_AlgorithmIdentifier
...
```

Création du champ `subjectPublicKey`

Le cas du champ `subjectPublicKey` de la structure `SubjectPublicKeyInfo` est plus subtil. Il s'agit d'une BIT STRING dont la valeur est le codage DER de la structure `RSAPublicKey` représentant la clé publique. Il est dit de cette BIT STRING qu'elle encapsule ou enrobe (*wrap* en anglais) la structure `RSAPublicKey`.

Le mécanisme d'enrobage permet d'inclure dans un champ (habituellement une BIT STRING ou une OCTET STRING) une structure dont le type n'est pas connu lors de l'élaboration du module. Par exemple, dans le cas des `SubjectPublicKeyInfo`, une BIT STRING peut stocker tout type de clé publique (RSA, DSA etc.), et l'apparition d'un nouveau type de clé publique ne nécessite pas de faire évoluer le module.

Ce mécanisme de champ générique impose qu'un autre élément permette de déterminer le type du champ. En l'occurrence, la valeur du champ `algorithm` détermine le type de structure enrobé par la BIT STRING de `subjectPublicKey`, par exemple une `RSAPublicKey` si `algorithm` contient l'OID `rsaEncryption`.

La structure à enrober dans le champ `subjectPublicKey` dans le cas d'une clé publique RSA est de type `RSAPublicKey` :

```
RSAPublicKey ::= SEQUENCE {
    modulus INTEGER,
    publicExponent INTEGER
}
```


Créer le fichier `ee-RSAPublicKey.asn.cnf` suivant (les valeurs sont celles de la clé publique créée dans le fichier `ee-key.pem`, le caractère littéral « \ » est employé pour découper une valeur sur plusieurs lignes) :

```
asn1=SEQUENCE:RSAPublicKey
```

```
[RSAPublicKey]
n = INTEGER:0xB86F48F999F19971C66F8064D1CA0C1A6EC88AF3B939FD07088D97B2BE1E2795B\
D1E8688FF0E6172B73D37A5B81935C7C3AE57A27E5D46F38383089C4410EDA95D1EFA99C4931586\
CE57490060003902034A358E07F90FF2D347342E6BF95139E06E63F3C997874F4B35E8DA3A87F4F\
5188A8674C1B11BA0F329FC5C2EB6CD26F465751C3789C6B97E3963368AA1881910262BA9D8FEE7\
9D3434E722A2302DF4E82D3979355B6254D032A3A52C317FA5D298403693A6C3A1665F3FF6FCED4\
54A064597F5EF417E68BB7FD2D3892B4F04CD359F2B0133460012BE7B5282DB23AAD23FFB37640F\
655F98F3806F10D9B4BF752F41A7E5BD173AB8345B81FD5C8DD1
e = INTEGER:0x010001
```

Générer le codage DER de la structure ASN.1 correspondante.

```
$ openssl asn1parse -genconf ee-RSAPublicKey.asn.cnf -i -out ee-RSAPublicKey.der
0:d=0 hl=4 l= 266 cons: SEQUENCE
4:d=1 hl=4 l= 257 prim:  INTEGER          :B86F48F999F19971C66F8064D1CA0C1
...
012BE7B5282DB23AAD23FFB37640F655F98F3806F10D9B4BF752F41A7E5BD173AB8345B81FD5C8DD
1
265:d=1 hl=2 l=   3 prim:  INTEGER          :010001
```

Le résultat affiché par cette opération doit être strictement identique à celui de la commande :

```
$ openssl rsa -in ee-key.pem -outform DER -RSAPublicKey_out
| openssl asn1parse -inform DER -i
```

La représentation hexadécimale de ce codage DER peut être déterminée ainsi :

```
$ od -tx1 -An ee-RSAPublicKey.der | tr -d "\n\r "
3082010a0282010100b86f48f999f19971c66f8064d1ca0c1a6ec88af3b939fd07088d97b2be1e27
...
5f98f3806f10d9b4bf752f41a7e5bd173ab8345b81fd5c8dd10203010001
```

La BIT STRING du champ `subjectPublicKey` de la structure `SubjectPublicKeyInfo` à générer doit prendre la valeur hexadécimale ci-dessus.

Voici le fichier `ee-SubjectPublicKeyInfo.asn.cnf` ainsi obtenu.

```
asn1=SEQUENCE:subjectPublicKeyInfo
```

```
[subjectPublicKeyInfo]
algorithm = SEQUENCE:rsa_AlgorithmIdentifier
subjectPublicKey = FORMAT:HEX,BITSTRING:3082010a0282010100b86f48f999f19971c66f8\
064d1ca0c1a6ec88af3b939fd07088d97b2be1e2795bd1e8688ff0e6172b73d37a5b81935c7c3ae\
57a27e5d46f38383089c4410eda95d1efa99c4931586ce57490060003902034a358e07f90ff2d34\
7342e6bf95139e06e63f3c997874f4b35e8da3a87f4f5188a8674c1b11ba0f329fc5c2eb6cd26f4\
65751c3789c6b97e3963368aa1881910262ba9d8fee79d3434e722a2302df4e82d3979355b6254d\
032a3a52c317fa5d298403693a6c3a1665f3ff6fced454a064597f5ef417e68bb7fd2d3892b4f04\
cd359f2b0133460012be7b5282db23aad23ffb37640f655f98f3806f10d9b4bf752f41a7e5bd173\
```

3. Cryptographie asymétrique

ab8345b81fd5c8dd10203010001

```
[rsa_AlgorithmIdentifier]
algorithm = OID:rsaEncryption
parameters = NULL
```

La ligne de commande UNIX/Linux suivante est proposée pour convertir un fichier binaire en sa représentation hexadécimale, avec découpage en lignes de 64 caractères et ajout du caractère « \ » en fin de ligne, le résultat pouvant ainsi être inclus directement dans un champ tel que `subjectPublicKey` ci-dessus (en prenant le soin de supprimer le dernier caractère « \ »):

```
$ od -tx1 -An ee-SubjectPublicKeyInfo.der | tr -d " \r\n" | sed 's/$/\\/'
30820122300d06092a864886f70d01010105000382010f003082010a02820101\
...
aad23ffb37640f655f98f3806f10d9b4bf752f41a7e5bd173ab8345b81fd5c8d\
d10203010001\
```

Sous Windows, la commande `sed` à utiliser est `sed 's/$/\\/'`, en remplaçant les apostrophes délimiteurs par des guillemets.

Générer la structure ASN.1 `SubjectPublicKeyInfo`.

```
$ openssl asn1parse -genconf ee-SubjectPublicKeyInfo.asn.cnf -i \
-out ee-SubjectPublicKeyInfo.der
0:d=0 hl=4 l= 290 cons: SEQUENCE
4:d=1 hl=2 l= 13 cons: SEQUENCE
6:d=2 hl=2 l= 9 prim: OBJECT :rsaEncryption
17:d=2 hl=2 l= 0 prim: NULL
19:d=1 hl=4 l= 271 prim: BIT STRING
```

Le fichier `ee-SubjectPublicKeyInfo.der` généré doit être identique au fichier `ee-key.pub.der` résultant de la commande suivante :

```
$ openssl rsa -in ee-key.pem -outform DER -out ee-key.der
```

Dans les deux cas, l'utilisation de la commande `openssl asn1parse` avec `-strparse 24`, où `24 = 19` octets (début de la `BIT STRING`) + `4` octets (longueur de l'en-tête) + `1` (premier octet de la `BIT STRING` représentant les bits inutilisés), permet d'extraire la structure `RSAPublicKey` encapsulée.

Réaliser manuellement l'enrobage d'une structure `SubjectPublicKeyInfo` dans une `BIT STRING` permet de mieux comprendre l'articulation entre les structures, mais est assez peu pratique. Heureusement, OpenSSL propose un mécanisme pour enrober une structure dans une `BIT STRING`, via le mot clé `BITWRAP`. En utilisant le type `BITWRAP`, le fichier `ee-SubjectPublicKeyInfo.asn.cnf` devient :

```
asn1=SEQUENCE:subjectPublicKeyInfo
```

```
[subjectPublicKeyInfo]
algorithm = SEQUENCE:rsa_AlgorithmIdentifier
subjectPublicKey = BITWRAP,SEQUENCE:subjectPublicKey
```

```
[rsa_AlgorithmIdentifier]
algorithm = OID:rsaEncryption
parameters = NULL
```

```
[subjectPublicKey]
n = INTEGER:0xB86F48F999F19971C66F8064D1CA0C1A6EC88AF3B939FD07088D97B2BE1E2795B\
D1E8688FF0E6172B73D37A5B81935C7C3AE57A27E5D46F38383089C4410EDA95D1EFA99C4931586\
CE57490060003902034A358E07F90FF2D347342E6BF95139E06E63F3C997874F4B35E8DA3A87F4F\
5188A8674C1B11BA0F329FC5C2EB6CD26F465751C3789C6B97E3963368AA1881910262BA9D8FEE7\
9D3434E722A2302DF4E82D3979355B6254D032A3A52C317FA5D298403693A6C3A1665F3FF6FCED4\
54A064597F5EF417E68BB7FD2D3892B4F04CD359F2B0133460012BE7B5282DB23AAD23FFB37640F\
655F98F3806F10D9B4BF752F41A7E5BD173AB8345B81FD5C8DD1
e = INTEGER:0x010001
```

Ce fichier final est plus lisible et plus facile à modifier que de passer par le codage DER intermédiaire d'une structure RSAPublicKey.

En cas d'oubli du mot clé BITWRAP, la structure générée (représentée ci-dessous) sera différente de celle souhaitée et, étant incompatible avec la définition donnée dans le module ASN.1, ne pourra pas être traitée par les applications utilisatrices.

```
0:d=0 hl=4 l= 285 cons: SEQUENCE
4:d=1 hl=2 l= 13 cons: SEQUENCE
6:d=2 hl=2 l= 9 prim: OBJECT :rsaEncryption
17:d=2 hl=2 l= 0 prim: NULL
19:d=1 hl=4 l= 266 cons: SEQUENCE
23:d=2 hl=4 l= 257 prim: INTEGER :B86F48F999F19971C66F8064D1CA0C
...
D1
284:d=2 hl=2 l= 3 prim: INTEGER :010001
```

Utiliser `dumpasn1` (cf. annexe B.3) pour analyser le fichier `ee-SubjectPublicKeyInfo.der`, en notant que la BIT STRING enrobante est détricotée.

```
$ dumpasn1 -a ee-SubjectPublicKeyInfo.der
0 290: SEQUENCE {
4 13: SEQUENCE {
6 9: OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
17 0: NULL
: }
19 271: BIT STRING, encapsulates {
24 266: SEQUENCE {
28 257: INTEGER
: 00 B8 6F 48 F9 99 F1 99 71 C6 6F 80 64 D1 CA 0C
...
: D1
289 3: INTEGER 65537
: }
: }
: }
```

3. Cryptographie asymétrique

Chapitre 4 — Certificats

Ce chapitre est consacré à la génération pas à pas de certificats. Pour permettre de vérifier le profil des certificats à l'aide d'un outil externe, le profil retenu est celui défini en France par l'annexe A14 du référentiel général de sécurité⁶ (RGS), dans un document intitulé *Profils de Certificats / LCR / OSCP et Algorithmes Cryptographiques*.

Il existait un outil, nommé outil Validation Profils RGS⁷, permettant de valider la conformité des certificats, listes de certificats révoqués, jetons OSCP et jetons d'horodatage par rapport aux profils définis par le RGS. Cet outil a disparu pendant le mois de juin 2012, et est mentionné au cas où il réapparaîtrait. Les éléments générés dans ce document ont tous été validés par cet outil.

4.1. Génération des bi-clés

Générer un bi-clé RSA de 2048 bits pour l'autorité de certification.

```
$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048
.....+++
.....
.....+++
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBCwggSjAgEAAoIBAQCcLbpiB/OVoLSz
RzENFJEwdE8UieC1UPNWxOSATXbIg+v9aSnpwikD6laI/DiN0jgiLK76H0LHqg/3
xXiaJyh4PRkzHG01muVj/6uE3G4CGR6gUWQ6c12qWmJ08GyLhPyEbiZ9FGysQQE+
Z3RLkg90nW2KRWnfrZxxtCXHb9zrMA/qlCB9rfdahs8ZqLwAiXueg4Hsq7e0IV4
6012U02jppXEkjC6+LL9H7c+uh0w85KgkHc9SNx896KQM/nX+uBA0QuSAId2uiH
2g8HrmSWecL/4AxIiIzqzX+H2FRIPtPtQ02/Xto+EQ1Wy/F6ML27NkoHL87VbrOC/
4+q/P4XBAgMBAAECggEAF3+AwcLUtQFK8pIuXQB1G2eExw8u9gEeYGS111yaS8c
KIvPBTzENCdPvUfC0xOwYNmOY6nDNMFhXYbcfZy1w0JaFm+uzm/EB4PtQ7gC5ERx
zNkCdsDbr9Hp/JHx2JXQ88WtVu9ff4EiqScUSvUbGGHQKo0NgqeMH43Ndq9kp9eF
JVc/TCX6KcpmopK0hKIRdOPow2MK9jCGkidxV700f641pLagQvW9o5siIbkIbidh
u82p0EAu5gyreLkDUpU6sEmSig70x9LmSgG7UyIhqD3bvVYLYw68/R4UZd3cwKRX
yceNyZKWNtmyurj1RLka1z3l0y2zqBeqz0pbKtgAQKBgQDMegYN5PtJD9lDwnfR
s4/Qj7Z9w5Unp6QMhVionE7NkIOKZ3E42PFGTbK4r99x/D26Sc/fKf2bHJbZS0bn
TMpHUSvnuvYQFNlsV1PWfg4VbCrRw9D26LG8ESdje1W3iXYcau/45J9bEvP1/vyI
WvWd/ITUxjN7WDyQWja14X2FpQKBgQDDiDDCfyGN0/SXmb1pnUCMMe/WxPje8RDN
lBeyDVvYpBTFKQDSbKONOnY8xKS5HLdCmR9817Sqpm9UJ/7oc3NeYKXYF9BGE08w
C5UfdA3QsdoSUS7TseTunn/yhmKJBq4m9AaJRPZCH49Ipk2FQIhQg9YPUdwd6klV
tDJPNZ3c7QKBgCWHd61lJS1Xm0xdQjheedQs5a9XQfyxK86QeqEXbu8TYtD4AzOn
vUBl6jMzNBM7dCjIezo9/qmFKbpfQuGYmldUcnZp9tdkvDLCJETthbAw1fsBsUxf
kJ8v1scPIMQ6mk7E4Z4Qd57Db7rqPXBZG/+vTMGwTqqImzTzGayAIGVxAoGBAKFk
Mkr3wGxefM79GPip0XNmbH6rSnhKJgJpsD1JBXyFwpcSRx1ollaIVvWFi+k3KJSR
6wmSyg7pHY8rDB413Q4TXBBHZ6PPoFcZ2FaD5jtR8ZuY4rvdZAcJULaP8ZkEqI6C
```

6. <http://www.modernisation.gouv.fr/rgs-securite>

7. <http://bao.dgme.fr>

4. Certificats

```
cTqwB0sY3Z2rluTb5SgACZnPiY4vqaRR/gyfRtPpAoGAfqPmbd8xLlKVaN7Rjlrn
IoOrDYaL27o2fhwmsiKtiMq/sbUQWstr+S34u7/BhnNk4gBFj8CnimK4QYdrpy8d
OprHbQU1Tvh400LPsMS5ZCfWzuhhLj09TABFW7Fbj4y+2gI3g0GcWd+BoY0x/NKL
rPewPJU3mc8wj2Y9ue8e1Lg=
-----END PRIVATE KEY-----
```

Copier la clé ci-dessus dans un fichier nommé `ca-key.pem` (ca pour *certificate/certification authority* ou autorité de certification), ou utiliser la commande ci-dessus avec l'option `-out ca-key.pem` pour poursuivre avec une autre clé.

Pour obtenir les mêmes valeurs que celles de ce document, renommer la clé `rsa-key.pem` générée au début en `ee-key.pem` (ee pour *end entity* ou entité finale). Sinon générer un bi-clé RSA de 2048 bits dans un fichier `ee-key.pem`.

4.2. Génération des requêtes de signature de certificat

Génération avec objet passé en ligne de commande

Les options de la commande `openssl req -new` permettent théoriquement de générer une CSR (*certificate signature request*, ou requête de signature de certificat) à elles seules., mais du fait d'un *bug* (connu) dans OpenSSL, un fichier de configuration supplémentaire — même fonctionnellement vide — reste nécessaire. Créer le fichier `req-empty.cnf` avec le contenu suivant.

```
[ req ]
distinguished_name = req_distinguished_name

[ req_distinguished_name ]
```

Générer la CSR, dans le fichier `ca-req.pem`.

```
$ openssl req -new -key ca-key.pem \
  -subj "/C=FR/O=Mon Entreprise/OU=0002 123456789/OU=OpenSSL Root CA" -sha256 \
  -config req-empty.cnf -out ca-req.pem
```

L'ordre des éléments composant l'objet (ou *subject* pour la terminologie anglaise) fait l'objet d'une section spécifique en annexe.

Analyse d'une requête de signature de certificat

Visualiser le contenu de la requête.

```
$ openssl req -in ca-req.pem -noout -text
Certificate Request:
  Data:
    Version: 0 (0x0)
    Subject: C=FR, O=Mon Entreprise, OU=0002 123456789, OU=OpenSSL Root CA
    Subject Public Key Info:
```

```

Public Key Algorithm: rsaEncryption
  Public-Key: (2048 bit)
  Modulus:
    00:9c:2d:ba:62:07:f3:95:a0:b4:b3:47:31:0d:14:
    ...
    85:c1
  Exponent: 65537 (0x10001)
Attributes:
  a0:00
Signature Algorithm: sha256WithRSAEncryption
  8b:0f:a7:83:95:25:01:b4:8c:67:9e:27:43:b3:d8:90:95:a8:
  ...
  ff:54:53:9a

```

La CSR reflète la structure ASN.1 CertificationRequest :

```

CertificationRequest ::= SEQUENCE {
  certificationRequestInfo CertificationRequestInfo,
  signatureAlgorithm AlgorithmIdentifier{{ SignatureAlgorithms }},
  signature             BIT STRING
}

```

où CertificationRequestInfo a la structure suivante :

```

CertificationRequestInfo ::= SEQUENCE {
  version          INTEGER { v1(0) } (v1,...),
  subject          Name,
  subjectPKInfo    SubjectPublicKeyInfo{{ PKInfoAlgorithms }},
  attributes       [0] Attributes{{ CRIAttributes }}
}

```

Le champ attribut contient les octets 0xa0 0x00, correspondant au codage DER d'un SET OF ASN.1 vide, l'élément attributes étant obligatoire même si la CSR ne contient aucun attribut, comme expliqué dans la documentation de la commande req d'OpenSSL.

Génération avec objet renseigné de manière interactive

Pour la génération de la CSR de l'entité finale, une autre méthode est proposée, permettant de renseigner les informations constituant la CSR de manière interactive. Créer le fichier de configuration ee-req-dynamic.cnf.utf8 suivant, et l'enregistrer avec le codage UTF-8 sans BOM (*byte order mark*).

La problématique du codage des caractères, qui prend ici tout son sens, est abordée en annexe.

```

[ req ]
distinguished_name = req_distinguished_name
string_mask = MASK:0x2002

[ req_distinguished_name ]
C = Code du pays où est enregistrée l'entité (en majuscules)
C_default = FR
C_min = 2

```

4. Certificats

C_max = 2

O = Nom officiel complet de l'entité

1.OU = Identification de l'entité au format ISO 6523

2.OU = Identification complémentaire (optionnelle, [Entrée] pour ignorer)

CN = Prénom et nom de l'état civil du porteur

La syntaxe non documentée MASK:... permet d'autoriser uniquement certains formats de chaînes de caractères. La valeur 0x2002 autorise uniquement les chaînes de type PrintableString (0x0002) et UTF8String (0x2000). Plus précisément, une chaîne de caractères est codée en PrintableString par défaut, sauf si elle contient des caractères qui n'existent pas dans le jeu de caractères PrintableString, et dans ce cas elle est codée sous la forme d'une UTF8String. Pour obtenir la liste complète des valeurs possibles pour MASK, se reporter aux constantes B_ASN1_* dans le fichier d'en-tête asn1.h de la libcrypto.

Générer la CSR, en renseignant les informations demandées.

Les commandes ci-dessous ne fonctionnent pas sous Windows avec la page de code UTF-8 (chcp 65001) de l'invite de commande, qui semble mal coexister avec openssl : s'attendre au mieux à pouvoir saisir uniquement des caractères ASCII (avec un message d'erreur fatale problems making Certificate Request à la saisie l'un caractère non ASCII), et au pire à une erreur empêchant le démarrage d'OpenSSL (The device is not ready., observé sous Windows XP, à moins que l'exécutable openssl.exe ne soit appelé depuis un fichier batch, auquel cas – toujours sous Windows XP – le fichier batch ne sera même pas lu !).

Une astuce (particulièrement inélégante... une méthode plus pragmatique est proposée un peu plus loin) consiste à utiliser une page de code telle que Windows-1252 (chcp 1252), et à copier-coller des chaînes de caractères UTF-8 représentées en Windows-1252 vers l'invite de commande. Par exemple, sous Notepad++⁸, créer un nouveau fichier, choisir le codage UTF-8 (Encoding > Encode in UTF-8), saisir la chaîne de caractères souhaitée (par exemple « Entité »), choisir le codage ANSI (Encoding > Encode in ANSI), copier la chaîne résultante (« Mon entitÃ© »), et la coller dans la fenêtre d'invite de commande. Dans cet exemple, les deux octets représentant le caractère « é » en UTF-8 (0xc3 0xa9) sont représentés en Windows-1252 par les deux caractères « Ã© ». OpenSSL traitant uniquement les octets déduits de la représentation de la page de code en cours, il n'y voit que du feu.

```
$ openssl req -new -key ee-key.pem -config ee-req-dynamic.cnf.utf8 \
-utf8 -out ee-req.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.

Code du pays où est enregistrée l'entité (en majuscules) [FR]:

Nom officiel complet de l'entité []: **Mon Entité**

Identification de l'entité au format ISO 6523 []: **0002 987654321**

Identification complémentaire (optionnelle, [Entrée] pour ignorer) []:

Prénom et nom de l'état civil du porteur []: **Prénom NOM**

Pour ajouter des contrôles sur la longueur des champs et/ou ajouter des attributs dans la CSR, se reporter à la documentation⁹ de la fonction openssl req.

8. <http://notepad-plus-plus.org/>

Génération avec objet fixé dans un fichier de configuration

La dernière méthode proposée consiste à définir les valeurs des champs dans le fichier de configuration. Créer le fichier `ee-req-static.cnf.utf8` suivant, et l'enregistrer avec le codage UTF-8 (sans BOM).

Cette méthode est la plus pratique sous Windows pour éviter les problèmes liés au codage UTF-8 car OpenSSL n'a aucun souci à lire les fichiers de configuration au format UTF-8 (tant que l'option `-utf8` est utilisée).

```
[ req ]
distinguished_name = req_distinguished_name
string_mask = MASK:0x2002
prompt = no

[ req_distinguished_name ]
C = FR
O = Mon Entité
OU = 0002 987654321
CN = Prénom Nom
```

Exécuter ensuite la commande suivante :

```
$ openssl req -new -key ee-key.pem -config ee-req-static.cnf.utf8 -utf8 \
-out ee-req2.pem
```

Si le fichier `ee-req.pem` a été créé précédemment, alors s'assurer que les fichiers `ee-req.pem` et `ee-req2.pem` sont identiques puis supprimer `ee-req2.pem`, sinon renommer `ee-req2.pem` en `ee-req.pem` avant de poursuivre.

Exercice — Générer un bi-clé RSA `ee2-key.pem` de 2048 bits, puis, à l'aide d'une méthode au choix, la CSR `ee2-req.pem` correspondant à cette clé et au DN `/C=FR/O=Mon Organisation/OU=0002 963852741/CN=Entité Finale`.

Représentation de l'objet d'une requête de signature de certificat

Pour conclure partiellement sur les différences de comportement par rapport au codage UTF-8, la fin de cette section s'intéresse aux différents affichages possibles du sujet de la CSR.

Affichage de la CSR ou du sujet de la CSR avec la commande `openssl req` : les caractères non ASCII sont remplacés par une représentation hexadécimale (le caractère « é » a pour codage UTF-8 `0xc3 0xa9`, comme indiqué par exemple dans cette table de caractères¹⁰).

```
$ openssl req -in ee-req.pem -noout -text
Certificate Request:
  Data:
    Version: 0 (0x0)
    Subject: C=FR, O=Mon Entit\xC3\xA9, OU=0002 987654321, CN=Pr\xC3\xA9nom Nom
```

9. <http://www.openssl.org/docs/apps/req.html>

10. <http://www.utf8-chartable.de/>

4. Certificats

...

```
$ openssl req -in ee-req.pem -noout -subject
subject=/C=FR/O=Mon Entit\xC3\xA9/OU=0002 987654321/CN=Pr\xC3\xA9nom Nom
```

Analyse ASN.1 de la CSR sous Linux : les chaînes UTF8String ne sont pas affichées directement, mais leur représentation hexadécimale montre la valeur attendue (avec toujours c3 a9 pour « é »).

```
$ openssl req -in ee-req.pem -outform DER | openssl asn1parse \
-inform DER -i -dump
 0:d=0  hl=4 l= 663 cons: SEQUENCE
 4:d=1  hl=4 l= 383 cons: SEQUENCE
 8:d=2  hl=2 l=   1 prim: INTEGER               :00
11:d=2  hl=2 l=  82 cons: SEQUENCE
13:d=3  hl=2 l=  11 cons: SET
15:d=4  hl=2 l=   9 cons: SEQUENCE
17:d=5  hl=2 l=   3 prim: OBJECT                :countryName
22:d=5  hl=2 l=   2 prim: PRINTABLESTRING      :FR
26:d=3  hl=2 l=  20 cons: SET
28:d=4  hl=2 l=  18 cons: SEQUENCE
30:d=5  hl=2 l=   3 prim: OBJECT                :organizationName
35:d=5  hl=2 l=  11 prim: UTF8STRING
      0000 - 4d 6f 6e 20 45 6e 74 69-74 c3 a9          Mon Entit..
48:d=3  hl=2 l=  23 cons: SET
50:d=4  hl=2 l=  21 cons: SEQUENCE
52:d=5  hl=2 l=   3 prim: OBJECT                :organizationalUnitName
57:d=5  hl=2 l=  14 prim: PRINTABLESTRING      :0002 987654321
73:d=3  hl=2 l=  20 cons: SET
75:d=4  hl=2 l=  18 cons: SEQUENCE
77:d=5  hl=2 l=   3 prim: OBJECT                :commonName
82:d=5  hl=2 l=  11 prim: UTF8STRING
      0000 - 50 72 c3 a9 6e 6f 6d 20-4e 6f 6d          Pr..nom Nom
```

...

Analyse ASN.1 de la CSR sous Windows, sous une page de code autre que 65001 (UTF-8), par exemple 850 : OpenSSL tente d'afficher la représentation de la chaîne dans la page de code en cours (avec la page 65001, si openssl peut être lancé, par exemple sous Windows 7, la commande s'interrompt immédiatement après l'affichage du premier caractère non ASCII).

```
> openssl req -in ee-req.pem -outform DER | openssl asn1parse -inform DER -i \
-dump
 0:d=0  hl=4 l= 663 cons: SEQUENCE
 4:d=1  hl=4 l= 383 cons: SEQUENCE
 8:d=2  hl=2 l=   1 prim: INTEGER               :00
11:d=2  hl=2 l=  82 cons: SEQUENCE
13:d=3  hl=2 l=  11 cons: SET
15:d=4  hl=2 l=   9 cons: SEQUENCE
17:d=5  hl=2 l=   3 prim: OBJECT                :countryName
22:d=5  hl=2 l=   2 prim: PRINTABLESTRING      :FR
26:d=3  hl=2 l=  20 cons: SET
28:d=4  hl=2 l=  18 cons: SEQUENCE
```

```

30:d=5  hl=2 l= 3 prim:  OBJECT          :organizationName
35:d=5  hl=2 l= 11 prim:  UTF8STRING      :Mon Entit  
48:d=3  hl=2 l= 23 cons:  SET
50:d=4  hl=2 l= 21 cons:  SEQUENCE
52:d=5  hl=2 l= 3 prim:  OBJECT          :organizationalUnitName
57:d=5  hl=2 l= 14 prim:  PRINTABLESTRING :0002 987654321
73:d=3  hl=2 l= 20 cons:  SET
75:d=4  hl=2 l= 18 cons:  SEQUENCE
77:d=5  hl=2 l= 3 prim:  OBJECT          :commonName
82:d=5  hl=2 l= 11 prim:  UTF8STRING      :Pr  nom Nom

```

...

Dans tous les cas, l'analyse ASN.1 par `openssl asn1parse` met en   vidence que les cha  nes qui admettent un codage ASCII sont repr  sent  es par une `PrintableString`, r  servant aux `UTF8String` les cas o   un codage ASCII n'est pas possible.

4.3. G  n  ration des certificats

G  n  ration du certificat de l'autorit   de certification

Cr  er le fichier de configuration `ca-crt.cnf` suivant, d  finissant le profil du certificat de l'autorit   de certification (AC). Pour plus d'informations sur le format du fichier de configuration OpenSSL pour les profils de certificats, consulter cette page¹¹.

```

[ca_ext]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
keyUsage=critical,keyCertSign,cRLSign
certificatePolicies \
= 1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.1
basicConstraints=critical,CA:TRUE

```

Le caract  re « \ » est ici litt  ral quel que soit l'environnement, et permet de d  couper une ligne « fonctionnelle » sur plusieurs lignes « physiques ».

L'OID (*object identifier*, ou identifiant d'objet) de la politique de certification est normalement issu d'un sous-arc de l'OID attribu      l'organisme en charge de l'autorit   de certification. Les possibilit  s d'obtention d'OID sont d  crites dans la FAQ du site OID Repository¹². Ainsi, l'OID dans le fichier de configuration ci-dessus est d  fini sous un identifiant de sous-arc g  n  r   al  atoirement sous un arc pr  vu par Microsoft    l'usage de ses clients,    l'aide du script `oidgen.vbs` (disponible ici¹³).    noter que la piste de l'arc UUID¹⁴ de l'ITU, qui est sans doute plus satisfaisante car ind  pendante d'un organisme commercial, n'est pas recommand  e    la date d'  criture car les OID associ  s ne sont support  s ni par Windows, ni par Firefox (version 11), ni par bien d'autres applications¹⁵.

11. http://www.openssl.org/docs/apps/x509v3_config.html

12. <http://www.oid-info.com>

13. <http://gallery.technet.microsoft.com/scriptcenter/56b78004-40d0-41cf-b95e-6e795b2e8a06>

14. <http://www.itu.int/ITU-T/asn1/uuid.html>

15. http://www.viathinksoft.de/~daniel-marschall/asn.1/oid_facts.html

4. Certificats

Générer le certificat de l'AC à partir de la CSR `ca-req.pem` et du fichier de configuration précédent, pour une durée de validité de 10 ans, et signée en SHA256 et RSA avec la clé privée de l'AC `ca-key.pem`.

```
$ openssl x509 -req -in ca-req.pem -extfile ca-crt.cnf -extensions ca_ext \
  -signkey ca-key.pem -sha256 -days 3652 -out ca-crt.pem
Loading 'screen' into random state - done
Signature ok
subject=/C=FR/O=Mon Entreprise/OU=0002 123456789/OU=OpenSSL Root CA
Getting Private key
```

■ En l'absence d'option `-set_serial`, le numéro de série est généré aléatoirement sur 64 bits.

■ Si le fichier de configuration définit un seul profil, alors la balise de section [...] peut être omise, et dans ce cas ne pas utiliser l'option `-extensions`.

■ Les versions successives des RFC permettent d'employer un nombre croissant d'algorithmes de signature de certificat (en particulier RSASSA-PSS, dont l'algorithme de *padding* EMSA-PSS a été évoqué dans la section sur la signature numérique, introduit dans la RFC 4055, et les algorithmes GOST introduits dans la RFC 4491), mais en pratique les algorithmes de signature les plus fréquemment utilisés sont à base de RSA et SHA-1 (en mode PKCS#1 version 1.5, cf. RFC 3279) ou SHA-2 (également en mode PKCS#1 version 1.5, cf. RFC 4055).

Afficher le contenu du certificat.

```
$ openssl x509 -in ca-crt.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            ac:aa:ff:2f:9d:e9:3c:53
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=FR, O=Mon Entreprise, OU=0002 123456789, OU=OpenSSL Root CA
        Validity
            Not Before: Apr  7 14:12:48 2012 GMT
            Not After : Apr  7 14:12:48 2022 GMT
        Subject: C=FR, O=Mon Entreprise, OU=0002 123456789, OU=OpenSSL Root CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            Public-Key: (2048 bit)
            Modulus:
                00:9c:2d:ba:62:07:f3:95:a0:b4:b3:47:31:0d:14:
                ...
                85:c1
            Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                4C:6D:87:93:82:F7:2D:2C:07:23:A2:0F:E0:71:2D:17:3F:39:F3:8F
            X509v3 Authority Key Identifier:
                keyid:4C:6D:87:93:82:F7:2D:2C:07:23:A2:0F:E0:71:2D:17:3F:39:F3:8
F

            X509v3 Key Usage: critical
```

```

Certificate Sign, CRL Sign
X509v3 Certificate Policies:
  Policy: 1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224
.8393003.10972002.1.1

X509v3 Basic Constraints: critical
  CA:TRUE
Signature Algorithm: sha256WithRSAEncryption
  96:a4:82:08:6e:7a:3a:a7:32:7c:db:54:13:f1:0f:a0:3e:dc:
...
  7c:32:c1:bb

```

Génération d'un certificat d'authentification et signature pour l'entité finale

Créer le fichier de configuration `ee-crt.cnf` suivant, définissant le profil du certificat d'authentification et signature du porteur.

```

[authsig_ext]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
keyUsage=critical,nonRepudiation,digitalSignature
certificatePolicies \
= 1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.2
crlDistributionPoints=URI:http://tiny.cc/LatestCRL
basicConstraints=critical,CA:FALSE

```

L'élément `crlDistributionPoints` doit contenir, pour que le profil du certificat soit conforme au RGS, l'URL d'une liste de certificats révoqués (LCR) publiquement accessible et au format DER. Même si l'outil de validation des profils RGS n'effectue pas cette vérification à la date de rédaction, il est recommandé de renseigner une URL publique pointant réellement sur la LCR. Une solution possible est de créer un site web gratuit (une recherche des mots clés « hébergement web gratuit » ou l'offre du FAI du lecteur sont deux pistes à creuser) et d'y publier la LCR. Pour plus de flexibilité, l'auteur a utilisé un réducteur d'URL pour produire une URL générique « pérenne » pouvant être paramétrée pour pointer vers l'hébergeur de la LCR du moment. (À un instant donné, il est peu probable que l'URL ci-dessus pointe réellement sur une LCR, l'auteur n'ayant pas prévu de tenir celle-ci à jour en dehors des périodes de rédaction de ce document !)

Avant de créer le certificat, générer aléatoirement le numéro de série du prochain certificat signé par l'AC, sur 8 octets (64 bits).

```
$ openssl rand -hex 8 -out ca-crt.srl
```

Déclencher la génération du certificat.

```
$ openssl x509 -req -in ee-req.pem -extfile ee-crt.cnf -extensions authsig_ext \
  -CA ca-crt.pem -CAkey ca-key.pem -CAserial ca-crt.srl -sha256 -days 730 \
  -out ee-crt-authsig.pem

```

Afficher le contenu du certificat.

```
$ openssl x509 -in ee-crt-authsig.pem -noout -text
Certificate:
```

4. Certificats

```
Data:
  Version: 3 (0x2)
  Serial Number:
    dc:d2:1e:e5:a2:b7:df:c7
  Signature Algorithm: sha256WithRSAEncryption
  Issuer: C=FR, O=Mon Entreprise, OU=0002 123456789, OU=OpenSSL Root CA
  Validity
    Not Before: Apr  7 15:17:47 2012 GMT
    Not After : Apr  7 15:17:47 2014 GMT
  Subject: C=FR, O=Mon Entit\xC3\xA9, OU=0002 987654321, CN=Pr\xC3\xA9nom
Nom
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (2048 bit)
    Modulus:
      00:b8:6f:48:f9:99:f1:99:71:c6:6f:80:64:d1:ca:
      ...
      8d:d1
    Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Subject Key Identifier:
      8B:28:E4:FE:77:43:B0:05:E3:67:1F:8A:EF:13:58:2B:CB:46:40:60
    X509v3 Authority Key Identifier:
      keyid:4C:6D:87:93:82:F7:2D:2C:07:23:A2:0F:E0:71:2D:17:3F:39:F3:8
F
    X509v3 Key Usage: critical
      Digital Signature, Non Repudiation
    X509v3 Certificate Policies:
      Policy: 1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224
      .8393003.10972002.1.2
    X509v3 CRL Distribution Points:
      Full Name:
        URI:http://tiny.cc/LatestCRL
    X509v3 Basic Constraints: critical
      CA:FALSE
  Signature Algorithm: sha256WithRSAEncryption
    09:2f:ba:bf:d9:86:fd:02:9c:8a:22:3b:a0:f2:e4:81:60:3a:
    ...
    77:cf:6e:8b
```

4.4. Vérification manuelle de la signature d'un certificat

Au niveau le plus général, un certificat X.509 contient une structure de données à signer nommée `tbsCertificate`, un algorithme de signature (`signatureAlgorithm`) et une signature numérique (`signatureValue`), visibles sur le certificat d'authentification précédent en filtrant

l'analyse de la commande `openssl asn1parse` sur les deux premiers niveaux d'imbrication (via `grep d=[01]`, la notation `d=` étant employée dans l'affichage de l'analyse ASN.1 pour désigner la profondeur — *depth* — de l'imbrication, où 0 est le niveau le plus global).

```
$ openssl asn1parse -in ee-crt-authsig.pem -i | grep d=[01]
  0:d=0  hl=4 l=1006 cons: SEQUENCE
  4:d=1  hl=4 l= 726 cons: SEQUENCE
 734:d=1  hl=2 l= 13 cons: SEQUENCE
 749:d=1  hl=4 l= 257 prim: BIT STRING
```

Pour vérifier manuellement la signature numérique, il faut extraire la structure `tbsCertificate` et en calculer l'empreinte, extraire la signature numérique de `signatureValue` et la déchiffrer avec la clé publique de l'AC, et vérifier que les deux valeurs obtenues sont égales.

La structure `tbsCertificate` est la première SEQUENCE de niveau 1 (`d=1`) sous la structure `Certificate` de niveau 0 :

```
4:d=1  hl=4 l= 726 cons: SEQUENCE
```

La SEQUENCE commence à l'octet 4. L'extraire en utilisant l'option `-strparse 4 \` de la commande `openssl asn1parse` :

```
$ openssl asn1parse -in ee-crt-authsig.pem -strparse 4 \
-out ee-crt-authsig.tbscertificate.der -noout
```

L'algorithme de signature du certificat étant `sha256WithRSAEncryption`, calculer l'empreinte SHA-256 de la structure `tbsCertificate` extraite.

```
$ openssl sha256 ee-crt-authsig.tbscertificate.der
SHA256(ee-crt-authsig.tbscertificate.der)= 45232ce57de879d9030142362a6c86186b0a1
c95d8e583387d9032b2e13ff677
```

La signature numérique du certificat est la BIT STRING de niveau 1 (`d=1`) sous la structure `Certificate`, à partir de l'octet 749 :

```
$ openssl asn1parse -in ee-crt-authsig.pem -i -offset 749 -dump
  0:d=0  hl=4 l= 257 prim: BIT STRING
    0000 - 00 09 2f ba bf d9 86 fd-02 9c 8a 22 3b a0 f2 e4  .. / .....";...
    ...
    00f0 - af 0e 13 4d af b8 d6 82-eb 24 1f 41 ea 77 cf 6e  ...M.....$.A.w.n
    0100 - 8b                                     .
```

Avec l'option `-offset 749`, la numérotation de la position des octets dans l'affichage ci-dessus est relative à l'octet 749 de la structure globale.

La signature numérique du certificat a une longueur de 2048 bits (soit 256 octets) : où commence-t-elle, et pourquoi la BIT STRING a-t-elle une longueur de `l= 257` octets ? L'octet 749 est le début du *codage* de la BIT STRING. Ce codage est constitué d'une en-tête (*header*) de longueur 4 octets (`hl=4`, avec `hl` pour *header length* ou longueur de l'en-tête), suivie du contenu, qui commence donc à l'octet 753. Une BIT STRING, comme son nom l'indique est une chaîne de *bits*, donc contrairement à une OCTET STRING par exemple, sa valeur peut être représentée par un nombre de bits qui n'est pas nécessairement un multiple de 8. Le premier octet de contenu (ici `0x00`) donne le

4.5. Génération d'un certificat de confidentialité

Cette section s'intéresse à la génération d'un certificat de confidentialité, qui sera utilisé pour protéger la confidentialité de données destinées à son porteur dans les chapitres suivants.

Générer un bi-clé RSA de 2048 bits.

```
$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 \
  -out ee2-key.pem
```

La clé ci-dessous sera utilisée pour `ee2-key.pem` :

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAvm+7N4g3fCuN2kp/Fg3dr8/fz4ImPOXwV90o3roN/Q6SKKn
2yNy/RU64WpM2kyVrj3Mn8izELYQBS+m50USEEqrikanx4EAY9QPgP6f6H0IczH0
T5mjhw2m5H1gXibulBhWpdrkYcQaRB2yQAo2owBDLRgRpi9vNZUGCLmMLkCmqeNr
k9njZ7FxDZp6tiYay64QMBRlc96aYSyuk8+WG3Ia3FIRze03H7IGouVFRJlkHon
NLbVWanBNTnfgkl4CQhzTeg/fEBXIRnF6gN56hhZ2KwVVt6Sgn06ISyv3IenSrim
iWFTddbXVP9PCSQpBjW7rq/ZRvB4Z3uoKzJr7QIDAQABAoIBABDgG3DEKRj0daON
rC5kDwLX920CNm7KTukGDECdva9UcusRdAjPUxmJru2CQELB3BvueCRNKIoZnfF4
5G1Ue5tts5D65gWbbFeDIaggIlwC5QC6kobZb60k+y00c625ul0TGJ5by1k3ogo6
xjupLl7EpT6iCuJfGIspPZMGWJ/DDiCYD5JywmrgfET/Iy+0d0nKlhm1kxCOKCGE
VcvLQXIA9wV0ogtD20JzXBGVF0cjQbI9j5kD2F5IGK1IyNNS6yTPTh0K6XSM3TYj
4dFLodRh8mlRiBtyXGeGaA7I1lPL2LrhVvWrGqql53fdYz0/eNlVHP7tTaHwn4z0
NXluBNkCgYEA8vi14FMb4KYHo6Jy99T9TOMBp6N34xLbp09+T2BRu53Bktq1oDgU
GPYv/ry1gfHPqERaLhmwugI8B1IqmjmtPLIwC5IROAdPW0qEb/NFeUZA8wobHyP
laGkzs9HgA4jW2EchXEUy0CaivVdcdmeCWznWl+luscCt3H0i7+akJcGyEAxqXS
r0v576hP2ewrCAANFqhYDm7Zbc6/bNIVbj/3r1zpyTqv+4F7rih8t95ws7pS2a+o
1t2Z5zpnzz60LMCilUCrLCU69pgD20Ge2fiZc/C0rXNsSvsxlH5ElKPTVSV+BZ5I
K5y0n/EISqC1MbaFT9S/Tz1035h6M+9tsxX9tBsCgYAWxzfFzYTPisEslg4xLcaR
bravQKP8pcfx54Hv7xe6fw+mtNpSKu+3Z/kX6Jkb1Y/iNoY9zCrFRXBbP1i/HKkh
B5ETRj5m4ki0DW5dEHFy7SU7NeiUQxky7fSFbulFGA44guIpoHBIPMXyQSiBg+VM
/OizkEP5Pq8Cg6xpbY0QAQKBgHtwzgLgvLade8lqxa66AWgkAlwtq/VddXzU67ZQ
D8UsUqbWTKdWxhmKVT4Y/yXgUd2uF5g1qo6Sr7GMKONMbXub7QKIvCRCgJr+Iw1G
021dffvw7smFYODqugk36HpPywkIu8ZGIcQGDuX5Cb+zsRHTnXJLIbk2UjH1xdB0
i0wjAoGBAM3Sa9mpHItntPvNg84kJer7Lbx7xNlhuKgCJ2+U10ngGExjQNDZjQF
6dJpt+MqWezAMmuJRyzy1zwc9cRz0NkuvXOAWHtep43qXk+jedaBpujGqBQCg3Q
e6RymtQKiGTyDdnum/HjKFEhDIHjzCEMSlCRs/Sv4FkhTt2i9xwZ
-----END RSA PRIVATE KEY-----
```

Créer le fichier de configuration `ee2-req-static.cnf.utf8` suivant, avec le codage UTF-8 sans BOM, pour la CSR :

```
[ req ]
distinguished_name      = req_distinguished_name
string_mask              = MASK:0x2002
prompt = no

[ req_distinguished_name ]
C = FR
O = Mon Organisation
```

4. Certificats

OU = 0002 963852741

CN = Entité Finale

Générer la CSR :

```
$ openssl req -new -key ee2-key.pem -config ee2-req-static.cnf.utf8 -utf8 \
  -out ee2-req.pem
```

Créer le fichier de configuration ee2-crt.cnf pour les extensions du certificat :

```
[confid_ext]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
keyUsage=critical,keyEncipherment
certificatePolicies \
= 1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.3
crlDistributionPoints=URI:http://tiny.cc/LatestCRL
basicConstraints=critical,CA:FALSE
```

Il est également possible d'ajouter la section [confid_ext] au fichier ee-crt.cnf existant et d'utiliser ce fichier de configuration ci-après.

Générer aléatoirement le numéro de série du prochain certificat signé par l'AC (il s'agit ici de l'AC créée dans le chapitre sur les certificats, et non celle du chapitre sur l'authentification).

```
$ openssl rand -hex 8 -out ca-crt.srl
```

Générer le certificat.

```
$ openssl x509 -req -in ee2-req.pem -extfile ee2-crt.cnf \
  -extensions confid_ext -CA ca-crt.pem -CAkey ca-key.pem -CAserial ca-crt.srl \
  -sha256 -days 730 -out ee-crt-authsig.pem
```

Valider la conformité du profil du certificat par rapport au RGS.

4.6. Importation dans le magasin de certificats de Windows

Pour pouvoir utiliser un certificat dans une application Microsoft, le certificat et sa clé privée doivent être importés dans le magasin de certificats de Windows. Le format utilisé pour l'importation est le format PKCS#12, qui permet de stocker un certificat, sa clé privée éventuellement protégée par un mot de passe, et d'autres certificats tels que ceux de la chaîne de certification.

Créer un fichier au format PKCS#12 contenant le certificat du porteur et sa clé privée, ainsi que le certificat de l'AC. Saisir le mot de passe au moment de la demande.

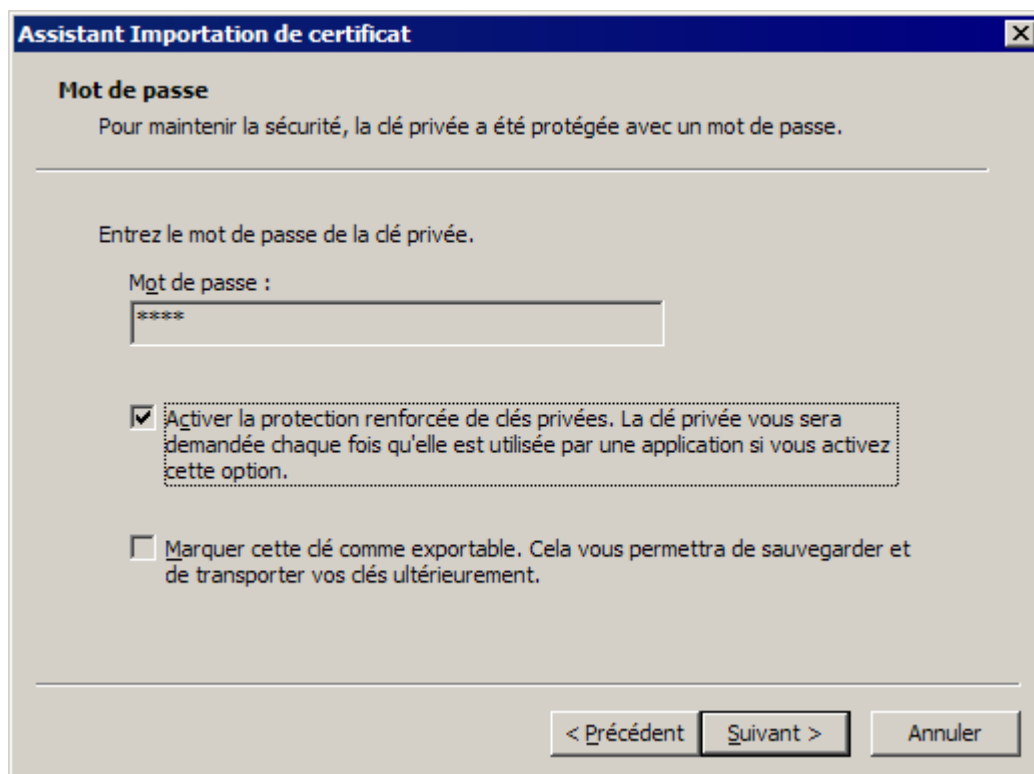
```
$ openssl pkcs12 -export -in ee-crt-authsig.pem -inkey ee-key.pem \
  -certfile ca-crt.pem -name "OpenSSL EE" -out ee-authsig.p12
Enter Export Password:
Verifying - Enter Export Password:
```

L'option `-name` définit le nom convivial du certificat, qui permet de distinguer un certificat dans l'affichage du magasin de certificats de Windows.

La manière la plus simple d'installer un certificat depuis un fichier au format PKCS#12 est de double-cliquer sur le nom du fichier dans l'explorateur Windows.

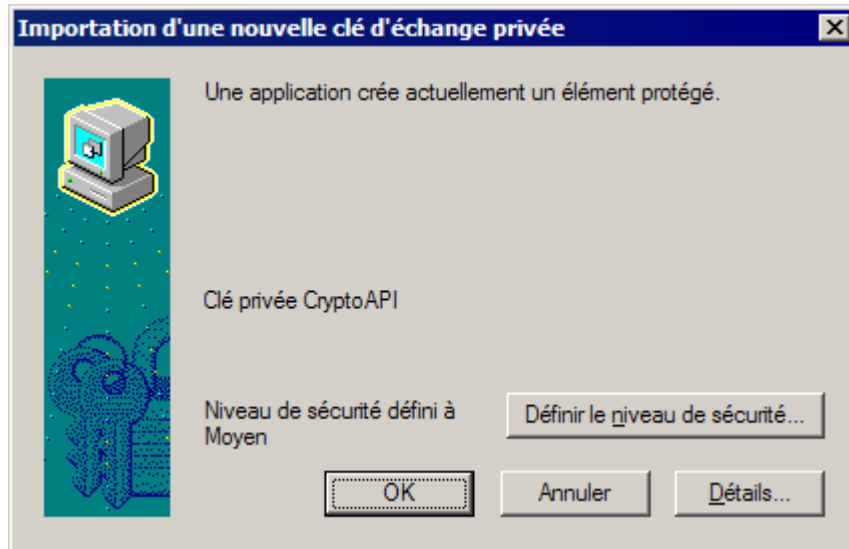
Windows associe les extensions `.p12` et `.pfx` aux fichiers au format PKCS#12.

Cliquer sur le bouton « Suivant » dans la boîte de dialogue de bienvenue, puis valider le choix du fichier à importer en cliquant sur le bouton « Suivant ». Dans la boîte de dialogue suivante, saisir le mot de passe, et cocher la case « Activer la protection renforcée des clés (...) », comme dans la figure ci-dessous.

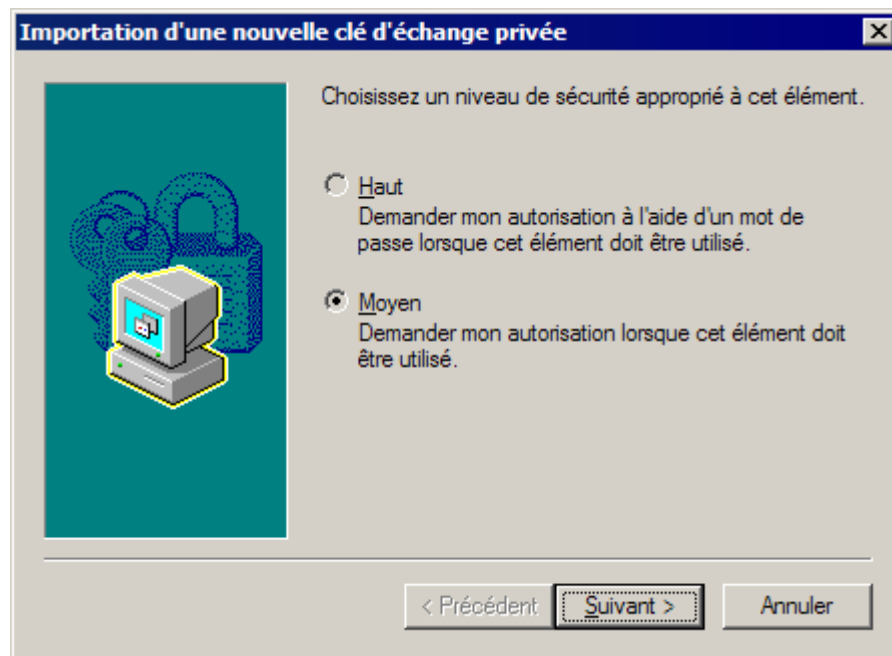


4. Certificats

Valider avec le bouton « Suivant ». Laisser l'option par défaut « Sélectionner automatiquement le magasin de certificats selon le type de certificats » et cliquer sur le bouton « Suivant ». Démarrer l'importation en cliquant sur le bouton « Terminer ». Conséquence de l'activation de la protection renforcée des clés privées, la boîte de dialogue suivante s'affiche.



Par défaut, le niveau de protection est moyen, c'est-à-dire qu'une alerte Windows est affichée dès qu'une application souhaite utiliser la clé privée. Il est également possible d'élever le niveau de sécurité au niveau haut, en cliquant sur le bouton « Définir le niveau de sécurité », et en sélectionnant l'option « Haut » dans la boîte de dialogue ci-dessous :

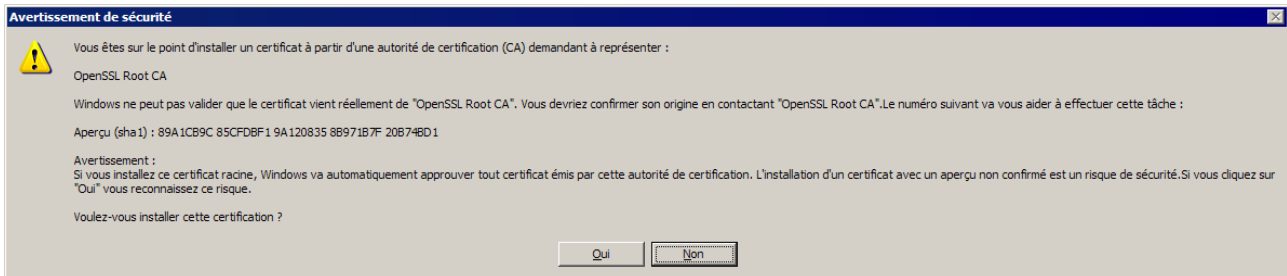


Après avoir cliqué sur le bouton « Suivant », l'utilisateur définit un mot de passe, lequel sera ensuite demandé lors de chaque tentative d'accès à la clé privée.

Le mot de passe n'étant pas régi par une politique de mot de passe et n'étant pas recouvrable en cas d'oubli (ce qui finit généralement par arriver s'il est rarement utilisé), il est recommandé d'activer le ni-

veau de protection haut avec précaution. Dans le cadre de ce document, il est supposé que le niveau de protection est moyen.

Après avoir validé à l'aide du bouton « OK », l'avertissement suivant s'affiche.



Le fond et la forme de cet avertissement (contenu alarmiste, mise en page inhabituelle, procédure et terminologie peu compréhensibles pour un néophyte) sont souvent cités pour illustrer la « non-convivialité » de la confiance électronique pour l'utilisateur final.

Utiliser la commande openssl x509 suivante pour calculer l'« Aperçu (sha1) » du certificat :

```
$ openssl x509 -fingerprint -in ca-crt.pem -noout
```

```
SHA1 Fingerprint=89:A1:CB:9C:85:CF:DB:F1:9A:12:08:35:8B:97:1B:7F:20:B7:4B:D1
```

L'empreinte ci-dessus est l'empreinte SHA-1 (par défaut, l'algorithme de hachage pouvant être précisé par une option `-<algorithme>`, par exemple `-sha256`) du codage DER du certificat.

Les empreintes correspondant, l'AC racine peut être importée sans crainte en cliquant sur le bouton « Oui ».

Enfin, la boîte de dialogue indiquant que l'importation s'est terminée correctement s'affiche : cliquer sur le bouton « OK ».

Le magasin de certificats de Windows peut être visualisé de deux manières :

- Depuis la console de gestion, soit via le menu Windows Démarrer > Exécuter... > saisir `certmgr.msc` et valider, soit (notamment si `certmgr.msc` n'existe pas) via Démarrer > Exécuter... > saisir `mmc` et valider puis choisir le menu Fichier > Ajouter/Supprimer un composant logiciel enfichable..., cliquer sur le bouton « Ajouter... », sélectionner « Certificats », cliquer sur le bouton « Ajouter », laisser la sélection par défaut « Mon compte d'utilisateur », cliquer sur le bouton « Terminer », puis sur le bouton « Fermer », puis sur « OK », et dérouler la liste « Certificats - Utilisateur actuel » (et optionnellement, sauvegarder cette configuration de console pour pouvoir y accéder directement ultérieurement). Les certificats de l'utilisateur sont sous Personnel/Certificats (y retrouver le certificat délivré à « Prénom Nom » portant le nom convivial « OpenSSL EE »), les AC intermédiaires sous Autorités intermédiaires/Certificats, et les AC racines sous Autorité de certification racines/Certificats (y retrouver le certificat délivré à « OpenSSL Root CA »).
- Depuis Internet Explorer, accéder au menu Outils > Options Internet... > onglet Contenu, cliquer sur le bouton « Certificats ». Les certificats de l'utilisateur sont sous Personnel (le certificat délivré à « Prénom Nom » y porte le nom convivial « OpenSSL EE »), les AC intermédiaires sous Autorités intermédiaires, et les AC racines (dont « OpenSSL Root CA ») sous Autorité principales de confiance.

4. Certificats

Le certificat du porteur importé dans le magasin de certificats de Windows peut être retrouvé par OpenSSL à partir de son nom convivial, en utilisant la commande `openssl engine` (après éventuelle configuration du moteur cryptographique `capi` d'OpenSSL, comme décrit en annexe).

```
> openssl engine -t capi -post lookup_method:2 -post lookup_cert:"OpenSSL EE"
(capi) CryptoAPI ENGINE
      [ available ]
[Success]: lookup_method:2
      Friendly Name "OpenSSL EE"
      Subject: C = FR, O = Mon Entit\C3\A9, OU = 0002 987654321, CN = Pr\C3\A9nom No
m
      Issuer: C = FR, O = Mon Entreprise, OU = 0002 123456789, OU = OpenSSL Root CA
[Success]: lookup_cert:OpenSSL EE
```

4.7. Construction d'un certificat

La construction manuelle d'un certificat s'effectue en trois étapes : la constitution de la structure à signer `TBSCertificate`, la signature de cette structure, et la constitution de la structure `Certificate` contenant ces éléments avec la syntaxe ASN.1 suivante :

```
Certificate ::= SEQUENCE {
    tbsCertificate      TBSCertificate,
    signatureAlgorithm  AlgorithmIdentifier,
    signatureValue      BIT STRING
}
```

Constitution d'une structure TBSCertificate

La structure `TBSCertificate` a la syntaxe ASN.1 suivante :

```
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,
    subjectUniqueID      [2] IMPLICIT UniqueIdentifier OPTIONAL,
    extensions           [3] EXPLICIT Extensions OPTIONAL
}
```

Les champs `version` et `serialNumber` sont de type `INTEGER`, avec pour valeurs respectives 2 et — pour reprendre le cas du certificat d'authentification et signature généré dans la section 4.3 (page 27) — `0xcdcd21ee5a2b7dfc7`. Les autres champs sont des `SEQUENCE`.

Créer le fichier `ee-TBSCertificate.asn.cnf` avec l'encodage UTF-8, contenant les lignes suivantes :

```
asn1=SEQUENCE:ee_tbsCertificate

[ee_tbsCertificate]
version = EXPLICIT:0,INTEGER:2
serialNumber = INTEGER:0xDCD21EE5A2B7DFC7
signature = SEQUENCE:ee_signature
issuer = SEQUENCE:ee_issuer
validity = SEQUENCE:ee_validity
subject = SEQUENCE:ee_subject
subjectPublicKeyInfo = SEQUENCE:ee_subjectPublicKeyInfo
extensions = EXPLICIT:3,SEQUENCE:ee_extensions
```

Le champ `signature` contenant l'algorithme de signature (ici SHA-256 et RSA) est de type `ASN.1 AlgorithmIdentifier`, et est défini de manière analogue au champ `algorithm` de la structure `SubjectPublicKeyInfo` (cf. section 3.4, page 13) :

```
[ee_signature]
algorithm = OID:sha256WithRSAEncryption
parameters = NULL
```

Les champs `issuer` et `subject` contiennent chacun une `SEQUENCE` de `RDN`, chaque `RDN` étant un `SET` de `SEQUENCE {type, value}`, où `type` est l'OID du type de `RDN`, et `value` la valeur du `RDN` (elle-même de type `PrintableString` ou `UTF8String` par exemple).

À titre illustratif, le champ `issuer` à produire pour le certificat considéré admet la structure `ASN.1` suivante (produite par l'outil `dumpasn1` présenté dans l'annexe B.3) :

```
SEQUENCE {
  SET {
    SEQUENCE {
      OBJECT IDENTIFIER countryName (2 5 4 6)
      PrintableString 'FR'
    }
  }
  SET {
    SEQUENCE {
      OBJECT IDENTIFIER organizationName (2 5 4 10)
      PrintableString 'Mon Entreprise'
    }
  }
  SET {
    SEQUENCE {
      OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
      PrintableString '0002 123456789'
    }
  }
  SET {
    SEQUENCE {
      OBJECT IDENTIFIER organizationalUnitName (2 5 4 11)
```

4. Certificats

```
        PrintableString 'OpenSSL Root CA'
    }
}
}
```

Cela se traduit par les lignes suivantes dans le fichier `ee-TBSCertificate.asn.cnf` :

```
[ee_issuer]
C = SET:ee_issuer_C_RDN
O = SET:ee_issuer_O_RDN
OU1 = SET:ee_issuer_OU1_RDN
OU2 = SET:ee_issuer_OU2_RDN

[ee_issuer_C_RDN]
rdn = SEQUENCE:ee_issuer_C_ATV

[ee_issuer_C_ATV]
type = OID:countryName
value = PRINTABLESTRING:FR

[ee_issuer_O_RDN]
rdn = SEQUENCE:ee_issuer_O_ATV

[ee_issuer_O_ATV]
type = OID:organizationName
value = PRINTABLESTRING:Mon Entreprise

[ee_issuer_OU1_RDN]
rdn = SEQUENCE:ee_issuer_OU1_ATV

[ee_issuer_OU1_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:0002 123456789

[ee_issuer_OU2_RDN]
rdn = SEQUENCE:ee_issuer_OU2_ATV

[ee_issuer_OU2_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:OpenSSL Root CA
```

De même, le DN de l'objet, `/C=FR/O=Mon Entité/OU=0002 987654321/CN=Prénom Nom` (selon la notation usuelle d'OpenSSL, cf. annexe A.3) se traduit par les lignes suivantes :

```
[ee_subject]
C = SET:ee_subject_C_RDN
O = SET:ee_subject_O_RDN
OU = SET:ee_subject_OU_RDN
CN = SET:ee_subject_CN_RDN

[ee_subject_C_RDN]
rdn = SEQUENCE:ee_subject_C_ATV
```



```

[ee_subject_C_ATV]
type = OID:countryName
value = PRINTABLESTRING:FR

[ee_subject_O_RDN]
rdn = SEQUENCE:ee_subject_O_ATV

[ee_subject_O_ATV]
type = OID:organizationName
value = FORMAT:UTF8,UTF8String:Mon Entité

[ee_subject_OU_RDN]
rdn = SEQUENCE:ee_subject_OU_ATV

[ee_subject_OU_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:0002 987654321

[ee_subject_CN_RDN]
rdn = SEQUENCE:ee_subject_CN_ATV

[ee_subject_CN_ATV]
type = OID:commonName
value = FORMAT:UTF8,UTF8String:Prénom Nom

```

La période de validité du certificat a pour syntaxe ASN.1 :

```

Validity ::= SEQUENCE {
    notBefore      Time,
    notAfter       Time
}

```

Time peut être de deux types possibles (UTCTime ou GeneralizedTime). Le type UTCTime est retenu pour le certificat à générer, avec la notation *aammjjhhmmssZ* (pour une heure UTC) pour les dates et heures dans le fichier de configuration :

```

[ee_validity]
notBefore = UTCTIME:120407151747Z
notAfter = UTCTIME:140407151747Z

```

La clé publique est de type SubjectPublicKeyInfo : les sections correspondantes (subjectPublicKeyInfo, rsa_AlgorithmIdentifier et subjectPublicKey) du fichier ee-SubjectPublicKeyInfo.asn.cnf (cf. section 3.4) peuvent être reprises tel quel, en ajoutant simplement le préfixe ee_ aux noms de sections :

```

[ee_subjectPublicKeyInfo]
algorithm = SEQUENCE:ee_rsaEncryption
subjectPublicKey = BITWRAP,SEQUENCE:ee_subjectPublicKey

[ee_rsaEncryption]
algorithm = OID:rsaEncryption

```

4. Certificats

```
parameters = NULL
```

```
[ee_subjectPublicKey]
n = INTEGER:0xB86F48F999F19971C66F8064D1CA0C1A6EC88AF3B939FD07088D97B2BE1E2795B\
D1E8688FF0E6172B73D37A5B81935C7C3AE57A27E5D46F38383089C4410EDA95D1EFA99C4931586\
CE57490060003902034A358E07F90FF2D347342E6BF95139E06E63F3C997874F4B35E8DA3A87F4F\
5188A8674C1B11BA0F329FC5C2EB6CD26F465751C3789C6B97E3963368AA1881910262BA9D8FEE7\
9D3434E722A2302DF4E82D3979355B6254D032A3A52C317FA5D298403693A6C3A1665F3FF6FCED4\
54A064597F5EF417E68BB7FD2D3892B4F04CD359F2B0133460012BE7B5282DB23AAD23FFB37640F\
655F98F3806F10D9B4BF752F41A7E5BD173AB8345B81FD5C8DD1
e = INTEGER:0x010001
```

Chaque extension est représentée sous la forme d'une SEQUENCE {extnID, extnValue}, où extnID est l'OID d'identification de l'extension, et extnValue est une OCTET STRING dont le contenu est la représentation DER de la structure ASN.1 constituant l'extension, ce qui nécessitera d'utiliser le mot clé OCTWRAP, dont le principe est analogue à BITWRAP pour les BIT STRING (cf. la démarche de création du champ subjectPublicKey d'une clé publique dans la section 3.4, page 14). Ainsi, chaque extension sera définie selon le modèle suivant :

```
[extensions]
...
nom_extension=SEQUENCE:nom_extension
...

[nom_extension]
extnID = OID:oid_extension
extnValue = OCTWRAP,structure_ASN1_extension
```

Ajouter tout d'abord la section ee_extensions ci-dessous :

```
[ee_extensions]
subjectKeyIdentifier=SEQUENCE:ee_subjectKeyIdentifier
authorityKeyIdentifier=SEQUENCE:ee_authorityKeyIdentifier
keyUsage=SEQUENCE:ee_keyUsage
certificatePolicies=SEQUENCE:ee_certificatePolicies
crlDistributionPoints=SEQUENCE:ee_crlDistributionPoints
basicConstraints=SEQUENCE:ee_basicConstraints
```

L'extension subjectKeyIdentifier contient une OCTET STRING dont la valeur est l'empreinte SHA-1 de la représentation DER de la structure RSAPublicKey représentant la clé publique de l'entité certifiée. Il suffit pour obtenir cette empreinte de hacher le fichier ee-RSAPublicKey.der généré dans la section 3.4, page 14.

```
$ openssl sha1 ee-RSAPublicKey.der
SHA1(ee-RSAPublicKey.der)= 8b28e4fe7743b005e3671f8aef13582bcb464060
```

Ajouter la section suivante dans le fichier ee-TBSCertificate.asn.cnf, pour l'extension subjectKeyIdentifier :

```
[ee_subjectKeyIdentifier]
extnID = OID:subjectKeyIdentifier
```

```
extnValue = OCTWRAP,FORMAT:HEX,OCTETSTRING:\
8b28e4fe7743b005e3671f8aef13582bcb464060
```

L'extension `authorityKeyIdentifier` référence la clé publique de l'autorité de certification, et contrairement au champ `subjectKeyIdentifier`, plusieurs modes de référence sont possibles, comme indiqué dans la définition ASN.1 de l'extension :

```
AuthorityKeyIdentifier ::= SEQUENCE {
    keyIdentifier          [0] KeyIdentifier          OPTIONAL,
    authorityCertIssuer    [1] GeneralNames          OPTIONAL,
    authorityCertSerialNumber [2] CertificateSerialNumber OPTIONAL
}
```

```
KeyIdentifier ::= OCTET STRING
```

Le type de référence retenu pour le certificat en cours de création est `KeyIdentifier`, identifié par la balise ASN.1 [0] IMPLICIT.

Le mot clé IMPLICIT est sous-entendu dans la définition de la structure `AuthorityKeyIdentifier` compte tenu de la ligne `DEFINITIONS IMPLICIT TAGS ::=` figurant en début du module ASN.1 correspondant.

Avec ce type de référence, le contenu de l'extension `authorityKeyIdentifier` est analogue à celui de `subjectKeyIdentifier`, mais il est calculé à partir de la structure `RSAPublicKey` représentant la clé publique de l'autorité de certification émettrice du certificat. La valeur hexadécimale à utiliser peut être déterminée de la manière suivante à partir du fichier `ca-key.pem` (le bi-clé de l'autorité de certification) :

```
$ openssl rsa -in ca-key.pem -RSAPublicKey_out -outform DER | openssl sha1
writing RSA key
(stdin)= 4c6d879382f72d2c0723a20fe0712d173f39f38f
```

Ou encore, à partir du certificat de l'autorité de certification :

```
$ openssl x509 -in ca-crt.pem -noout -pubkey | openssl rsa -pubin \
-RSAPublicKey_out -outform DER | openssl sha1
(stdin)= 4c6d879382f72d2c0723a20fe0712d173f39f38f
```

Ajouter les sections suivantes au fichier de configuration ASN.1 :

```
[ee_authorityKeyIdentifier]
extnID = OID:authorityKeyIdentifier
extnValue = OCTWRAP,SEQUENCE:ee_authorityKeyIdentifier_seq

[ee_authorityKeyIdentifier_seq]
keyIdentifier = IMPLICIT:0,FORMAT:HEX,OCTETSTRING:\
4c6d879382f72d2c0723a20fe0712d173f39f38f
```

4. Certificats

L'extension `keyUsage` présente la particularité, dans le cas du certificat étudié ici, d'être une extension critique, avec un champ `BOOLEAN` positionnée à une valeur vraie à inclure entre les champs `extnID` et `extnValue`. Par ailleurs, la valeur enrobée dans le champ `extnValue` est un champ de bits représenté par le type `BIT STRING`, et dont les valeurs possibles sont les suivantes :

```
KeyUsage ::= BIT STRING {  
    digitalSignature      (0),  
    nonRepudiation       (1),  
    keyEncipherment      (2),  
    dataEncipherment     (3),  
    keyAgreement         (4),  
    keyCertSign          (5),  
    cRLSign              (6),  
    encipherOnly         (7),  
    decipherOnly         (8)  
}
```

Pour le certificat d'authentification, les bits 0 (`digitalSignature`) et 1 (`nonRepudiation`) sont activés, les autres sont désactivés.

La section correspondant à l'extension `keyUsage` est la suivante :

```
[ee_keyUsage]  
extnID = OID:keyUsage  
critical = BOOLEAN:true  
extnValue = OCTWRAP,FORMAT:BITLIST,BITSTRING:0,1
```

La traduction de la définition ASN.1 de l'extension `certificatePolicies` ne pose aucun problème particulier :

```
[ee_certificatePolicies]  
extnID = OID:certificatePolicies  
extnValue = OCTWRAP,SEQUENCE:ee_policyInformation
```

```
[ee_policyInformation]  
policyInformation = SEQUENCE:ee_policyIdentifier
```

```
[ee_policyIdentifier]  
policyIdentifier = OID:\  
1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.2
```

L'extension `crlDistributionPoints` mérite quelques commentaires. Les premières sections sont classiques :

```
[ee_crlDistributionPoints]  
extnID = OID:crlDistributionPoints  
extnValue = OCTWRAP,SEQUENCE:ee_distributionPoints
```

```
[ee_distributionPoints]  
distributionPoint = SEQUENCE:ee_distributionPoint
```

Le type `DistributionPoint` s'appuie sur les définitions ASN.1 suivantes :

```
DistributionPoint ::= SEQUENCE {
    distributionPoint      [0]      DistributionPointName OPTIONAL,
    reasons                [1]      ReasonFlags OPTIONAL,
    cRLIssuer              [2]      GeneralNames OPTIONAL
}
```

```
DistributionPointName ::= CHOICE {
    fullName               [0]      GeneralNames,
    nameRelativeToCRLIssuer [1]      RelativeDistinguishedName
}
```

```
GeneralName ::= CHOICE {
    otherName               [0]      AnotherName,
    rfc822Name              [1]      IA5String,
    dNSName                 [2]      IA5String,
    x400Address             [3]      ORAddress,
    directoryName           [4]      Name,
    ediPartyName            [5]      EDIPartyName,
    uniformResourceIdentifier [6]      IA5String,
    ipAddress               [7]      OCTET STRING,
    registeredID            [8]      OBJECT IDENTIFIER
}
```

Pour pouvoir définir une URI, correspondant au choix [6] du type `GeneralName`, il faut représenter la structure schématique suivante :

```
[0] CHOICE {
    [0] CHOICE {
        [6] IA5String:...
    }
}
```

La problématique est de déterminer si les balises sont `IMPLICIT` ou `EXPLICIT`. La règle ici est que, par défaut, les balises sont `IMPLICIT` (le module ASN.1 de définition des types comportant l'entête `DEFINITIONS IMPLICIT TAGS`), sauf — en application de la clause 30.8 de [X.680] — les types `CHOICE` qui sont `EXPLICIT`. En restituant les mots clés `EXPLICIT` et `IMPLICIT`, la structure ci-dessus devient :

```
[0] EXPLICIT CHOICE {
    [0] EXPLICIT CHOICE {
        [6] IMPLICIT IA5String:...
    }
}
```

Le type `CHOICE` étant codé comme l'élément choisi, la structure ci-dessus admet la représentation suivante :

```
[0] EXPLICIT {
    [0] EXPLICIT {
        [6] IMPLICIT IA5String:...
    }
}
```

4. Certificats

```
}  
}
```

Cela se traduit par la section suivante :

```
[ee_distributionPoint]  
distributionPoint = EXPLICIT:0,EXPLICIT:0,IMPLICIT:6,IA5STRING:\  
http://tiny.cc/LatestCRL
```

Enfin, la valeur de l'extension basicConstraints a la syntaxe suivante :

```
BasicConstraints ::= SEQUENCE {  
    cA                      BOOLEAN DEFAULT FALSE,  
    pathLenConstraint       INTEGER (0..MAX) OPTIONAL  
}
```

Le certificat est un certificat d'entité finale sans contraintes sur la longueur de la chaîne de certification, donc les deux champs ci-dessus peuvent être omis, ce qui produit une SEQUENCE vide.

Cette extension est par ailleurs marquée critique (tout comme l'extension keyUsage), d'où la section suivante :

```
[ee_basicConstraints]  
extnID = OID:basicConstraints  
critical = BOOLEAN:true  
extnValue = OCTWRAP,SEQUENCE
```

Générer la représentation DER de la structure TBSCertificate ainsi définie :

```
$ openssl asn1parse -genconf ee-TBSCertificate.asn.cnf \  
-out ee-TBSCertificate.der -i
```

Signature du certificat

La signature du certificat correspond au chiffrement, par la clé privée RSA de l'autorité de certification, de la structure DigestInfo contenant l'empreinte SHA-256 du codage DER de la structure TBSCertificate.

Générer la représentation hexadécimale de cette empreinte SHA-256 :

```
$ openssl sha256 ee-TBSCertificate.der  
SHA256(ee-TBSCertificate.der)= 45232ce57de879d9030142362a6c86186b0a1c95d8e583387  
d9032b2e13ff677
```

Créer le fichier ee-DigestInfo.asn.cnf représentant la structure DigestInfo contenant cette empreinte :

```
asn1 = SEQUENCE:digestInfo  
  
[digestInfo]  
digestAlgorithm = SEQUENCE:digestAlgorithm  
digest = FORMAT:HEX,OCTETSTRING:\  
45232ce57de879d9030142362a6c86186b0a1c95d8e583387d9032b2e13ff677
```

```
[digestAlgorithm]
algorithm = OID:sha256
parameters = NULL
```

Générer le codage DER correspondant :

```
$ openssl asn1parse -genconf ee-DigestInfo.asn.cnf -i -out ee-DigestInfo.der
  0:d=0  hl=2 l= 49 cons: SEQUENCE
  2:d=1  hl=2 l= 13 cons: SEQUENCE
  4:d=2  hl=2 l=  9 prim: OBJECT                               :sha256
 15:d=2  hl=2 l=  0 prim: NULL
 17:d=1  hl=2 l= 32 prim: OCTET STRING                      [HEX DUMP]:45232CE57DE879D903014
2362A6C86186B0A1C95D8E583387D9032B2E13FF677
```

Générer la représentation hexadécimale (découpée sur plusieurs lignes à l'aide du caractère « \ ») de la signature du codage DER obtenu :

```
$ openssl pkeyutl -sign -in ee-DigestInfo.der -inkey ca-key.pem \
| od -tx1 -An -w | tr -d " " | sed 's/$/\\/'
092fbabfd986fd029c8a223ba0f2e481603a1a4ed698c179635e2452b8b3d9ae\
61074a802f1f3338bb19a0a09665a9dcbdbcbfd3f3cc8f410e634fa2b9775631\
72b0695b17394d50e6db308d62070d3873d264dc0d203c2fd652f2ccb798cce2\
2c451ba911c6a8949ec7c07034dff6382008325eda736057f9347bd3a90d7a67\
0607d4455cf6d91c94a87937d9e3e5b95c30b448dae857a35d7b4ea5c9360e1f\
155a598966cfe8a6c2f6b553ec433deaad75fa28a89a35985723874665999f8a\
8e7d19136dad9f0c2e52a0ed94a80c0367a83f444368453088baa61322cd6595\
08dbe499438568e9767473c891fe29af0e134dafb8d682eb241f41ea77cf6e8b\
```

Cette valeur représente le contenu de la BIT STRING constituant le champ signatureValue du certificat.

Finalisation du certificat

Copier le fichier ee-TBSCertificate.asn.cnf sous le nom ee-Certificate.asn.cnf.

Remplacer la première ligne de ce fichier par la ligne suivante :

```
asn1=SEQUENCE:ee_certificate
```

Ajouter la section certificate ci-dessous (la valeur de signatureValue est celle obtenue ci-avant, à l'exclusion du dernier caractère « \ », qui doit être supprimé) :

```
[ee_certificate]
tbsCertificate = SEQUENCE:ee_tbsCertificate
signatureAlgorithm = SEQUENCE:ee_signatureAlgorithm
signatureValue=FORMAT:HEX,BITSTRING:\
092fbabfd986fd029c8a223ba0f2e481603a1a4ed698c179635e2452b8b3d9ae\
61074a802f1f3338bb19a0a09665a9dcbdbcbfd3f3cc8f410e634fa2b9775631\
72b0695b17394d50e6db308d62070d3873d264dc0d203c2fd652f2ccb798cce2\
2c451ba911c6a8949ec7c07034dff6382008325eda736057f9347bd3a90d7a67\
0607d4455cf6d91c94a87937d9e3e5b95c30b448dae857a35d7b4ea5c9360e1f\
```

4. Certificats

```
155a598966cfe8a6c2f6b553ec433deaad75fa28a89a35985723874665999f8a\  
8e7d19136dad9f0c2e52a0ed94a80c0367a83f444368453088baa61322cd6595\  
08dbe499438568e9767473c891fe29af0e134dafb8d682eb241f41ea77cf6e8b
```

Il reste uniquement à créer une section `ee_signatureAlgorithm` pour compléter la description du certificat :

```
[ee_signatureAlgorithm]  
algorithm = OID:sha256WithRSAEncryption  
parameters = NULL
```

Noter que la section `ee_signatureAlgorithm` est identique à la section `ee_signature`, comme prévu dans la section 5.1.1.2 de la RFC 5280.

Générer le certificat :

```
$ openssl asn1parse -genconf ee-Certificate.asn.cnf -i -out ee-Certificate.der
```

Pour confirmer que le certificat généré est valide, le convertir au format PEM et le vérifier en utilisant le certificat de l'autorité de certification émettrice :

```
$ openssl x509 -in ee-Certificate.der -inform DER \  
| openssl verify -CAfile ca-crt.pem  
stdin: OK
```


Chapitre 5 — Liste de certificats révoqués

Ce chapitre décrit la génération d'une liste de certificats révoqués (LCR, ou CRL pour *certificate revocation list*) par l'autorité de certification créée dans le chapitre précédent. Le profil utilisé est celui du RGS.

5.1. Émission d'une liste de certificats révoqués

Créer le fichier `ca-crl.srl`, contenant la chaîne de caractères `01`, le numéro de série en hexadécimal à inclure dans l'extension `CRLNumber` de la LCR. Ce numéro est incrémenté à chaque nouvelle génération de LCR.

Créer le fichier vide `ca-db.txt`, constituant la base de données des certificats gérés par une AC via la commande `openssl ca`. Dans le cas présent, les certificats étant générés à l'aide de la commande `openssl x509`, la base de données est initialement vide, et est enrichie par la commande `openssl ca` lors de la révocation d'un certificat.

Chaque ligne du fichier de base de données est au format suivant (type vaut `R` pour un certificat révoqué) :

```
type  date_expiration  info_revocation  num_serie  nom_fichier  objet  numero
```

Générer le fichier de configuration `ca-crl.cnf`, avec le contenu suivant :

```
[ca_crl]
database = ca-db.txt
crlnumber = ca-crl.srl

[ca_crl_ext]
authorityKeyIdentifier=keyid
```

Générer la LCR.

```
$ openssl ca -gencrl -cert ca-crt.pem -keyfile ca-key.pem -crlhours 48 \
  -md sha256 -config ca-crl.cnf -name ca_crl -crlexts ca_crl_ext \
  -out ca-crl.pem
```

```
Using configuration from ca-crl.cnf
Loading 'screen' into random state - done
```

Suite à la génération de la LCR, observer que le numéro dans le fichier `ca-crl.srl` est incrémenté de 1 (après « `FF` », la valeur suivante est « `0100` »).

Afficher la LCR.

```
$ openssl crl -in ca-crl.pem -noout -text
Certificate Revocation List (CRL):
    Version 2 (0x1)
```

5. Liste de certificats révoqués

```
Signature Algorithm: sha256WithRSAEncryption
Issuer: /C=FR/O=Mon Entreprise/OU=0002 123456789/OU=OpenSSL Root CA
Last Update: Apr  7 19:11:29 2012 GMT
Next Update: Apr  9 19:11:29 2012 GMT
CRL extensions:
    X509v3 Authority Key Identifier:
        keyid:4C:6D:87:93:82:F7:2D:2C:07:23:A2:0F:E0:71:2D:17:3F:39:F3:8
```

F

```
X509v3 CRL Number:
    1
```

No Revoked Certificates.

```
Signature Algorithm: sha256WithRSAEncryption
6a:c6:90:ac:cc:7a:93:db:6c:d6:23:44:d8:6e:a2:ad:1d:54:
...
3a:7b:32:8a
```

5.2. Révocation d'un certificat

Avant d'effectuer les opérations de cette section, il est recommandé d'effectuer une copie de sauvegarde des fichiers `ca-db.txt`, `ca-crl.srl` et `ca-crl.pem`.

Révoquer le certificat de confidentialité pour cause de compromission de clé.

```
$ openssl ca -cert ca-crt.pem -keyfile ca-key.pem -config ca-crl.cnf \
  -md sha256 -name ca_crl -revoke ee2-crt-confid.pem \
  -crl_compromise 20120616172700Z
Using configuration from ca-crl.cnf
Adding Entry with serial number 89FC7231AEC956C0 to DB for /C=FR/O=Mon Organisat
ion/OU=0002 963852741/CN=Entit\xC3\xA9 Finale
Revoking Certificate 89FC7231AEC956C0.
Data Base Updated
```

Cette commande met uniquement à jour le fichier `ca-db.txt`, mais ne génère par la LCR.

En conséquence, il ne devrait pas être nécessaire de préciser l'algorithme de hachage, mais la commande `openssl ca` l'impose.

Observer que le contenu du fichier `ca-db.txt` a été mis à jour, avec un contenu semblable au suivant :

```
R 140407201901Z 120616153042Z,keyTime,20120616172700Z 89FC7231AEC956C0
  unknown /C=FR/O=Mon Organisation/OU=0002 963852741/CN=Entit\xC3\xA9 Finale
```

Générer la LCR mise à jour puis l'afficher.

```
$ openssl ca -gencrl -cert ca-crt.pem -keyfile ca-key.pem -crlhours 48 \
  -md sha256 -config ca-crl.cnf -name ca_crl -crlxts ca_crl_ext \
  -out ca-crl.pem
Using configuration from ca-crl.cnf
```

```
$ openssl crl -in ca-crl.pem -noout -text
```

```
Certificate Revocation List (CRL):
```

```
Version 2 (0x1)
```

```
Signature Algorithm: sha256WithRSAEncryption
```

```
Issuer: /C=FR/O=Mon Entreprise/OU=0002 123456789/OU=OpenSSL Root CA
```

```
Last Update: Jun 16 15:44:42 2012 GMT
```

```
Next Update: Jun 18 15:44:42 2012 GMT
```

```
CRL extensions:
```

```
X509v3 Authority Key Identifier:
```

```
keyid:4C:6D:87:93:82:F7:2D:2C:07:23:A2:0F:E0:71:2D:17:3F:39:F3:8
```

```
F
```

```
X509v3 CRL Number:
```

```
2
```

```
Revoked Certificates:
```

```
Serial Number: 89FC7231AEC956C0
```

```
Revocation Date: Jun 16 15:37:43 2012 GMT
```

```
CRL entry extensions:
```

```
X509v3 CRL Reason Code:
```

```
Key Compromise
```

```
Invalidity Date:
```

```
Jun 16 17:27:00 2012 GMT
```

```
Signature Algorithm: sha256WithRSAEncryption
```

```
3d:76:18:27:a0:9c:e6:62:8f:f0:5d:c2:78:1d:34:e7:b4:a2:
```

```
...
```

```
84:bd:7e:35
```

Si les fichiers ont été préalablement sauvegardés, les restaurer, de manière à maintenir la validité des certificats.

5.3. Construction d'une liste de certificats révoqués

La construction d'une liste de certificats révoqués est analogue à la construction d'un certificat. Dans l'exemple ci-après, la LCR reconstruite est celle contenant le numéro de série du certificat de confidentialité, générée dans la section 4.5.

Constitution d'une structure TBSCertList

Créer le fichier `crl-tbsCertList.asn.cnf` suivant.

```
asn1 = SEQUENCE:tbsCertList
```

```
[tbsCertList]
```

```
version = INTEGER:1
```

```
signature = SEQUENCE:crl_signature
```

```
issuer = SEQUENCE:crl_issuer
```

```
thisUpdate = UTCTIME:120616154442Z
```

```
nextUpdate = UTCTIME:120618154442Z
```

5. Liste de certificats révoqués

```
revokedCertificates = SEQUENCE:revokedCertificates  
crlExtensions = EXPLICIT:0,SEQUENCE:crlExtensions
```

```
[crl_signature]  
algorithm = OID:sha256WithRSAEncryption  
parameters = NULL
```

```
[crl_issuer]  
C = SET:crl_issuer_C_RDN  
O = SET:crl_issuer_O_RDN  
OU1 = SET:crl_issuer_OU1_RDN  
OU2 = SET:crl_issuer_OU2_RDN
```

```
[crl_issuer_C_RDN]  
rdn = SEQUENCE:crl_issuer_C_ATV
```

```
[crl_issuer_C_ATV]  
type = OID:countryName  
value = PRINTABLESTRING:FR
```

```
[crl_issuer_O_RDN]  
rdn = SEQUENCE:crl_issuer_O_ATV
```

```
[crl_issuer_O_ATV]  
type = OID:organizationName  
value = PRINTABLESTRING:Mon Entreprise
```

```
[crl_issuer_OU1_RDN]  
rdn = SEQUENCE:crl_issuer_OU1_ATV
```

```
[crl_issuer_OU1_ATV]  
type = OID:organizationalUnitName  
value = PRINTABLESTRING:0002 123456789
```

```
[crl_issuer_OU2_RDN]  
rdn = SEQUENCE:crl_issuer_OU2_ATV
```

```
[crl_issuer_OU2_ATV]  
type = OID:organizationalUnitName  
value = PRINTABLESTRING:OpenSSL Root CA
```

```
[revokedCertificates]  
revokedCertificate = SEQUENCE:revokedCertificate_1
```

```
[revokedCertificate_1]  
userCertificate = INTEGER:0x89fc7231aec956c0  
revocationDate = UTCTIME:120616153743Z  
crlEntryExtensions = SEQUENCE:crlEntryExtensions_1
```

```
[crlEntryExtensions_1]  
CRLReason = SEQUENCE:CRLReason_1
```

```

invalidityDate = SEQUENCE:invalidityDate_1

[CRLReason_1]
extnID = OID:CRLReason
extnValue = OCTWRAP,ENUMERATED:1

[invalidityDate_1]
extnID = OID:invalidityDate
extnValue = OCTWRAP,GENERALIZEDTIME:20120616172700Z

[crlExtensions]
authorityKeyIdentifier = SEQUENCE:crl_authorityKeyIdentifier
crlNumber = SEQUENCE:crlNumber

[crl_authorityKeyIdentifier]
extnID = OID:authorityKeyIdentifier
extnValue = OCTWRAP,SEQUENCE:crl_authorityKeyIdentifier_seq

[crl_authorityKeyIdentifier_seq]
keyIdentifier = IMPLICIT:0,FORMAT:HEX,OCTETSTRING:\
4c6d879382f72d2c0723a20fe0712d173f39f38f

[crlNumber]
extnID = OID:crlNumber
extnValue = OCTWRAP,INTEGER:2

```

La section `crl_issuer` (et ses dépendances) a été reprise de la section `ee_issuer` du fichier `ee-Certificate.asn.cnf`, en remplaçant le préfixe `ee_` par `crl_`. De même pour la section `crl_authorityKeyIdentifier`, copiée de la section `ee_authorityKeyIdentifier` en modifiant uniquement le préfixe.

Générer la représentation DER de la structure `TBSCertList` ainsi définie :

```
$ openssl asn1parse -genconf crl-tbsCertList.asn.cnf -out crl-tbsCertList.der
```

Signature de la liste de certificats révoqués

Selon le processus de signature utilisé pour les certificats, déterminer la représentation hexadécimale de l'empreinte SHA-256 de la structure `TBSCertList` générée précédemment :

```
$ openssl sha256 crl-tbsCertList.der
SHA256(crl-tbsCertList.der)= d330d6ca5180ff1bbe87b9edcfc092cb50b96bffd537af6fb2a
07df524bde609
```

Créer le fichier `crl-DigestInfo.asn.cnf` suivant, en copiant dans le champ `digest` l'empreinte hexadécimale :

```

asn1 = SEQUENCE:digestInfo

[digestInfo]
digestAlgorithm = SEQUENCE:digestAlgorithm

```

5. Liste de certificats révoqués

```
digest = FORMAT:HEX,OCTETSTRING:\
d330d6ca5180ff1bbe87b9edcfc092cb50b96bffd537af6fb2a07df524bde609
```

```
[digestAlgorithm]
algorithm = OID:sha256
parameters = NULL
```

Générer le codage DER de la structure DigestInfo :

```
$ openssl asn1parse -genconf crl-DigestInfo.asn.cnf -i -out crl-DigestInfo.der
0:d=0 hl=2 l= 49 cons: SEQUENCE
2:d=1 hl=2 l= 13 cons: SEQUENCE
4:d=2 hl=2 l= 9 prim: OBJECT :sha256
15:d=2 hl=2 l= 0 prim: NULL
17:d=1 hl=2 l= 32 prim: OCTET STRING [HEX DUMP]:D330D6CA5180FF1BBE87B
9EDCFC092CB50B96BFFD537AF6FB2A07DF524BDE609
```

Signer le fichier DER résultant à l'aide de la clé privée de l'autorité de certification, et en obtenir la représentation hexadécimale :

```
$ openssl pkeyutl -sign -in crl-DigestInfo.der -inkey ca-key.pem \
| od -tx1 -An -w | tr -d " " | sed 's/$/\\/'
3d761827a09ce6628ff05dc2781d34e7b4a2c059cc844cdd95745d2929171670\
815ad8a81b8fb71b39d3ac97976030a4116e83a7c52e3416e27d1925b624fd05\
f1201a2488758fc14f7be440311e2d2490d1049c09fdfadbe913b37cd4204a70\
606b3cdacc5ea2665d46d44f6189d9018a80e5dc7cf87e82398bf5a8e30ce53a\
7a3458f5a48f184f5b9f83c962eabdc503289df775af605d9ac257aa4189ff52\
ebb05097a6f7db40c210ae82e24d638d01dbd4fc8df90f3f32f510ffbf8253cf\
dd85f3f27e902c7c1c456d1bca8ce3643f349bc0b60fccc31f25402eb3aa2b7e\
ba11819a2001316d3c3d409b5501ba74bb0c0b8d14a2f2ef388b4bd484bd7e35\
```

Finalisation de la liste de certificats révoqués

Copier le fichier crl-tbsCertList.asn.cnf sous le nom crl-CertificateList.asn.cnf. Y remplacer la première ligne par celle-ci :

```
asn1 = SEQUENCE:certificateList
```

Ajouter les sections suivantes, où le champ signatureValue contient la valeur de la signature numérique obtenue précédemment :

```
[certificateList]
tbsCertList = SEQUENCE:tbsCertList
signatureAlgorithm = SEQUENCE:crl_signatureAlgorithm
signatureValue = FORMAT:HEX,BITSTRING:\
3d761827a09ce6628ff05dc2781d34e7b4a2c059cc844cdd95745d2929171670\
815ad8a81b8fb71b39d3ac97976030a4116e83a7c52e3416e27d1925b624fd05\
f1201a2488758fc14f7be440311e2d2490d1049c09fdfadbe913b37cd4204a70\
606b3cdacc5ea2665d46d44f6189d9018a80e5dc7cf87e82398bf5a8e30ce53a\
7a3458f5a48f184f5b9f83c962eabdc503289df775af605d9ac257aa4189ff52\
ebb05097a6f7db40c210ae82e24d638d01dbd4fc8df90f3f32f510ffbf8253cf\
dd85f3f27e902c7c1c456d1bca8ce3643f349bc0b60fccc31f25402eb3aa2b7e\
```

ba11819a2001316d3c3d409b5501ba74bb0c0b8d14a2f2ef388b4bd484bd7e35

```
[crl_signatureAlgorithm]
algorithm = OID:sha256WithRSAEncryption
parameters = NULL
```

Générer la LCR au format DER :

```
$ openssl asn1parse -genconf crl-CertificateList.asn.cnf -i \
  -out crl-CertificateList.der
```

5. Liste de certificats révoqués

Chapitre 6 — Confidentialité — PKCS#7 et CMS

Ce chapitre s'intéresse à la protection en confidentialité de données à l'aide de *Cryptographic Message Syntax*, plus connue à l'origine sous le nom de PKCS#7 d'après le document de RSA Security qui définit cette syntaxe, reprise par l'IETF sous le nom de PKCS#7 dans la RFC 2315, et étendue sous le nom de *Cryptographic Message Syntax*, ou CMS, dans la RFC 2630 (puis RFC 3369, RFC 3852, et RFC 5652, version actuelle à la date de rédaction). PKCS#7/CMS est au cœur de la norme S/MIME (*Secure/Multipurpose Internet Mail Extensions*), qui est notamment utilisée pour sécuriser les courriels (signature électronique et confidentialité). Le format PKCS#7 originel étant encore largement répandu (le support de la norme CMS dans OpenSSL a été introduit par la version 1.0.0, en mars 2010), dans ce chapitre, le terme PKCS#7 est employé lorsqu'aucune spécificité de CMS n'est employée.

Dans le cas d'usage le plus courant pour le chiffrement, PKCS#7/CMS génère une clé symétrique (ci-dessous une clé AES-256, avec l'option `-aes-256-cbc`), chiffre les données à l'aide de cette clé (ici en mode CBC), puis chiffre la clé symétrique avec la clé publique de chacun des destinataires et inclut chaque exemplaire de la clé symétrique chiffrée dans la structure PKCS#7/CMS cible.

6.1. Chiffrement d'un fichier

Chiffrer le fichier de test `data.txt` avec le certificat de confidentialité généré ci-avant (`ee2-crt-confid.pem`), au format PKCS#7 codé en DER.

```
$ openssl smime -encrypt -in data.txt -aes-256-cbc -outform DER \
  -out data.enc.p7.der ee2-crt-confid.pem
```

La commande historique `openssl smime` gère les structures PKCS#7 uniquement. Pour manipuler des structures CMS, utiliser `openssl cms` (dont la syntaxe est très proche).

6.2. Déchiffrement d'un fichier

Déchiffrer les données contenues dans la structure obtenue, en utilisant la clé privée de déchiffrement `ee2-key.pem` associée à la clé publique de chiffrement incluse dans le certificat de confidentialité :

```
$ openssl smime -decrypt -in data.enc.p7.der -inform DER -inkey ee2-key.pem
texte en clair
```

Sous Windows il est possible de déchiffrer un fichier à l'aide d'une clé privée associée à un certificat du magasin de certificats de Windows, comme décrit ci-après.

Créer le fichier PKCS#12 rassemblant le certificat de confidentialité et la clé privée. Il n'est pas utile d'ajouter le certificat de l'AC racine si celle-ci a déjà été importée dans le magasin de certificats de Windows (cf. section 4.6).

```
$ openssl pkcs12 -export -in ee2-crt-confid.pem -inkey ee2-key.pem \
  -name "OpenSSL EE2" -out ee2-confid.p12
Loading 'screen' into random state - done
Enter Export Password:
```

Dans l'explorateur Windows, double-cliquer sur le fichier ee2-confid.p12 généré et suivre la procédure d'importation.

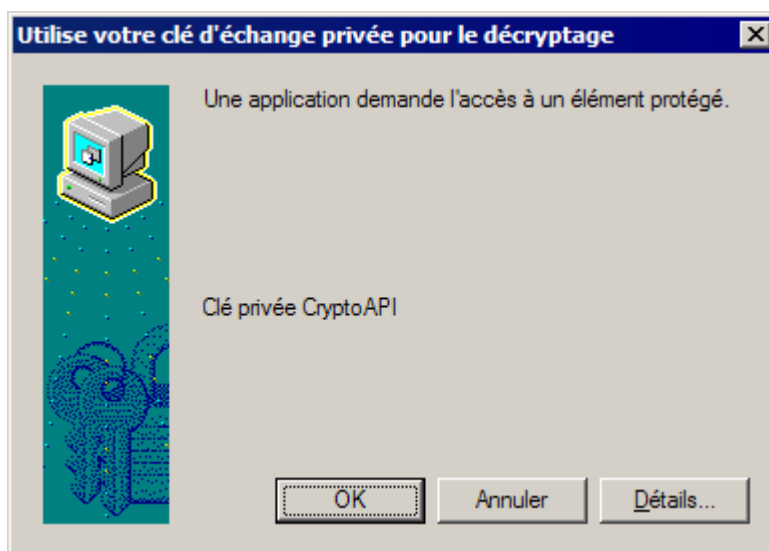
Au besoin configurer l'environnement pour utiliser le moteur cryptographique capi d'OpenSSL (cf. annexe E.1), et initier le déchiffrement du fichier chiffré, en utilisant l'option `-keyform ENGINE` et en passant en paramètre de `-inkey` une sous-chaîne de l'objet du certificat (en l'occurrence le contenu du CN).

```
$ openssl smime -decrypt -in data.enc.p7.der -inform DER -keyform ENGINE \
  -inkey "Entité Finale" -engine capi
```

Les accents et caractères spéciaux sont gérés correctement dans le paramètre passé à l'option `-inkey`.

Pour identifier la clé privée à partir du nom convivial du certificat au lieu d'une sous-chaîne de l'objet du certificat (par exemple dans le cas où le nom convivial permettrait de distinguer deux certificats portant le même objet), ajouter, dans la section du fichier de configuration concernant la configuration du moteur capi, la ligne `lookup_method=2` après la ligne `init=1`.

Si la protection renforcée de la clé privée a bien été activée lors de l'importation du fichier PKCS#12, alors une alerte Windows s'affiche, demandant l'autorisation d'utiliser la clé privée à des fins de déchiffrement.



Cliquer sur le bouton « Oui », et le déchiffrement s'effectue.

6.3. Analyse d'une structure PKCS#7/CMS

Au niveau le plus général, la syntaxe ASN.1 d'une structure PKCS#7/CMS est la suivante :

```
ContentInfo ::= SEQUENCE {
    contentType ContentType,
    content
    [0] EXPLICIT ANY DEFINED BY contentType OPTIONAL }
```

Pour des données chiffrées à l'intention d'un ou de plusieurs destinataires, le type de contenu (contentType) est pkcs7-envelopedData, qui correspond à la structure ASN.1 EnvelopedData de PKCS#7. Celle-ci contient les informations sur les destinataires et leur clé publique de chiffrement dans recipientInfos), et des données chiffrées dans encryptedContentInfo (à comparer au type de contenu EncryptedData, qui inclut uniquement les données chiffrées).

Analyser le contenu du fichier data.enc.p7.der, à l'aide de la commande openssl cms qui, contrairement à openssl smime, propose un affichage mis en forme (en combinant les options -cmsout et -print):

```
$ openssl cms -cmsout -in data.enc.p7.der -inform DER -print
CMS_ContentInfo:
  contentType: pkcs7-envelopedData (1.2.840.113549.1.7.3)
  d.envelopedData:
    version: <ABSENT>
    originatorInfo: <ABSENT>
    recipientInfos:
      d.ktri:
        version: <ABSENT>
        d.issuerAndSerialNumber:
          issuer: C=FR, O=Mon Entreprise, OU=0002 123456789, OU=OpenSSL Root CA
          serialNumber: 9942947635038607040
        keyEncryptionAlgorithm:
          algorithm: rsaEncryption (1.2.840.113549.1.1.1)
          parameter: NULL
        encryptedKey:
          0000 - 59 1c 7b 93 3a 30 17 a6-55 e5 af 7e 02 db 67   Y.{.:0..U...~..g
          ...
          00f0 - a6 bb 75 c5 09 de 8a 93-b0 c5 fc f9 69 2c e5   ...u.....i,.
          00ff - c2                                             .
      encryptedContentInfo:
        contentType: pkcs7-data (1.2.840.113549.1.7.1)
        contentEncryptionAlgorithm:
          algorithm: aes-256-cbc (2.16.840.1.101.3.4.1.42)
          parameter: OCTET STRING:
            0000 - a4 42 ee f2 d5 87 1b d0-d2 33 32 1d 36 98 25   .B.....32.6.%
            000f - de                                             .
        encryptedContent:
          0000 - ed fa 00 e0 1d d0 1a 96-59 bd f5 55 fd 7d ac   .....Y..U.}.
          000f - c8                                             .
  unprotectedAttrs:
    <EMPTY>
```

Les valeurs obtenues par le lecteur seront différentes de celles ci-dessus, la clé de chiffrement et le vecteur d'initialisation étant générés aléatoirement.

Même mis en forme par `-print`, l'affichage manque un peu de finition : numéros de version marqués absents au lieu de nuls, champ `KeyTransRecipientInfo` (correspondant à `RecipientInfo` dans la nomenclature de PKCS#7) abrégé en `d.ktri` etc. À titre d'exercice, le lecteur peut installer la bibliothèque `pyasn1`¹⁶ pour Python¹⁷, avec le module `pyasn1_modules`¹⁸ qui propose dans le sous-répertoire `tools` un outil de mise en forme d'une structure PKCS#7 au format PEM, nommé `pkcs7dump.py`. Le résultat obtenu est une vue plus fidèle de la structure (au détriment du contenu, parfois affiché en hexadécimal sans décodage DER, et des OID non convertis en nom convivial) :

```
$ openssl pkcs7 -in data.enc.p7.der -inform DER -outform PEM | \
python chemin_vers/pkcs7dump.py
ContentInfo:
  contentType=1.2.840.113549.1.7.3
  content=0x308201c7020100318201823082017e02010030663059310b300906035504061302465
...
3321d369825de8010edfa00e01dd01a9659bdf555fd7dacc8

EnvelopedData:
  version=0
  recipientInfos=RecipientInfos:
    RecipientInfo:
      version=0
      issuerAndSerialNumber=IssuerAndSerialNumber:
        issuer=Name:
          =RDNSequence:
            RelativeDistinguishedName:
              AttributeTypeAndValue:
                type=2.5.4.6
                value=0x13024652
            RelativeDistinguishedName:
              AttributeTypeAndValue:
                type=2.5.4.10
                value=0x130e4d6f6e20456e7472657072697365
            RelativeDistinguishedName:
              AttributeTypeAndValue:
                type=2.5.4.11
                value=0x130e3030303220313233343536373839
            RelativeDistinguishedName:
              AttributeTypeAndValue:
                type=2.5.4.11
                value=0x130f4f70656e53534c20526f6f74204341

        serialNumber=9942947635038607040

      keyEncryptionAlgorithm=KeyEncryptionAlgorithmIdentifier:
        algorithm=1.2.840.113549.1.1.1
        parameters=0x0500

      encryptedKey=0x591c7b933a3017a655e5af7e02db67d26a5f981c2cbc9dddbbc26c72a112ce
...
```

16. <http://sourceforge.net/projects/pyasn1/>

17. <http://www.python.org/>

18. <http://sourceforge.net/projects/pyasn1/files/pyasn1-modules/>

```
d1e4a840115f51192da6bb75c509de8a93b0c5fcf9692ce5c2
```

```
encryptedContentInfo=EncryptedContentInfo:
  contentType=1.2.840.113549.1.7.1
  contentEncryptionAlgorithm=ContentEncryptionAlgorithmIdentifier:
    algorithm=2.16.840.1.101.3.4.1.42
    parameters=0x0410a442eef2d5871bd0d233321d369825de

  encryptedContent=0xedfa00e01dd01a9659bdf555fd7dacc8
```

6.4. Déchiffrement manuel d'un fichier

La suite de cette section s'intéresse au déchiffrement manuel des données, à travers les opérations successives suivantes :

- Extraire la clé symétrique chiffrée contenue dans le champ `encryptedKey` du `recipientInfo` de l'unique destinataire.
- Déchiffrer la clé symétrique chiffrée, à l'aide de la clé privée du destinataire.
- Extraire le contenu chiffré par la clé symétrique.
- Extraire le vecteur d'initialisation.
- Déchiffrer le contenu chiffré, en utilisant la clé symétrique et le vecteur d'initialisation.

Afficher tout d'abord l'analyse ASN.1 de la structure PKCS#7, de manière à déterminer l'octet de début et la longueur des champs à extraire.

```
$ openssl asn1parse -in data.enc.p7.der -inform DER -i
 0:d=0  hl=4 l= 474 cons: SEQUENCE
 4:d=1  hl=2 l=   9 prim: OBJECT                  :pkcs7-envelopedData
15:d=1  hl=4 l= 459 cons: cont [ 0 ]
19:d=2  hl=4 l= 455 cons: SEQUENCE
23:d=3  hl=2 l=   1 prim: INTEGER                  :00
26:d=3  hl=4 l= 386 cons: SET
30:d=4  hl=4 l= 382 cons: SEQUENCE
34:d=5  hl=2 l=   1 prim: INTEGER                  :00
37:d=5  hl=2 l= 102 cons: SEQUENCE
39:d=6  hl=2 l=  89 cons: SEQUENCE
41:d=7  hl=2 l=  11 cons: SET
43:d=8  hl=2 l=   9 cons: SEQUENCE
45:d=9  hl=2 l=   3 prim: OBJECT                  :countryName
50:d=9  hl=2 l=   2 prim: PRINTABLESTRING        :FR
54:d=7  hl=2 l=  23 cons: SET
56:d=8  hl=2 l=  21 cons: SEQUENCE
58:d=9  hl=2 l=   3 prim: OBJECT                  :organizationName
63:d=9  hl=2 l=  14 prim: PRINTABLESTRING        :Mon Entreprise
79:d=7  hl=2 l=  23 cons: SET
81:d=8  hl=2 l=  21 cons: SEQUENCE
83:d=9  hl=2 l=   3 prim: OBJECT                  :organizationalUnitName
88:d=9  hl=2 l=  14 prim: PRINTABLESTRING        :0002 123456789
```

```

104:d=7  hl=2 l= 24 cons:      SET
106:d=8  hl=2 l= 22 cons:      SEQUENCE
108:d=9  hl=2 l=  3 prim:      OBJECT          :organizationalUnitName
113:d=9  hl=2 l= 15 prim:      PRINTABLESTRING :OpenSSL Root CA
130:d=6  hl=2 l=  9 prim:      INTEGER          :89FC7231AEC956C0
141:d=5  hl=2 l= 13 cons:      SEQUENCE
143:d=6  hl=2 l=  9 prim:      OBJECT          :rsaEncryption
154:d=6  hl=2 l=  0 prim:      NULL
156:d=5  hl=4 l= 256 prim:      OCTET STRING      [HEX DUMP]:591C7B933A3017A65
5E5AF7E02DB67D26A5F981C2CBC9DDDBBC26C72A112CEBFDE42EC86C8CC2E8F180348A0EC0288675
...
0C5FCF9692CE5C2
 416:d=3  hl=2 l= 60 cons:      SEQUENCE
 418:d=4  hl=2 l=  9 prim:      OBJECT          :pkcs7-data
 429:d=4  hl=2 l= 29 cons:      SEQUENCE
 431:d=5  hl=2 l=  9 prim:      OBJECT          :aes-256-cbc
 442:d=5  hl=2 l= 16 prim:      OCTET STRING      [HEX DUMP]:A442EEF2D5871BD0D
233321D369825DE
 460:d=4  hl=2 l= 16 prim:      cont [ 0 ]

```

Se reporter aux affichages mis en forme obtenus précédemment pour mieux visualiser la correspondance entre le décodage ASN.1 brut et la structure PKCS#7.

Extraire la clé symétrique chiffrée, qui démarre dans le cas ci-dessus à l'octet 160 (le champ `encryptedKey` démarre à l'octet 156, et son contenu démarre après l'en-tête du champ, soit `hl=4` octets plus loin), et a une longueur de `l=256` octets (sans surprise, puisque la taille du bloc RSA est égale à la taille de la clé RSA, soit 2048 bits, et que la taille de la clé AES-256 à chiffrer est de taille inférieure à la taille du bloc, donc un seul bloc de 256 octets suffit pour stocker la clé symétrique chiffrée).

```
$ openssl asn1parse -inform DER -in data.enc.p7.der -offset 160 -length 256 \
-out data.enc.p7.encryptedkey.bin -noout
```

Déchiffrer la clé symétrique chiffrée, à l'aide de la clé privée du destinataire (`ee2-key.pem`) :

```
$ openssl pkeyutl -decrypt -in data.enc.p7.encryptedkey.bin -inkey ee2-key.pem \
-out data.enc.p7.decryptedkey.bin
```

Inutile ici d'employer l'option `-pkeyopt rsa_padding_mode`, le *padding* employé est celui par défaut, c'est-à-dire celui d'EME-PKCS1-v1_5 de PKCS#1.

Extraire également le contenu chiffré (`encryptedContent`), d'une longueur de 16 octets à partir de l'octet 462 (= 460 + 2).

```
$ openssl asn1parse -inform DER -in data.enc.p7.der -i -offset 462 -length 16 \
-out data.enc.p7.encryptedcontent.bin -noout
```

Dans le cas où l'algorithme AES est utilisé par PKCS#7/CMS en mode CBC, le champ (`contentEncryptionAlgorithm`), de type `AlgorithmIdentifier` (dont la syntaxe est donnée ci-dessous), doit contenir l'identifiant de l'algorithme (l'OID d'AES-256-CBC dans le champ `algorithm`) et le vecteur d'initialisation (dans `parameters`).

```
AlgorithmIdentifier ::= SEQUENCE {
    algorithm OBJECT IDENTIFIER,
    parameters ANY DEFINED BY algorithm OPTIONAL
}
```

Extraire le vecteur d'initialisation (début à l'octet 444 = 442 + 2, longueur de 16 octets) :

```
$ openssl asn1parse -inform DER -in data.enc.p7.der -i -offset 444 -length 16 \
  -out data.enc.p7.iv.bin -noout
```

Convertir la clé AES-256 et le vecteur d'initialisation en leur représentation hexadécimale. Pour le vecteur d'initialisation, il suffit de copier la chaîne de caractères suivant [HEX DUMP] dans le dernier OCTET STRING du résultat de la commande `openssl asn1parse` initiale. Une autre option, basée sur les commandes `od` et `tr` est proposée pour convertir des données binaires en leur représentation hexadécimale :

```
$ od -An -tx1 data.enc.p7.iv.bin | tr -d "\r\n "
a442eef2d5871bd0d233321d369825de
```

```
$ od -An -tx1 data.enc.p7.decryptedkey.bin | tr -d "\r\n "
decbe54a194b77363c591fc747cd08eba8ba75e654d28b657b5b50815e68cfdc
```

Déchiffrer enfin les données du fichier `data.enc.p7.encryptedcontent.bin` à l'aide d'`openssl pkeyutl` et des représentations hexadécimales de la clé AES-256 et du vecteur d'initialisation.

```
$ openssl aes-256-cbc -d -in data.enc.p7.encryptedcontent.bin \
  -K decbe54a194b77363c591fc747cd08eba8ba75e654d28b657b5b50815e68cfdc \
  -iv a442eef2d5871bd0d233321d369825de
texte en clair
```

Le résultat obtenu est bien identique au texte en clair initial.

6.5. Spécificités de CMS

L'option `-keyid` de la commande `openssl cms`, spécifique à CMS par rapport à PKCS#7 dans le cadre du chiffrement, permet de référencer les certificats des destinataires par leur champ `subjectKeyIdentifier` (s'il est présent dans le certificat, ce qui est le cas pour les certificats RGS) plutôt que par DN de l'émetteur et numéro de série. Noter qu'avec cette option, le champ `version` de `KeyTransRecipientInfo` est valué à 2 (au lieu de 0 pour PKCS#7).

```
$ openssl cms -encrypt -in data.txt -aes-256-cbc -outform DER \
  -keyid ee2-crt-confid.pem | openssl cms -cmsout -inform DER -print
CMS_ContentInfo:
  contentType: pkcs7-envelopedData (1.2.840.113549.1.7.3)
```

```

d.envelopedData:
  version: <ABSENT>
  originatorInfo: <ABSENT>
  recipientInfos:
    d.ktri:
      version: 2
      d.subjectKeyIdentifier:
        0000 - 56 50 65 3f af 52 89 e6-ff 4f 1b d7 7b cd 74  VPe?.R...0...{.t
        000f - e6 87 33 19 9f                                ..3..
      keyEncryptionAlgorithm:

```

...

L'autre fonctionnalité spécifique à la production de capsules CMS de type EnvelopedData de la commande `openssl cms`, est la possibilité de chiffrer la clé symétrique à l'aide d'une clé symétrique AES, laquelle est supposée être connue des destinataires. La première clé est dite enrobée — *wrapped* en anglais — par la seconde, ce qui est reflété dans l'identifiant et le nom de l'algorithme de chiffrement symétrique, comme illustré ci-dessous (et cf. la RFC 3565 sur l'utilisation d'AES avec CMS). Ce cas d'usage est rare.

```

$ openssl rand -hex 32
dd91174bf24b73d3e995d47c7fa649f974dc32b048a77495d4ce8876db4bf6c7

$ openssl cms -encrypt -in data.txt -aes-256-cbc -outform DER \
  -secretkey dd91174bf24b73d3e995d47c7fa649f974dc32b048a77495d4ce8876db4bf6c7 \
  -secretkeyid 0123 | openssl cms -cmsout -inform DER -print
CMS_ContentInfo:
  contentType: pkcs7-envelopedData (1.2.840.113549.1.7.3)
  d.envelopedData:
    version: <ABSENT>
    originatorInfo: <ABSENT>
    recipientInfos:
      d.kekri:
        version: 4
        kekid:
          keyIdentifier:
            0000 - 01 23                                .#
          date: <ABSENT>
          other: <ABSENT>
        keyEncryptionAlgorithm:
          algorithm: id-aes256-wrap (2.16.840.1.101.3.4.1.45)
          parameter: <ABSENT>
        encryptedKey:
          0000 - ef af da 73 6d 03 92 db-57 57 16 c6 97 1d e8  ...sm...WW.....

```

...

Chapitre 7 — Confidentialité — XML Encryption

La norme XML Encryption (souvent appelé « XML Enc ») a été conçue par le W3C pour pouvoir encapsuler des données (XML ou non) chiffrées dans un document XML, par exemple au sein d'un message SOAP dans le cas de *web services* (services web).

Le lecteur intéressé pourra lire, dans cette présentation¹⁹, formalisée par ce document²⁰, comment, dans certains contextes d'utilisation de XML Encryption, et notamment pour chiffrer des messages de type *web service*, les caractéristiques de XML Encryption constituent une faille permettant de déchiffrer un texte chiffré. Cette faille résulte essentiellement de l'associativité de l'opération ou-exclusif employée par le mode CBC, et du caractère prévisible de la structure d'un message XML et des caractères qu'il contient (en particulier le délimiteur de balises <).

Pour protéger la confidentialité de données, XML Encryption substitue à celles-ci un élément XML `xenc:EncryptedData` (le préfixe `xenc:` désigne ci-après l'espace de nommage — ou *namespace* — `http://www.w3.org/2001/04/xmlenc#`), contenant obligatoirement les données chiffrées (`xenc:CipherData`), souvent l'algorithme de chiffrement (`xenc:EncryptionMethod`) et les informations d'identification de la (ou des) clé(s) de chiffrement (`ds:KeyInfo`, où le préfixe `ds:` représente l'espace de nommage `http://www.w3.org/2000/09/xmldsig#`), et quelquefois des informations complémentaires (`xenc:EncryptionProperties`).

Parmi les modes d'utilisation proposés par XML Encryption, celui retenu dans les exemples ci-après consiste à chiffrer (en mode CBC) les données à protéger avec une clé symétrique (AES-256), elle-même chiffrée par la clé publique (RSA) du destinataire, et à inclure clé chiffrée et données chiffrées dans l'élément `xenc:EncryptedData` résultant.

Cette section s'appuie sur l'outil en ligne de commande `xmlsec` issu de XML Security Library, cet outil étant l'équivalent XML de la commande `openssl smime` ou `openssl cms` du monde binaire, aussi bien pour la fonction de confidentialité que pour la fonction de signature électronique (abordée dans le prochain chapitre).

La commande `xmlenc --encrypt` fonctionne sur la base d'un fichier d'entrée (XML ou non) et d'un fichier dit *template* (ou modèle) représentant le squelette de l'élément `xenc:EncryptedData` à constituer pour remplacer l'élément à chiffrer, lequel peut être le fichier d'entrée dans son intégralité s'il s'agit d'un fichier non XML, ou tout ou partie d'un fichier XML.

19. https://www.owasp.org/images/5/5a/07A_Breaking_XML_Signature_and_Encryption_-_Juraj_Somorovsky.pdf

20. <http://www.nds.rub.de/media/nds/veroeffentlichungen/2011/10/22/HowToBreakXMLenc.pdf>

7.1. Chiffrement d'un fichier non XML

Créer le fichier modèle `xenc-tmpl-ki-certificate.xml` (`ki-certificate` indique que l'élément `ds:KeyInfo` référencera la clé publique de confidentialité via son certificat), avec le contenu suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<xenc:EncryptedData
  xmlns:xenc="http://www.w3.org/2001/04/xmenc#"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  <xenc:EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmenc#aes256-cbc"/>
  <ds:KeyInfo>
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod
        Algorithm="http://www.w3.org/2001/04/xmenc#rsa-1_5"/>
      <ds:KeyInfo>
        <ds:X509Data>
          <ds:X509Certificate/>
        </ds:X509Data>
      </ds:KeyInfo>
    <xenc:CipherData>
      <xenc:CipherValue/>
    </xenc:CipherData>
  </xenc:EncryptedKey>
</ds:KeyInfo>
<xenc:CipherData>
  <xenc:CipherValue/>
</xenc:CipherData>
</xenc:EncryptedData>
```

Les trois nœuds fils de `xenc:EncryptedData` sont les éléments évoqués ci-dessus :

- `xenc:EncryptionMethod` identifie l'algorithme de chiffrement des données, en l'occurrence AES-256 en mode CBC.
- `ds:KeyInfo` référence la clé de chiffrement AES-256, comme décrit ci-après.
- `xenc:CipherData` contient les données chiffrées ou plus exactement, dans le cas du chiffrement par bloc (ce qui est le cas de CBC), cet élément contient la concaténation du vecteur d'initialisation et des données chiffrées.

L'élément `ds:KeyInfo` sous `xenc:EncryptedData` contient un seul nœud fils, `xenc:EncryptedKey`. Cet élément a un schéma très proche de celui de `xenc:EncryptedData` mais est prévu pour contenir des clés de chiffrement (à l'exclusion de tout autre type de donnée) chiffrées, en l'occurrence la clé symétrique AES-256. Il contient ainsi les éléments suivants :

- `xenc:EncryptionMethod`, qui identifie l'algorithme de chiffrement de la clé symétrique, RSA (en mode PKCS#1 version 1.5).

- `ds:KeyInfo` référence la clé publique de confidentialité du destinataire. Comme celle-ci est contenue dans un certificat X.509, il a été choisi de la représenter par des données X.509 (`ds:X509Data`), et plus précisément par le certificat lui-même (`ds:X509Certificate`), d'autres possibilités étant proposées plus loin.
- `xenc:CipherData` contient la clé chiffrée.

Dans le fichier squelette `xenc-tmpl-ki-certificate.xml`, les trois éléments laissés vides sont à renseigner par `xmlenc` lors de l'opération de chiffrement :

- `ds:Certificate` contiendra le codage Base64 du codage DER du certificat de confidentialité contenant la clé publique avec laquelle a été chiffrée la clé symétrique de chiffrement (ou clé de session dans la terminologie de `xmlsec`).
- Le premier élément `xenc:CipherData`, sous `xenc:EncryptedKey`, contiendra le codage Base64 de la clé symétrique de chiffrement chiffrée

Chiffrer le fichier `data.txt` créé au début de ce document, à l'aide d'une clé de session AES-256, elle-même chiffrée avec la clé publique générée en début de chapitre. Ajouter l'option `--output data.xenc-ki-certificate.xml` avant le nom du fichier *template* pour sauvegarder

```
$ xmlsec --encrypt --binary-data data.txt --pubkey-cert-pem ee2-crt-confid.pem \
  --session-key aes-256 --output data.xenc-ki-certificate.xml \
  xenc-tmpl-ki-certificate.xml
```

Le fichier `data.xenc-ki-certificate.xml` résultant est le suivant (au contenu des champs `CipherValue` près, la clé de session étant générée aléatoirement) :

```
<?xml version="1.0" encoding="UTF-8"?>
<xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <xenc:EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc"/>
  <ds:KeyInfo>
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
      <ds:KeyInfo>
        <ds:X509Data>
          <ds:X509Certificate>
MIID9jCCAt6gAwIBAgIJAIIn8cjGuyVbAMAOGCSqGSIB3DQEBCwUAMFkxCzAJBgNV
...
Xu2IRj0LE3ZRthFl+PLDZeaKDC10q/DEtJAV/K25DKzEJ9/Fin0nhGGPHHwjkuuj
y2/ys5JBXFMB0w==</ds:X509Certificate>
          </ds:X509Data>
        </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>
j+eQRCIXznXjvHnJS3dK+a/UGeuXlmA+JV6B/Rl5053XG/zjvmY0qv2NUb7fxM3S
T/c0xy8RV9T83P6fYtFcELpY+sas0kHdFqBAvCihAYNd/y8MIz/wv+wM7iCiMdU6
tKGvDs3L6ATYe93dxt8YEMPQj9zo4G3FdZMRHGdr2Y8mCORVKQ9GyBuPymi6ZGKr
```

```
2gQXzSeI+SjCMcpsRMw3tr/yLRqwf0gzfqDigr70GLJxwJl5uBUiMwItXG+zvPVA
HApPMFyBbGQpA9tnmHYVQ6h3yqivlzzKM+vGzcbbgFcAt/U68ndPn4FlELM9sIXM
cyM6Hmo58yS3e3HJQS5ihQ==
  </xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedKey>
</ds:KeyInfo>
<xenc:CipherData>
  <xenc:CipherValue>
    9f4cxjwvWx2yP6amxRJzwgscq4w+Yv3yA+B2Fje9JRs=
  </xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedData>
```

Le résultat a été remis en page pour tenir compte des contraintes d'affichage de ce document, mais ces modifications n'impactent nullement l'intégrité du contenu, le format XML et le codage Base64 étant très permissifs quant à l'utilisation des espaces (au sens large, incluant les tabulations et les retours chariot).

7.2. Déchiffrement d'un fichier

Déchiffrer ce fichier, en notant que `xmlsec` a besoin de la chaîne de certification (option `--trusted-pem`, voire `--untrusted-pem`) pour vérifier le certificat de chiffrement, sans quoi le résultat du déchiffrement sera assorti d'un message d'erreur (`unable to get local issuer certificate`).

```
$ xmlsec --decrypt --privkey-pem ee2-key.pem --trusted-pem ca-crt.pem \
  data.xenc-ki-certificate.xml
texte en clair
```

La commande `xmlsec` ne permet pas d'utiliser le moteur `capi` d'OpenSSL.

7.3. Déchiffrement manuel d'un fichier

Le déchiffrement manuel d'un fichier XML Encryption s'appuie sur l'outil en ligne de commande `XMLStarlet`²¹, dont la commande `xml sel` permet notamment d'extraire les valeurs de nœuds XML à partir d'un chemin XPath.

L'extraction manuelle de ces valeurs est évidemment possible, mais l'objectif secondaire ici est d'ajouter un outil à l'arsenal du lecteur.

21. <http://xmlstar.sourceforge.net/>

Pour illustrer la commande `xml sel`, voici comment afficher la valeur du nœud `xenc:CipherData` contenant le vecteur d'initialisation et la clé de session chiffrée par la clé publique du destinataire :

```
$ xml sel -N enc=http://www.w3.org/2001/04/xmlenc# \
-N ds=http://www.w3.org/2000/09/xmldsig# -t -v \
/enc:EncryptedData/ds:KeyInfo/enc:EncryptedKey/enc:CipherData/enc:CipherValue \
data.xenc-ki-certificate.xml
j+eQRCIXznXjvHnJS3dK+a/UGeuXlma+JV6B/Rl5053XG/zjvmY0qv2NUb7fxM3S
T/cOxy8RV9T83P6fYtFcELpY+sas0kHdFqBAVCihAYNd/y8MIz/wv+wM7iCiMdU6
tKGvDs3L6ATYe93dxt8YEMPQj9zo4G3FdZMRHGdr2Y8mCORVKQ9GyBuPymi6ZGKr
2gQXzSeI+SjCMcpsRMw3tr/yLRqwf0gzfqDigr7OGLJxwJl5uBUiMwItXG+zvPVA
HApPMFyBbGQpA9tnmHYVQ6h3yqivlzzKM+vGzcbbgFcAt/U68ndPn4FlELM9sIXM
cyM6Hmo58yS3e3HJQS5ihQ==
```

L'option `-N` permet de déclarer un espace de nommage, `-t` introduit les traitements à appliquer, et le paramètre de l'option `-v` définit le chemin XPath du nœud dont la valeur est à extraire.

La commande `xml sel` applique une feuille de style XSLT, dont chaque élément `xsl:template` (le préfixe `xsl:` représente ici l'espace de nommage `http://www.w3.org/1999/XSL/Transform`) est généré par une option `-t`. Dans le cas ci-dessus, l'option `-v` fait appel à la fonction XSLT `xsl:value-of`. L'option globale `-C` permet d'afficher la XSLT intermédiaire appliquée (rarement optimale mais toujours efficace !).

La bibliothèque `libxml2`, utilisée par `xmlsec` et `xml`, inclut un outil en ligne de commande, `xmllint`, permettant de manipuler des structures XML, et proposant un mini-shell pour effectuer des opérations en mode interactif. L'affichage de la valeur d'un nœud par son chemin XPath par le shell s'effectue de la manière suivante :

```
$ xmllint --shell data.xenc-ki-certificate.xml
/ > setns enc=http://www.w3.org/2001/04/xmlenc#
/ > setns ds=http://www.w3.org/2000/09/xmldsig#
/ > cat /enc:EncryptedData/ds:KeyInfo/enc:EncryptedKey/enc:CipherData/enc:Cipher
Value/text()
-----
j+eQRCIXznXjvHnJS3dK+a/UGeuXlma+JV6B/Rl5053XG/zjvmY0qv2NUb7fxM3S
T/cOxy8RV9T83P6fYtFcELpY+sas0kHdFqBAVCihAYNd/y8MIz/wv+wM7iCiMdU6
tKGvDs3L6ATYe93dxt8YEMPQj9zo4G3FdZMRHGdr2Y8mCORVKQ9GyBuPymi6ZGKr
2gQXzSeI+SjCMcpsRMw3tr/yLRqwf0gzfqDigr7OGLJxwJl5uBUiMwItXG+zvPVA
HApPMFyBbGQpA9tnmHYVQ6h3yqivlzzKM+vGzcbbgFcAt/U68ndPn4FlELM9sIXM
cyM6Hmo58yS3e3HJQS5ihQ==
```

Pour sauvegarder la valeur du nœud dans un fichier, utiliser, dans la suite de la session shell ci-dessus :

```
/ > cd /enc:EncryptedData/ds:KeyInfo/enc:EncryptedKey/enc:CipherData/enc:CipherV
alue/text()
text > write data.xenc-ki-certificate.encryptedkey.b64
```

La valeur intermédiaire extraite n'a pas d'intérêt particulier codée en Base64, et peut donc être transmise à `openssl base64` sans fichier intermédiaire pour obtenir la valeur finale :

```
$ xml sel -N enc=http://www.w3.org/2001/04/xmlenc# \
-N ds=http://www.w3.org/2000/09/xmldsig# -t -v \
/enc:EncryptedData/ds:KeyInfo/enc:EncryptedKey/enc:CipherData/enc:CipherValue \
```

```
data.xenc-ki-certificate.xml \
| openssl base64 -d -out data.xenc-ki-certificate.encryptedkey.bin
```

Pour passer par un fichier intermédiaire, utiliser le caractère de redirection > pour écrire le résultat de la commande `xml sel` dans un fichier.

Les valeurs codées en Base64 par `xmlsec` sont découpées en lignes de 64 caractères. Cette limite de 64 caractères était initialement imposée dans la RFC 1421 qui définit le codage Base64, avant d'être relevée à 76 caractères par MIME dans la RFC 2045. Ces limites se retrouvent notamment dans OpenSSL, qui découpe ses codages Base64 en lignes de 64 caractères par souci de compatibilité, et refuse de traiter des lignes de Base64 de plus de 76 caractères. Les valeurs codées en Base64 dans les structures XML sont souvent représentées par le type `base64Binary` défini par la norme XML Schema. Ce type recommande une limite de 76 caractères sans l'imposer, et il n'est pas rare d'avoir à traiter des nœuds XML contenant des données codées en Base64 sur une seule ligne de longueur (parfois très largement) supérieure à 76 caractères, ce qui n'est pas du goût de tous les décodeurs Base64, à commencer par OpenSSL. La commande UNIX/GNU/Gnuwin32 `fold -w 64` permet de découper une telle ligne en lignes de 64 caractères.

Extraire le contenu chiffré codé en Base64 et le décoder :

```
$ xml sel -N enc=http://www.w3.org/2001/04/xmlenc# \
-t -v /enc:EncryptedData/enc:CipherData/enc:CipherValue \
data.xenc-ki-certificate.xml \
| openssl base64 -d -out data.xenc-ki-certificate.cipherdata.bin
```

Le contenu obtenu concaténant le vecteur d'initialisation et les données chiffrées, ces deux éléments doivent être séparés.

Le vecteur d'initialisation a une taille égale à la taille d'un bloc de l'algorithme symétrique de chiffrement, c'est-à-dire 128 bits pour AES, soit 16 octets, qui peuvent être extraits par la commande UNIX/GNU/GnuWin `head` (via `head -c 16`). Le déchiffrement de la clé de chiffrement s'effectuant à l'aide de la commande `openssl aes-256-cbc`, qui attend des valeurs hexadécimales pour le vecteur d'initialisation et la clé AES-256 (cf. chapitre 1), il faut obtenir la représentation hexadécimale du vecteur d'initialisation, ce qui est par exemple possible en utilisant `od -An -tx1` pour obtenir une représentation mise en forme (indentée et sur éventuellement sur plusieurs lignes), à filtrer par `tr -d "\n\r "` qui supprime espaces et retours chariot.

```
$ head -c 16 data.xenc-ki-certificate.cipherdata.bin | od -An -tx1 \
| tr -d "\n\r "
f5fe1cc63c2f5b1db23fa6a6c51273c2
```

Extraire les données chiffrées, situées après les 16 premiers octets, à l'aide de la commande UNIX/GNU/GnuWin `dd` :

```
$ dd bs=1 skip=16 if=data.xenc-ki-certificate.cipherdata.bin \
of=data.xenc-ki-certificate.decryptedcontent.bin
16+0 enregistrements lus.
16+0 enregistrements écrits.
16 bytes (16 B) copied, 0 seconds, Infinity B/s
```

Déchiffrer à présent la clé de session chiffrée, en utilisant la clé privée du destinataire, et en obtenir la représentation hexadécimale attendue par `openssl aes-cbc-256`, comme précédemment :

```
$ openssl pkeyutl -decrypt -in data.xenc-ki-certificate.encryptedkey.bin \
  -inkey ee2-key.pem | od -An -tx1 | tr -d "\n\r "
50b858866904ede6d01f1d09b4bead9595d6a280e4cfc1c7e8373f50090f959b
```

Tenter de déchiffrer le contenu chiffré en utilisant la clé de session et le vecteur d'initialisation :

```
$ openssl aes-256-cbc -d -iv f5fe1cc63c2f5b1db23fa6a6c51273c2 \
  -K 50b858866904ede6d01f1d09b4bead9595d6a280e4cfc1c7e8373f50090f959bc29 \
  -in data.xenc-ki-certificate.decryptedcontent.bin
bad decrypt
6960:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad decrypt:.\
crypto\evp\evp_enc.c:548:
```

Le déchiffrement n'échoue pas à cause d'une clé ou d'un vecteur d'initialisation incorrect mais à cause de l'impossibilité pour OpenSSL de décoder le *padding* mis en œuvre par XML Encryption, qui fixe uniquement le dernier octet au nombre d'octets de remplissage et les autres octets de remplissage à une valeur aléatoire, contrairement au *padding* standard PKCS#5 qui fixe les n derniers octets au nombre n d'octets de remplissage.

Le déchiffrement peut réussir ci-dessus si les octets de remplissage aléatoires prennent tous la valeur n , ce qui se produit avec une probabilité de 1 sur 256^{n-1} .

Déchiffrer les données en désactivant le décodage du *padding* et en injectant le résultat dans la commande `od` :

```
$ openssl aes-256-cbc -d -iv f5fe1cc63c2f5b1db23fa6a6c51273c2 \
  -K 50b858866904ede6d01f1d09b4bead9595d6a280e4cfc1c7e8373f50090f959bc29 \
  -in data.xenc-ki-certificate.decryptedcontent.bin -nopad | od -tx1z
0000000 74 65 78 74 65 20 65 6e 20 63 6c 61 69 72 59 02 >texte en clairY.<
0000020
```

Le texte en clair est bien rétabli, et les octets de *padding* 0x59 0x02 sont visualisés (avec le dernier octet 0x02 indiquant le nombre d'octets de *padding* à ignorer).

7.4. Chiffrement d'un élément XML

Le chiffrement d'un élément XML s'appuie sur un fichier squelette semblable à celui utilisé pour le chiffrement d'un fichier non XML, mais l'élément `EncryptedData` contient un attribut `Type` dont la valeur (à savoir `http://www.w3.org/2001/04/xmlenc#Element`) indique que les données à chiffrées sont un élément XML.

Dans le fichier squelette ci-dessous (nommé `xenc-tmpl-element-ki-ski.xml`), il a été choisi de référencer le certificat de chiffrement dans la structure finale par la valeur de son champ `SubjectKeyIdentifier` en utilisant l'élément `X509SKI`.

Il est également possible de référencer le certificat par le DN de son émetteur et son numéro de série (élément `X509IssuerSerial`) et/ou par le DN de son objet (`X509SubjectName`), comme défini dans la section 4 de [XML-DSIG]²².

```
<?xml version="1.0" encoding="UTF-8"?>
<xenc:EncryptedData
  xmlns:xenc="http://www.w3.org/2001/04/xmenc#"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  Type="http://www.w3.org/2001/04/xmenc#Element">
  <xenc:EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmenc#aes256-cbc"/>
  <ds:KeyInfo>
  <xenc:EncryptedKey>
  <xenc:EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmenc#rsa-1_5"/>
  <ds:KeyInfo>
  <ds:X509Data>
  <ds:X509SKI/>
  </ds:X509Data>
  </ds:KeyInfo>
  <xenc:CipherData>
  <xenc:CipherValue/>
  </xenc:CipherData>
  </xenc:EncryptedKey>
  </ds:KeyInfo>
  <xenc:CipherData>
  <xenc:CipherValue/>
  </xenc:CipherData>
  </xenc:EncryptedData>
```

Créer ensuite le fichier de test `data.xml` suivant.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<data>Texte en clair</data>
</root>
```

Procéder au chiffrement du nœud dont le chemin XPath est `/root/data`, en utilisant l'option `--node-xpath` de `xmlsec` pour spécifier ce chemin. Le contenu de ce nœud sera remplacé par une arborescence XML correspondant au fichier squelette.

```
$ xmlsec --encrypt --xml-data data.xml --pubkey-cert-pem ee2-crt-confid.pem
  --session-key aes-256 --node-xpath /root/data xenc-tmpl-element-ki-ski.xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <EncryptedData xmlns="http://www.w3.org/2001/04/xmenc#"
    Type="http://www.w3.org/2001/04/xmenc#Element">
    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmenc#aes256-cbc"/>
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <EncryptedKey xmlns="http://www.w3.org/2001/04/xmenc#"
        <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmenc#rsa-1_5"/>
```

22. <http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/#sec-X509Data>


```

<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
  <X509Data>
    <X509SKI>V1BlP69Sieb/TxvXe8105oczGZ8=</X509SKI>
  </X509Data>
</KeyInfo>
<CipherData>
  <CipherValue>
Dsjod0+WEHWY0t8MYxUZeVmpba9ByvM5wzB73HxDtcVIqSRzxW2YH1wpSvBuLlps
i5qiB7U9rybQOebulZ2fFyTooxB1JMMJAItmDBGwgBNbyFTPxjDBTpNQRlM9RUV3
EscmZ0KLlUSqmikx3DYh4M7pawmAkzYBiIhh3/5jpL3yWhnEHSjq8Q3Xdv6ysOF
Hlr4gS+KxKudNn8xlBj+MR0yxHK9WBdnbL5TkKnC9HfNM1hqmJUgrwIv60QtZs38
tGcHiV9ouP2DJyGSc0hMtdDBVN5x5VMs700mosTpt70vKA0q0w0r4PUWnA9ABns+
62wLfUxyB6qcfhJ86p3+AQ==
  </CipherValue>
</CipherData>
</EncryptedKey>
</KeyInfo>
<CipherData>
  <CipherValue>
Eaar+WnM3H/t0hVC9ynXzEs0QbTquYQ7Z8Kt1AkB/7KlHRAyiCmDOS/TLWIA2kuj
  </CipherValue>
</CipherData>
</EncryptedData>
</root>

```

Pour chiffrer la valeur de l'élément (c'est-à-dire le contenu textuel) au lieu de l'élément entier, utiliser le chemin XPath `/root/data/text()` en paramètre de `--node-xpath` :

```

$ xmlsec --encrypt --xml-data data.xml --pubkey-cert-pem ee2-crt-confid.pem \
--session-key aes-256 --node-xpath /root/data/text() \
xenc-tmpl-element-ki-ski.xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <data><xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    Type="http://www.w3.org/2001/04/xmlenc#Element">
    ...
  </xenc:EncryptedData></data>
</root>

```


Chapitre 8 — Signature électronique — PKCS#7/CMS

La structure de données *Cryptographic Message Syntax* présentée dans le chapitre précédent avec les structures PKCS#7 et CMS, permet d'inclure des données signées, et est adaptée à la signature de données binaires (au sens large, c'est-à-dire par opposition aux données structurées au format XML). Elle est à la base des formats de signature électronique avancée CAdES et PAdES.

Indépendamment du format de signature électronique, une signature et les données faisant l'objet de la signature peuvent coexister de plusieurs manières :

- La signature et les données sont dans deux fichiers séparés : la signature est dite *détachée*, et contient parfois — en plus de l'empreinte des données — une référence technique au contenu signé, à l'exemple d'un URI (*uniform resource identifier*, ou identifiant de ressource uniforme, dont les URL usuelles sont un cas particulier).
- La structure de données constituant la signature électronique contient les données : la signature est *enveloppante*.
- Les données englobent la signature : la signature est *enveloppée*.

Une structure PKCS#7/CMS peut contenir les données signées pour produire une signature enveloppante. Sinon elle peut être détachée ou, par exemple dans le cas de PAdES, enveloppée.

8.1. Signature d'un fichier

Signer le fichier de test `data.txt` avec le certificat d'authentification et signature généré dans la section 4.3 (`ee-crt-authsig.pem`), au format PKCS#7 détaché, codé en DER, en utilisant l'algorithme de signature RSA (implicite) avec SHA-256.

```
$ openssl smime -sign -in data.txt -signer ee-crt-authsig.pem \
  -inkey ee-key.pem -outform DER -out sig-p7.der -md sha256 \
  -nosmimcap
```

L'option `-nosmimcap` est non documentée pour la commande `openssl smime` (elle l'est pour `openssl cms`) : elle évite d'inclure l'attribut `SMIMECapabilities`, spécifique à S/MIME (cf. RFC 3851, section 2.5.2) dans les attributs signés.

Produire une signature PKCS#7 enveloppante avec le même fichier de test, à l'aide de l'option `-nodetach` :

```
$ openssl smime -sign -in data.txt -signer ee-crt-authsig.pem \
  -inkey ee-key.pem -outform DER -nodetach -out data.sig-enveloping.p7.der \
  -md sha256 -nosmimcap
```

Sous Windows il est possible de signer un fichier à l'aide d'une clé privée associée à un certificat du magasin de certificats de Windows. Importer le certificat d'authentification et signature dans le magasin de certificats de Windows si cela n'a pas déjà été fait (cf. section 4.6). Configurer au besoin l'environnement (fichier de configuration spécifique et variable d'environnement `OPENSSL_CONF`, cf. annexe E.1) pour utiliser le moteur cryptographique `capi` d'OpenSSL, puis initier la signature du fichier en utilisant les options `-keyform ENGINE` et `-engine capi`, et en passant en paramètre de `-inkey` une sous-chaîne de l'objet du certificat (ci-dessous le contenu du CN).

```
$ openssl smime -sign -in data.txt -signer ee-crt-authsig.pem \
  -keyform ENGINE -engine capi -inkey "Nom Prénom" -outform DER \
  -out sig-p7.der -md sha1 -nosmimecap
```

Les accents et caractères spéciaux sont gérés correctement dans le paramètre passé à l'option `-inkey`.

L'algorithme de hachage a été dégradé à SHA-1, le moteur `capi` d'OpenSSL supportant uniquement l'utilisation des algorithmes de hachage du fournisseur cryptographique Microsoft Enhanced Cryptographic Provider v1.0 (SHA-1 et MD5), excluant les algorithmes de hachage de la famille SHA-2, qui sont seulement supportés par le fournisseur cryptographique Microsoft Enhanced RSA and AES Cryptographic Provider, introduit dans le *service pack 3* de Windows XP.

Pour prendre en compte SHA-256, en théorie, il suffirait d'ajouter quelques lignes de code dans la fonction `capi_rsa_sign()` du fichier source `engines\e_capi.c` d'OpenSSL pour associer l'identifiant d'algorithme `NID_sha256` d'OpenSSL à l'identifiant `CALG_SHA_256` de la CryptoAPI de Microsoft, de recompiler le code source d'OpenSSL pour obtenir une nouvelle bibliothèque `capi.dll`, et de configurer l'utilisation du moteur `capi` pour utiliser le bon type de fournisseur cryptographique (`csp_type = 24` dans le fichier de configuration d'OpenSSL, correspondant au type `PROV_RSA_AES` de la CryptoAPI²³) et le bon fournisseur cryptographique (`csp_name = Microsoft Enhanced RSA and AES Cryptographic Provider`), mais en pratique, le fournisseur cryptographique est réinitialisé à celui par défaut lors du chargement de la clé privée (il s'agit sans doute d'un bug), laissant l'utilisation de SHA-256 hors de portée.

Si la protection renforcée de la clé privée a bien été activée lors de l'importation du fichier PKCS#12, alors une alerte Windows s'affiche, demandant l'autorisation d'utiliser la clé privée à des fins de signature.



23. <http://msdn.microsoft.com/en-us/library/windows/desktop/aa387447%28v=vs.85%29.aspx>

Cliquer sur le bouton « Oui », et la signature s'effectue.

8.2. Vérification d'une signature

Vérifier la signature détachée. Le fichier d'origine doit être fourni pour effectuer cette opération.

```
$ openssl smime -verify -inform DER -in sig-p7.der \
  -content data.txt -CAfile ca-crt.pem
texte en clairVerification successful
```

Le contenu signé est affiché sur la sortie standard. Pour éviter cet affichage, qui n'est pas utile dans le cas d'une signature détachée, rediriger la sortie standard vers nul (Windows) ou /dev/null (environnements de type UNIX).

Vérifier la signature enveloppante, en restituant les données enveloppées :

```
$ openssl smime -verify -inform DER -in data.sig-enveloping.p7.der \
  -CAfile ca-crt.pem -out data-restored.txt
Verification successful
```

```
$ cat data-restored.txt
texte en clair
```

8.3. Analyse d'une structure PKCS#7/CMS

Analyser le contenu du fichier constituant la signature détachée :

```
$ openssl cms -in sig-p7.der -inform DER -cmsout -print
CMS_ContentInfo:
  contentType: pkcs7-signedData (1.2.840.113549.1.7.2)
  d.signedData:
    version: 1
    digestAlgorithms:
      algorithm: sha256 (2.16.840.1.101.3.4.2.1)
      parameter: NULL
    encapContentInfo:
      eContentType: pkcs7-data (1.2.840.113549.1.7.1)
      eContent: <ABSENT>
    certificates:
      d.certificate:
        ... certificat du signataire...
    crls:
      <EMPTY>
    signerInfos:
      version: 1
      d.issuerAndSerialNumber:
        issuer: C=FR, O=Mon Entreprise, OU=0002 123456789, OU=OpenSSL Root CA
        serialNumber: 15911814405079687111
```

```

digestAlgorithm:
  algorithm: sha256 (2.16.840.1.101.3.4.2.1)
  parameter: NULL
signedAttrs:
  object: contentType (1.2.840.113549.1.9.3)
  value.set:
    OBJECT:pkcs7-data (1.2.840.113549.1.7.1)

  object: signingTime (1.2.840.113549.1.9.5)
  value.set:
    UTCTIME:Apr 27 19:19:32 2012 GMT

  object: messageDigest (1.2.840.113549.1.9.4)
  value.set:
    OCTET STRING:
      0000 - 89 bd 92 28 6d 6c 80 14-c0 60 30 b2 5f    ...(ml...`0._
      000d - 8b 40 cc 1d 56 56 d4 b3-b7 b4 83 18 74    .@..VV.....t
      001a - f5 0d 6f 55 57 f3                        ..oUW.
signatureAlgorithm:
  algorithm: rsaEncryption (1.2.840.113549.1.1.1)
  parameter: NULL
signature:
  0000 - 25 86 40 af f2 9b d2 79-76 5a 63 c8 dc 4b f0    %.@....yvZc..K.
  ...
  00f0 - 6d 15 5a af 5a aa b5 dc-a7 9d d0 f6 94 20 0f    m.Z.Z..... .
  00ff - 7f                                              .
unsignedAttrs:
  <EMPTY>

```

Le type de contenu (contentType) de la capsule PKCS#7 est pkcs7-signedData, correspondant au type ASN.1 SignedData de PKCS#7, conçu comme son nom l'indique pour stocker des données signées. Le type SignedData contient la liste des algorithmes de hachage employés par les signataires (champ digestAlgorithms), les données signées (encapContentInfo, dont le champ eContent est absent, s'agissant d'une signature détachée et les données étant donc externes à la signature), les certificats des signataires (certificates), rarement les LCR associées aux certificats (crls, vide ici) et, sous le champ signerInfos, autant d'éléments de type SignerInfo que de signataires (en réalité de signatures).

La signature portée par chaque champ signerInfo ne s'applique pas uniquement aux données à signer mais à un ensemble d'attributs (dits authentifiés dans la terminologie de PKCS#7, ou signés dans celle de CMS), parmi lesquels se trouve l'empreinte des données à signer. Chaque champ signerInfo référence le certificat du signataire par le DN de son AC émettrice et son numéro de série (issuerAndSerialNumber), ou (dans le champ sid spécifique à CMS) par son champ subjectKeyIdentifier. Il précise l'algorithme de hachage appliqué aux données à signer (digestAlgorithm), les attributs authentifiés (champ authenticatedAttributes) ou signés (champ signedAttrs), l'algorithme de signature ou de chiffrement asymétrique de l'empreinte (signatureAlgorithm), la signature numérique (signature), et d'éventuels attributs non authentifiés/non signés (champ unauthenticatedAttributes ou unsignedAttrs, absent ici).

À noter dans les attributs authentifiés/signés la présence de l'heure de signature (`signingTime`) : celle-ci est déclarative et correspond à l'heure de la machine de l'utilisateur. Pour obtenir une heure de signature opposable, le format de signature électronique avancée CAdES étend CMS en définissant un attribut `contentTimestamp` permettant de stocker un jeton d'horodatage portant sur les données à signer et lui-même signé par une autorité d'horodatage (supposée digne de confiance, au même titre qu'une autorité de certification pour la délivrance de certificats).

Les commandes `openssl smime` et `openssl cms` produisent des signatures dont l'algorithme de signature, lorsqu'il met en œuvre des clés RSA, est identifié comme étant `rsaEncryption`, sans préciser l'algorithme de hachage : c'est le comportement par défaut prévu par la RFC 3370 (section 3.2). Si l'algorithme de hachage est précisé (par exemple `sha256WithRSAEncryption`, introduit dans CMS par la RFC 5754 avec les autres algorithmes à base des algorithmes de hachage de la famille SHA-2), alors OpenSSL est capable de vérifier la signature. (D'ailleurs, OpenSSL considère la signature valide même si l'algorithme de hachage référencé par l'algorithme de signature est incorrect, car il utilise l'algorithme de hachage identifié dans le champ `digestAlgorithm` de la structure `SignerInfo` associée.)

8.4. Vérification manuelle d'une signature

Le mécanisme de vérification manuelle d'une signature PKCS#7/CMS est analogue à celui utilisé pour vérifier la signature d'un certificat (cf. section 4.4), la particularité résidant dans la détermination des données sur lesquelles portent la signature.

Les premiers niveaux de la structure d'une signature PKCS#7 sont les suivants :

```
ContentInfo ::= SEQUENCE {
    contentType ContentType,
    content [0] EXPLICIT SEQUENCE {
        version Version,
        digestAlgorithms DigestAlgorithmIdentifiers,
        contentInfo ContentInfo,
        certificates [0] IMPLICIT ExtendedCertificatesAndCertificates OPTIONAL,
        crls [1] IMPLICIT CertificateRevocationLists OPTIONAL,
        signerInfos SET OF SEQUENCE {
            version Version,
            issuerAndSerialNumber IssuerAndSerialNumber,
            digestAlgorithm DigestAlgorithmIdentifier,
            authenticatedAttributes [0] IMPLICIT SET OF Attribute OPTIONAL,
            digestEncryptionAlgorithm DigestEncryptionAlgorithmIdentifier,
            encryptedDigest EncryptedDigest,
            unauthenticatedAttributes [1] IMPLICIT SET OF Attribute OPTIONAL
        }
    }
}
```

Pour une signature CMS, la structure correspondante est la suivante :

```
ContentInfo ::= SEQUENCE {
    contentType ContentType,
    content [0] EXPLICIT SEQUENCE {
        version CMSVersion,
```

```

    digestAlgorithms DigestAlgorithmIdentifiers,
    encapContentInfo EncapsulatedContentInfo,
    certificates [0] IMPLICIT CertificateSet OPTIONAL,
    crls [1] IMPLICIT RevocationInfoChoices OPTIONAL,
    signerInfos SET OF SEQUENCE {
        version CMSVersion,
        sid SignerIdentifier,
        digestAlgorithm DigestAlgorithmIdentifier,
        signedAttrs [0] IMPLICIT SET OF Attribute OPTIONAL,
        signatureAlgorithm SignatureAlgorithmIdentifier,
        signature SignatureValue,
        unsignedAttrs [1] IMPLICIT SET OF Attribute OPTIONAL
    }
}
}

```

Les données faisant l'objet de la signature, laquelle est incluse dans le champ `signature` de chaque élément `SignerInfo` du champ `signerInfos`, dépendent de la présence ou non du champ `authenticatedAttributes` (PKCS#7) ou `signedAttrs` (CMS). Si ce champ est absent, alors les données signées sont les octets de contenu DER (hors octets d'identifiant de balise et de longueur) du champ `contentInfo` (PKCS#7) ou `encapContentInfo` (CMS). Si le champ `authenticatedAttributes/signedAttrs` est présent, alors les données à signer sont le codage DER du SET OF Attribute constituant ce champ, avec un balisage EXPLICIT, c'est-à-dire sans le balisage [0] IMPLICIT.

Ces règles sont définies dans la section 9.3 de [PKCS#7] et dans la section 5.4 des RFC successives définissant CMS ([RFC 2630], [RFC 3369] et [RFC 3852]).

Deux méthodes sont proposées pour générer le codage DER faisant l'objet de la signature :

- Soit remplacer le codage DER de la balise [0] par le codage DER de la balise SET dans le codage DER du champ `authenticatedAttributes/signedAttrs` : cette méthode est plus simple à réaliser mais plus compliquée à comprendre.
- Soit reconstruire l'ensemble du champ en créant un fichier de configuration ASN.1 interprétable par la commande `openssl asn1parse -genconf` : cette méthode est plus longue, mais sera réutilisable pour ajouter de nouveaux attributs signés dans le cadre de la constitution de signatures avancées au format CAdES.

Dans les deux cas, la première étape est d'extraire le champ `authenticatedAttributes/signedAttrs`.

Extraction du champ `authenticatedAttributes/signedAttrs`

Afficher tout d'abord les six premiers niveaux de l'analyse ASN.1 de la signature détachée générée précédemment pour faciliter la mise en correspondance de celle-ci avec la structure ASN.1 :

```

$ openssl asn1parse -in sig-p7.der -inform DER -i | grep d=[0-5]
0:d=0 hl=4 l=1578 cons: SEQUENCE

```



```

 4:d=1  hl=2 l=  9 prim:  OBJECT                :pkcs7-signedData
15:d=1  hl=4 l=1563 cons:  cont [ 0 ]
19:d=2  hl=4 l=1559 cons:  SEQUENCE
23:d=3  hl=2 l=  1 prim:  INTEGER                :01
26:d=3  hl=2 l= 15 cons:  SET
28:d=4  hl=2 l= 13 cons:  SEQUENCE
30:d=5  hl=2 l=  9 prim:  OBJECT                :sha256
41:d=5  hl=2 l=  0 prim:  NULL
43:d=3  hl=2 l= 11 cons:  SEQUENCE
45:d=4  hl=2 l=  9 prim:  OBJECT                :pkcs7-data
56:d=3  hl=4 l=1010 cons:  cont [ 0 ]
60:d=4  hl=4 l=1006 cons:  SEQUENCE
64:d=5  hl=4 l= 726 cons:  SEQUENCE
794:d=5  hl=2 l= 13 cons:  SEQUENCE
809:d=5  hl=4 l= 257 prim:  BIT STRING
1070:d=3 hl=4 l= 508 cons:  SET
1074:d=4 hl=4 l= 504 cons:  SEQUENCE
1078:d=5 hl=2 l=  1 prim:  INTEGER                :01
1081:d=5 hl=2 l= 102 cons:  SEQUENCE
1185:d=5 hl=2 l= 13 cons:  SEQUENCE
1200:d=5 hl=2 l= 105 cons:  cont [ 0 ]
1307:d=5 hl=2 l= 13 cons:  SEQUENCE
1322:d=5 hl=4 l= 256 prim:  OCTET STRING          [HEX DUMP]:258640AFF29BD2797
...
79DD0F694200F7F

```

Le champ `authenticatedAttributes/signedAttrs` de l'unique `SignerInfo` se situe à l'octet 1200 : l'extraire (pour les besoins de la première méthode) et l'afficher (pour clarifier les opérations à réaliser dans le cadre de la deuxième méthode).

```

$ openssl asn1parse -in sig-p7.der -inform DER -i -strparse 1200 \
-out sig-p7.signedAttrs.der
 0:d=0  hl=2 l= 105 cons:  cont [ 0 ]
 2:d=1  hl=2 l= 24 cons:  SEQUENCE
 4:d=2  hl=2 l=  9 prim:  OBJECT                :contentType
15:d=2  hl=2 l= 11 cons:  SET
17:d=3  hl=2 l=  9 prim:  OBJECT                :pkcs7-data
28:d=1  hl=2 l= 28 cons:  SEQUENCE
30:d=2  hl=2 l=  9 prim:  OBJECT                :signingTime
41:d=2  hl=2 l= 15 cons:  SET
43:d=3  hl=2 l= 13 prim:  UTCTIME                :120427191932Z
58:d=1  hl=2 l= 47 cons:  SEQUENCE
60:d=2  hl=2 l=  9 prim:  OBJECT                :messageDigest
71:d=2  hl=2 l= 34 cons:  SET
73:d=3  hl=2 l= 32 prim:  OCTET STRING          [HEX DUMP]:89BD92286D6C8014C06
030B25F8B40CC1D5656D4B3B7B4831874F50D6F5557F3

```

Remplacement du codage DER de la balise [0]

Pour la première méthode, il s'agit de remplacer la balise [CONTEXT 0] (affichée sous la forme `cont [0]` ci-dessus, à l'octet 0) par un type SET OF. Pour un type construit (à l'exemple de SET OF), le codage DER de la balise [CONTEXT 0] est l'octet 0xa0 (ce que révèle aisément la commande `od -tx1` appliquée au fichier extrait). Le type SET OF est représenté en DER par l'octet 0x31. Les règles DER n'ont pas d'impact sur cette valeur par rapport aux règles BER.

Plus précisément, le type SET OF est représenté par une balise de classe UNIVERSAL numéro 17 ou 0x11 (règle 27.2 de [X.680]), et admet un codage construit (*constructed*, par opposition à *primitive*) d'après la règle 8.12.1 de [X.690]. En application des règles de codage BER définies dans la famille de règles 8.1.2 de [X.690], la représentation binaire de l'identifiant du type SET OF est 0b00110001, soit 0x31. Dans le détail, la valeur binaire se découpe en 00.1.10001, le séparateur « . » étant employé pour séparer les bits représentant successivement : la classe UNIVERSAL (00), le codage construit (1), et le numéro de balise (10001, soit 17 en décimal). Les règles DER n'ont pas d'impact sur cette valeur par rapport aux règles BER.

Le remplacement du premier octet 0xa0 par l'octet 0x31 peut s'effectuer de plusieurs manières. La méthode la plus simple est d'utiliser un éditeur hexadécimal, à l'exemple de HexEdit²⁴ sous Windows. Une deuxième méthode consiste, sous UNIX/Linux, à convertir le fichier binaire en hexadécimal à l'aide de l'outil `xxd`, à modifier l'octet, puis à convertir le fichier hexadécimal résultant en binaire. Une troisième option s'appuie sur OpenSSL, et consiste à convertir le fichier en Base64, à effectuer la modification, puis à convertir le fichier obtenu en binaire.

Coder le fichier `sig-p7.signedAttrs.der` en Base64.

```
$ openssl base64 -in sig-p7.signedAttrs.der \
  -out sig-p7.signedAttrs.b64
```

Le contenu du fichier obtenu est le suivant :

```
oGkwGAYJKoZIhvcNAQkDMQsGCSqGSIb3DQEHATAcBgkqhkiG9w0BCQUxDxcNMTIw
NDI3MTkxOTMyWjAvBgkqhkiG9w0BCQQxIgQgib2SKG1sgBTAYDCyX4tAzB1WVtSz
t7SDGHT1DW9VV/M=
```

Le premier octet est codé sur les deux premiers caractères, `oG`, soit les indices 40 et 6 de l'alphabet Base64, donc, en binaire sur 6 bits, 101000 et 000110, soit, en regroupant les huit premiers bits représentant l'octet 0xa0, 10100000 0110. Le remplacement du premier octet par 0x31 sans modifier les quatre derniers bits donne 00110001 0110, soit, en regroupant par blocs de 6 bits, 001100 010110, soit 12 et 22 en décimal, soit encore `MW` dans l'alphabet Base64.

Copier le fichier `sig-p7.signedAttrs.b64` sous le nom `sig-p7.explicit-signedAttrs.b64`, y remplacer les deux premiers caractères `oG` par `MW`, puis reconstituer le fichier binaire.

```
$ openssl base64 -d -in sig-p7.explicit-signedAttrs.b64 \
  -out sig-p7.explicit-signedAttrs.der
```

24. <http://www-physics.mps.ohio-state.edu/~prewett/hexedit/index.html>

Analyser le fichier résultant pour observer que la balise [CONTEXT 0] a bien été remplacée par un type SET.

```
$ openssl asn1parse -i -inform DER \
-in sig-p7.explicit-signedAttrs.der
 0:d=0  hl=2 l= 105 cons: SET
 2:d=1  hl=2 l=  24 cons: SEQUENCE
 4:d=2  hl=2 l=   9 prim: OBJECT           :contentType
15:d=2  hl=2 l=  11 cons: SET
17:d=3  hl=2 l=   9 prim: OBJECT           :pkcs7-data
28:d=1  hl=2 l=  28 cons: SEQUENCE
30:d=2  hl=2 l=   9 prim: OBJECT           :signingTime
41:d=2  hl=2 l=  15 cons: SET
43:d=3  hl=2 l=  13 prim: UTCTIME          :120427191932Z
58:d=1  hl=2 l=  47 cons: SEQUENCE
60:d=2  hl=2 l=   9 prim: OBJECT           :messageDigest
71:d=2  hl=2 l=  34 cons: SET
73:d=3  hl=2 l=  32 prim: OCTET STRING     [HEX DUMP]:89BD92286D6C8014C06
030B25F8B40CC1D5656D4B3B7B4831874F50D6F5557F3
```

Reconstruction du champ `authenticatedAttributes/signedAttrs`

La deuxième méthode permettant de générer le fichier `sig-p7.explicit-signedAttrs.der` consiste à créer un fichier de configuration définissant la structure ASN.1 souhaitée, interprétable par la commande `openssl asn1parse -genconf`, à l'image de ce qui a été fait pour constituer manuellement une clé publique (cf. section 3.4) ou un certificat (cf. section 4.7). Créer pour cela le fichier `sig-p7.explicit-signedAttrs.asn.cnf` suivant :

```
asn1 = SET:signedAttrs

[signedAttrs]
attr_contentType = SEQUENCE:attr_contentType
attr_signingTime = SEQUENCE:attr_signingTime
attr_messageDigest = SEQUENCE:attr_messageDigest

[attr_contentType]
attrType = OID:contentType
attrValues = SET:attr_contentType_values

[attr_contentType_values]
attr_contentType_value = OID:pkcs7-data

[attr_signingTime]
attrType = OID:signingTime
attrValues = SET:attr_signingTime_values

[attr_signingTime_values]
attr_signingTime_value = UTCTIME:120427191932Z

[attr_messageDigest]
```

```
attrType = OID:messageDigest
attrValues = SET:attr_messageDigest_values

[attr_messageDigest_values]
attr_messageDigest_value = FORMAT:HEX,OCTETSTRING:\
89bd92286d6c8014c06030b25f8b40cc1d5656d4b3b7b4831874f50d6f5557f3
```

Générer le codage DER de la structure ASN.1 correspondante :

```
$ openssl asn1parse -genconf sig-p7.explicit-signedAttrs.asn.cnf \
-i -out sig-p7.explicit-signedAttrs.der
  0:d=0  hl=2 l= 105 cons: SET
  2:d=1  hl=2 l=  24 cons: SEQUENCE
  4:d=2  hl=2 l=   9 prim: OBJECT           :contentType
 15:d=2  hl=2 l=  11 cons: SET
 17:d=3  hl=2 l=   9 prim: OBJECT           :pkcs7-data
 28:d=1  hl=2 l=  28 cons: SEQUENCE
 30:d=2  hl=2 l=   9 prim: OBJECT           :signingTime
 41:d=2  hl=2 l=  15 cons: SET
 43:d=3  hl=2 l=  13 prim: UTCTIME          :120427191932Z
 58:d=1  hl=2 l=  47 cons: SEQUENCE
 60:d=2  hl=2 l=   9 prim: OBJECT           :messageDigest
 71:d=2  hl=2 l=  34 cons: SET
 73:d=3  hl=2 l=  32 prim: OCTET STRING      [HEX DUMP]:89BD92286D6C8014C06
030B25F8B40CC1D5656D4B3B7B4831874F50D6F5557F3
```

Noter que le résultat est identique à celui obtenu à l'aide de la première méthode.

Vérification de la signature

Une fois le fichier `sig-p7.explicit-signedAttrs.der` généré à l'aide de l'une des méthodes ci-dessus, en calculer l'empreinte SHA-256 :

```
$ openssl sha256 sig-p7.explicit-signedAttrs.der
SHA256(sig-p7.explicit-signedAttrs.der)= 1e656e089e0e9c8196eb547af
f9fb3808fe837a35351b9b1a3a8dc4de8769479
```

Extraire ensuite la signature numérique correspondant au champ signature (pour CMS, ou encryptedDigest pour PKCS#7) de `SignerInfo` : elle est située à l'octet 1326 (1322 + hl), pour une longueur de 256 octets.

```
$ openssl asn1parse -in sig-p7.der -inform DER -i -offset 1326 \
-length 256 -out sig-p7.signature.bin
```

L'erreur de décodage est normale (les données extraites sont le contenu d'une OCTET STRING « arbitraire » et non une structure DER valide) et peut être ignorée. Alternativement, la commande `dd` peut être utilisée pour réaliser l'extraction des octets avec un outil plus « naturel ».

Déchiffrer la signature avec la clé publique du signataire et analyser la structure `DigestInfo` résultante :

```
$ openssl pkeyutl -verifyrecover -in sig-p7.signature.bin \
-inkey ee-key.pem -pubin | openssl asn1parse -i -inform DER
  0:d=0  hl=2 l= 49 cons: SEQUENCE
  2:d=1  hl=2 l= 13 cons: SEQUENCE
  4:d=2  hl=2 l=  9 prim: OBJECT                  :sha256
 15:d=2  hl=2 l=  0 prim: NULL
 17:d=1  hl=2 l= 32 prim: OCTET STRING          [HEX DUMP]:1E656E089E0E9C8196EB5
47AFF9FB3808FE837A35351B9B1A3A8DC4DE8769479
```

L'empreinte portée par cette structure `DigestInfo` est égale à l'empreinte des données signées/authentifiées calculée précédemment, ce qui achève la vérification de la signature numérique.

Par ailleurs, l'empreinte portée par l'attribut `messageDigest` du `SignerInfo` est l'empreinte du fichier `data.txt` (cf. chapitre 2), ce qui permet d'affirmer que la signature PKCS#7/CMS porte sur le fichier `data.txt`.

8.5. Construction d'une signature électronique PKCS#7/CMS

La construction d'une signature électronique PKCS#7/CMS ne pose aucun problème en partant de la structure `ContentInfo` rappelée dans la section 8.4 et en utilisant les méthodes de construction déjà vues pour les certificats.

Constitution des attributs authentifiés/signés

La première étape est de créer la structure SET OF `Attributes` correspondant aux attributs signés/authentifiés : reprendre le fichier `sig-p7.explicit-signedAttrs.asn.cnf` défini dans la section 8.4, en l'adaptant au besoin (heure de signature dans la section `attr_signingTime_values`, et empreinte dans la section `attr_messageDigest_values`), puis générer le codage DER de la structure :

```
$ openssl asn1parse -genconf sig-p7.explicit-signedAttrs.asn.cnf \
-i -out sig-p7.explicit-signedAttrs.der
```

Signature des attributs authentifiés/signés

La démarche pour générer la signature est semblable à celle utilisée pour signer la structure `TBSCertificate` d'un certificat (cf. section 4.7).

Calculer la valeur hexadécimale de l'empreinte SHA-256 du fichier DER généré :

```
$ openssl sha256 sig-p7.explicit-signedAttrs.der
SHA256(sig-p7.explicit-signedAttrs.der)= 1e656e089e0e9c8196eb547aff9fb3808fe837a
35351b9b1a3a8dc4de8769479
```

Créer le fichier `sig-p7.DigestInfo.asn.cnf` suivant, représentant la structure `DigestInfo` associée à cette empreinte, en reportant la valeur hexadécimale de l'empreinte ci-dessus dans le champ `digest` de la section `digestInfo` :

```
asn1 = SEQUENCE:digestInfo
```

```
[digestInfo]
digestAlgorithm = SEQUENCE:digestAlgorithm
digest = FORMAT:HEX,OCTETSTRING:\
1e656e089e0e9c8196eb547aff9fb3808fe837a35351b9b1a3a8dc4de8769479

[digestAlgorithm]
algorithm = OID:sha256
parameters = NULL
```

Générer le codage DER de la structure `DigestInfo` :

```
$ openssl asn1parse -genconf sig-p7.DigestInfo.asn.cnf -i \
-out sig-p7.DigestInfo.der
 0:d=0  hl=2 l= 49 cons: SEQUENCE
 2:d=1  hl=2 l= 13 cons: SEQUENCE
 4:d=2  hl=2 l=  9 prim: OBJECT                  :sha256
15:d=2  hl=2 l=  0 prim: NULL
17:d=1  hl=2 l= 32 prim: OCTET STRING          [HEX DUMP]:1E656E089E0E9C8196EB5
47AFF9FB3808FE837A35351B9B1A3A8DC4DE8769479
```

Signer ce codage DER à l'aide de la clé publique du signataire, et produire la représentation hexadécimale de cette signature :

```
$ openssl pkeyutl -sign -in sig-p7.DigestInfo.der -inkey ee-key.pem \
| od -tx1 -An -w | tr -d " " | sed 's/$/\\/'
258640aff29bd279765a63c8dc4bf0b0b2a74acd14dae89ba5703a129c0518bb\
b7bc88c91998c820974f9ad3f409ffd55ec55cabb48eafbc5cac57f672a8b05a\
906918ed3575beca60fc40bdf4e04ae340647e86f31a732bffa239912515f670\
4d5cd3215455da2ac196efab2e94a79b88c7409bc53279c73a5801bed8d4e585\
0d705384953a1271d29a225389eb705167bb25f8028ea9f2ffe2b86b448f01bd\
9e52484d3a6d8e78e6f902d0592041ccdb430acdc5126d590db7e5e54e34282\
3a1e134c1c8f33b91be8ffefdf3dec82903deb51d4378608c8d5c756d19ae301\
f9b90e502d2ae6917f6e4d2a1b3745e26d155aaf5aaab5dca79dd0f694200f7f\
```

Construction de la structure *SignerInfo*

Dans un fichier `sig-p7.asn.cnf`, créer la section suivante correspondant à la structure `SignerInfo` (la valeur de l'OCTET STRING du champ `signature` est la valeur hexadécimale de la signature obtenue précédemment) :

```
[signerInfo]
version = INTEGER:1
sid = SEQUENCE:sid
digestAlgorithm = SEQUENCE:signerInfo_digestAlgorithm
signedAttrs = IMPLICIT:0,SET:signedAttrs
```

```
signatureAlgorithm = SEQUENCE:signatureAlgorithm
signature = FORMAT:HEX,OCTETSTRING:\
258640aff29bd279765a63c8dc4bf0b0b2a74acd14dae89ba5703a129c0518bb\
b7bc88c91998c820974f9ad3f409ffd55ec55cabb48eafbc5cac57f672a8b05a\
906918ed3575beca60fc40bdf4e04ae340647e86f31a732bffa239912515f670\
4d5cd3215455da2ac196efab2e94a79b88c7409bc53279c73a5801bed8d4e585\
0d705384953a1271d29a225389eb705167bb25f8028ea9f2ffe2b86b448f01bd\
9e52484d3a6d8e78e6f902d0592041ccdb430acdc5126d590db7e5e54e34282\
3a1e134c1c8f33b91be8ffefdf3dec82903deb51d4378608c8d5c756d19ae301\
f9b90e502d2ae6917f6e4d2a1b3745e26d155aaf5aaab5dca79dd0f694200f7f
```

Le champ sid identifie le certificat du signataire par son numéro de série et par le DN de l'émetteur. Les champs et sections correspondantes du fichier ee-Certificate.asn.cnf peuvent être repris :

```
[sid]
issuer = SEQUENCE:issuer
serialNumber = INTEGER:0xdcd21ee5a2b7dfc7
```

```
[issuer]
issuer_C_RDN = SET:issuer_C_RDN
issuer_O_RDN = SET:issuer_O_RDN
issuer_OU1_RDN = SET:issuer_OU1_RDN
issuer_OU2_RDN = SET:issuer_OU2_RDN
```

```
[issuer_C_RDN]
issuer_C_ATV = SEQUENCE:issuer_C_ATV
```

```
[issuer_C_ATV]
type = OID:countryName
value = PRINTABLESTRING:FR
```

```
[issuer_O_RDN]
issuer_O_ATV = SEQUENCE:issuer_O_ATV
```

```
[issuer_O_ATV]
type = OID:organizationName
value = PRINTABLESTRING:Mon Entreprise
```

```
[issuer_OU1_RDN]
issuer_OU1_ATV = SEQUENCE:issuer_OU1_ATV
```

```
[issuer_OU1_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:0002 123456789
```

```
[issuer_OU2_RDN]
issuer_OU2_ATV = SEQUENCE:issuer_OU2_ATV
```

```
[issuer_OU2_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:OpenSSL Root CA
```

Ajouter la section suivante pour le champ `digestAlgorithm` :

```
[signerInfo_digestAlgorithm]
algorithm = OID:sha256
parameters = NULL
```

Inclure ensuite toutes les sections du fichier `sig-p7.explicit-signedAttrs.asn.cnf` (en prenant soin d'omettre la ligne initiale `asn1 = SET:signedAttrs`).

Terminer avec la section suivante :

```
[signatureAlgorithm]
algorithm = OID:rsaEncryption
parameters = NULL
```

Il serait possible d'utiliser `algorithm = OID:sha256WithRSAEncryption` (cf. note page 77) sans rompre la signature et en préservant sa validité.

Finalisation de la structure PKCS#7/CMS

Ajouter au début du fichier `sig-p7.asn.cnf` les lignes suivantes :

```
asn1 = SEQUENCE:contentInfo

[contentInfo]
contentType = OID:pkcs7-signedData
content = EXPLICIT:0,SEQUENCE:content

[content]
version = INTEGER:1
digestAlgorithms = SET:content_digestAlgorithms
encapContentInfo = SEQUENCE:encapContentInfo
certificates = IMPLICIT:0,SEQUENCE:certificates
signerInfos = SET:signerInfos

[content_digestAlgorithms]
digestAlgorithm = SEQUENCE:content_digestAlgorithm

[content_digestAlgorithm]
algorithm = OID:sha256
parameters = NULL

[encapContentInfo]
eContentType = OID:pkcs7-data

[certificates]
certificate = SEQUENCE:ee_certificate

[signerInfos]
signerInfo = SEQUENCE:signerInfo
```


Faire suivre ces lignes de la copie de toutes les sections du fichier `ee-Certificate.asn.cnf` (en excluant la ligne initiale `asn1 = SEQUENCE:ee_certificate`), correspondant au certificat du signataire, qui est référencé dans la section `certificates` ci-dessus.

Générer le codage DER de la signature PKCS#7/CMS :

```
$ openssl asn1parse -genconf sig-p7.asn.cnf -i -out sig-p7.der
```

Vérifier la signature électronique ainsi obtenue :

```
$ openssl smime -verify -inform DER -in sig-p7.der -content data.txt \  
-CAfile ca-crt.pem
```

```
texte en clairVerification successful
```


Chapitre 9 — Signature électronique — XML Signature

La norme XML Signature (souvent appelée « XML DSig », avec « DSig » pour *digital signature* ou signature numérique) du W3C²⁵ permet de signer des données (XML ou non, mais seul le cas XML est abordé ici, le cas non XML étant rare), et est à la base du format de signature électronique avancée XAdES.

Une signature XML Signature portant sur des données XML se matérialise par un élément XML `ds:Signature` (où le préfixe `ds:` représente l'espace de nommage <http://www.w3.org/2000/09/xmldsig#>), qui peut être inclus dans les données à signer (signature enveloppée), ou contenir les données à signer (signature enveloppante), ou être séparé (nœud disjoint d'un même document XML ou document XML séparé) des données à signer (signature détachée). Le cas le plus courant est celui de la signature enveloppée.

L'élément `ds:Signature` contient les informations sur la signature (algorithmes, identification et empreinte des données signées) dans un élément `ds:SignedInfo`, la valeur de la signature numérique (élément `ds:SignatureValue`) de cet élément `ds:SignedInfo`, souvent les informations d'identification de la clé de signature (élément `ds:KeyInfo` introduit dans le chapitre 7, page 63), et dans certains cas un élément `ds:Object` (contenant par exemple les données à signer dans le cas d'une signature enveloppante, ou encore les propriétés avancées de la signature dans le cas d'une la signature XAdES).

L'élément `ds:SignedInfo` contient les éléments suivants :

- `ds:CanonicalizationMethod`, qui définit l'algorithme de canonicalisation (abrégé en « c14n », c'est-à-dire la première lettre, 14 pour le nombre de lettres omises, et la dernière lettre). Le rôle de la canonicalisation est crucial dans le contexte de la signature électronique pour le maintien de l'intégrité des données. En effet, contrairement au monde binaire où l'information est représentée sans ambiguïté par une suite d'octets, dans le monde XML une information admet plusieurs représentations. Par exemple, les éléments `<tag attr1 = "val1" attr2 = "val2" />` et `<tag attr2="val2" attr1="val1"/>` (noter les espaces et l'ordre des attributs) représentent la même information, même si les suites d'octets de leur représentation sont différentes. La canonicalisation d'un document XML consiste à appliquer un ensemble de règles permettant d'obtenir un document XML canonique, représentation unique de l'information contenue dans un document XML. Dans le contexte de la signature électronique, des informations représentées différemment admettent la même empreinte après canonicalisation. Le W3C définit une méthode de canonicalisation générale²⁶, enrichie par des règles complémentaires sous le nom de canonicalisation exclusive²⁷ visant à préserver les déclarations des espaces de nommage lors d'extractions de sous-arbres à partir d'un document

25. <http://www.w3.org/TR/xmldsig-core/>

26. <http://www.w3.org/TR/xml-c14n>

27. <http://www.w3.org/TR/xml-exc-c14n/>

XML, ce qui est particulièrement utile dans le cadre de la signature électronique de portions d'un document XML employant plusieurs espaces de nommage.

- `ds:SignatureMethod`, qui spécifie l'algorithme de signature.
- Un ou plusieurs nœuds `ds:Reference`, contenant l'algorithme de hachage et l'empreinte d'un ensemble de données, lesquelles peuvent être référencées dans le nœud.

9.1. Signature d'un fichier

À l'image de XML Encryption (cf. chapitre 7), l'outil `xmlsec` ajoute des signatures électroniques à un document XML en s'appuyant sur des squelettes d'éléments `ds:Signature`. À la différence de XML Encryption, ces modèles d'éléments `ds:Signature` doivent être contenus dans le document incluant la signature.

Voici un exemple de modèle d'élément `ds:Signature` :

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
    <ds:Reference URI="">
      <ds:Transforms>
        <ds:Transform
          Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </ds:Transforms>
      <ds:DigestMethod
        Algorithm="http://www.w3.org/2001/04/xmldsig#sha256" />
      <ds:DigestValue/>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue/>
  <ds:KeyInfo>
    <ds:X509Data>
      <ds:X509Certificate/>
    </ds:X509Data>
  </ds:KeyInfo>
</ds:Signature>
```

Les nœuds fils de `ds:Signature` contiennent les informations présentées ci-dessus :

L'élément `ds:SignedInfo` de `ds:Signature` définit les caractéristiques de la signature à appliquer :

- L'algorithme de canonicalisation exclusive, référencé dans l'attribut `Algorithm` de `ds:CanonicalizationMethod`, utilisé pour canonicaliser l'élément `ds:SignedInfo` faisant l'objet de la signature numérique inscrite dans `ds:SignatureValue`.

- L'algorithme de signature SHA-256 et RSA (ds:SignatureMethod/@Algorithm). À noter que la famille des algorithmes SHA-2 a été introduite dans la version de travail de la version 1.1 de la norme XML Signature²⁸.
- Les données faisant l'objet de la signature électronique, c'est-à-dire ici l'ensemble du document XML parent de l'élément ds:Signature (identifié par l'attribut vide URI de l'élément ds:Reference), auquel est appliqué la transformation de signature enveloppée, qui supprime ledit élément ds:Signature (voir l'encadré page 95 pour une explication plus précise), puis une canonicalisation exclusive. L'élément ds:DigestValue est vide, et sera complété lors de la signature par l'empreinte des données, calculée en utilisant l'algorithme SHA-256, identifié par l'élément ds:DigestMethod.

L'élément ds:SignatureValue est laissé vide, et sera complété par la signature numérique lors de la signature, tout comme la référence à la clé de signature ds:KeyInfo, dont le sous-élément ds:X509Certificate sera complété avec le codage en Base64 de la représentation DER du certificat du signataire.

Générer le fichier data.dsig-tmpl.xml suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <data>Texte en clair</data>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      <ds:SignatureMethod
        Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
      <ds:Reference URI="">
        <ds:Transforms>
          <ds:Transform
            Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
          <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        </ds:Transforms>
        <ds:DigestMethod
          Algorithm="http://www.w3.org/2001/04/xmenc#sha256" />
        <ds:DigestValue />
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue />
  </ds:Signature>
  <ds:KeyInfo>
    <ds:X509Data>
      <ds:X509Certificate />
    </ds:X509Data>
  </ds:KeyInfo>
</root>
```

28. <http://www.w3.org/TR/2009/WD-xmldsig-core1-20090226/>

Signer le fichier à l'aide de la clé privée et du certificat d'authentification et signature créé dans la section 4.3, en prenant garde à la syntaxe particulière de l'option `--privkey-pem`, qui prend pour paramètre obligatoire le nom du fichier de la clé privée (au format PEM) suivi, pour pouvoir renseigner l'élément `ds:X509Data`, du nom du fichier du certificat associé, les noms de fichier étant séparés par une virgule.

```
$ xmlsec --sign --privkey-pem ee-key.pem,ee-crt-authsig.pem data.dsig-tmpl.xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <data>Texte en clair</data>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      <ds:SignatureMethod
        Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
      <ds:Reference URI="">
        <ds:Transforms>
          <ds:Transform
            Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
          <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmenc#sha256" />
        <ds:DigestValue>
          o00jEBylngTpuEsj0e+RkjwymUf5e+wiDE151+Z7Zv0=
        </ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>
      RtMv8LrmyjHSU3ZU1Tb+BPnpirsAlfBEWwT93TXDdfEM2JDMbDSEE+G0jNczNhU
      7OottfaSMFVrMELbUjhx/qAen1vtb+XyMe1aZClcAyeNAhqhWM9KqTJKmeFxxhwe
      rHya6XcSZaMLKNE7erODbKL05GArtoQposC680MT4y03uwc+hAwpnqAxc45N1B1
      NS86K35E5F+xxGTbWFqodan8m6T01t3c4vuoUCBvGZK7eSvtC2ufirkmxhtm8hw
      pjFFRjchlqyY2tISseWYfdUEA8pZi5w3aSXtCC1No59Ae5L/zfXPxCVzrAvp3rAn
      1mvQHwQ3Q6X9HNqvzgmZdQ==</ds:SignatureValue>
    <ds:KeyInfo>
      <ds:X509Data>

        <ds:X509Certificate>
MIID7jCCAtagAwIBAgIJANzSHuWit9/HMAOGCSqGSIb3DQEBCwUAMFkxCzAJBgNV
...
bos=</ds:X509Certificate>
</ds:X509Data>
      </ds:KeyInfo>
    </ds:Signature>
  </root>
```

Les « espaces blancs » (*whitespaces*) — de type espace, tabulation et retour chariot — entre les balises faisant partie des données à signer, le résultat obtenu peut différer.

Utiliser la même ligne de commande avec l'option `--output data.dsig.xml` en prévision de la vérification de la signature :

```
$ xmlsec --sign --privkey-pem ee-key.pem,ee-crt-authsig.pem \
  --output data.dsig.xml data.dsig-tmpl.xml
```

Automatisation de l'ajout d'un modèle de `ds:Signature` à un document XML existant

Pour pouvoir maintenir séparément un document XML et le fichier modèle de l'élément `ds:Signature`, le fichier de transformation XSLT suivant, nommé `dsig-addafternode.xslt` est proposé.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <!-- insert-doc parameter:
    specifies the filename of the document to be inserted after the node -->
  <xsl:param name="insert-doc"/>

  <xsl:template match="/root/data"> <!-- change XPath value as required -->
    <xsl:copy-of select="."/> <!-- copies matching XPath node -->
    <xsl:copy-of select="document($insert-doc)"/> <!-- inserts file -->
  </xsl:template>

  <!-- identity transform template:
    copies nodes recursively unless another template applies -->
  <xsl:template match="node() | @*">
    <xsl:copy>
      <xsl:apply-templates select="node() | @*" />
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

Cette transformation ajoute le contenu du fichier passé en tant que paramètre `insert-doc` après le nœud dont le chemin XPath est spécifié dans l'attribut `match` du premier élément `xsl:template` (soit `/root/data` ci-dessus, à adapter en fonction du besoin).

À titre d'exemple, copier le contenu du modèle d'élément `ds:Signature` dans un fichier `dsig-tmpl.xml`, et exécuter la commande suivante pour obtenir un le fichier `data.dsig-tmpl.xml` :

```
$ xsltproc --stringparam insert-doc dsig-tmpl.xml dsig-addafternode.xslt \
  data.xml
```

9.2. Vérification d'une signature

Vérifier la signature électronique du fichier signé obtenu précédemment.

```
$ xmlsec --verify --trusted-pem ca-crt.pem data.dsig.xml
OK
SignedInfo References (ok/all): 1/1
Manifests References (ok/all): 0/0
```

Le certificat du signataire étant inclus dans l'élément `ds:KeyInfo`, il est utilisé pour valider la signature : seul le certificat de l'AC émettrice est à fournir en plus.

Si le certificat n'est pas inclus dans l'élément `ds:KeyInfo`, par exemple si la clé est référencée par le champ `SubjectKeyIdentifier` du certificat dans un élément `ds:X509SKI` sous `ds:X509Data`, alors il doit être précisé en ligne de commande avec l'option `--pubkey-cert-pem`.

9.3. Vérification manuelle d'une signature

La vérification manuelle d'une signature XML Signature implique de réaliser les opérations suivantes pour un élément `ds:Signature` donné :

- Extraire l'élément `ds:SignedInfo`, et le canonicaliser en utilisant l'algorithme défini dans l'élément `ds:CanonicalizationMethod`.
- Pour chacun des éléments `ds:Reference` sous `ds:SignedInfo`, obtenir les données référencées (généralement via l'attribut `URI` si la signature est contenue dans le même document XML que les données sur lesquelles elle porte), et vérifier que le codage Base64 de leur empreinte — calculée en utilisant l'algorithme de hachage défini dans `ds:DigestMethod` — est égal au contenu de l'élément `ds:DigestValue`.
- Vérifier à l'aide de la clé référencée dans l'élément `ds:KeyInfo` que la signature numérique de l'élément `ds:SignedInfo` par l'algorithme défini dans l'élément `ds:SignatureMethod` correspond à la valeur renseignée dans l'élément `ds:SignatureValue`.

La suite de cette section s'appuie sur le fichier signé `data.dsig.xml` généré précédemment.

Extraction et canonicalisation de l'élément `ds:SignedInfo`

Extraire l'élément `ds:SignedInfo` en utilisant l'outil XMLStarlet présenté dans la section 7.3.

```
$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# -t \
  -c /root/ds:Signature/ds:SignedInfo data.dsig.xml > data.dsig.signedinfo.xml
```

Effectuer une canonicalisation exclusive de cet élément :

```
$ xmllint --exc-c14n data.dsig.signedinfo.xml > data.dsig.signedinfo.c14n.xml
```


À ce stade, une petite précision s'impose sous Windows. La canonicalisation normalise les retours chariot, en supprimant les caractères *carriage return* (ou CR, codé par l'octet 0x0d, et représenté par le caractère spécial \r)... mais certaines applications sous Windows (dont les outils `xmllint` et `xml`) les restituent lorsqu'elles rencontrent un caractère *line feed* (ou LF, codé par l'octet 0x0a, et représenté par le caractère spécial \n). Dans ce cas, il convient, après la canonicalisation exclusive, de supprimer les caractères CR ou \r, ce qui peut s'effectuer en chaînant la commande `xmllint` ci-dessus avec une commande `tr`, comme ceci :

```
> xmllint --exc-c14n data.dsig.signedinfo.xml | tr -d "\r" \
> data.dsig.signedinfo.c14n.xml
```

Il est possible de vérifier avec la commande `od -tx1` qu'aucun octet 0x0d ne subsiste avant les octets 0x0a.

Extraction et transformation des références

L'attribut URI de l'unique élément `ds:Reference` du fichier exemple contenant une chaîne vide, la référence à considérer est le fichier dans son intégralité.

La première transformation à appliquer, spécifier par le premier élément `ds:Transform`, est la transformation de signature enveloppée, qui supprime l'élément `ds:Signature` contenant l'élément `ds:Transform` considéré.

Comprendre la transformation de signature enveloppée

La transformation de signature enveloppée réalise une opération simple à comprendre, mais dont le principe mérite quelques explications.

L'extrait suivant du fichier d'exemple est repris pour illustrer :

```
<root>
  <data>Texte en clair</data>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      ...
      <ds:Reference URI="">
        <ds:Transforms>
          <ds:Transform
            Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
          ...
        </ds:Transforms>
      ...
    </ds:Signature>
  </root>
```

Il s'agit d'une signature enveloppée, portant sur l'ensemble du fichier XML. Les données faisant l'objet de la signature sont l'ensemble des nœuds du fichier, à l'exception du nœud

ds:Signature, car la signature ne peut pas porter sur elle-même (cela créerait une dépendance circulaire impossible à traiter), d'où la transformation de signature enveloppée référencée dans l'élément ds:Transform: Algorithm="#enveloped-signature", qui supprime d'un document XML l'élément ds:Signature incriminé. Après application de la transformation, le fichier obtenu est donc :

```
<root>
  <data>Texte en clair</data>

</root>
```

Intuitivement, la notion d'« élément ds:Signature incriminé » à supprimer est claire, mais plus formellement, la norme XML Signature définit le résultat de la transformation de signature enveloppée comme étant l'ensemble des nœuds dont un ancêtre ou soi (au sens de l'axe ancestor-or-self:: de XPath) est le nœud ds:Signature parent de l'élément ds:Transform concerné, c'est-à-dire que la transformation doit avoir un résultat identique à la transformation suivante :

```
<ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
  <ds:XPath>
    count(ancestor-or-self::ds:Signature |
    here()/ancestor::ds:Signature[1]) >
    count(ancestor-or-self::ds:Signature)</XPath>
```

Dit autrement, tous les nœuds pour lesquels l'expression XPath ci-dessus est évaluée à vrai sont conservés.

Pour un nœud donné, l'expression count(ancestor-or-self::ds:Signature) est égale au nombre d'éléments ds:Signature de niveau supérieur ou égal au nœud considéré. L'expression count(ancestor-or-self::ds:Signature | here()/ancestor::ds:Signature[1]) dénombre quant à elle les éléments ds:Signature qui sont de niveau supérieur ou égal au nœud considéré, ainsi que le premier élément ds:Signature rencontré en remontant l'arborescence XML en partant du nœud ds:Transform considéré (en d'autres mots, le ds:Signature contenant le ds:Transform), en excluant les doublons.

Pour illustrer numériquement les valeurs obtenues, soit l'arborescence XML schématisée ci-dessous :

```
doc (1)
  ...
  ds:Signature (2)
  ...
  ds:Signature (3)
    ...
    ds:Transform (3a)
    ...
```

L'élément `ds:Transform` en (3a) référençant la transformation signature enveloppée définie par l'expression XPath précédente, soit à déterminer le document XML résultant de l'application de la transformation.

Au nœud (1), l'expression XPath se simplifie en $1 > 0$: il n'existe aucun élément `ds:Signature` de niveau supérieur à cet emplacement (d'où le terme de droite 0), et l'élément `ds:Signature` du nœud (3) compte pour 1 dans le terme de gauche. Donc l'expression est vraie et le nœud (1) est inclus dans le document résultant. Cela s'applique également à tous les nœuds en dehors des deux éléments `ds:Signature`.

Au nœud (2), il existe un élément `ds:Signature` de niveau supérieur ou égal (le nœud lui-même), donc le terme de droite de l'expression XPath vaut 1. Le terme de gauche dénombre les éléments `ds:Signature` de niveau supérieur ou égal (le nœud (2) uniquement) ainsi que l'élément `ds:Signature` du nœud (3), soit deux nœuds en tout. L'expression résultante, $2 > 1$ est vraie, et le nœud (2) est inclus, tout comme ses sous-éléments pour lesquels le même raisonnement s'applique.

Au nœud (3), il existe un élément `ds:Signature` de niveau supérieur ou égal, qui est le nœud (3) lui-même, donc le terme de droite de l'expression XPath vaut 1. Le terme de gauche dénombre les éléments `ds:Signature` de niveau supérieur ou égal (le nœud (3) uniquement) ainsi que l'élément `ds:Signature` du nœud (3), soit un seul nœud distinct, qui est le nœud (3). L'expression devient $1 > 1$, donc une valeur booléenne fausse, et le nœud (3) n'est pas inclus dans le résultat de la transformation, tout comme ses sous-éléments.

À l'issue de la transformation, le document obtenu a l'arborescence suivante :

```
doc (1)
  ...
  ds:Signature (2)
  ...
```

Cela correspond au résultat attendu : l'élément `ds:Signature` contenant la transformation de signature enveloppée a été supprimée en préparation de la production de la signature à l'emplacement du nœud (3).

En pratique, les API de signature utilisent des algorithmes optimisés pour effectuer la transformation de signature enveloppée via l'algorithme <http://www.w3.org/2000/09/xmldsig#enveloped-signature> au lieu d'appliquer la transformation XPath qui, bien que fonctionnellement équivalente, est peu performante.

Le fichier XSLT suivant (nommé `enveloped-signature.xslt`) effectue la transformation de signature enveloppée en utilisant l'expression XPath définie dans la section 6.6.4 de la norme XML Signature²⁹ :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="no"/>

  <xsl:template match="node() | @*">
    <xsl:if test="count(ancestor-or-self::ds:Signature
      | /root/ds:Signature/ds:SignedInfo/ds:Reference/ds:Transforms
      /ds:Transform/ancestor::ds:Signature[1])
      > count(ancestor-or-self::ds:Signature)">
      <xsl:copy>
        <xsl:apply-templates select="node() | @*" />
      </xsl:copy>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

La fonction `here()` n'étant pas une fonction XPath standard (elle est définie dans la section 6.6.3 de la norme XML Signature³⁰), elle doit donc être remplacée par le chemin XPath explicite de l'élément `ds:Transform` de référence, soit, dans l'exemple précédent, `/root/ds:Signature/ds:SignedInfo/ds:Reference/ds:Transforms/ds:Transform`.

Pour appliquer la transformation XSLT au fichier d'entrée `data.dsig.xml`, utiliser la ligne de commande suivante :

```
$ xsltproc enveloped-signature.xslt data.dsig.xml \
  > data.dsig.transform-enveloped-signature.xml
```

Le document XML résultant doit ensuite subir une canonicalisation exclusive, tel que prévu par le deuxième élément `ds:Transform` de `ds:SignedInfo`.

```
$ xmllint --exc-c14n data.dsig.transform-enveloped-signature.xml \
  > data.dsig.transforms.xml
```

Sous Windows, penser à chaîner cette commande dans la commande `tr -d "\r"` pour supprimer les caractères CR superflus.

29. <http://www.w3.org/TR/xmldsig-core/#sec-EnvelopedSignature>

30. <http://www.w3.org/TR/xmldsig-core/#sec-XPath>

Vérification de l'élément `ds:DigestValue`

Calculer l'empreinte du document XML `data.dsig.transforms.xml`, obtenu — pour rappel — après application aux données référencées par l'élément `ds:Reference` des transformations indiquées dans l'élément `ds:Transforms`, en utilisant l'algorithme SHA-256 spécifié dans l'élément `ds:DigestMethod`, puis coder cette empreinte en Base64.

```
$ openssl sha256 -binary data.dsig.transforms.xml | openssl base64
o00jEBylngTpuEsj0e+RkjwymUf5e+wiDE151+Z7Zv0=
```

La valeur obtenue est identique à la valeur du nœud `ds:DigestValue` de l'élément `ds:Reference` considéré :

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  ...
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
      ...
      <ds:Reference URI="">
        ...
        <ds:DigestValue>
          o00jEBylngTpuEsj0e+RkjwymUf5e+wiDE151+Z7Zv0=
        </ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    ...
  </ds:Signature>
</root>
```

La référence est donc valide.

Vérification de la signature numérique

Les références étant validées, il reste à vérifier la signature à proprement parler.

La valeur du nœud `ds:SignatureValue` est le codage Base64 de la signature numérique de l'élément `ds:SignedInfo` canonicalisé par l'algorithme indiqué dans l'élément `ds:CanonicalizationMethod`. Cet élément a été extrait précédemment dans le fichier `data.dsig.signedinfo.c14n.xml`, et est reproduit ci-dessous (sans remise en forme des lignes trop longues, afin de préserver l'intégrité du document canonicalisé).

```
<ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></ds:CanonicalizationMethod>
  <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"></ds:SignatureMethod>
  <ds:Reference URI="">
    <ds:Transforms>
      <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-s
```

```

signature"></ds:Transform>
    <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></ds
:Transform>
    </ds:Transforms>
    <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"></d
s:DigestMethod>
    <ds:DigestValue>o00jEBylngTpuEsj0e+RkjwymUf5e+wiDE151+Z7Zv0=</ds:DigestV
alue>
    </ds:Reference>
</ds:SignedInfo>

```

En vertu du principe idéal de WYSIWYS (*what you see is what you sign*, soit littéralement « ce que vous voyez est ce que vous signez ») en vigueur dans le domaine de la signature électronique (par exemple dans certains contextes réglementaires), c'est ce document XML canonique qui devrait être affiché à l'utilisateur pour signature. En pratique et par souci de convivialité, lorsque le principe WYSIWYS n'est pas explicitement imposé, c'est le document XML référencé par l'élément `ds:Reference` qui est affiché à l'utilisateur.

L'algorithme de signature utilisé est SHA-256 et RSA, comme indiqué dans l'élément `ds:SignatureMethod`.

Calculer l'empreinte SHA-256 de l'élément `ds:SignedInfo` canonicalisé :

```

$ openssl sha256 data.dsig.signedinfo.c14n.xml
SHA256(data.dsig.signedinfo.c14n.xml)= 2d4d71e14ebf862476ffd6b3da3666c50661e139c
08f3fb9f6c845a2f4c5fe41

```

Extraire, à partir du certificat de signature inclus dans l'élément `ds:X509Certificate` du nœud `ds:KeyInfo`, la clé publique du signature, dans le fichier `ee-key.pub.pem`.

```

$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# -t \
-v /root/ds:Signature/ds:KeyInfo/ds:X509Data/ds:X509Certificate \
data.dsig.xml | openssl base64 -d | openssl x509 -inform DER -pubkey -noout \
> ee-key.pub.pem

```

Le lecteur étant supposé familiarisé avec les outils `xml` et `openssl`, les commandes successives ont été chaînées sans passer par des fichiers intermédiaires.

Déchiffrer la signature numérique contenue dans l'élément `ds:SignatureValue` à l'aide de la clé publique de signature extraite, et analyser la structure ASN.1 `DigestInfo` (cf. section 3.3) résultante.

```

$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# -t \
-v /root/ds:Signature/ds:SignatureValue data.dsig.xml | openssl base64 -d \
| openssl pkeyutl -verifyrecover -pubin -inkey rsakey.pub.pem \
| openssl asn1parse -i -inform DER
0:d=0 hl=2 l= 49 cons: SEQUENCE
2:d=1 hl=2 l= 13 cons: SEQUENCE
4:d=2 hl=2 l= 9 prim: OBJECT :sha256
15:d=2 hl=2 l= 0 prim: NULL
17:d=1 hl=2 l= 32 prim: OCTET STRING [HEX DUMP]:2D4D71E14EBF862476FFD
6B3DA3666C50661E139C08F3FB9F6C845A2F4C5FE41

```

La valeur de l'empreinte à partir de l'octet 18 concorde avec l'empreinte de l'élément `ds:SignedInfo` canonicalisé calculée précédemment, donc la signature numérique est valide.

■ Pour achever la vérification de la signature électronique, il faut vérifier que le certificat est valide, ce qui a été traité dans la section 4.4.

Chapitre 10 — Authentification web

Les tutoriaux sur l'utilisation de certificats SSL avec les serveurs web Apache et IIS étant légion, cette section s'intéresse à la génération de certificats EV³¹ (*Extended Validation* ou validation étendue) et à leur utilisation avec le serveur web nginx, recompilé pour pouvoir utiliser TLSv1.2, première version du protocole TLS à supporter l'algorithme de hachage SHA-256 (du fait d'attaques partielles à son encontre, SHA-1 est interdit par le RGS en France à partir du 6 mai 2013, et commence à être déconseillé ou est interdit ailleurs). Côté client, TLSv1.2 est encore très peu implémenté³² à la date de rédaction, y compris sur les versions les plus récentes des navigateurs web, donc le cas d'étude ci-après s'appuiera sur Internet Explorer 8 et 9 sous Windows 7.

Le document³³ définissant les exigences liées aux certificats EV tolère SHA-1 tant que les algorithmes de la famille SHA-2 ne sont pas largement répandus dans les navigateurs : si la conformité au RGS n'est pas requise, alors ce chapitre peut être suivi pour mettre en place un serveur web avec un certificat ayant un profil EV avec une version de TLS antérieure à 1.2. Une section est dédiée à la configuration de Firefox en mode *debug* pour accepter un certificat de test comme étant de type EV.

10.1. Génération des certificats et des listes de certificats révoqués

Sommairement, les principales spécificités d'un certificat EV sont les suivantes :

- Référencer l'entité en charge du serveur dans le champ objet à l'aide des RDN usuels (tels que C, O, et CN — qui peut être omis au profit d'un `dNSName` dans l'extension `SubjectAlternativeName`), mais aussi des RDN moins couramment utilisés (pour l'adresse et le type d'entité : `postalCode`, `serialNumber`, etc.) et enfin des RDN propriétaires (dont `jurisdictionOfIncorporationCountryName`).
- Référencer, dans l'extension `certificatePolicies` du certificat serveur, l'OID de la politique de certification associée aux certificats EV. Internet Explorer s'attend à ce que les certificats de niveau supérieur contiennent soit ce même OID (peu réaliste d'un point de vue fonctionnel), soit `anyPolicy` (la solution retenue ci-après).
- Référencer les ressources permettant de vérifier le statut de révocation des certificats, OSCP (*Online Certificate Status Protocol* ou protocole de statut de certificat en ligne) étant imposé depuis 2011. L'URL du serveur (*responder* ou répondeur) OSCP est à inclure — sauf cas particulier de l'agrafage OSCP (*OCSP stapling*, cf. RFC 4366) qui n'est pas traité ici pour le moment — dans l'extension `authorityInformationAccess` du certificat d'AC et des certificats d'AC intermédiaires).

31. <http://www.cabforum.org/certificates.html>

32. http://en.wikipedia.org/wiki/Transport_Layer_Security#Browser_implementations

33. http://www.cabforum.org/Baseline_Requirements_V1.pdf

10. Authentification web

Pour un peu plus de « réalisme » (toutes proportions gardées !), une hiérarchie de certification indépendante de celle créée précédemment est générée, comprenant une AC racine et une AC fille, de laquelle sera issu le certificat d'authentification serveur. Le RGS recommandant l'usage d'OCSP, et les exigences sur les certificats EV ainsi que Firefox l'imposant pour les certificats EV, un serveur OCSP est également mis en place.

L'objet d'un certificat EV d'entité final doit contenir des RDN qui ne sont pas connus d'OpenSSL et doivent donc être définis dans un fichier d'OID complémentaires. Créer le fichier `ev-oid.tsv` avec le contenu suivant :

```
1.3.6.1.4.1.311.60.2.1.1  jurL  jurisdictionOfIncorporationLocalityName
1.3.6.1.4.1.311.60.2.1.2  jurST jurisdictionOfIncorporationStateOrProvinceName
1.3.6.1.4.1.311.60.2.1.3  jurC  jurisdictionOfIncorporationCountryName
```

Créer le fichier `req-empty-ev.cnf` suivant, référençant le fichier d'OID.

```
oid_file = ev-oid.tsv

[ req ]
distinguished_name = req_distinguished_name

[ req_distinguished_name ]
```

Générer en une seule passe — successivement pour l'AC racine, l'AC fille, le serveur OCSP rattaché à l'AC racine, le serveur OCSP rattaché à l'AC fille, et le serveur TLS — un bi-clé RSA de 2048 bits et une CSR contenant un objet (*subject*) conforme aux exigences associées aux certificats EV et aux profils de certificats RGS (les structures de DN utilisées précédemment conviennent).

```
$ openssl req -new -newkey rsa:2048 -nodes -keyout tls-rootca-key.pem \
  -subj "/C=FR/O=Mon Organisme/OU=0002 147258369/OU=OpenSSL TLS Root CA" \
  -sha256 -config req-empty-ev.cnf -out tls-rootca-req.pem
$ openssl req -new -newkey rsa:2048 -nodes -keyout tls-subca-key.pem \
  -subj "/C=FR/O=Mon Organisme/OU=0002 147258369/OU=OpenSSL TLS Subordinate CA" \
  -sha256 -config req-empty-ev.cnf -out tls-subca-req.pem
$ openssl req -new -newkey rsa:2048 -nodes -keyout tls-server-key.pem \
  -subj "/jurC=FR/businessCategory=Private Organization/serialNumber=789456123/C=FR/postalCode=99000/ST=France/L=Ma Ville/street=1 rue du Village/O=Ma Boutique/OU=0002 789456123/CN=localhost" -sha256 -config req-empty-ev.cnf \
  -out tls-server-req.pem
$ openssl req -new -newkey rsa:2048 -nodes -keyout tls-rootca-ocsp-key.pem \
  -subj "/C=FR/O=Mon Organisme/OU=0002 147258369/CN=OpenSSL TLS Root CA OCSP Responder" -sha256 -config req-empty-ev.cnf -out tls-rootca-ocsp-req.pem
$ openssl req -new -newkey rsa:2048 -nodes -keyout tls-subca-ocsp-key.pem \
  -subj "/C=FR/O=Mon Organisme/OU=0002 147258369/CN=OpenSSL TLS Subordinate CA OCSP Responder" -sha256 -config req-empty-ev.cnf -out tls-subca-ocsp-req.pem
```

■ Le DN objet de la CSR des serveurs TLS et OCSP est à saisir sur la même ligne.

■ Noter l'emploi du nom court `jurC` pour désigner le type de RDN associé à l'OID 1.3.6.1.4.1.311.60.2.1.3.

Contrôler le contenu de l'objet du serveur TLS (le petit souci d'affichage du premier RDN est dû à la longueur du nom long du type du RDN).

```
$ openssl req -in tls-server-req.pem -config req-empty-ev.cnf -noout -subject \
-nameopt multiline
subject=
= FRurisdictionOfIncorporationCountryName
  businessCategory      = Private Organization
  serialNumber          = 789456123
  countryName           = FR
  postalCode            = 99000
  stateOrProvinceName   = France
  localityName          = Ma Ville
  streetAddress         = 1 rue du Village
  organizationName       = Ma Boutique
  organizationalUnitName = 0002 789456123
  commonName            = localhost
```

Créer le fichier de configuration pour les certificats et LCR émis par l'AC racine, `tls-rootca.cnf`.

```
[tls_rootca]
database = tls-rootca-db.txt
serial = tls-rootca-crt.srl
crlnumber = tls-rootca-crl.srl
certificate = tls-rootca-crt.pem
private_key = tls-rootca-key.pem
default_md = sha256
default_crl_hours = 48
new_certs_dir = certs

[tls_ca_cert_dnpolicy]
C = supplied
O = supplied
OU = supplied

[tls_server_cert_dnpolicy]
C = supplied
O = supplied
OU = supplied
CN = supplied

[tls_rootca_cert_ext]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
keyUsage=critical,keyCertSign,cRLSign
certificatePolicies \
=1.2.840.113556.1.8000.2554.29563.49294.43847.16581.44480.6974059.2493436.1.1,\
  anyPolicy
basicConstraints=critical,CA:TRUE

[tls_rootca_crl_ext]
authorityKeyIdentifier=keyid
```

10. Authentication web

```
[tls_subca_cert_ext]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
keyUsage=critical,keyCertSign,cRLSign
certificatePolicies \
=1.2.840.113556.1.8000.2554.29563.49294.43847.16581.44480.6974059.2493436.1.2,\
  anyPolicy
crlDistributionPoints=URI:http://localhost/tls-rootca-crl.der
basicConstraints=critical,CA:TRUE
authorityInfoAccess=OCSP;URI:http://ocsp-rootca
```

```
[tls_rootca_ocsp_cert_ext]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
keyUsage=critical,digitalSignature
certificatePolicies=@tls_rootca_ocsp_cert_pol
crlDistributionPoints=URI:http://localhost/tls-rootca-crl.der
basicConstraints=critical,CA:FALSE
extendedKeyUsage=OCSPSigning
```

```
[tls_rootca_ocsp_cert_pol]
policyIdentifier = \
  1.2.840.113556.1.8000.2554.29563.49294.43847.16581.44480.6974059.2493436.1.4
CPS = http://localhost/rootca-ocsp-CPS.pdf
```

Ce fichier de configuration mérite quelques explications par rapport à ce qui a été fait précédemment :

- L'utilisation du serveur OCSP inclus dans OpenSSL suppose que la tenue à jour de la base des certificats émis par l'AC racine (database = tls-rootca-db.txt ci-dessus), ce qui impose l'utilisation de la commande `openssl ca` (au lieu de `openssl x509`) pour émettre les certificats.
- La commande `openssl ca` impose la définition dans le fichier de configuration d'une politique de nommage régissant les RDN obligatoires ou acceptés de la CSR pour inclusion dans le certificat, d'où les sections `[..._dnpolicy]`.
- Cette commande impose aussi la publication des certificats dans un répertoire local (`new_certs_dir = certs`), répertoire qui n'est plus utilisé par la suite.
- Le nom de domaine du serveur OCSP associé à l'AC racine (c'est-à-dire capable de déterminer le statut d'un certificat émis par l'AC racine) est `ocsp-rootca` (un alias de la machine locale, qui sera configuré plus tard), et est à remplacer par un autre (vrai) nom de domaine s'il est hébergé sur un serveur autre que la machine locale.
- De même, la LCR est publiée en local dans cette infrastructure d'exemple : la publier sur un autre serveur et modifier le champ `crlDistributionPoints` à sa convenance.
- L'AC racine et l'AC fille référencent deux politiques de certification : la vraie politique de certification (au sens du RGS), et la méta-politique `anyPolicy` (comparable au métacaractère « * » pour un nom de fichier), facultative d'après le guide des certificats EV, mais obligatoire en réalité pour fonctionner avec Internet Explorer.

Créer le fichier de configuration pour les certificats et LCR émis par l'AC fille, `tls-subca.cnf`.

```
oid_file = ev-oid.tsv

[tls_subca]
database = tls-subca-db.txt
serial = tls-subca-crt.srl
crlnumber = tls-subca-crl.srl
certificate = tls-subca-crt.pem
private_key = tls-subca-key.pem
default_md = sha256
default_crl_hours = 48
new_certs_dir = certs

[tls_server_cert_dnpolicy]
C = supplied
O = supplied
OU = supplied
CN = supplied

[tls_server_ev_cert_dnpolicy]
jurC = supplied
jurST = optional
jurL = optional
businessCategory = supplied
serialNumber = supplied
C = supplied
postalCode = optional
ST = supplied
L = supplied
street = optional
O = supplied
OU = supplied
CN = supplied

[tls_subca_crl_ext]
authorityKeyIdentifier=keyid

[tls_subca_ocsp_cert_ext]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
keyUsage=critical,digitalSignature
certificatePolicies=@tls_subca_ocsp_cert_pol
crlDistributionPoints=URI:http://localhost/tls-subca-crl.der
basicConstraints=critical,CA:FALSE
extendedKeyUsage=OCSPSigning

[tls_subca_ocsp_cert_pol]
policyIdentifier = \
    1.2.840.113556.1.8000.2554.29563.49294.43847.16581.44480.6974059.2493436.1.5
CPS = http://localhost/subca-ocsp-CPS.pdf
```

10. Authentication web

```
[tls_server_cert_ext]
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
keyUsage=critical,keyEncipherment,digitalSignature
certificatePolicies=@tls_server_cert_pol
crlDistributionPoints=URI:http://localhost/tls-subca-crl.der
basicConstraints=critical,CA:FALSE
extendedKeyUsage=serverAuth
authorityInfoAccess=OCSP;URI:http://ocsp-subca

[tls_server_cert_pol]
policyIdentifier = \
    1.2.840.113556.1.8000.2554.29563.49294.43847.16581.44480.6974059.2493436.1.3
CPS = http://localhost/server-auth-CPS.pdf
```

Quelques commentaires s'imposent ici aussi :

- Le fichier d'OID est référencé (`oid_file = ev-oid.tsv`) pour que les RDN correspondants puissent être utilisés dans la politique de nommage `tls_server_ev_cert_dnpolicy`.
- En dépit de l'exigence du RGS, deux bits du `keyUsage` sont positionnés (`digitalSignature` pour l'authentification, et `keyEncipherment` pour le chiffrement de la clé de session TLS) : du fait de la criticité du champ `keyUsage`, il s'agit d'éviter un jour d'être confronté au cas d'une application empêchant l'établissement d'une session TLS pour cause de bit manquant. Cet écart est provisoirement autorisé par le RGS depuis avril 2012³⁴ (par effet de bord, l'erratum étant initialement prévu pour gérer la coexistence de l'échange de clés via RSA et via Diffie-Hellman).
- 1.2.840.113556.1.8000.2554.29563.49294.43847.16581.44480.6974059.2493436.1.3 est l'identifiant de la politique de certification faisant l'objet du processus de référencement EV.
- Les remarques précédentes sur l'emplacement du point de distribution des LCR et du serveur OCSP s'appliquent ici aussi.

Créer les fichiers vides `tls-rootca-db.txt` et `tls-subca-db.txt` pour les bases de données des AC (utiliser par exemple la commande `touch` sous Linux/UNIX ou `cat nul > nomfichier` en ligne de commande DOS).

Créer le sous-répertoire `certs`.

Générer le certificat de l'AC racine, puis successivement — après avoir généré pour chacun un numéro de série aléatoire sur huit octets — celui de l'AC fille,

```
$ openssl x509 -req -in tls-rootca-req.pem -extfile tls-rootca.cnf \
    -extensions tls_rootca_cert_ext -signkey tls-rootca-key.pem -sha256 \
    -days 3652 -out tls-rootca-cert.pem
Signature ok
subject=/C=FR/O=Mon Organisme/OU=0002 147258369/OU=OpenSSL TLS Root CA
Getting Private key

$ openssl rand -hex 8 -out tls-rootca-cert.srl
```

34. http://www.ssi.gouv.fr/fr/reglementation-ssi/referentiel-general-de-securite/modification-immediate_var_mode_calcul.html

```
$ openssl ca -batch -in tls-subca-req.pem -config tls-rootca.cnf \  
-name tls_rootca -extensions tls_subca_cert_ext -policy tls_ca_cert_dnpolicy \  
-days 2191 -out tls-subca-cert.pem
```

Using configuration from tls-rootca.cnf

Check that the request matches the signature

Signature ok

The Subject's Distinguished Name is as follows

countryName :PRINTABLE:'FR'

organizationName :PRINTABLE:'Mon Organisme'

organizationalUnitName:PRINTABLE:'0002 147258369'

organizationalUnitName:PRINTABLE:'OpenSSL TLS Subordinate CA'

Certificate is to be certified until Apr 15 13:21:52 2018 GMT (2191 days)

Write out database with 1 new entries

Data Base Updated

```
$ openssl rand -hex 8 -out tls-rootca-cert.srl
```

```
$ openssl ca -batch -in tls-rootca-ocsp-req.pem -config tls-rootca.cnf \  
-name tls_rootca -extensions tls_rootca_ocsp_cert_ext \  
-policy tls_server_cert_dnpolicy -days 2191 -out tls-rootca-ocsp-cert.pem
```

Using configuration from tls-rootca.cnf

Check that the request matches the signature

Signature ok

The Subject's Distinguished Name is as follows

countryName :PRINTABLE:'FR'

organizationName :PRINTABLE:'Mon Organisme'

organizationalUnitName:PRINTABLE:'0002 147258369'

commonName :PRINTABLE:'OpenSSL TLS Root CA OCSP Responder'

Certificate is to be certified until Apr 15 13:21:54 2018 GMT (2191 days)

Write out database with 1 new entries

Data Base Updated

```
$ openssl rand -hex 8 -out tls-subca-cert.srl
```

```
$ openssl ca -batch -in tls-server-req.pem -config tls-subca.cnf \  
-name tls_subca -extensions tls_server_cert_ext \  
-policy tls_server_ev_cert_dnpolicy -days 365 -out tls-server-cert.pem
```

Using configuration from tls-subca.cnf

Check that the request matches the signature

Signature ok

The Subject's Distinguished Name is as follows

:PRINTABLE:'FR'ncorporationCountryName

businessCategory :PRINTABLE:'Private Organization'

serialNumber :PRINTABLE:'789456123'

countryName :PRINTABLE:'FR'

postalCode :PRINTABLE:'99000'

stateOrProvinceName :PRINTABLE:'France'

localityName :PRINTABLE:'Ma Ville'

streetAddress :PRINTABLE:'1 rue du Village'

organizationName :PRINTABLE:'Ma Boutique'

organizationalUnitName:PRINTABLE:'0002 789456123'

10. Authentication web

```
commonName          :PRINTABLE:'localhost'
Certificate is to be certified until Apr 15 13:21:55 2013 GMT (365 days)
```

Write out database with 1 new entries
Data Base Updated

```
$ openssl rand -hex 8 -out tls-subca-crt.srl
$ openssl ca -batch -in tls-subca-ocsp-req.pem -config tls-subca.cnf \
  -name tls_subca -extensions tls_subca_ocsp_cert_ext \
  -policy tls_server_cert_dnpolicy -days 2191 -out tls-subca-ocsp-crt.pem
Using configuration from tls-subca.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName          :PRINTABLE:'FR'
organizationName     :PRINTABLE:'Mon Organisme'
organizationalUnitName:PRINTABLE:'0002 147258369'
commonName           :PRINTABLE:'OpenSSL TLS Subordinate CA OCSP Responder'
Certificate is to be certified until Apr 15 13:21:57 2018 GMT (2191 days)
```

Write out database with 1 new entries
Data Base Updated

Créer le fichier `tls-rootca-crl.srl` et le fichier `tls-subca-crl.srl`, contenant chacun la chaîne de caractères « 01 ».

Générer la LCR émise par l'AC racine et par l'AC fille.

```
$ openssl ca -gencrl -config tls-rootca.cnf -name tls_rootca \
  -crlxtns tls_rootca_crl_ext -out tls-rootca-crl.pem
$ openssl ca -gencrl -config tls-subca.cnf -name tls_subca \
  -crlxtns tls_subca_crl_ext -out tls-subca-crl.pem
```

Vérifier la cohérence globale de la chaîne en validant le certificat du serveur TLS :

```
$ cat tls-subca-crl.pem tls-rootca-crl.pem tls-rootca-crt.pem \
  > tls-cafile-chain.pem
$ openssl verify -untrusted tls-subca-crt.pem -CAfile tls-cafile-chain.pem \
  -crl_check_all -verbose tls-server-crt.pem
tls-server-crt.pem: OK
```

Convertir les deux LCR au format DER.

```
$ openssl crl -in tls-rootca-crl.pem -outform DER -out tls-rootca-crl.der
$ openssl crl -in tls-subca-crl.pem -outform DER -out tls-subca-crl.der
```

Si les points de distribution des LCR sont sur une machine autre que la machine locale, déposer les LCR au format DER aux points de distribution appropriés.

10.2. Compilation et installation du serveur web nginx

À la date de rédaction, les serveurs web libres usuels ne supportent pas nativement TLSv1.2, et il est nécessaire de procéder à leur recompilation (pour Apache, avant la parution de la branche 1.0.1 d'OpenSSL, la solution était de compiler Apache avec GnuTLS³⁵). Cette section s'intéresse à la compilation du serveur web nginx avec OpenSSL 1.0.1 sous Windows (les instructions générales sont disponibles ici³⁶), ce serveur web ayant été choisi car plus simple à compiler et à configurer qu'Apache. En environnement Linux/UNIX, la compilation par les outils classiques `make` et `gcc` ne pose aucun problème particulier, mais des explications complémentaires semblent nécessaires pour Windows.

La compilation proposée de nginx sous Windows repose sur l'environnement MSYS pour les outils GNU usuels, et sur la suite Microsoft Visual Studio Express 2010 pour le compilateur C.

Télécharger³⁷ et installer Visual Studio Express 2010, disponible gratuitement sur le site de Microsoft.

Télécharger MinGW depuis la page principale³⁸, suivre le lien *Downloads*, et sur le site Sourceforge télécharger la dernière version de l'exécutable `mingw-get-inst-...exe`, situé dans le dossier *Installer* puis `mingw-get-inst`. Installer MinGW, puis ouvrir une invite de commande DOS, et saisir la commande suivante :

```
> mingw-get install msys
```

Cette commande suppose que répertoire des binaires de MinGW a bien été ajouté à la variable d'environnement `PATH` lors de l'installation, sinon saisir le chemin complet de `mingw-get` (le répertoire des binaires de MinGW est le sous-répertoire `bin` du répertoire d'installation de MinGW).

Fermer l'invite de commande DOS, puis démarrer l'invite de commande de MSYS. Exécuter le script de post-installation :

```
$ /postinstall/pi.sh
```

Le lecteur prévoyant d'utiliser MSYS au-delà de cette section peut être intéressé par l'invite de commande améliorée `Console2`³⁹, installable pour MSYS via `$ mingw-get install msys-console` puis `$ mingw-get install msys-console` (l'exécutable devrait se trouver dans le sous-répertoire `msys\1.0\lib\Console2` du répertoire d'installation de MinGW).

Installer un client Subversion en préparation du téléchargement de la dernière branche de nginx.

35. <http://www.linuxunbound.com/2011/07/using-gnutls-with-apache-httpd-2-2/>

36. http://nginx.org/en/docs/howto_build_on_win32.html

37. <http://www.microsoft.com/express>

38. `mingw-get-inst`

39. <http://sourceforge.net/projects/console/>

Voici une piste possible pour intégrer les binaires de SVN à l'installation de MinGW/MSYS sous MSYS (cette procédure est très aisément réalisable sans passer par MSYS, mais présentant l'avantage d'installer les outils `wget` et `unzip`). La version de Subversion pour Windows⁴⁰ disponible à la date d'écriture est la version 1.7.4. Les commandes ci-dessous sont à adapter pour une version différente.

```
$ mingw-get install msys-wget
$ mingw-get install msys-unzip
$ wget http://sourceforge.net/projects/win32svn/files/1.7.4/svn-win32-1.7.4.zip/download
$ unzip svn-win32-1.7.4.zip
$ cd svn-win32-1.7.4
$ mv bin/* /mingw/bin/
$ cd ..
```

Faire un *checkout* Subversion du code source de la dernière version de nginx. En ligne de commande, la commande correspondante est (pour la version 1.1.18, qui n'est déjà plus la dernière version à la date de rédaction !) :

```
$ svn co svn://svn.nginx.org/nginx/tags/release-1.1.18
```

Télécharger le code source de `pcres`⁴¹, de `zlib`⁴² et d'`OpenSSL`⁴³. Il sera supposé ci-après que les archives zip et tar.gz correspondantes ont été téléchargées dans le même répertoire que celui d'où a été téléchargé le code source de nginx.

Si à la date de lecture la version en cours de `zlib` est toujours la version 1.2.6, alors télécharger la version beta (version 1.2.6.1)⁴⁴ (évoquée ici⁴⁵) pour éviter d'être confronté à des problèmes de compilation⁴⁶ liés à la fonction `gzflags()` sous Windows.

Préparer le code source pour la compilation de nginx, sous MSYS (en modifiant les noms des fichiers et l'emplacement des répertoires au besoin) :

```
$ cd release-1.1.18
$ mkdir -p objs/lib
$ cd objs/lib
$ unzip ../../../../pcres-8.30.zip
$ tar -xzf ../../../../zlib-1.2.6.1.tar.gz
$ tar -xzf ../../../../openssl-1.0.1.tar.gz
```

Pour `OpenSSL` version 1.0.1, deux retouches du paquetage source sont nécessaires. La première est liée à la gestion des retours chariot sous Windows dans le script `mk1mf.pl`, et est décrite dans l'annexe sur la compilation d'`OpenSSL`. La seconde modification a trait à l'utilisation de l'assembleur `MASM` (non officiellement supporté pour la compilation d'`OpenSSL` sous Windows, mais imposé par nginx) : le problème est décrit ici⁴⁷, et la solution ici⁴⁸.

40. <http://alagazam.net/>

41. <http://www.pcre.org/>

42. <http://zlib.net/>

43. <http://www.openssl.org>

44. <http://zlib.net/current/beta/zlib-1.2.6.1.tar.gz>

45. http://mail.madler.net/pipermail/zlib-devel_madler.net/2012-February/002759.html

46. http://mail.madler.net/pipermail/zlib-devel_madler.net/2012-January/002733.html

47. <http://www.mail-archive.com/openssl-dev@openssl.org/msg30681.html>

48. <http://cvs.openssl.org/chngview?cn=22302>

Configurer nginx pour la compilation (en veillant aux numéros de version dans les noms des répertoires) :

```
$ auto/configure --with-cc=cl --builddir=objs --prefix= \
--conf-path=conf/nginx.conf --pid-path=logs/nginx.pid \
--http-log-path=logs/access.log --error-log-path=logs/error.log \
--sbin-path=nginx.exe --http-client-body-temp-path=temp/client_body_temp \
--http-proxy-temp-path=temp/proxy_temp \
--http-fastcgi-temp-path=temp/fastcgi_temp \
--with-cc-opt=-DFD_SETSIZE=1024 --with-pcre=objs/lib/pcre-8.30 \
--with-zlib=objs/lib/zlib-1.2.6 --with-openssl=objs/lib/openssl-1.0.1 \
--with-select_module --with-http_ssl_module --with-ipv6
```

Fermer MSYS, puis ouvrir l'invite de commande de Visual Studio (installée dans le menu Démarrer sous un nom tel que *Visual Studio Command Prompt (2010)*), se rendre dans le répertoire racine du code source de nginx (.../release-1.1.18 ci-dessus), et démarrer la compilation, dont l'aboutissement devrait être la production de l'exécutable objs/nginx.exe :

```
> nmake -f objs/Makefile
```

Créer un répertoire d'installation, et y copier objs/nginx.exe du répertoire de compilation.

Dans le répertoire d'installation, créer les sous-répertoires logs, conf, temp et www.

10.3. Paramétrage du serveur web nginx

Dans le sous-répertoire conf, créer le fichier nginx.conf, avec le contenu suivant (en remplaçant *[racine_nginx]* par le chemin du répertoire racine de nginx, avec le caractère séparateur de sous-répertoires « / » — par exemple C:/tp-ce/nginx sous Windows) :

```
events {
    worker_connections 1024;
}

http {
    server {
        listen 80 ;

        server_name localhost;
        location / {
            root [répertoire d'installation de nginx]/www;
            index index.html;
        }
    }
}
```

10. Authentication web

Créer le fichier `index.html` ci-dessous (avec le codage UTF-8) et le placer dans le sous-répertoire `www` de `nginx`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>nginx/TLSv1.2/EV</title>
</head>
<body>
<h1>Bonjour à tous !</h1>
</body>
</html>
```

Pour valider la configuration, démarrer le serveur `nginx` (via la commande `nginx`, et s'assurer que la page `http://localhost` correspond bien au fichier `index.html` ci-dessus. Arrêter le serveur par la commande `nginx -s stop` pour passer à la configuration de SSL/TLS.

Concaténer le certificat du serveur TLS et celui de son AC émettrice (AC non racine) dans le fichier `tls-server-chain.pem`.

```
$ cat tls-server-crt.pem tls-subca-crt.pem > tls-server-chain.pem
```

Copier le fichier ainsi produit, ainsi que la clé privée du serveur `tls-server-key.pem` dans le sous-répertoire `conf` de `nginx`, et ajouter les lignes en gras ci-dessous au fichier de configuration `nginx.conf`.

```
...
http {
    server {
        listen 80 ;
        listen 443 ssl ;

        ssl_certificate  tls-server-chain.pem ;
        ssl_certificate_key  tls-server-key.pem ;

        server_name localhost;
    }
    ...
}
```

Démarrer le serveur `nginx` et tenter d'accéder à la page `https://localhost`, ce qui devrait provoquer un avertissement du navigateur, qui ne connaît pas l'AC racine de la chaîne de certification du certificat du serveur TLS. Installer l'AC racine dans le magasin de certificats de Windows (pour Internet Explorer et Chrome) ou du navigateur (pour la plupart des autres navigateurs) et vérifier que l'accès à la page web ne provoque plus d'erreur (avec au contraire une signalétique de type cadenas indiquant que la connexion est sécurisée).

Si les points de distribution des LCR sont locaux (comme dans les fichiers de configuration proposés), alors déposer les fichiers des LCR dans les répertoires appropriés : avec les fichiers ci-dessus, `tls-rootca-crl.der` et `tls-subca-crl.der` sont à placer dans le sous-répertoire `www` de `nginx`.

10.4. Mise en place des serveurs OCSP

Pour démarrer localement sur des ports HTTP arbitraires (81 et 82 ci-après) les serveurs OCSP associés d'une part aux certificats délivrés l'AC racine, et d'autre part à ceux délivrés par l'AC fille, exécuter les commandes ci-dessous (dans deux invites de commande différentes, les commandes étant bloquantes) dans le répertoire dans lequel ont été générés les certificats :

```
$ openssl ocsp -sha256 -index tls-rootca-db.txt -port 0.0.0.0:81 \
  -rsigner tls-rootca-ocsp-crt.pem -rkey tls-rootca-ocsp-key.pem \
  -CA tls-rootca-crt.pem -text
```

pour l'AC racine, et

```
$ openssl ocsp -index tls-subca-db.txt -port 0.0.0.0:82 \
  -rsigner tls-subca-ocsp-crt.pem -rkey tls-subca-ocsp-key.pem \
  -CA tls-subca-crt.pem -text
```

pour l'AC intermédiaire.

Sous les systèmes d'exploitation supportant IPv6, le serveur OCSP écoute par défaut uniquement sur les adresses IPv6 de la machine (constaté par l'auteur sous Windows 7, le problème ayant été rencontré par d'autres⁴⁹ sous Linux), d'où l'utilisation de la syntaxe non documentée `-port 0.0.0.0:port`, qui force le serveur à écouter sur les adresses IPv4 uniquement (cf. aussi ici⁵⁰ pour d'autres cas d'utilisation non documentés de `-port` en rapport avec ce sujet).

L'option `-text` permet d'afficher les requêtes reçues et les réponses envoyées dans la fenêtre du serveur, et peut être omise si par exemple le serveur OCSP est exécuté en tant que démon.

Tester le serveur OCSP depuis le répertoire des certificats à l'aide de la commande suivante :

```
$ openssl ocsp -sha256 -issuer tls-rootca-crt.pem -cert tls-subca-crt.pem \
  -CAfile tls-rootca-crt.pem -host localhost:81
```

Response verify OK

tls-subca-crt.pem: good

This Update: Apr 19 13:05:58 2012 GMT

```
$ cat tls-subca-crt.pem tls-rootca-crt.pem > tls-subca.cafile
```

```
$ openssl ocsp -sha256 -issuer tls-subca-crt.pem -cert tls-server-crt.pem \
  -CAfile tls-subca.cafile -host localhost:82
```

Response verify OK

tls-server-crt.pem: good

This Update: Apr 19 13:14:57 2012 GMT

La fenêtre des serveurs OCSP respectifs affiche les requêtes et réponses OCSP, par exemple la requête portant sur le statut du certificat de l'AC fille :

OCSP Request Data:

Version: 1 (0x0)

Requestor List:

Certificate ID:

Hash Algorithm: sha256

49. <http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=632833>

50. <http://comments.gmane.org/gmane.comp.encryption.openssl.user/40849>

10. Authentication web

```
Issuer Name Hash: EB98C924CC18F1235F2F900AE21016CB57EB3AF3F1BCABBB02E3
3A4C9B999E7C
Issuer Key Hash: 9FA4C24D06DB661D043B914527650BA763D9EBOA4181FC5BCE467
E6F72E939EF
Serial Number: E6572284531C7987
Request Extensions:
  OCSLP Nonce:
    04100B44A571D0453E8B235701C1A511FE97
```

suivie de la réponse indiquant que le certificat est valide :

```
OCSLP Response Data:
  OCSLP Response Status: successful (0x0)
  Response Type: Basic OCSLP Response
  Version: 1 (0x0)
  Responder Id: C = FR, O = Mon Organisme, OU = 0002 147258369, CN = OpenSSL T
LS Root CA OCSLP Responder
  Produced At: Apr 19 13:06:46 2012 GMT
  Responses:
    Certificate ID:
      Hash Algorithm: sha256
      Issuer Name Hash: EB98C924CC18F1235F2F900AE21016CB57EB3AF3F1BCABBB02E33A4C
9B999E7C
      Issuer Key Hash: 9FA4C24D06DB661D043B914527650BA763D9EBOA4181FC5BCE467E6F7
2E939EF
      Serial Number: E6572284531C7987
      Cert Status: good
      This Update: Apr 19 13:06:46 2012 GMT

    Response Extensions:
      OCSLP Nonce:
        04100B44A571D0453E8B235701C1A511FE97
      Signature Algorithm: sha1WithRSAEncryption
        8d:2c:75:32:3e:2b:44:84:4f:fd:d7:f9:2d:d2:99:48:4a:7e:
        ...
        c8:ea:b7:92
Certificate:
  ... certificat du répondeur OCSLP...
```

Le lecteur attentif remarquera que la signature de la réponse OCSLP utilise SHA-1, malgré le démarrage du serveur OCSLP avec l'option `-sha256`. Il se trouve que préciser l'algorithme de hachage lorsque la commande `openssl ocspl` est utilisée en mode serveur OCSLP ne sert à rien : SHA-1, algorithme de hachage par défaut d'OpenSSL dans le cas d'une signature à l'aide d'une clé RSA, est utilisé systématiquement. Pour signer avec RSA et un autre algorithme de hachage, il faut modifier le code source d'OpenSSL et le recompiler, ou créer une nouvelle commande de serveur OCSLP, ce qui fait l'objet de l'annexe F.1.

Pour faire le lien entre les URL référencées dans l'extension `authorityInfoAccess` des certificats et les serveurs OCSLP qui ont été démarrés, deux nouveaux serveurs virtuels (servant les hôtes `ocsp-rootca` et `ocsp-subca`) doivent être créés et configurés en tant que *reverse proxy* : toutes les requêtes adressées à ces hôtes sont redirigées vers une autre adresse, en l'occurrence celle des serveurs OCSLP.

Tout d'abord, ajouter dans `C:\Windows\System32\drivers\etc\hosts` (Windows) ou `/etc/hosts` (Linux/UNIX) les lignes suivantes, définissant deux alias de l'adresse IP 127.0.0.1 (appelée adresse de rebouclage ou *loopback*, et qui désigne la machine locale, dont le nom de domaine est `localhost`):

```
127.0.0.1  obsp-rootca
127.0.0.1  obsp-subca
```

Ajouter dans `nginx.conf` les blocs de configuration pour les serveurs virtuels associés à ces alias :

```
...
http {
    server {
        ...
    }

    server {
        listen    80;

        server_name obsp-rootca;
        location / {
            proxy_pass http://localhost:81/ ;
            proxy_set_header Host obsp-rootca;
        }

    }

    server {
        listen    80;

        server_name obsp-subca;
        location / {
            proxy_pass http://localhost:82/ ;
        }

    }
}
```

Démarrer `nginx`, mais ne pas tenter d'utiliser `openssl obsp` avec `-host obsp-rootca` ou `-host obsp-subca` au risque d'être déçu : en effet, cette commande génère une requête POST HTTP extrêmement basique, ne renseignant pas le nom de l'hôte dans l'en-tête HTTP `Host` (ce qui est assez peu représentatif du comportement normal d'un client HTTP), si bien que `nginx` ne sait pas qu'il doit rediriger la requête vers l'hôte virtuel `obsp-rootca` ou `obsp-subca`, et renverra une erreur HTTP (typiquement une erreur 405, *method not allowed* ou méthode non autorisée, pour avoir tenté d'adresser une requête POST à un fichier statique). Une solution simple est d'utiliser un client HTTP plus standard, qui génère une en-tête HTTP `Host`, pour envoyer la requête : l'exemple ci-après s'appuie sur `curl`⁵¹ (le site web propose de télécharger l'outil en version binaire ou source).

51. <http://curl.haxx.se/>

Générer une requête OCSP dans le fichier `tls-subca-ocspreq.der` (elle est toujours créée au format DER) :

```
$ openssl ocsp -sha256 -issuer tls-rootca-crt.pem -cert tls-subca-crt.pem \
  -reqout tls-subca-ocspreq.der
```

Constituer une requête POST HTTP avec pour en-tête `Content-Type` la valeur `application/ocsp-request` (optionnel, mais attendu par la RFC 2560), l'envoyer au serveur, et écrire la réponse dans le fichier `tls-subca-ocspresp.der` :

```
$ curl -o tls-subca-ocspresp.der --data-binary @tls-subca-ocspreq.der \
  -H "Content-Type:application/ocsp-request" http://ocsp-rootca
```

Afficher la réponse dans un format lisible :

```
$ openssl ocsp -respin tls-subca-ocspresp.der -CAfile tls-rootca-crt.pem \
  -resp_text
```

Vérifier que le serveur OCSP de l'AC fille fonctionne aussi comme prévu :

```
$ openssl ocsp -sha256 -issuer tls-rootca-crt.pem -cert tls-subca-crt.pem \
  -reqout tls-subca-ocspreq.der
$ curl -o tls-subca-ocspresp.der --data-binary @tls-subca-ocspreq.der \
  -H "Content-Type:application/ocsp-request" http://ocsp-rootca
$ openssl ocsp -respin tls-subca-ocspresp.der -CAfile tls-rootca-crt.pem \
  -resp_text
```

10.5. Paramétrage de TLSv1.2

Les instructions de cette section ne doivent pas être suivies si le navigateur utilisé ne supporte pas TLSv1.2.

Les algorithmes cryptographiques SSL/TLS implémentés par OpenSSL peuvent être recensés par la commande suivante :

```
$ openssl ciphers -v
ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH      Au=RSA  Enc=AESGCM(256) Mac=AEAD
ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH    Au=ECDSA Enc=AESGCM(256) Mac=AEAD
...
EXP-RC2-CBC-MD5          SSLv3 Kx=RSA(512) Au=RSA  Enc=RC2(40)  Mac=MD5  export
EXP-RC4-MD5              SSLv3 Kx=RSA(512) Au=RSA  Enc=RC4(40)  Mac=MD5  export
```

Restreindre cette liste aux algorithmes utilisant des clés RSA pour le chiffrement de la clé de session et pour l'authentification (RSA) et l'algorithme de hachage SHA-256 (SHA256), et imposer le chiffrement symétrique du canal (excluant l'absence de chiffrement : !NULL) :

```
$ openssl ciphers -v SHA256+RSA!NULL
AES256-SHA256          SSLv3 Kx=RSA      Au=RSA  Enc=AES(256)  Mac=SHA256
AES128-SHA256          SSLv3 Kx=RSA      Au=RSA  Enc=AES(128)  Mac=SHA256
```


Dans la liste obtenue, les algorithmes de chiffrement symétrique sont AES128 ou AES256, tous deux conformes au RGS. La chaîne consolidée définissant les algorithmes autorisés est ainsi SHA256+RSA+AES.

Il est possible de s'assurer à l'aide de l'option `-tlsv1` que les algorithmes ci-dessus sont bien supportés par TLSv1.

Ajouter les lignes en gras ci-dessous au fichier de configuration `nginx.conf` pour forcer l'utilisation du protocole TLSv1.2, et imposer l'utilisation des algorithmes conformes au RGS :

```
...
http {
    server {
        listen 80 ;
        listen 443 ssl ;

        ssl_certificate  tls-server-chain.pem ;
        ssl_certificate_key  tls-server-key.pem ;
ssl_protocols TLSv1.2 ;
ssl_ciphers SHA256+RSA+AES ;

        server_name localhost;
    }
    ...
}
```

Le fichier de configuration final est le suivant :

```
events {
    worker_connections 1024;
}

http {
    server {
        listen 80 ;
        listen 443 ssl ;

        ssl_certificate  tls-server-chain.pem ;
        ssl_certificate_key  tls-server-key.pem ;
        ssl_protocols TLSv1.2 ;
        ssl_ciphers SHA256+RSA+AES ;

        server_name localhost;
        location / {
            root [répertoire d'installation de nginx]/www;
            index index.html;
        }
    }

    server {
        listen 80;

        server_name ojsp-rootca;
        location / {
```

10. Authentification web

```
        proxy_pass http://localhost:81/ ;
        proxy_set_header Host obsp-rootca;
    }

}

server {
    listen 80;

    server_name obsp-subca;
    location / {
        proxy_pass http://localhost:82/ ;
    }
}

}
```

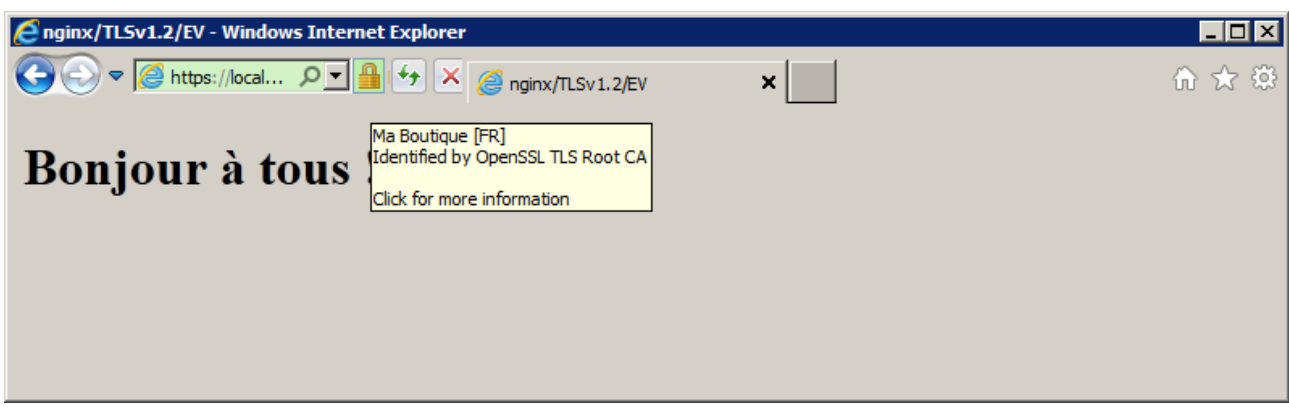
L'agrafage OCSP évoqué ci-dessus devrait être supporté⁵² dans la version 2.0 de nginx. Le lecteur intéressé par cette fonctionnalité peut mettre en place un serveur web IIS 7 ou supérieur, ou Apache (cf. instructions ici⁵³ ou ce patch⁵⁴ pour la version 2.2.6).

10.6. Internet Explorer

Installer le certificat de l'AC racine dans le magasin de certificats de Windows.

Pour qu'un certificat serveur dont la chaîne de certification mène à l'AC racine soit reconnu comme étant un certificat EV, sous Windows 7 (et Windows 2008), afficher les propriétés du certificat de l'AC racine depuis le magasin de certificats, accéder à l'onglet Validation étendue, saisir l'OID de la PC associée aux certificats EV, c'est-à-dire l'OID de la PC tel qu'inscrit dans l'extension `certificatePolicies` du certificat serveur TLS, et valider.

Accéder à l'adresse `https://localhost`, et observer que l'URL et le nom de l'entité certifiée (telle que renseignée dans le champ 0 du certificat serveur, pour rappel le champ CN identifie l'application) sont affichées sur fond vert, représentant l'utilisation de certificats EV.



52. <http://mailman.nginx.org/pipermail/nginx-devel/2011-June/000956.html>

53. <http://www.imperialviolet.org/2009/12/20/setting-up-ocsp.html>

54. https://issues.apache.org/bugzilla/show_bug.cgi?id=43822

Dans la configuration mise en place ici, IE9 vérifie le statut de révocation en consultant les LCR.

Pour s'assurer que le statut de révocation est bien pris en compte, une LCR contenant le numéro de série du certificat peut être émise et déposée dans le répertoire servant les LCR. Il est alors nécessaire de purger les caches des LCR de Windows pour forcer le chargement de la nouvelle LCR, à l'aide des commandes suivantes (la première vide le cache disque, la seconde invalide le cache mémoire) :

```
>certutil -urlcache CRL delete
> certutil -setreg chain\ChainCacheResyncFiletime @now
```

La commande `certutil -v urlcache -CRL` affiche le contenu du cache disque des LCR. Pour plus d'informations sur la gestion du statut de révocation des certificats sous Windows et la commande `certutil`, se reporter à la documentation officielle⁵⁵.

Le serveur OCSP n'est pas utilisable en l'état par IE9, celui-ci générant des requêtes OCSP au format RFC 5019 (section 5) via HTTP GET, qui n'est pas supporté par le serveur OCSP en ligne de commande d'OpenSSL⁵⁶. Un script CGI pourrait par exemple être développé pour convertir la requête d'IE9 en requête POST « classique » avant de la passer à OpenSSL.

À titre d'illustration, voici le journal des requêtes HTTP du serveur, mettant en évidence l'interrogation (en échec) du serveur OCSP et le téléchargement des LCR.

```
127.0.0.1 - - [15/Apr/2012:10:11:32 +0200] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)"
127.0.0.1 - - [15/Apr/2012:10:11:34 +0200] "-" 400 0 "-" "-"
127.0.0.1 - - [15/Apr/2012:10:11:55 +0200] "GET /MEowSDBGMEQwQjAJBgUrDgMCGGUABBT
npomF%2FJkCIe62ShdoLJJBGBfQ0AQU9DQZRK0Uy40EscKbCj6oDJB5aXwCCQDmVyKEUxx5hw%3D%3D
HTTP/1.1" 200 5 "-" "Microsoft-CryptoAPI/6.1"
127.0.0.1 - - [15/Apr/2012:10:11:55 +0200] "GET /MEowSDBGMEQwQjAJBgUrDgMCGGUABBT
npomF%2FJkCIe62ShdoLJJBGBfQ0AQU9DQZRK0Uy40EscKbCj6oDJB5aXwCCQDmVyKEUxx5hw%3D%3D
HTTP/1.1" 502 173 "-" "Microsoft-CryptoAPI/6.1"
127.0.0.1 - - [15/Apr/2012:10:12:00 +0200] "GET /tls-rootca-crl.der HTTP/1.1" 20
0 474 "-" "Microsoft-CryptoAPI/6.1"
127.0.0.1 - - [15/Apr/2012:10:12:05 +0200] "GET /MEkwRzBFMEMwQTAJBgUrDgMCGGUABBT
2IXg%2BD7s0Dwo6NZoErGbC4LF2ZwQUMmQEAQYr2Zgp2wEl0Y026synN4QCCDaoL4MkMnHI HTTP/1.1
" 200 5 "-" "Microsoft-CryptoAPI/6.1"
127.0.0.1 - - [15/Apr/2012:10:12:07 +0200] "-" 400 0 "-" "-"
127.0.0.1 - - [15/Apr/2012:10:12:07 +0200] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)"
127.0.0.1 - - [15/Apr/2012:10:12:08 +0200] "-" 400 0 "-" "-"
127.0.0.1 - - [15/Apr/2012:10:12:10 +0200] "GET /tls-rootca-crl.der HTTP/1.1" 30
4 0 "-" "Microsoft-CryptoAPI/6.1"
127.0.0.1 - - [15/Apr/2012:10:12:10 +0200] "GET /tls-subca-crl.der HTTP/1.1" 200
481 "-" "Microsoft-CryptoAPI/6.1"
127.0.0.1 - - [15/Apr/2012:10:12:12 +0200] "GET /tls-subca-crl.der HTTP/1.1" 200
481 "-" "Microsoft-CryptoAPI/6.1"
127.0.0.1 - - [15/Apr/2012:10:12:20 +0200] "POST / HTTP/1.1" 499 0 "-" "Microsof
t-CryptoAPI/6.1"
```

55. <http://technet.microsoft.com/en-us/library/ee619754%28v=ws.10%29.aspx>

56. <http://comments.gmane.org/gmane.comp.encryption.openssl.user/42191>

10.7. Firefox

L'acceptation d'une AC de test EV sous Firefox nécessite une version de débogage de Firefox, soit compilée soi-même (une opération relativement laborieuse décrite sur le site de Mozilla⁵⁷), soit pré-compilée (à récupérer dans le dépôt FTP de Mozilla⁵⁸, dans le sous-répertoire mozilla-release-plate-forme-debug, en notant que la stabilité de ces versions intermédiaires est variable).

L'AC racine EV de test doit être déclarée dans un fichier de configuration *ad hoc*, nommé `test_ev_roots.txt`. Ce fichier a la structure suivante :

```
1_fingerprint empreinte SHA1 au format hexadécimal xx:xx:...:xx
2_readable_oid OID de la politique de certification du certificat EV
3_issuer codage Base64 du DER du champ Issuer du certificat
4_serial codage Base64 de la représentation DER du numéro de série du certificat
```

Pour la ligne 1_fingerprint, utiliser l'option `-fingerprint` d'`openssl x509` :

```
$ openssl x509 -in tls-rootca-crt.pem -fingerprint -noout
SHA1 Fingerprint=BC:1F:00:D2:05:57:9F:9F:64:6F:FB:40:AE:AC:DE:46:F7:57:49:A1
```

Pour obtenir la ligne 3, déterminer d'abord où démarre le champ Issuer :

```
$ openssl x509 -in tls-rootca-crt.pem -outform DER | openssl asn1parse \
-inform DER -i
 0:d=0  hl=4 l= 987 cons: SEQUENCE
 4:d=1  hl=4 l= 707 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=   9 prim: INTEGER           :BE6D461CD887515D
24:d=2  hl=2 l=  13 cons: SEQUENCE
26:d=3  hl=2 l=   9 prim: OBJECT            :sha256WithRSAEncryption
37:d=3  hl=2 l=   0 prim: NULL
39:d=2  hl=2 l=  92 cons: SEQUENCE
41:d=3  hl=2 l=  11 cons: SET
43:d=4  hl=2 l=   9 cons: SEQUENCE
45:d=5  hl=2 l=   3 prim: OBJECT            :countryName
50:d=5  hl=2 l=   2 prim: PRINTABLESTRING  :FR
54:d=3  hl=2 l=  22 cons: SET
...
```

Ici la SEQUENCE définissant le champ Issuer démarre à l'octet 39. Extraire ce champ puis coder sa représentation DER en Base64 :

```
$ openssl x509 -in tls-rootca-crt.pem -outform DER | openssl asn1parse \
-inform DER -strparse 39 -out issuer.der
 0:d=0  hl=2 l=  92 cons: SEQUENCE
 2:d=1  hl=2 l=  11 cons: SET
 4:d=2  hl=2 l=   9 cons: SEQUENCE
 6:d=3  hl=2 l=   3 prim: OBJECT            :countryName
```

57. https://developer.mozilla.org/en-US/docs/Developer_Guide/Build_Instructions

58. <ftp://ftp.mozilla.org/pub/mozilla.org/firefox/tinderbox-builds/>

```

11:d=3  hl=2 l= 2 prim: PRINTABLESTRING :FR
15:d=1  hl=2 l= 22 cons: SET
17:d=2  hl=2 l= 20 cons: SEQUENCE
19:d=3  hl=2 l= 3 prim: OBJECT :organizationName
24:d=3  hl=2 l= 13 prim: PRINTABLESTRING :Mon Organisme
39:d=1  hl=2 l= 23 cons: SET
41:d=2  hl=2 l= 21 cons: SEQUENCE
43:d=3  hl=2 l= 3 prim: OBJECT :organizationalUnitName
48:d=3  hl=2 l= 14 prim: PRINTABLESTRING :0002 147258369
64:d=1  hl=2 l= 28 cons: SET
66:d=2  hl=2 l= 26 cons: SEQUENCE
68:d=3  hl=2 l= 3 prim: OBJECT :organizationalUnitName
73:d=3  hl=2 l= 19 prim: PRINTABLESTRING :OpenSSL TLS Root CA

```

```
$ openssl base64 -in issuer.der
```

```
MFwxCzAJBgNVBAYTAkZSMRYwFAYDVQQKEw1Nb24gT3JnYW5pc21lMRcwFQYDVQQL
Ew4wMDAyIDEONzI1ODM2OTEcMBoGA1UECxMTT3B1b1NTTCBUTFMgUm9vdCBDQQ==
```

La ligne 4_serial nécessite un peu plus de précaution, car il s'agit ici d'obtenir le codage Base64 de la représentation DER du numéro de série du certificat sans les en-têtes DER (lesquelles en-têtes identifient le type INTEGER et précisent la longueur du contenu).

Repérer l'emplacement du numéro de série :

```

$ openssl x509 -in tls-rootca-crt.pem -outform DER | openssl asn1parse \
-inform DER -i
 0:d=0  hl=4 l= 979 cons: SEQUENCE
 4:d=1  hl=4 l= 699 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= 9 prim: INTEGER :AD49132769D200C5

```

Le contenu du numéro de série démarre à l'octet 15 (la structure complète est à l'octet 13, dont les hl=2 premiers octets sont à ignorer) et a une longueur de l=9 octets. Extraire ce contenu, en ignorant les erreurs (ce contenu n'étant pas une structure DER valide), puis le coder en Base64.

```

$ openssl x509 -in tls-rootca-crt.pem -outform DER | openssl asn1parse \
-inform DER -i -offset 15 -length 9 -out serialnum.bin
Error in encoding
6360:error:0D07207B:asn1 encoding routines:ASN1_get_object:header too long:.\crypto\asn1\asn1_lib.c:150:

```

```

$ openssl base64 -in serialnum.bin
AL5tRhzyH1Fd

```

À partir des données ci-dessus créer le fichier test_ev_roots.txt.

```

1_fingerprint BC:1F:00:D2:05:57:9F:9F:64:6F:FB:40:AE:AC:DE:46:F7:57:49:A1
2_readable_oid 1.2.840.113556.1.8000.2554.29563.49294.43847.16581.44480.6974059.
2493436.1.3
3_issuer MFwxCzAJBgNVBAYTAkZSMRYwFAYDVQQKEw1Nb24gT3JnYW5pc21lMRcwFQYDVQQL
Ew4wMDAyIDEONzI1ODM2OTEcMBoGA1UECxMTT3B1b1NTTCBUTFMgUm9vdCBDQQ==
4_serial AL5tRhzyH1Fd

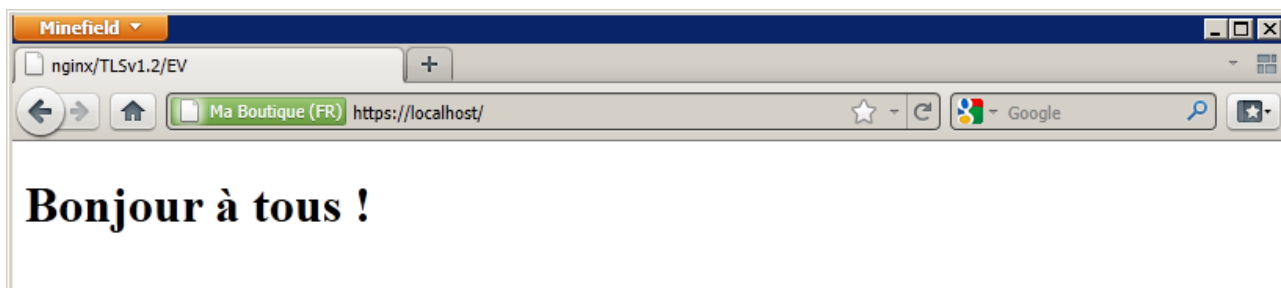
```

L'annexe F.2 propose un outil de génération du fichier `test_ev_roots.txt` à partir d'un certificat et de l'OID de la politique de certification EV.

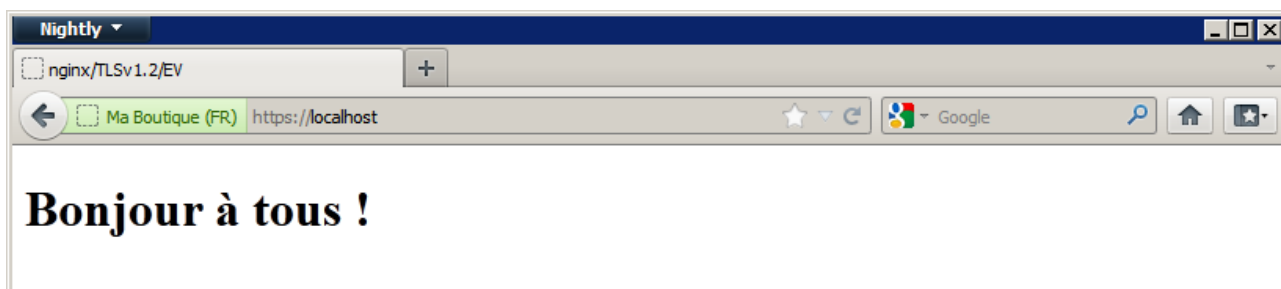
Déplacer ce fichier dans le répertoire du profil utilisateur de Firefox⁵⁹.

Donner à la variable d'environnement `ENABLE_TEST_EV_ROOTS_FILE` la valeur 1, démarrer Firefox et se rendre à l'URL `https://localhost`.

Les captures d'écran ci-dessous représentent deux rendus, le premier sous une version de Firefox 4.0 beta 8 *debug* précompilée.



Voici le rendu sous une version Firefox 11 *nightly* compilée à partir du code source.



Le nom de l'organisation « Ma Boutique » sur fond vert dans la barre l'URL indique que le serveur est authentifié à l'aide d'un certificat EV.

En mode EV, Firefox ne consulte pas les CRL⁶⁰, l'accès aux serveurs OCSP est donc indispensable.

Pour plus d'informations sur le comportement des navigateurs en matière de vérification du statut des certificats, se reporter à cet article⁶¹.

59. <http://support.mozilla.org/en-US/kb/profiles-where-firefox-stores-user-data>

60. https://wiki.mozilla.org/CA:EV_Revocation_Checking

61. <http://blog.spiderlabs.com/2011/04/certificate-revocation-behavior-in-modern-browsers.html>

Chapitre 11 — Horodatage

Dans ce chapitre, les profils retenus pour l'unité d'horodatage et les jetons d'horodatage sont ceux définis dans l'annexe 2 de l'annexe A12 du [RGS].

11.1. Horodatage de données

Génération du certificat de l'unité d'horodatage

Générer un bi-clé RSA de 2048 bits pour l'unité d'horodatage.

```
$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048
.....+++
.....+++
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBAwggSjAgEAAoIBAQDIkMQ/ueoN/rag
2BHfa6SwKUtoF2GatLyGJwzzBPbd0ITcSRZLw/otceBR1B9r18zLIF7dvG2IDgQ
e1Zwe1TmYAlgxRrZzyZz9zQtEfk/rxVkiZP6W4R/GL0igy+5+ob+NAn+UC2PZ6ff
2XVBogNpAbIUhNqhIj34YtcLMdyEH+h76rczwBNPFGVpKUsf/dHE/3+eAnbpJILz
IhJC5e1Y4winp01tN12RGRfjZ77qoKFXf4Wf0Y/hZ+g01QULGm0UMApp+34wHgFA
rTy3+80NeR1n/kdxq0+Msf0/5N2wK9KFnZPMY+AaTmahPTwo7K1KRWW5F6kV/aYa
WR0oNXwVAgMBAAECggEBAL7TDgVby9b466XpLkgWUKDTCU4pM+rY7IHu7F9T4ejD
dPiug1jFMBHMMKRMJ8cNX1SAGcnwh1UzHmSAFOP8U/DEmueZYUyujPV01A21tiHj
YUVAqKxg+pU6Ymk9y+D0/g9KT9/JRS/V/aE7cxa5IO191YNOS9dXjwK0d+/MxvRk
/cxJIUt42ynba08ahFf53L0LW1YMkWj+ZwZRktyOFVZOEHsY0uVUd6fWYLLpSq58
08Ud6cE7dv/1F8zqRDf2y29btz0jZcj5XWoBN2hMGntdFSxPbAXjfZlqXGhXC4kA
jbTPu9oZQcKEq2mxWDsgCk JrGGeEn/pp488HFREwZsECgYEA/FdIPCY6janCDwr/
qQDbb2m8NofRfyrgKgIa4hzYfBHEcwZIUQ2aA6ZZmpRPZ12U8LEyoEsdxFXV8Q4N
8Sc6625lnmPIeoVD4GNuh02Zhb2i6VWKF0wnBbJj2tueegJl/Behqioidl5LdpLW
jwKWNoyXCdbNx8pVT+4TnXA4wMkCgYEAy3lJxM+3I72ujcyldc3JCRp32ZCanXJY
4BQDcnhGfJ93S+halTRhqXdm80Hd7uglHF3Why9aVVTZuT7oYCNh502qqrVNrBf3
Oipcnb6mkD4xVDJcLV2kZXR8kKYeWidUrSmvi8H1Tz7+lgkcJulT2u6QnVljNrAD
4dVJYM2F8u0CgYBXtRKzbWousRF1XxJLsLOUvwCfR4fmlowBtGeZrgME5LwOauG+
CF8+44QDeYc7gk1xd3WsR0+aXWdWONdZuZP+HsoNSot1InrcNFNHjg4pSy+jwIqV
3G83yLBycqFbsRe3jpMvbS07Vr00Aq222WRSo30f+1xdFB0hC5QfxdDEGQKBgDSM
6K557xS+An0A46Lf0RKuOVFRdX1AcQ++W3207rV1AfyK0ApE8wwt6pfc5RK0xhDj
c2qKokvz1B+xzZY2zSuw7ikWQ/IOZ1eRZUYgXShXx6m1L/VPgAvr39gD70bdbZMj
dwEjYNXMsTfStoCeNlg0OS6yTVtsbjQ/P6p0qk+pAoGAFOWAMF9qvqefgy8X1yOr
UVQLY/vMT04wGQ8Tx1WSdQ5+W51TdfTp2itz4kZyOYQN0V5FmrQsdy+vzOCWKb88
gZcIO+y5YVVxdqgC2rr8Ss5i49lH8XivQoISMtES8ZSDhhxhru+SsTZiG8cUY63q
eUQ/xPQVKac394yeY8kcfS0=
-----END PRIVATE KEY-----
```

11. Horodatage

Copier la clé ci-dessus dans un fichier nommé `tsa-key.pem` (ts pour *timestamp* ou horodatage), ou utiliser la commande ci-dessus avec l'option `-out tsa-key.pem` pour poursuivre avec une autre clé.

Générer la CSR, dans le fichier `tsa-req.pem`.

```
$ openssl req -new -key tsa-key.pem \  
-subj "/C=FR/O=Mon Entreprise/OU=0002 123456789/CN=OpenSSL TSA" -sha256 \  
-config req-empty.cnf -out tsa-req.pem
```

Créer le fichier de configuration `tsa-crt.cnf` suivant, définissant le profil du certificat de l'unité d'horodatage.

```
[ts_ext]  
subjectKeyIdentifier = hash  
authorityKeyIdentifier = keyid  
keyUsage = critical,digitalSignature  
certificatePolicies = \  
    1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.4  
crlDistributionPoints = URI:http://tiny.cc/LatestCRL  
basicConstraints = critical,CA:FALSE  
extendedKeyUsage = critical,timeStamping
```

Générer aléatoirement le numéro de série du certificat de l'unité d'horodatage.

```
$ openssl rand -hex 8 -out ca-crt.srl
```

Déclencher la génération du certificat.

```
$ openssl x509 -req -in tsa-req.pem -extfile tsa-crt.cnf -extensions tsa_ext \  
-CA ca-crt.pem -CAkey ca-key.pem -CAserial ca-crt.srl -sha256 -days 730 \  
-out tsa-crt.pem  
Signature ok  
subject=/C=FR/O=Mon Entreprise/OU=0002 123456789/CN=OpenSSL TSA  
Getting CA Private Key
```

Afficher le contenu du certificat.

```
$ openssl x509 -in tsa-crt.pem -noout -text  
Certificate:  
    Data:  
        Version: 3 (0x2)  
        Serial Number:  
            d3:b9:4f:04:dd:d1:a0:40  
    Signature Algorithm: sha256WithRSAEncryption  
    Issuer: C=FR, O=Mon Entreprise, OU=0002 123456789, OU=OpenSSL Root CA  
    Validity  
        Not Before: Jun 16 20:34:01 2012 GMT  
        Not After : Jun 16 20:34:01 2014 GMT  
    Subject: C=FR, O=Mon Entreprise, OU=0002 123456789, CN=OpenSSL TSA  
    Subject Public Key Info:  
        Public Key Algorithm: rsaEncryption  
        Public-Key: (2048 bit)
```



```

Modulus:
  00:c8:90:c4:3f:b9:ea:0d:fe:b6:a0:d8:11:df:6b:
  ...
  7c:15
Exponent: 65537 (0x10001)
X509v3 extensions:
  X509v3 Subject Key Identifier:
    BA:0B:BB:AF:E3:25:46:56:FC:13:86:92:D2:15:40:62:DB:16:6A:4A
  X509v3 Authority Key Identifier:
    keyid:4C:6D:87:93:82:F7:2D:2C:07:23:A2:0F:E0:71:2D:17:3F:39:F3:8

```

F

```

X509v3 Key Usage: critical
  Digital Signature
X509v3 Certificate Policies:
  Policy: 1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224
.8393003.10972002.1.4

X509v3 CRL Distribution Points:

  Full Name:
    URI:http://tiny.cc/LatestCRL

X509v3 Basic Constraints: critical
  CA:FALSE
X509v3 Extended Key Usage: critical
  Time Stamping
Signature Algorithm: sha256WithRSAEncryption
  0f:9a:d3:af:e4:56:65:ba:7c:3b:92:ca:a0:fe:2d:49:37:8d:
  ...
  ed:c8:c9:9a

```

Émission d'un jeton d'horodatage

Générer une requête d'horodatage portant sur le fichier data.txt.

```
$ openssl ts -query -data data.txt -sha256 -out data.tsq
```

Afficher le contenu de la requête, en notant que l'empreinte référencée est bien l'empreinte SHA-256 du fichier data.txt.

```
$ openssl ts -query -in data.tsq -text
```

```

Version: 1
Hash Algorithm: sha256
Message data:
  0000 - 89 bd 92 28 6d 6c 80 14-c0 60 30 b2 5f 8b 40 cc   ... (ml...`0._.@.
  0010 - 1d 56 56 d4 b3 b7 b4 83-18 74 f5 0d 6f 55 57 f3   .VV.....t..oUW.
Policy OID: unspecified
Nonce: 0x4C815608C816687B
Certificate required: no
Extensions:

```

11. Horodatage

Cette requête correspond au type `TimeStampReq` défini dans la RFC 3161, qui spécifie TSP (*Time-Stamp Protocol*, ou protocole d'horodatage).

Créer le fichier de configuration `tsa-ts.cnf` avec le contenu suivant, pour définir les caractéristiques du jeton d'horodatage.

```
[tsa]
serial = tsa-ts.srl
certs = ca-crt.pem
crypto_device = builtin
default_policy = \
    1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.5
other_policies = \
    1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.5
digests = sha256
```

Contrairement à ce que laisse entendre la documentation de la commande `openssl ts`, tous les champs définis ci-dessus sont obligatoires si leur éventuelle option en ligne de commande équivalente n'a pas été utilisée.

Générer une réponse à partir de la requête.

```
$ openssl ts -reply -config tsa-ts.cnf -section tsa -queryfile data.tsq \
    -inkey tsa-key.pem -signer tsa-crt.pem -out data.tsr
Using configuration from tsa-ts.cnf
Warning: could not open file tsa-ts.srl for reading, using serial number: 1
Response has been generated.
```

Afficher le contenu de la réponse.

```
$ openssl ts -reply -in data.tsr -text
Status info:
Status: Granted.
Status description: unspecified
Failure info: unspecified

TST info:
Version: 1
Policy OID: 1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.5
Hash Algorithm: sha256
Message data:
    0000 - 89 bd 92 28 6d 6c 80 14-c0 60 30 b2 5f 8b 40 cc    ...(ml...`0._.@.
    0010 - 1d 56 56 d4 b3 b7 b4 83-18 74 f5 0d 6f 55 57 f3    .VV.....t..oUW.
Serial number: 0x01
Time stamp: Jun 16 21:13:09 2012 GMT
Accuracy: unspecified
Ordering: no
Nonce: 0x4C815608C816687B
TSA: unspecified
Extensions:
```

L'algorithme de hachage utilisé par openssl ts dans les jetons d'horodatage est codé en dur, et est SHA-1. La section de code concernée se situe dans le fichier source `crypto/ts/ts_rsp_sign.c`, dans la fonction `TS_RESP_sign()` :

```
if (!(si = PKCS7_add_signature(p7, ctx->signer_cert,
                             ctx->signer_key, EVP_sha1())))
```

Il n'est pas possible de modifier cet algorithme sans modifier et recompiler l'ensemble du code source d'OpenSSL. Une autre solution est de passer par la construction manuelle du jeton d'horodatage.

Valider la réponse par rapport à la requête d'horodatage.

```
$ openssl ts -verify -queryfile data.tsq -in data.tsr -CAfile ca-crt.pem \
-untrusted tsa-crt.pem
```

Verification: OK

La réponse correspond à la structure `TimeStampResp` de la RFC 3161, dont la syntaxe est la suivante :

```
TimeStampResp ::= SEQUENCE {
    status                PKIStatusInfo,
    timeStampToken         TimeStampToken OPTIONAL
}
```

Le jeton d'horodatage correspond au champ `timeStampToken` de cette structure. L'extraire à l'aide de la commande suivante.

```
$ openssl ts -reply -in data.tsr -token_out -out data.tst
```

Valider le jeton d'horodatage par rapport aux données faisant l'objet de l'horodatage.

```
$ openssl ts -verify -data data.txt -in data.tst -CAfile ca-crt.pem \
-untrusted tsa-crt.pem -token_in
```

Verification: OK

11.2. Serveur d'horodatage

Certaines applications de signature électronique permettent d'interroger un serveur d'horodatage pour obtenir une réponse à une requête d'horodatage en ligne. Cette section propose un script Perl implémentant un serveur d'horodatage minimaliste capable de produire une réponse `TimeStampResp` à une requête `TimeStampReq`.

En pré-requis à l'utilisation de ce script, Perl doit être installé, ainsi que le module Perl `HTTP::Daemon`⁶², qui inclus avec les distributions usuelles de Perl (y compris `ActivePerl`⁶³ et `Strawberry Perl`⁶⁴ sous Windows).

62. <http://search.cpan.org/~gaas/HTTP-Daemon-6.01/>

63. <http://www.activestate.com/activeperl>

64. <http://strawberryperl.com/>

11. Horodatage

Créer le fichier `tsadaemon.pl` suivant :

```
use strict ;
use warnings ;

use HTTP::Daemon;

# Change values below as required
my $port = 80 ;

my $tsa_cnf_file = 'tsa-ts.cnf' ;
my $tsa_section = 'tsa' ;
my $tsa_key_file = 'tsa-key.pem' ;
my $tsa_crt_file = 'tsa-crt.pem' ;
# Change values above as required

my $tmp_infile = 'temp.tsq' ;
my $tmp_outfile = 'temp.tsr' ;

my $daemon = HTTP::Daemon->new(LocalPort => $port) || die;

print 'Timestamp server running at URL ' . $daemon->url . "\n";

while (my $conn = $daemon->accept) {
    while (my $req = $conn->get_request) {
        if ($req->method eq 'POST') {
            # Write binary TimeStampReq from incoming HTTP POST request
            # to a temporary file
            open REQ, '>'.$tmp_infile ;
            binmode REQ ;
            print REQ $req->content ;
            close REQ ;

            # Generate TimeStampResp from TimeStampReq by invoking openssl
            unlink $tmp_outfile if -e $tmp_outfile ;
            system ('openssl', 'ts', '-reply', '-config', $tsa_cnf_file,
                '-section', $tsa_section, '-queryfile', $tmp_infile,
                '-inkey', $tsa_key_file, '-signer', $tsa_crt_file,
                '-out', $tmp_outfile) ;

            # If a file was generated by the previous command, then send it back to
            # the client...
            if (-e $tmp_outfile) {
                $conn->send_status_line ;
                $conn->send_header('Content-Type', 'application/timestamp-reply') ;
                $conn->send_header('Content-Length', -s $tmp_outfile) ;
                $conn->send_crlf ;
                $conn->send_file($tmp_outfile) ;
                $conn->close ;
            }
            # ... otherwise send an HTTP error
        } else {
```

```

        $conn->send_error() ;
    }
}
else {
    $conn->send_error()
}
}
$conn->close;
undef($conn);
}

```

Le serveur d'horodatage peut être interrogé à l'aide d'une requête POST HTTP sur le port 80 (par défaut, modifier la variable `$port` au besoin), et produit — si tout se passe correctement — une réponse HTTP avec pour en-tête `Content-Type: application/timestamp-reply`, comme prévu par la section 3.4 de la RFC 3161.

Placer le script dans le même répertoire que le fichier de configuration (`tsa-ts.cnf` par défaut, dans la variable `$tsa_cnf_file`), le fichier de la clé privée (`tsa-key.pem`, défini par la variable `$tsa_key_file`) et le fichier PEM du certificat de l'unité d'horodatage (`tsa-crt.pem`, variable `$tsa_crt_file`). Au besoin modifier la variable `$tsa_section` si la section du fichier de configuration pour l'unité d'horodatage est différente de `tsa`. Placer dans le même répertoire les fichiers référencés par cette section du fichier de configuration (en particulier : numéro de série et certificats supplémentaires).

Démarrer le serveur d'horodatage :

```
$ perl tsadaemon.pl
Timestamp server running at URL http://monserveur/
```

Tester le serveur d'horodatage en lui soumettant une requête d'horodatage à l'aide du client en ligne de commande `curl` (cf. section 10.4) :

```
$ curl -v -H "Content-Type: application/timestamp-query" \
  --data-binary @data.tsq -o data.tsadaemon.tsr http://localhost/
```

% Total	% Received	% Xferd	Average Speed		Time	Time	Time	Current
			Dload	Upload	Total	Spent	Left	Speed
100	802	100	736	100	66	1004	90	--:--:-- --:--:-- --:--:-- 1004

Afficher la réponse contenue dans le fichier `data.tsadaemon.tsr` et vérifier qu'il est valide, en utilisant les commandes proposées précédemment.

Le serveur d'horodatage proposé est volontairement extrêmement basique. Parmi ses limites, il ne supporte pas les connexions simultanées, ne vérifie pas que l'en-tête HTTP `Content-Type: application/timestamp-query` est présente, ne contrôle pas les données reçues avant de les écrire et de les envoyer à la commande `openssl`, et ne verrouille pas ni ne supprime les fichiers temporaires qu'il utilise.

11.3. Construction d'un jeton d'horodatage

Un jeton d'horodatage conforme à la RFC 3161 est une signature électronique au format CMS, enveloppant notamment les informations contenues dans la requête à laquelle fait suite le jeton ain-

si que la data et l'heure. Les méthodes utilisées dans la section 8.5 sont donc réutilisables pour constituer le jeton d'horodatage. La structure `TimeStampToken` à générer pour reconstruire le jeton d'horodatage est la suivante (les champs optionnels non utilisés ont été omis) :

```
TimeStampToken ::= SEQUENCE {
    contentType ContentType,
    content [0] EXPLICIT SEQUENCE {
        version CMSVersion,
        digestAlgorithms SET OF DigestAlgorithmIdentifier,
        encapContentInfo SEQUENCE {
            eContentType ContentType,
            eContent [0] EXPLICIT OCTET STRING OPTIONAL
        },
        signerInfos SET OF SEQUENCE {
            version CMSVersion,
            sid SignerIdentifier,
            digestAlgorithm DigestAlgorithmIdentifier,
            signedAttrs [0] IMPLICIT SET OF Attribute OPTIONAL,
            signatureAlgorithm SignatureAlgorithmIdentifier,
            signature SignatureValue,
        }
    }
}
```

Les données enveloppées par la signature électronique, ou plus exactement les données dont l'empreinte est référencée dans les attributs signés de la signature, est porté par le champ `eContent`, qui est le codage DER de la structure `TSTInfo` suivante :

```
TSTInfo ::= SEQUENCE {
    version INTEGER { v1(1) },
    policy TSAPolicyId,
    messageImprint SEQUENCE {
        hashAlgorithm AlgorithmIdentifier,
        hashedMessage OCTET STRING
    },
    serialNumber INTEGER,
    genTime GeneralizedTime,
    nonce INTEGER OPTIONAL,
}
```

Constitution de la structure `TSTInfo`

Créer le fichier `tst-TSTInfo.asn.cnf` ci-dessous, représentant la structure `TSTInfo`.

```
asn1 = SEQUENCE:tstInfo
```

```
[tstInfo]
version = INTEGER:1
policy = OID:\
1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.5
messageImprint = SEQUENCE:tsq_messageImprint
```

```

serialNumber = INTEGER:1
genTime = GENERALIZEDTIME:20120616211309Z
nonce = INTEGER:0x4C815608C816687B

[tsq_messageImprint]
hashAlgorithm = SEQUENCE:tsq_hashAlgorithm
hashedMessage = FORMAT:HEX,OCTETSTRING:\
89bd92286d6c8014c06030b25f8b40cc1d5656d4b3b7b4831874f50d6f5557f3

[tsq_hashAlgorithm]
algorithm = OID:sha256
parameters = NULL

```

Dans ce fichier, les champs `messageImprint` et `nonce` reprennent les valeurs présentes dans la requête d'horodatage. En particulier, le champ `hashedMessage` contient l'empreinte SHA-256 des données faisant l'objet de l'horodatage.

Générer le codage DER de la structure `TSTInfo`.

```

$ openssl asn1parse -genconf tst-TSTInfo.asn.cnf -i -out tst-TSTInfo.der
  0:d=0  hl=2 l= 122 cons: SEQUENCE
    2:d=1  hl=2 l=   1 prim:  INTEGER           :01
    5:d=1  hl=2 l=  36 prim:  OBJECT           :1.2.840.113556.1.8000.2554.4731
1.54169.61548.20478.40224.8393003.10972002.1.5
    43:d=1  hl=2 l=  49 cons: SEQUENCE
    45:d=2  hl=2 l=  13 cons: SEQUENCE
    47:d=3  hl=2 l=   9 prim:  OBJECT           :sha256
    58:d=3  hl=2 l=   0 prim:  NULL
    60:d=2  hl=2 l=  32 prim:  OCTET STRING     [HEX DUMP]:89BD92286D6C8014C060
30B25F8B40CC1D5656D4B3B7B4831874F50D6F5557F3
    94:d=1  hl=2 l=   1 prim:  INTEGER           :01
    97:d=1  hl=2 l=  15 prim:  GENERALIZEDTIME :20120616211309Z
   114:d=1  hl=2 l=   8 prim:  INTEGER           :4C815608C816687B

```

Calculer l'empreinte SHA-1 du fichier résultant.

```

$ openssl sha1 tst-TSTInfo.der
SHA1(tst-TSTInfo.der)= 0f6d6f384c1cba1e97921fff3d192920af7feae8

```

Comme indiqué dans la note page 129, OpenSSL utilise l'algorithme de hachage SHA-1 dans les jetons d'horodatage. Le lecteur peut remplacer cet algorithme par SHA-256 (et toutes les références à `sha1` par `sha256`) s'il souhaite générer un jeton d'horodatage conforme au RGS.

Constitution et signature des attributs signés

Créer le fichier `tst-signedAttrs.asn.cnf` suivant, qui décrit le SET OF `Attribute` constituant les attributs signés du jeton d'horodatage.

```

asn1 = SET:tst_signedAttrs

[tst_signedAttrs]
attr_contentType = SEQUENCE:tst_attr_contentType

```

11. Horodatage

```
attr_signingTime = SEQUENCE:tst_attr_signingTime
attr_messageDigest = SEQUENCE:tst_attr_messageDigest
attr_signingCertificate = SEQUENCE:tst_attr_signingCertificate
```

```
[tst_attr_contentType]
attrType = OID:contentType
attrValues = SET:tst_attr_contentType_values
```

```
[tst_attr_contentType_values]
contentType = OID:id-smime-ct-TSTInfo
```

```
[tst_attr_signingTime]
attrType = OID:signingTime
attrValues = SET:tst_attr_signingTime_values
```

```
[tst_attr_signingTime_values]
attr_signingTime_value = UTCTIME:120616211309Z
```

```
[tst_attr_messageDigest]
attrType = OID:messageDigest
attrValues = SET:tst_attr_messageDigest_values
```

```
[tst_attr_messageDigest_values]
messageDigest = FORMAT:HEX,OCTETSTRING:\
0f6d6f384c1cba1e97921fff3d192920af7feae8
```

```
[tst_attr_signingCertificate]
attrType = OID:id-smime-aa-signingCertificate
attrValues = SET:tst_attr_signingCertificate_values
```

Les trois premiers attributs ont été vus dans la section 8.5. Une petite mention toutefois pour l'attribut messageDigest, qui contient l'empreinte de la structure TSTInfo générée précédemment, c'est-à-dire l'empreinte de la valeur de l'OCTET STRING constituant le champ eContent.

L'attribut signingCertificate référence le certificat de signature de l'unité d'horodatage, et est de type SigningCertificate, défini par la RFC 2634, avec la structure ASN.1 suivante :

```
SigningCertificate ::= SEQUENCE {
    certs SEQUENCE OF ESSCertID,
    policies SEQUENCE OF PolicyInformation OPTIONAL
}
```

```
ESSCertID ::= SEQUENCE {
    certHash Hash,
    issuerSerial IssuerSerial OPTIONAL
}
```

```
Hash ::= OCTET STRING
```


La valeur du champ `certHash` est l'empreinte SHA-1 du codage DER du certificat considéré. Calculer cette empreinte.

```
$ openssl x509 -in tsa-crt.pem -outform DER | openssl sha1
(stdin)= 4328a6e719c924d2c414c24ae71950939561c509
```

Compléter le fichier de configuration `tst-signedAttrs.asn.cnf` avec les sections suivantes :

```
[tst_attr_signingCertificate_values]
signingCertificate = SEQUENCE:tst_signingCertificate_certs

[tst_signingCertificate_certs]
certs = SEQUENCE:tst_signingCertificate_essCertIDs

[tst_signingCertificate_essCertIDs]
essCertID = SEQUENCE:tst_signingCertificate_essCertID

[tst_signingCertificate_essCertID]
certHash = FORMAT:HEX,OCTETSTRING:4328a6e719c924d2c414c24ae71950939561c509
```

Générer à présent le codage DER des attributs signés.

```
$ openssl asn1parse -genconf tst-signedAttrs.asn.cnf -i -out tst-signedAttrs.der
 0:d=0  hl=3 l= 140 cons: SET
 3:d=1  hl=2 l=  26 cons: SEQUENCE
 5:d=2  hl=2 l=   9 prim: OBJECT           :contentType
16:d=2  hl=2 l=  13 cons: SET
18:d=3  hl=2 l=  11 prim: OBJECT           :id-smime-ct-TSTInfo
31:d=1  hl=2 l=  28 cons: SEQUENCE
33:d=2  hl=2 l=   9 prim: OBJECT           :signingTime
44:d=2  hl=2 l=  15 cons: SET
46:d=3  hl=2 l=  13 prim: UTCTIME          :120616211309Z
61:d=1  hl=2 l=  35 cons: SEQUENCE
63:d=2  hl=2 l=   9 prim: OBJECT           :messageDigest
74:d=2  hl=2 l=  22 cons: SET
76:d=3  hl=2 l=  20 prim: OCTET STRING      [HEX DUMP]:0F6D6F384C1CBA1E979
21FFF3D192920AF7FEAE8
 98:d=1  hl=2 l=  43 cons: SEQUENCE
100:d=2  hl=2 l=  11 prim: OBJECT           :id-smime-aa-signingCertificate

113:d=2  hl=2 l=  28 cons: SET
115:d=3  hl=2 l=  26 cons: SEQUENCE
117:d=4  hl=2 l=  24 cons: SEQUENCE
119:d=5  hl=2 l=  22 cons: SEQUENCE
121:d=6  hl=2 l=  20 prim: OCTET STRING      [HEX DUMP]:4328A6E719C924D2
C414C24AE71950939561C509
```

En préparation de la signature des attributs signés, calculer l'empreinte SHA-1 du fichier DER obtenu.

```
$ openssl sha1 tst-signedAttrs.der
SHA1(tst-signedAttrs.der)= efa58d98e76914770500fb223052bb0b5b94915b
```

11. Horodatage

Créer le fichier `tst-DigestInfo.asn.cnf` suivant, représentant la structure `DigestInfo` associée à cette empreinte :

```
asn1 = SEQUENCE:digestInfo

[digestInfo]
digestAlgorithm = SEQUENCE:digestAlgorithm
digest = FORMAT:HEX,OCTETSTRING:\
efa58d98e76914770500fb223052bb0b5b94915b

[digestAlgorithm]
algorithm = OID:sha1
parameters = NULL
```

Générer le codage DER de cette structure `DigestInfo` :

```
$ openssl asn1parse -genconf tst-DigestInfo.asn.cnf -i -out tst-DigestInfo.der
 0:d=0  hl=2 l= 33 cons: SEQUENCE
 2:d=1  hl=2 l=  9 cons:  SEQUENCE
 4:d=2  hl=2 l=  5 prim:  OBJECT                  :sha1
11:d=2  hl=2 l=  0 prim:  NULL
13:d=1  hl=2 l= 20 prim:  OCTET STRING            [HEX DUMP]:EFA58D98E76914770500F
B223052BB0B5B94915B
```

Signer ce fichier à l'aide de la clé privée de l'unité d'horodatage, et produire la représentation hexadécimale de cette signature.

```
$ openssl pkeyutl -sign -in tst-DigestInfo.der -inkey tsa-key.pem \
| od -tx1 -An -w | tr -d " " | sed 's/$/\\/'
222f835b63d35010e14941c4700ca3f1c07808051760e33820c9538e63a327f2\
19a1683e57ca75d2e79f67e1a730ca1ffc2c257c6a939405abdfde4e4aa24961\
9057a161e9ddf6ebc8f4f503c2f401fe47b8e21fcf70e96316646a026602bd74\
998905b6b9cf9d3a5cc00fa91ce90d30d90dc8820695b4782b7a0888cc06a948\
1707de349c1dfb2518e5dce1a2613e70627ee723dfa98fabe03e0cf64a812018\
5eda49ad4251cdccc3aa9d9beb31a79f13987562c02f4de179a2b967a9db0a0d\
6502d0baf2f4f9a1fb118244f47e0371453181f1ba5994f9bfac4ee2970a0e2f\
6d758a07b92bad329538dfc77ded7e594276f5410594c167c4b3de928c53b1b3\
```

Consolidation de la structure `TimeStampToken`

Créer le fichier `tst-TimeStampToken.asn.cnf`, avec le contenu suivant.

```
asn1 = SEQUENCE:tst_timeStampToken

[tst_timeStampToken]
contentType = OID:pkcs7-signedData
content = EXPLICIT:0,SEQUENCE:tst_content

[tst_content]
version = INTEGER:3
digestAlgorithms = SET:tst_digestAlgorithms
```

```
encapContentInfo = SEQUENCE:tst_encapContentInfo
signerInfos = SET:tst_signerInfos
```

```
[tst_digestAlgorithms]
digestAlgorithm = SEQUENCE:tst_digestAlgorithm
```

```
[tst_digestAlgorithm]
algorithm = OID:sha1
parameters = NULL
```

```
[tst_encapContentInfo]
eContentType = OID:id-smime-ct-TSTInfo
eContent = EXPLICIT:0,OCTWRAP,SEQUENCE:tstInfo
```

Ajouter ensuite les sections du fichier `tst-TSTInfo.asn.cnf`.

Poursuivre avec les sections suivantes :

```
[tst_signerInfos]
signerInfo = SEQUENCE:tst_signerInfo
```

```
[tst_signerInfo]
version = INTEGER:1
sid = SEQUENCE:tst_sid
digestAlgorithm = SEQUENCE:tst_signerInfo_digestAlgorithm
signedAttrs = IMPLICIT:0,SET:tst_signedAttrs
signatureAlgorithm = SEQUENCE:tst_signerInfo_signatureAlgorithm
signature = FORMAT:HEX,OCTETSTRING:\
222f835b63d35010e14941c4700ca3f1c07808051760e33820c9538e63a327f2\
19a1683e57ca75d2e79f67e1a730ca1ffc2c257c6a939405abdfde4e4aa24961\
9057a161e9ddf6ebc8f4f503c2f401fe47b8e21fcf70e96316646a026602bd74\
998905b6b9cf9d3a5cc00fa91ce90d30d90dc8820695b4782b7a0888cc06a948\
1707de349c1dfb2518e5dce1a2613e70627ee723dfa98fabe03e0cf64a812018\
5eda49ad4251cdccc3aa9d9beb31a79f13987562c02f4de179a2b967a9db0a0d\
6502d0baf2f4f9a1fb118244f47e0371453181f1ba5994f9bfac4ee2970a0e2f\
6d758a07b92bad329538dfc77ded7e594276f5410594c167c4b3de928c53b1b3
```

```
[tst_sid]
issuer = SEQUENCE:tst_signerInfo_issuer
serial = INTEGER:0xd3b94f04ddd1a040
```

```
[tst_signerInfo_issuer]
issuer_C_RDN = SET:issuer_C_RDN
issuer_O_RDN = SET:issuer_O_RDN
issuer_OU1_RDN = SET:issuer_OU1_RDN
issuer_OU2_RDN = SET:issuer_OU2_RDN
```

```
[tst_signerInfo_digestAlgorithm]
algorithm = OID:sha1
parameters = NULL
```

```
[tst_signerInfo_signatureAlgorithm]
```

11. Horodatage

```
algorithm = OID:rsaEncryption
parameters = NULL
```

Ci-dessus, le champ `signature` contient la représentation hexadécimale de la signature des attributs signés, telle qu'obtenue précédemment. Le champ `serial` contient le numéro de série du certificat de l'unité d'horodatage (qui peut être extrait à l'aide de la commande `openssl x509 -in tsa-crt.pem -noout -serial`).

Les sections ci-dessous sont reprises des fichiers de configuration des chapitres précédents.

```
[issuer_C_RDN]
issuer_C_ATV = SEQUENCE:issuer_C_ATV
```

```
[issuer_C_ATV]
type = OID:countryName
value = PRINTABLESTRING:FR
```

```
[issuer_O_RDN]
issuer_O_ATV = SEQUENCE:issuer_O_ATV
```

```
[issuer_O_ATV]
type = OID:organizationName
value = PRINTABLESTRING:Mon Entreprise
```

```
[issuer_OU1_RDN]
issuer_OU1_ATV = SEQUENCE:issuer_OU1_ATV
```

```
[issuer_OU1_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:0002 123456789
```

```
[issuer_OU2_RDN]
issuer_OU2_ATV = SEQUENCE:issuer_OU2_ATV
```

```
[issuer_OU2_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:OpenSSL Root CA
```

Ajouter enfin les sections du fichier `tst-signedAttrs.asn.cnf`.

Générer le jeton d'horodatage.

```
$ openssl asn1parse -genconf tst-TimeStampToken.asn.cnf -i \
  -out tst-TimeStampToken.der
```

Vérifier que le fichier généré est un jeton d'horodatage valide pour le fichier de données `data.txt`.

```
$ openssl ts -verify -data data.txt -in tst-TimeStampToken.der \
  -CAfile ca-crt.pem -untrusted tsa-crt.pem
Verification: OK
```

Si les valeurs n'ont pas été modifiées par rapport au jeton d'horodatage de référence `data.tst`, vérifier, par exemple en comparant les empreintes, que le jeton d'horodatage initial et celui généré sont identiques.

Chapitre 12 — Signature électronique avancée — CAdES

La série des normes AdES (*Advanced Electronic Signature* ou signature électronique avancée) — XAdES, CAdES et PAdES — a été définie par l'ETSI pour permettre la production de signatures électroniques avancées, au sens défini par la directive 1999/93/EC du Parlement européen et du conseil de l'Union européenne⁶⁵, dont l'objectif est de favoriser et d'encourager la reconnaissance légale de la signature électronique en Europe, en particulier en permettant de prolonger la validité des signatures électroniques au-delà de leur durée de vie « naturelle » (par défaut, tant que le certificat du signataire est valide).

PAdES

Héritant à la fois de CAdES et de PDF⁶⁶, la norme PAdES⁶⁷ (*PDF Advanced Electronic Signature*) n'a pas été abordée dans ce document, la constitution de ces signatures n'étant pas possible à l'aide exclusive d'outils en ligne de commande librement disponibles.

Depuis la version 9, Adobe Reader est en mesure de produire des signature PAdES⁶⁸ sur des documents possédant le droit Reader étendu (*extended Reader right*) de signature, lequel peut être accordé⁶⁹ par Adobe Acrobat Professional ou Adobe LiveCycle Reader Extensions.

Sans ces logiciels, la piste recommandée est d'utiliser la bibliothèque iText⁷⁰, nominalement en Java, et portée en C# sous le nom d'iTextSharp : la documentation de référence d'iText est le livre *iText in Action*, dont la section 12.4 de la deuxième édition concerne la signature électronique de fichiers PDF et évoque brièvement PAdES. Le lecteur intéressé peut se reporter au code source Java associé⁷¹ (des exemples de signature avec inclusion de jetons d'horodatage et de jetons OCSF sont proposés), ainsi qu'aux exemples de code source Java et C# proposés par Paulo Soares⁷².

La norme CAdES (spécifiée dans le document ETSI portant la référence TS 101 733) définit un ensemble de propriétés, dites qualifiantes, qui peuvent être incluses dans une signature électronique CMS, et dont certaines sont signées avec les données.

65. <http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2000:013:0012:0020:FR:PDF>

66. http://www.adobe.com/devnet/pdf/pdf_reference.html

67. <http://www.padesfaq.net/> (non officiel)

68. <http://blogs.adobe.com/security/ConfigurationofAdobeAcrobat9forPAdES.pdf>

69. <http://blog.tallcomponents.com/2011/02/reader-extensions-under-hood.html>

70. <http://itextpdf.com>

71. <http://itextpdf.com/book/chapter.php?id=12>

72. <http://itextpdf.sourceforge.net/howtosign.html>

Plusieurs versions de CAdES coexistent : la version 1.7.4 est la plus facilement récupérable, ayant été reprise par l'IETF dans la RFC 5126 (elle-même une mise à jour de la RFC 3126 qui correspond à la version 1.2.2 de CAdES), la version 1.8.1 est identifiée comme étant la version en cours sur le portail de la signature électronique de l'ETSI ⁷³, la version 1.8.4 est la dernière version de la branche 1.x de la norme, et la version 2.1.1 de mars 2012 est la dernière version à la date de rédaction. Ci-après, la version 2.1.1 est utilisée (récupérer la spécification à l'aide d'une recherche sur le mot clé « CAdES » dans le moteur de recherche proposé par l'ETSI ⁷⁴).

La version 2.1.1 de la norme CAdES définit plusieurs formats de signature électronique :

- CAdES-BES (*Basic Electronic Signature*, ou signature électronique de base) inclut le certificat du signataire dans les attributs signés de CMS, et permet d'intégrer une contresignature en tant que propriété non signée.
- CAdES-EPES (*Explicit Policy based Electronic Signature*, ou signature électronique avec politique explicite) inclut dans les propriétés signées l'identifiant d'une politique de signature, qui doit être utilisée pour valider la signature électronique.
- CAdES-T (T pour *time*, ou heure) intègre une heure de signature fiable dans les propriétés non signées.
- CAdES-C (C pour *complete validation data references*, ou références complètes aux données de validation) complète le format CAdES-T en référençant, au sein de propriétés non signées, les certificats (hors certificat de signature) et informations de révocation (listes de certificats révoqués et jetons OCSP) permettant de valider la signature électronique.
- CAdES-X Long (*extended long*, ou format long étendu) complète une structure CAdES-C avec les données de validation référencées.
- CAdES-X (*extended with time*, ou étendu avec heure) ajoute à une structure CAdES-C un jeton d'horodatage portant soit sur la structure CAdES-C complète (CAdES-X de type 1), soit sur les références aux données de validation uniquement (CAdES-X de type 2).
- CAdES-X Long Type 1 et CAdES-X Long Type 2 (*extended long with time*, ou format long étendu avec heure) ajoutent à une structure CAdES-C les informations introduites par les formats CAdES-X Long et CAdES-X de type 1 ou de type 2.
- CAdES-A (*archival form*, ou format d'archivage) ajoute aux propriétés non signées d'une structure CAdES-X Long (de base, de type 1 ou de type 2) ou CAdES-A un ou plusieurs jetons d'horodatage de manière à sceller la signature électronique dans la durée.
- CAdES-LT (*long-term form*, ou format long terme), introduit dans la version 2.1.1, a le même objectif que CAdES-A, mais permet d'enrichir une signature CAdES-T (ou plus) au lieu d'une signature CAdES-X Long (ou plus), et est donc bien plus simple à mettre en œuvre.

■ La conformité à la norme CAdES n'impose pas l'implémentation des formats CAdES-X et supérieurs.

La suite de ce chapitre s'intéresse à la construction d'une signature CAdES-LT. Pour cela, le fichier de configuration ASN.1 utilisé pour construire une signature CMS dans la section 8.5 est adapté et complété avec les éléments nécessaires à la constitution d'une structure CAdES-BES, puis celle-ci

73. <http://www.etsi.org/WebSite/Technologies/ElectronicSignature.aspx>

74. <http://webapp.etsi.org/WorkProgram/expert/queryform.asp>

est successivement enrichie avec les données permettant d'obtenir une signature avancée au format CAdES-T, puis CAdES-LT.

12.1. Constitution de la structure CAdES-BES

La démarche de constitution de la structure CAdES-BES est analogue à la construction d'une signature électronique PKCS#7/CMS, tel que décrit dans la section 8.5 : constituer et signer les attributs signés, puis créer la structure `SignerInfo`, et enfin consolider la structure CAdES-BES.

Constitution des attributs signés

CAdES-BES enrichit une structure CMS avec les attributs signés `contentType`, `messageDigest`, et `signingCertificate` ou `SigningCertificateV2`. Les deux premiers sont déjà inclus dans la structure CMS constituée dans la section 8.5. Les attributs `signingCertificate` et `SigningCertificateV2` permettent d'identifier le certificat du signataire à l'aide de son empreinte (ce mécanisme a été utilisé précédemment, dans la section 11.3, pour référencer le certificat de l'unité d'horodatage dans un jeton d'horodatage) : dans le cas où l'algorithme de hachage choisi est SHA-1, l'attribut `signingCertificate` doit être utilisé, sinon c'est l'attribut `SigningCertificateV2` qui doit être utilisé. Dans le cas présent, l'algorithme de hachage choisi est SHA-256, il reste simplement à ajouter à la structure CMS initiale l'attribut signé `signingCertificateV2` contenant le certificat du signataire pour obtenir une signature CAdES-BES.

L'attribut `signingCertificateV2` est défini dans la RFC 5035. Sa structure est la suivante :

```
SigningCertificateV2 ::= SEQUENCE {
    certs SEQUENCE OF
        SEQUENCE {
            hashAlgorithm AlgorithmIdentifier
                DEFAULT {algorithm id-sha256},
            certHash OCTET STRING,
            issuerSerial SEQUENCE {
                issuer GeneralNames,
                serialNumber CertificateSerialNumber
            } OPTIONAL
        },
    policies SEQUENCE OF PolicyInformation OPTIONAL
}
```

Il est choisi d'omettre les éléments optionnels ainsi que le champ `hashAlgorithm` (l'algorithme par défaut, SHA-256, est utilisé). La structure à générer pour l'attribut signé `SigningCertificateV2` est donc la suivante :

```
SigningCertificateV2 ::= SEQUENCE {
    certs SEQUENCE OF
        SEQUENCE {
            certHash OCTET STRING,
```

```
    },
}
```

La valeur du champ `certHash` est l'empreinte SHA-256 du codage DER du certificat du signataire. Calculer cette empreinte.

```
$ openssl sha256 ee-crt-authsig.pem
SHA256(ee-crt-authsig.pem)= d2df6647707afb2911a96e3d619aeebe79e2141db177fcd127f3
295075e3e39d
```

Copier le fichier `sig-p7.explicit-signedAttrs.asn.cnf` créé dans la section 8.4 sous le nom `data.cades-bes.signedAttrs.asn.cnf`, et ajouter à ce dernier les sections suivantes, en reportant la valeur obtenue ci-dessus dans le champ `certHash` de la section `signingCertificateV2_ESSCertIDv2` :

```
[attr_signingCertificateV2]
attrType = OID:SigningCertificateV2
attrValues = SET:attr_signingCertificateV2_values

[attr_signingCertificateV2_values]
attr_signingCertificateV2_value = SEQUENCE:signingCertificateV2

[signingCertificateV2]
certs = SEQUENCE:signingCertificateV2_ESSCertIDv2s

[signingCertificateV2_ESSCertIDv2s]
signingCertificateV2_ESSCertIDv2 = SEQUENCE:signingCertificateV2_ESSCertIDv2

[signingCertificateV2_ESSCertIDv2]
certHash = FORMAT:HEX,OCTETSTRING:\
d2df6647707afb2911a96e3d619aeebe79e2141db177fcd127f3295075e3e39d
```

Ajouter ensuite la ligne en gras ci-dessous dans la section `signedAttrs` :

```
[signedAttrs]
attr_contentType = SEQUENCE:attr_contentType
attr_signingTime = SEQUENCE:attr_signingTime
attr_messageDigest = SEQUENCE:attr_messageDigest
attr_signingCertificateV2 = SEQUENCE:attr_signingCertificateV2
```

L'attribut contenant la date et heure de signature (champ `attr_signingTime`) pourrait être supprimé, n'étant pas requis par CAdES-BES.

L'OID correspondant à l'attribut `SigningCertificateV2` n'est pas connu d'OpenSSL. Créer le fichier `cades-oid.tsv` suivant pour le définir :

```
1.2.840.113549.1.9.16.2.47 id-aa-signingCertificateV2 SigningCertificateV2
```

La commande `openssl asn1parse -genconf` semble imposer un retour chariot simple de type UNIX (`\n`) ou Mac (`\r`) en fin de ligne, le retour chariot double de Windows (`\n\r`) provoquant une erreur. Sous Windows, si l'éditeur de texte ne permet pas de convertir les caractères de fin de ligne, alors utiliser la commande `tr -d "\r"` pour remplacer les retours chariot par des `\n` simples.

Générer la représentation DER de la structure SET of Attributes, en n'oubliant pas de référencer le fichier d'OID supplémentaires créé ci-dessus :

```
$ openssl asn1parse -genconf data.cades-bes.signedAttrs.asn.cnf -i \
-oid cades-oid.tsv -out data.cades-bes.signedAttrs.der
  0:d=0  hl=3 l= 162 cons: SET
  3:d=1  hl=2 l=  24 cons: SEQUENCE
  5:d=2  hl=2 l=   9 prim: OBJECT                :contentType
 16:d=2  hl=2 l=  11 cons: SET
 18:d=3  hl=2 l=   9 prim: OBJECT                :pkcs7-data
 29:d=1  hl=2 l=  28 cons: SEQUENCE
 31:d=2  hl=2 l=   9 prim: OBJECT                :signingTime
 42:d=2  hl=2 l=  15 cons: SET
 44:d=3  hl=2 l=  13 prim: UTCTIME                :120427191932Z
 59:d=1  hl=2 l=  47 cons: SEQUENCE
 61:d=2  hl=2 l=   9 prim: OBJECT                :messageDigest
 72:d=2  hl=2 l=  34 cons: SET
 74:d=3  hl=2 l=  32 prim: OCTET STRING          [HEX DUMP]:89BD92286D6C8014C06
030B25F8B40CC1D5656D4B3B7B4831874F50D6F5557F3
 108:d=1  hl=2 l=  55 cons: SEQUENCE
 110:d=2  hl=2 l=  11 prim: OBJECT                :SigningCertificateV2
 123:d=2  hl=2 l=  40 cons: SET
 125:d=3  hl=2 l=  38 cons: SEQUENCE
 127:d=4  hl=2 l=  36 cons: SEQUENCE
 129:d=5  hl=2 l=  34 cons: SEQUENCE
 131:d=6  hl=2 l=  32 prim: OCTET STRING          [HEX DUMP]:D2DF6647707AFB29
11A96E3D619AEEBE79E2141DB177FCD127F3295075E3E39D
```

Signature des attributs signés

Le fichier généré doit désormais être signé pour être inclus dans le champ signature de la structure SignerInfo.

Calculer l'empreinte SHA-256 de ce fichier.

```
$ openssl sha256 data.cades-bes.signedAttrs.der
SHA256(data.cades-bes.signedAttrs.der)= 91c0d3903a337db5bc75644952d82845cf2c8766
73d5aafd81e0ab2c7163d04b
```

Encapsuler cette valeur dans une structure DigestInfo, en créant tout d'abord le fichier data.cades-bes.DigestInfo.asn.cnf suivant :

```
asn1 = SEQUENCE:digestInfo

[digestInfo]
digestAlgorithm = SEQUENCE:digestAlgorithm
digest = FORMAT:HEX,OCTETSTRING:\
91c0d3903a337db5bc75644952d82845cf2c876673d5aafd81e0ab2c7163d04b

[digestAlgorithm]
```

```
algorithm = OID:sha256
parameters = NULL
```

Générer ensuite le codage DER correspondant.

```
$ openssl asn1parse -genconf data.cades-bes.DigestInfo.asn.cnf -i \
-out data.cades-bes.DigestInfo.der
 0:d=0  hl=2 l= 49 cons: SEQUENCE
 2:d=1  hl=2 l= 13 cons: SEQUENCE
 4:d=2  hl=2 l=  9 prim: OBJECT                    :sha256
15:d=2  hl=2 l=  0 prim: NULL
17:d=1  hl=2 l= 32 prim: OCTET STRING      [HEX DUMP]:91COD3903A337DB5BC756
44952D82845CF2C876673D5AAFD81E0AB2C7163D04B
```

Signer le fichier généré à l'aide de la clé publique du signataire.

```
$ openssl pkeyutl -sign -in data.cades-bes.DigestInfo.der -inkey ee-key.pem \
-out data.cades-bes.signature.bin
```

Produire enfin la représentation hexadécimale de cette signature :

```
$ od -tx1 -An -w data.cades-bes.signature.bin | tr -d "\"" | sed 's/$/\\/'
a28aff00ebade94eb7e3e3e29909b23ad4e85a53f216c1a7bfd48145e7c7a279\
8327eda577f831ac90c29ed12d4729afd7846a715f7ce5343139563bd391fce4\
8511523d00f077cc1cd1921673f6f2dd273bf256531c860eb021fb18e1fc8347\
9688a5bc934015fd6cbb9a5dd4c872d87c0fb0a946462e3c59aef16c7b1f2a3a\
b9d388a9cb84ce37edf3ffb7c4c189e9d47ded1d89c82c577928afaa283b9709\
48a9ab4ac5cdcb4a846947746e252c8d7262416a1286c6a44ec9d48ded51cf14\
7e74719191b1f5ce2e21af4a8a2af938d98767ee1dde29b04798c8e0d1c6d07d\
b8d2b960e0e443da61248b4189fd62de224866dbfa02d2b7f9ca1cb4d9e5e7b6\
```

Finalisation de la structure CAdES-BES

Copier le fichier sig-p7.asn.cnf généré dans la section 8.5 sous le nom data.cades-bes.asn.cnf. Dans ce nouveau fichier, remplacer la section signedAttrs et ses dépendances par les sections du fichier data.cades-bes.signedAttrs.asn.cnf, et remplacer la valeur du champ signature de la section signerInfo par la signature des attributs signés obtenue ci-dessus :

```
[signerInfo]
version = INTEGER:1
sid = SEQUENCE:sid
digestAlgorithm = SEQUENCE:signerInfo_digestAlgorithm
signedAttrs = IMPLICIT:0,SET:signedAttrs
signatureAlgorithm = SEQUENCE:signatureAlgorithm
signature = FORMAT:HEX,OCTETSTRING:\
a28aff00ebade94eb7e3e3e29909b23ad4e85a53f216c1a7bfd48145e7c7a279\
...
b8d2b960e0e443da61248b4189fd62de224866dbfa02d2b7f9ca1cb4d9e5e7b6
```

Générer le codage DER de la signature CAdES-BES.

```
$ openssl asn1parse -genconf data.cades-bes.asn.cnf -oid cades-oid.tsv -i \
  -out data.cades-bes.der
```

Vérifier que le fichier généré est une signature électronique CMS valide.

```
$ openssl cms -verify -inform DER -in data.cades-bes.der -content data.txt \
  -CAfile ca-crt.pem
texte en clairVerification successful
```

12.2. Constitution de la structure CAdES-T

Pour produire la structure CAdES-T, la structure CAdES-BES est enrichie d'un jeton d'horodatage portant sur la valeur de la signature.

Générer une requête d'horodatage portant sur la valeur binaire de la signature :

```
$ openssl ts -query -data data.cades-bes.signature.bin -sha256 \
  -out signatureTimeStamp.tsq
```

Générer la réponse à cette requête.

```
$ openssl ts -reply -config tsa-ts.cnf -section tsa \
  -queryfile signatureTimeStamp.tsq -inkey tsa-key.pem -signer tsa-crt.pem \
  -out signatureTimeStamp.tsr
```

Using configuration from tsa-ts.cnf

Response has been generated.

Il est également possible de solliciter le serveur d'horodatage proposé dans la section 11.2 pour générer ce jeton.

```
$ openssl ts -reply -in signatureTimeStamp.tsr -token_out \
  -out signatureTimeStamp.tst
```

En utilisant le processus exposé notamment dans la section 8.5 et/ou en utilisant la méthode proposée en annexe B.2, générer la description du jeton d'horodatage au format consommable par la commande `openssl asn1parse -genconf`. Pour le jeton généré ci-dessus, le résultat obtenu est le suivant :

```
[signatureTST]
contentType = OID:pkcs7-signedData
content = EXPLICIT:0,SEQUENCE:signatureTST_content

[signatureTST_content]
version = INTEGER:3
digestAlgorithms = SET:signatureTST_digestAlgorithms
encapContentInfo = SEQUENCE:signatureTST_encapContentInfo
signerInfos = SET:signatureTST_signerInfos

[signatureTST_digestAlgorithms]
digestAlgorithm = SEQUENCE:signatureTST_digestAlgorithm
```

```
[signatureTST_digestAlgorithm]
algorithm = OID:sha1
parameters = NULL

[signatureTST_encapContentInfo]
eContentType = OID:id-smime-ct-TSTInfo
eContent = EXPLICIT:0,OCTWRAP,SEQUENCE:signatureTST_TSTInfo

[signatureTST_TSTInfo]
version = INTEGER:1
policy = OID:\
1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.5
messageImprint = SEQUENCE:signatureTST_messageImprint
serialNumber = INTEGER:11
genTime = GENERALIZEDTIME:20120626201609Z
nonce = INTEGER:0x00e1e79be83f82b8af

[signatureTST_messageImprint]
hashAlgorithm = SEQUENCE:signatureTST_TSTInfo_hashAlgorithm
hashedMessage = FORMAT:HEX,OCTETSTRING:\
3da761db17a08fdb46d95cdcecf234c461dac887b4bf0a2d65d8c288621bc235

[signatureTST_TSTInfo_hashAlgorithm]
algorithm = OID:sha256
parameters = NULL

[signatureTST_signerInfos]
signerInfo = SEQUENCE:signatureTST_signerInfo

[signatureTST_signerInfo]
version = INTEGER:1
sid = SEQUENCE:signatureTST_sid
digestAlgorithm = SEQUENCE:signatureTST_digestAlgorithm
signedAttrs = IMPLICIT:0,SEQUENCE:signatureTST_signedAttrs
signatureAlgorithm = SEQUENCE:signatureTST_signatureAlgorithm
signature = FORMAT:HEX,OCTETSTRING:\
1fa03c62cc5648513e8972b049841b32ea753b3763643edb1b2bf196a4b91b7d\
b91c6a3bfc154e866bda19ab3da4d178dd8c454ded0fb808a5a23f2d80b6d81d\
c04afc47bb6eff2edb000a7b195146a7d6c9b97ba1e6cf9ac472ec79ca7c387f\
939de7dbfdd5011cc4f94dc207fb346f0ff4c039941a88f930c4ccc30989e9ba\
08309699e5ac72270e3dab04683b54965da4c349106b1153fe1c64e393ded157\
cdfc6aea6eebd2ffe3ef4dbf0c5d3ea6bf0f3ed2066b4cc23df95ba55921fa66\
87ed16ffe87bfa0c58397cc9bed01f47905af2a2db48395004205edbf8f06d3\
6ceb384859c85cdee276672d25137c396def80cbd5b2317fc63aef3d8b47c659

[signatureTST_sid]
issuer = SEQUENCE:tsa_issuer
serial = INTEGER:0xd3b94f04ddd1a040

[tsa_issuer]
```

```

C = SET:tsa_issuer_C_RDN
O = SET:tsa_issuer_O_RDN
OU1 = SET:tsa_issuer_OU1_RDN
OU2 = SET:tsa_issuer_OU2_RDN

[tsa_issuer_C_RDN]
rdn = SEQUENCE:tsa_issuer_C_ATV

[tsa_issuer_C_ATV]
type = OID:countryName
value = PRINTABLESTRING:FR

[tsa_issuer_O_RDN]
rdn = SEQUENCE:tsa_issuer_O_ATV

[tsa_issuer_O_ATV]
type = OID:organizationName
value = PRINTABLESTRING:Mon Entreprise

[tsa_issuer_OU1_RDN]
rdn = SEQUENCE:tsa_issuer_OU1_ATV

[tsa_issuer_OU1_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:0002 123456789

[tsa_issuer_OU2_RDN]
rdn = SEQUENCE:tsa_issuer_OU2_ATV

[tsa_issuer_OU2_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:OpenSSL Root CA

[signatureTST_digestAlgorithm]
algorithm = OID:sha1
parameters = NULL

[signatureTST_signedAttrs]
attr_contentType = SEQUENCE:signatureTST_attr_contentType
attr_signingTime = SEQUENCE:signatureTST_attr_signingTime
attr_messageDigest = SEQUENCE:signatureTST_attr_messageDigest
attr_signingCertificate = SEQUENCE:signatureTST_attr_signingCertificate

[signatureTST_attr_contentType]
attrType = OID:contentType
attrValues = SET:signatureTST_attr_contentType_values

[signatureTST_attr_contentType_values]
contentType = OID:id-smime-ct-TSTInfo

[signatureTST_attr_signingTime]

```

```
attrType = OID:signingTime
attrValues = SET:signatureTST_attr_signingTime_values

[signatureTST_attr_signingTime_values]
attr_signingTime_value = UTCTIME:120626201609Z

[signatureTST_attr_messageDigest]
attrType = OID:messageDigest
attrValues = SET:signatureTST_attr_messageDigest_values

[signatureTST_attr_messageDigest_values]
messageDigest = FORMAT:HEX,OCTETSTRING:\
18fe06c583bd2a1226648ee7b09d25902d8c089c

[signatureTST_attr_signingCertificate]
attrType = OID:id-smime-aa-signingCertificate
attrValues = SET:signatureTST_attr_signingCertificate_values

[signatureTST_attr_signingCertificate_values]
signingCertificate = SEQUENCE:signatureTST_signingCertificate_certs

[signatureTST_signingCertificate_certs]
certs = SEQUENCE:signatureTST_signingCertificate_essCertIDs

[signatureTST_signingCertificate_essCertIDs]
essCertID = SEQUENCE:signatureTST_signingCertificate_essCertID

[signatureTST_signingCertificate_essCertID]
certHash = FORMAT:HEX,OCTETSTRING:4328a6e719c924d2c414c24ae71950939561c509

[signatureTST_signatureAlgorithm]
algorithm = OID:rsaEncryption
parameters = NULL
```

Les sections `tsa_issuer*` sont identiques aux sections `ee_issuer*` du fichier `data.cades-bes.asn.cnf`, puisque l'autorité de certification émettrice du certificat du signataire est également émettrice du certificat de l'unité d'horodatage.

Copier le fichier `data.cades-bes.asn.cnf` sous le nom `data.cades-t.asn.cnf`, et y ajouter les sections ci-dessus.

Ajouter ensuite le champ `unsignedAttrs` dans la section `signerInfo`, ainsi que les sections correspondant aux attributs non signés, comme ci-dessous :

```
[signerInfo]
...
unsignedAttrs = IMPLICIT:1,SET:unsignedAttrs

[unsignedAttrs]
attr_signatureTimeStamp = SEQUENCE:attr_signatureTimeStamp

[attr_signatureTimeStamp]
```



```
attrType = OID:SignatureTimeStampToken
attrValues = SET:attr_signatureTimeStamp_values
```

```
[attr_signatureTimeStamp_values]
attr_contentType_value = SEQUENCE:signatureTST
```

Enfin, ajouter la ligne suivante dans le fichier `ca-des-oid.tsv` pour définir l'OID correspondant à l'attribut `SignatureTimeStampToken` :

```
1.2.840.113549.1.9.16.2.14 id-aa-signatureTimeStampToken SignatureTimeStampToken
```

Générer la représentation DER de la structure CAdES-T :

```
$ openssl asn1parse -genconf data.cades-t.asn.cnf -i -oid cades-oid.tsv \
  -out data.cades-t.der
```

Vérifier que la signature reste valide.

```
$ openssl smime -verify -inform DER -in data.cades-t.der -content data.txt \
  -CAfile ca-crt.pem
texte en clairVerification successful
```

12.3. Constitution de la structure CAdES-LT

Attention, la structure produite n'a pas été validée par un outil tiers, l'auteur n'ayant connaissance d'aucune application gratuite permettant de valider totalement une structure CAdES-LT en version 2.1.1. Si le lecteur souhaite produire commercialement des signatures électroniques avancées aux formats CAdES, alors il lui est recommandé de s'intéresser aux *Plugtests*⁷⁵, événements organisés par l'ETSI pour permettre aux fournisseurs de produits implémentant certaines normes de l'ETSI, dont CAdES, d'effectuer des tests d'interopérabilité et ainsi de déterminer le niveau de conformité de leurs produits par rapport à ces normes.

Le format CAdES-LT enrichit une structure de type CAdES-T (ou plus) d'un attribut non signé, `LongTermValidation`, contenant un jeton d'horodatage ou un enregistrement de preuve (ou *evidence record*) ERS (*Evidence Record Syntax*, tel que défini dans la RFC 4998) portant sur la plupart des informations contenues dans la signature (y compris des attributs non signés), et d'éventuelles données de validation (certificats, listes de révocation etc.) permettant à la signature de contenir les éléments nécessaires à sa propre validation.

L'objectif des enregistrements de preuve ERS est de maintenir la valeur probante de données arbitraires sur le long terme par ré-horodatages successifs, avant que les algorithmes cryptographiques ne soient cassés et que les certificats associés aux données ou aux enregistrements de preuve ne deviennent invalides. Le principe mis en œuvre est semblable à celui utilisé pour ré-horodater des structures *AdES-A, et le mécanisme de constitution s'apparente à celui mis en jeu pour construire un attribut `LongTermValidation` de CAdES-LT. Un enregistrement de preuve ERS est un objet ASN.1, `EvidenceRecord`, qu'il serait aisé de constituer et d'inclure dans la signature CAdES-LT, mais en pratique l'implémentation et l'utilisation de cette structure sont marginales, donc cette option est écartée au profit d'un jeton d'horodatage classique.

75. <http://www.etsi.org/Website/OurServices/Plugtests/home.aspx>

La structure de l'attribut LongTermValidation est la suivante.

```
LongTermValidation ::= SEQUENCE {
    poeDate GeneralizedTime,
    poeValue CHOICE {
        timeStamp [0] EXPLICIT TimeStampToken,
        evidenceRecord [1] EXPLICIT EvidenceRecord
    } OPTIONAL,
    extraCertificates [0] IMPLICIT CertificateSet OPTIONAL,
    extraRevocation [1] IMPLICIT RevocationInfoChoices OPTIONAL
}
```

Le champ poeDate (poe pour *proof of existence* ou preuve d'existence) contient la date et l'heure (de préférence identique, à défaut le plus proche possible) du jeton d'horodatage (ou enregistrement de preuve) inclus dans le champ poeValue. Le champ extraCertificates contiendra le certificat de l'autorité de certification racine, qui a émis le certificat de l'unité d'horodatage et du signataire, ainsi que le certificat de l'unité d'horodatage. Le champ extraRevocation contiendra la dernière liste de certificats révoqués émise par l'autorité de certification racine avant la constitution de l'attribut.

La norme CAdES 2.1.1 introduit, dans le cadre du format CAdES-LT, la notion d'*empreinte-arbre* (ou *tree-hash*), qui permet d'obtenir une empreinte unique à partir d'éléments dont l'ordre est arbitraire, à l'exemple de certificats ou de listes de certificats révoqués. Cette empreinte-arbre s'obtient en calculant les empreintes des éléments, en concaténant ces empreintes par ordre croissant, et en calculant l'empreinte de la donnée binaire ainsi générée.

Plus formellement, le classement des empreintes par ordre croissant est le classement par ordre lexicographique de leur représentation binaire, où par exemple $000 < 001 < 100 < 111$.

Les données à horodater sont la concaténation des empreintes suivantes :

- L'empreinte du type du contenu signé (champ eContentType).
- L'empreinte des données faisant l'objet de la signature (ici, le fichier data.txt ; le champ eContent quand la signature est enveloppante).
- L'empreinte-arbre des certificats contenus dans la signature, c'est-à-dire les éléments CertificateChoices du champ certificates de la structure de plus haut niveau, et de ceux du champ extraCertificates de l'attribut LongTermValidation en cours de constitution.
- L'empreinte-arbre des données de révocation contenues dans la signature dans le champ crls, ainsi que de celles contenues dans le champ extraRevocation de l'attribut LongTermValidation en cours de constitution.
- L'empreinte de la concaténation des champs version, sid et digestAlgorithm de la structure SignerInfo.
- L'empreinte des attributs signés de la structure SignerInfo.
- L'empreinte de la concaténation des champs signatureAlgorithm et signature de la structure SignerInfo.
- L'empreinte-arbre des attributs non signés.

L'algorithme de hachage utilisé dans ce processus doit être un algorithme référencé dans le champ `digestAlgorithms` de la signature. Dans le cas présent, il s'agit de l'algorithme SHA-256.

Calcul de l'empreinte du type du contenu signé

L'empreinte du champ `eContentType` est calculée sur l'ensemble de sa représentation DER, incluant les octets de type et de longueur.

Deux méthodes de calcul sont proposées. La première s'appuie sur la structure CAdES-T générée précédemment, et consiste à extraire les octets à hacher à l'aide des options `-offset` et `-length` de la commande `openssl asn1parse`. Afficher le début de l'analyse ASN.1 de cette structure.

```
$ openssl asn1parse -in data.cades-t.der -inform DER -i | head -n15
 0:d=0  hl=4 l=2389 cons: SEQUENCE
 4:d=1  hl=2 l= 9 prim: OBJECT                  :pkcs7-signedData
15:d=1  hl=4 l=2374 cons: cont [ 0 ]
19:d=2  hl=4 l=2370 cons: SEQUENCE
23:d=3  hl=2 l= 1 prim: INTEGER                  :01
26:d=3  hl=2 l= 15 cons: SET
28:d=4  hl=2 l= 13 cons: SEQUENCE
30:d=5  hl=2 l= 9 prim: OBJECT                  :sha256
41:d=5  hl=2 l= 0 prim: NULL
43:d=3  hl=2 l= 11 cons: SEQUENCE
45:d=4  hl=2 l= 9 prim: OBJECT                  :pkcs7-data
56:d=3  hl=4 l=1010 cons: cont [ 0 ]
60:d=4  hl=4 l=1006 cons: SEQUENCE
64:d=5  hl=4 l= 726 cons: SEQUENCE
68:d=6  hl=2 l= 3 cons: cont [ 0 ]
```

Le début de la syntaxe ASN.1 d'une structure CMS/CAdES est la suivante :

```
ContentInfo ::= SEQUENCE {
    contentType ContentType,
    content [0] EXPLICIT SEQUENCE {
        version CMSVersion,
        digestAlgorithms DigestAlgorithmIdentifiers,
        encapContentInfo SEQUENCE {
            eContentType ContentType,
            eContent [0] EXPLICIT OCTET STRING OPTIONAL },
        certificates [0] IMPLICIT CertificateSet OPTIONAL,
        ...
    }
}
```

En mettant en correspondance l'analyse et la syntaxe, il apparaît que le champ `eContentType` commence à l'octet 45, pour une longueur de 11 octets : 2 octets d'en-tête (`hl`) et 9 de contenu. Extraire le champ.

```
$ openssl asn1parse -in data.cades-t.der -inform DER -i -offset 45 -length 11 \
-out data.cades-lt.eContentType.bin
0:d=0 hl=2 l= 9 prim: OBJECT :pkcs7-data
```

La deuxième méthode proposée pour obtenir le champ `eContentType` consiste à générer celui-ci à partir d'un fichier de configuration ASN.1 interprétable par la commande `openssl asn1parse -genconf`, ou d'une ligne de commande simple `openssl asn1parse -genstr` dans le cas d'un type ASN.1 primitif (ce qui est le cas du champ `eContentType`, qui est de type primitif OBJECT IDENTIFIER). Le champ `eContentType` est défini ainsi dans le fichier `data.cades-t.asn.cnf` :

```
...
[encapContentInfo]
eContentType = OID:pkcs7-data
...
```

Générer le codage DER du champ `eContentType` :

```
$ openssl asn1parse -genstr OID:pkcs7-data -i -out data.cades-lt.eContentType.bin
0:d=0 hl=2 l= 9 prim: OBJECT :pkcs7-data
```

De manière équivalente, il est possible de créer le fichier `data.cades-lt.eContentType.asn.cnf` suivant :

```
asn1 = OID:pkcs7-data
```

Générer ensuite le codage DER correspondant :

```
$ openssl asn1parse -genconf data.cades-lt.eContentType.asn.cnf -i \
-out data.cades-lt.eContentType.bin
0:d=0 hl=2 l= 9 prim: OBJECT :pkcs7-data
```

Une fois le fichier `data.cades-lt.eContentType.bin` produit à l'aide de l'une de ces méthodes, en générer l'empreinte SHA-256 :

```
$ openssl sha256 -binary data.cades-lt.eContentType.bin \
> data.cades-lt.eContentType.sha256
```

Calcul de l'empreinte des données signées

La signature étant détachée (et ne contenant donc pas de champ `eContent`), générer le fichier contenant l'empreinte binaire des données signées :

```
$ openssl sha256 -binary data.txt > data.cades-lt.content.sha256
```

Calcul de l'empreinte-arbre des certificats

L'empreinte-arbre à générer ensuite porte sur les certificats contenus dans la signature et dans l'attribut LongTermValidation en cours de constitution. Il s'agit des certificats suivants : le certificat du signataire, le certificat de l'autorité de certification racine, et le certificat de l'unité d'horodatage. Calculer les empreintes de ces certificats, préalablement codés en DER.

```
$ openssl x509 -in ee-crt-authsig.pem -outform DER | openssl sha256 -binary \
> data.cades-lt.certificate.ee.sha256
```

```
$ openssl x509 -in ca-crt.pem -outform DER | openssl sha256 -binary \
> data.cades-lt.certificate.ca.sha256
```

```
$ openssl x509 -in tsa-crt.pem -outform DER | openssl sha256 -binary \
> data.cades-lt.certificate.tsa.sha256
```

Il faut à présent classer les empreintes par ordre croissant et les concaténer. La manière la plus immédiate de procéder est d'utiliser une commande de type `od -tx1 -An` sur chacun des fichiers pour obtenir les valeurs hexadécimales de chaque empreinte, les classer, puis de concaténer les fichiers d'empreintes dans l'ordre obtenu. Un script shell (pour systèmes UNIX/Linux) et un fichier batch (pour Windows) sont proposés ci-après pour automatiser la procédure.

Pour les systèmes UNIX/Linux, créer le script `genhashlist` suivant (lui attribuer le droit d'exécution via par exemple `chmod +x genhashlist`):

```
#!/bin/bash
for i in $* ; do \
    echo `od -tx1 -An "$i" | tr -d "\r\n" " `: $i ; \
done
```

Pour les systèmes Windows, créer le fichier batch `genhashlist.bat` suivant :

```
@echo off
for %%a in (%) do (
od -tx1 -An "%%a" | tr -d "\n\r "
echo.|set /P=:%%a
echo.)
```

Ces scripts fonctionnent de manière identique : ils prennent en argument une liste de fichiers, comprenant éventuellement des métacaractères (typiquement, `*`) et retournent pour chaque fichier le contenu du fichier codé en hexadécimal, suivi d'un caractère `:`, suivi du nom du fichier. À titre illustratif, sous UNIX/Linux :

```
$ ./genhashlist data.cades-lt.certificate.*.sha256
8f6bc593235515f87869afbab31199543ab779688f2b21ea19da28606c456e39:data.cades-lt.c
ertificate.ca.sha256
1797d10a26237bd5b7ddff81e1848a17d07168646b47b9a083fafa6f713e1957:data.cades-lt.c
ertificate.ee.sha256
68b22f49939e398872086e8e36c83b34a4e45ff521ae86c99d165817322c87cb:data.cades-lt.c
ertificate.tsa.sha256
```

La commande équivalente sous Windows est :

```
> genhashlist data.cades-lt.certificate.*.sha256
```

Attention, ces scripts ne gèrent pas les noms de fichiers passés en argument contenant des espaces. Par ailleurs, sous UNIX/Linux, si un argument contenant un métacaractère est susceptible de retourner un nom de fichier contenant un espace, alors elle doit être entourée de guillemets. Pour pouvoir automatiser le traitement du résultat des scripts comme décrit ci-après, les noms de fichiers ne doivent pas contenir d'espace.

Le résultat des script peut désormais être trié par ordre lexicographique croissant, en chaînant la commande ci-dessus dans la commande `sort` :

```
... | sort
1797d10a26237bd5b7ddff81e1848a17d07168646b47b9a083fafa6f713e1957:data.cades-lt.certificate.ee.sha256
68b22f49939e398872086e8e36c83b34a4e45ff521ae86c99d165817322c87cb:data.cades-lt.certificate.tsa.sha256
8f6bc593235515f87869afb31199543ab779688f2b21ea19da28606c456e39:data.cades-lt.certificate.ca.sha256
```

Une commande `cut` sur le résultat de ce tri permet d'obtenir le nom du fichier (le deuxième champ et suivants en considérant que le séparateur de champs est un espace) :

```
... | cut -d " " -f 2-
data.cades-lt.certificate.ee.sha256
data.cades-lt.certificate.tsa.sha256
data.cades-lt.certificate.ca.sha256
```

Chacun de ces fichiers peut alors être concaténé, via `xargs` (qui boucle par défaut sur chaque ligne passée en entrée, et exécute la commande passée en paramètre) et `cat` (cf. note ci-après concernant la concaténation de fichiers binaires sous Windows), puis le résultat peut être haché, produisant ainsi l'empreinte-arbre attendue :

```
... | xargs cat | openssl sha256 -binary > data.cades-lt.certificates.sha256
```

L'ensemble de ce processus peut bien entendu être automatisé pour produire une empreinte-arbre à partir des fichiers empreintes individuels, en enrobant la commande ci-dessus dans un script *ad hoc*.

Sous Windows, la commande `cat` employée dans la commande ci-dessus est supposée être celle de la suite GnuWin, qui est capable de gérer indifféremment des fichiers textes ou binaires, contrairement à la commande `CAT` native de Windows. Si pour une raison ou une autre la commande `cat` de GnuWin n'était pas disponible, il faudrait utiliser la commande `COPY /b` de manière manuelle, avec la syntaxe suivante :

```
> copy /b data.cades-lt.certificate.ee.sha256 \
+ data.cades-lt.certificate.tsa.sha256 \
+ data.cades-lt.certificate.ca.sha256 \
data.cades-lt.certificates.bin
```

Le fichier obtenu doit ensuite être haché :

```
> openssl sha256 -binary data.cades-lt.certificates.bin \
> data.cades-lt.certificates.sha256
```

Calcul de l'empreinte-arbre des listes de certificats révoqués

Émettre une nouvelle liste de certificats révoqués (LCR), à inclure en tant qu'élément du champ `extraRevocation` de l'attribut `LongTermValidation`.

```
$ openssl ca -gencrl -cert ca-crt.pem -keyfile ca-key.pem -crlhours 48 \
  -md sha256 -config ca-crl.cnf -name ca_crl -crlexts ca_crl_ext \
  -out data.cades-lt.crl.ltv.pem
```

Le lecteur souhaitant utiliser les mêmes valeurs que celles obtenues par l'auteur peut créer le fichier `data.cades-lt.crl.ltv.pem` avec le contenu suivant :

```
-----BEGIN X509 CRL-----
MIIBOzCBvAIBATANBgkqhkiG9w0BAQsFADBZMQswCQYDVQQGEwJGUjEXMBUGA1UE
ChMOTW9uIEVudHJlcHJpc2UxZzAVBgNVBAsTDjAwMDIzMjIzNDU2Nzg5MRgwFgYD
VQQLLEw9PcGVuU1NMIFJvb3QgQ0EXDTEyMDYyNzE5NDUxOFoXDTEyMDYyOTE5NDUx
OFqgLzAtMB8GA1UdIwQYMBAAFEExth50C9y0sByOiD+BxLRc/OfOPMAoGA1UdFAQD
AgEJMAOGCSqGSIb3DQEBChwUAA4IBAQA+f2f3r47B1MhJYD1L1kcfTzBFIn00QYY2
RWYFFufa/vStxhu50zW7qk9t92NRMiytz6wowU6dPu+dpm/2bxh4CEgmDN72LXKW
VBWkqQKiFJjKl+ky/pJ3Wm3lqQfqqwv+K34+ciEMG+sSAJnwzN04loKW5fylkvb6
LTaSmJg3fxJk74cbMob9uZTZkW+olyVero5fIC/CMwU3Tgrv9VELkjFca3CF61Mq
KD6yMUDk+S18rTvbsma1zgotiLflx7Nm1dG24WsDNdfnFZqst9mZUtYodXl19oZx
F+E0stj7yuESB0trMhmqDOGQBDLVExyd0oActCrNdxBsJC3aCN
-----END X509 CRL-----
```

Convertir le codage en DER.

```
$ openssl crl -in data.cades-lt.crl.ltv.pem -outform DER \
  -out data.cades-lt.crl.ltv.der
```

Une seule LCR figurant dans la structure `CAdES-LT`, l'empreinte-arbre des LCR est simplement l'empreinte de l'empreinte de la LCR : la générer.

```
$ openssl sha256 -binary data.cades-lt.crl.ltv.der | openssl sha256 -binary \
  > data.cades-lt.crls.sha256
```

Calcul de l'empreinte des champs `version`, `sid` et `digestAlgorithm`

L'empreinte suivante à produire est celle de la concaténation du codage DER des champs `version`, `sid` et `digestAlgorithm` de la structure `SignerInfo` à laquelle sera ajouté l'attribut `LongTermValidation`. La méthode proposée ici est de retrouver les octets correspondant à ces champs dans l'analyse ASN.1 de la signature `CAdES-T`, de les extraire et de les hacher (ci-dessous, les principaux champs et structures de la signature ont été ajoutés en gras).

```
$ openssl asn1parse -in data.cades-t.der -inform DER -i
  0:d=0  hl=4 l=2389 cons: SEQUENCE -- ContentInfo
  4:d=1  hl=2 l= 9 prim: OBJECT          :pkcs7-signedData
 15:d=1  hl=4 l=2374 cons: cont [ 0 ]  -- content
 19:d=2  hl=4 l=2370 cons: SEQUENCE
...
1070:d=3  hl=4 l=1319 cons: SET -- signerInfos
1074:d=4  hl=4 l=1315 cons: SEQUENCE -- SignerInfo
1078:d=5  hl=2 l= 1 prim: INTEGER       :01 -- version
```

```

1081:d=5  hl=2 l= 102 cons:      SEQUENCE -- sid
1083:d=6  hl=2 l=  89 cons:      SEQUENCE
1085:d=7  hl=2 l=  11 cons:      SET
1087:d=8  hl=2 l=   9 cons:      SEQUENCE
1089:d=9  hl=2 l=   3 prim:      OBJECT          :countryName
1094:d=9  hl=2 l=   2 prim:      PRINTABLESTRING :FR
1098:d=7  hl=2 l=  23 cons:      SET
1100:d=8  hl=2 l=  21 cons:      SEQUENCE
1102:d=9  hl=2 l=   3 prim:      OBJECT          :organizationName
1107:d=9  hl=2 l=  14 prim:      PRINTABLESTRING :Mon Entreprise
1123:d=7  hl=2 l=  23 cons:      SET
1125:d=8  hl=2 l=  21 cons:      SEQUENCE
1127:d=9  hl=2 l=   3 prim:      OBJECT          :organizationalUnitName
1132:d=9  hl=2 l=  14 prim:      PRINTABLESTRING :0002 123456789
1148:d=7  hl=2 l=  24 cons:      SET
1150:d=8  hl=2 l=  22 cons:      SEQUENCE
1152:d=9  hl=2 l=   3 prim:      OBJECT          :organizationalUnitName
1157:d=9  hl=2 l=  15 prim:      PRINTABLESTRING :OpenSSL Root CA
1174:d=6  hl=2 l=   9 prim:      INTEGER          :DCD21EE5A2B7DFC7
1185:d=5  hl=2 l=  13 cons:      SEQUENCE -- digestAlgorithm
1187:d=6  hl=2 l=   9 prim:      OBJECT          :sha256
1198:d=6  hl=2 l=   0 prim:      NULL
1200:d=5  hl=3 l= 162 cons:      cont [ 0 ] -- signedAttrs

```

...

Les données à hacher commencent à l'octet 1078 et terminent à l'octet précédant l'octet 1200, soit une longueur de 122 octets. Extraire ces données.

```
$ openssl asn1parse -in data.cades-t.der -inform DER -i -offset 1078 \
-length 122 -out data.cades-lt.v-sid-dig.bin
```

Le fait que cette opération ne produise aucune erreur suggère que les champs ont été extraits intégralement et sans déborder sur les champs suivants (tenter la même commande avec `-length 121` ou `-length 123` pour s'en convaincre).

Calculer l'empreinte du fichier obtenu.

```
$ openssl sha256 -binary data.cades-lt.v-sid-dig.bin \
> data.cades-lt.v-sid-dig.sha256
```

Calcul de l'empreinte des attributs signés

Cette opération a déjà été vue page 154. Exécuter la commande suivante.

```
$ openssl sha256 -binary data.cades-bes.signedAttrs.der \
> data.cades-lt.signedAttrs.sha256
```


Calcul de l'empreinte des champs *signatureAlgorithm* et *signature*

L'empreinte des champs *signatureAlgorithm* et *signature* s'obtient à l'aide de la même méthode que celle proposée pour calculer l'empreinte des champs *version*, *sid* et *digestAlgorithm* (cf. page 157). Afficher l'analyse ASN.1 de la signature CAdES-T (les principaux champs et structures de la signature ont été ajoutés en gras).

```
$ openssl asn1parse -in data.cades-t.der -inform DER -i
  0:d=0  hl=4 l=2389 cons: SEQUENCE -- ContentInfo
    4:d=1  hl=2 l=  9 prim: OBJECT          :pkcs7-signedData
   15:d=1  hl=4 l=2374 cons: cont [ 0 ] -- content
   19:d=2  hl=4 l=2370 cons: SEQUENCE
...
 1070:d=3  hl=4 l=1319 cons: SET -- signerInfos
 1074:d=4  hl=4 l=1315 cons: SEQUENCE -- SignerInfo
...
 1365:d=5  hl=2 l= 13 cons: SEQUENCE -- signatureAlgorithm
 1367:d=6  hl=2 l=  9 prim: OBJECT          :rsaEncryption
 1378:d=6  hl=2 l=  0 prim: NULL
 1380:d=5  hl=4 l= 256 prim: OCTET STRING      [HEX DUMP]:A28AFF00EBAD94EB
7E3E3E29909B23AD4E85A53F216C1A7BFD48145E7C7A2798327EDA577F831AC90C29ED12D4729AFD
7846A715F7CE5343139563BD391FCE48511523D00F077CC1CD1921673F6F2DD273BF256531C860EB
021FB18E1FC83479688A5BC934015FD6CBB9A5DD4C872D87C0FB0A946462E3C59AEF16C7B1F2A3AB
9D388A9CB84CE37EDF3FFB7C4C189E9D47DED1D89C82C577928AFAA283B970948A9AB4AC5CDBC4A8
46947746E252C8D7262416A1286C6A44EC9D48DED51CF147E74719191B1F5CE2E21AF4A8A2AF938D
98767EE1DDE29B04798C8E0D1C6D07DB8D2B960E0E443DA61248B4189FD62DE224866DBFA02D2B7F
9CA1CB4D9E5E7B6 -- signature
 1640:d=5  hl=4 l= 749 cons: cont [ 1 ] -- unsignedAttrs
...
```

Les données à hacher commencent à l'octet 1365 et terminent à l'octet précédant l'octet 1640, soit une longueur de 275 octets. Extraire ces données.

```
$ openssl asn1parse -in data.cades-t.der -inform DER -i -offset 1365 \
  -length 275 -out data.cades-lt.sigalg-sig.bin
```

Calculer l'empreinte du fichier obtenu.

```
$ openssl sha256 -binary data.cades-lt.sigalg-sig.bin \
  > data.cades-lt.sigalg-sig.sha256
```

Calcul de l'empreinte-arbre des attributs non signés

Chacun des attributs non signés doit être soit extrait de la structure CAdES-T en utilisant par exemple la méthode employée pour calculer les empreintes des champs *version*, *sid*, etc., soit construit à partir d'un fichier de configuration ASN.1 interprétable par la commande `openssl asn1parse -genconf`. Cette seconde méthode est utilisée ci-après.

Copier le fichier `data.cades-t.asn.cnf` sous le nom `data.cades-lt.unsignedAttrs.sigts.asn.cnf`, correspondant à `SignatureTimeStamp`, jeton d'horodatage spécifique à CAdES-T et unique attribut non signé.

Remplacer la première ligne du nouveau fichier par la ligne suivante :

```
asn1 = SEQUENCE:attr_signatureTimeStamp
```

Générer le codage DER de l'attribut.

```
$ openssl asn1parse -genconf data.cades-lt.unsignedAttrs.sigts.asn.cnf -i \
  -oid cades-oid.tsv -out data.cades-lt.unsignedAttrs.sigts.bin
  0:d=0  hl=4 l= 745 cons: SEQUENCE
  4:d=1  hl=2 l=  11 prim: OBJECT                  :SignatureTimeStampToken
 17:d=1  hl=4 l= 728 cons: SET
 21:d=2  hl=4 l= 724 cons: SEQUENCE
...
```

Générer l'empreinte SHA-256 du fichier obtenu.

```
$ openssl sha256 -binary data.cades-lt.unsignedAttrs.sigts.bin \
  > data.cades-lt.unsignedAttrs.sigts.sha256
```

Dans le cas d'une structure CAdES-T contenant plusieurs attributs non signés, reproduire cette opération pour chacun des attributs.

L'empreinte-arbre des attributs non signés est simplement l'empreinte de cette empreinte. La générer.

```
$ openssl sha256 -binary data.cades-lt.unsignedAttrs.sigts.sha256 \
  > data.cades-lt.unsignedAttrs.sha256
```

Dans le cas d'une structure contenant plusieurs attributs non signés, employer la méthode décrite pour calculer l'empreinte-arbre des certificats (cf. page 155).

Émission du jeton d'horodatage

Les données à horodater sont la concaténation des huit empreintes et empreintes-arbres calculées précédemment.

À titre de contrôle, les valeurs calculées sont les suivantes :

```
$ genhashlist *.sha256
fa40fd248fc05fdf1e3951569df7c460652183083adada7f90e92648dc5f0cc8:data.cades-lt.c
ertificates.sha256
...
89bd92286d6c8014c06030b25f8b40cc1d5656d4b3b7b4831874f50d6f5557f3:data.cades-lt.c
ontent.sha256
a19ca72ee6d8f8dbd0c8c63ff515efa1aeb1a60975dca0d27db74a897c49046f:data.cades-lt.c
rls.sha256
68eaa891b4a70436878676f07513f96b4f39aceeb54beee6d9d1a9ef62409991:data.cades-lt.e
ContentType.sha256
0edfd0615a3a38b89c5b808bd71d47dbe8ee36cc162d08ea73e541dccda870a0:data.cades-lt.s
igalg-sig.sha256
```

```
91c0d3903a337db5bc75644952d82845cf2c876673d5aafd81e0ab2c7163d04b:data.cades-lt.s
ignedAttrs.sha256
a25e214c81673e4ed1d70b477a7b7e5867e82df632f449adbd774a8b14893033:data.cades-lt.u
nsignedAttrs.sha256
...
bb261bf6901764e741da00eab09536b5e789329e4e2dd45dbe3499eaf6ec84d6:data.cades-lt.v
-sid-dig.sha256
```

Concaténer les empreintes dans l'ordre prévu par la norme CAdES.

```
$ cat \
data.cades-lt.eContentType.sha256 \
data.cades-lt.content.sha256 \
data.cades-lt.certificates.sha256 \
data.cades-lt.crls.sha256 \
data.cades-lt.v-sid-dig.sha256 \
data.cades-lt.signedAttrs.sha256 \
data.cades-lt.sigalg-sig.sha256 \
data.cades-lt.unsignedAttrs.sha256 \
> data.cades-lt.LongTermValidation.data_to_hash
```

L'empreinte de ce fichier est la suivante.

```
$ openssl sha256 data.cades-lt.LongTermValidation.data_to_hash
SHA256(data.cades-lt.LongTermValidation.data_to_hash)= 1296cc40e45a61c64f60229ee
835ed15193e2b162927fd1f287ae49754b2a9c6
```

Générer la requête d'horodatage associée à ce fichier.

```
$ openssl ts -query -data data.cades-lt.LongTermValidation.data_to_hash \
-sha256 -out LongTermValidation.TimeStamp.tsq
```

Générer la réponse d'horodatage.

```
$ openssl ts -reply -config tsa-ts.cnf -section tsa \
-queryfile LongTermValidation.TimeStamp.tsq -inkey tsa-key.pem \
-signer tsa-crt.pem -out LongTermValidation.TimeStamp.tsr
```

Extraire le jeton d'horodatage de cette réponse.

```
$ openssl ts -reply -in LongTermValidation.TimeStamp.tsr -token_out \
-out LongTermValidation.TimeStamp.tst
```

Constitution du champ `timeStamp` de l'attribut `LongTermValidation`

À l'aide d'une méthode au choix, créer le fichier de configuration ASN.1 `LongTermValidation.TimeStampToken.asn.cnf` à partir du jeton d'horodatage. La méthode la plus rapide consiste à effectuer une copie du fichier `signatureTimeStampToken.asn.cnf`, à remplacer toutes les occurrences de la chaîne de caractères `signatureTST` par la chaîne `LTV_TST`, et à effectuer les modifications en gras ci-dessous :

```
asn1 = SEQUENCE:LTV_TST
```

```
[LTV_TST]
...

[LTV_TST_TSTInfo]
version = INTEGER:1
policy = OID:\
1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.5
messageImprint = SEQUENCE:LTV_TST_messageImprint
serialNumber = INTEGER:12
genTime = GENERALIZEDTIME:20120628204038Z
nonce = INTEGER:0x0090a7870811e427e7

[LTV_TST_messageImprint]
hashAlgorithm = SEQUENCE:LTV_TST_TSTInfo_hashAlgorithm
hashedMessage = FORMAT:HEX,OCTETSTRING:\
1296cc40e45a61c64f60229ee835ed15193e2b162927fd1f287ae49754b2a9c6

[LTV_TST_TSTInfo_hashAlgorithm]
...

[LTV_TST_signerInfo]
version = INTEGER:1
sid = SEQUENCE:LTV_TST_sid
digestAlgorithm = SEQUENCE:LTV_TST_digestAlgorithm
signedAttrs = IMPLICIT:0,SEQUENCE:LTV_TST_signedAttrs
signatureAlgorithm = SEQUENCE:LTV_TST_signatureAlgorithm
signature = FORMAT:HEX,OCTETSTRING:\
68fd52c20f948eec7cde8f818d3c3ab1d0959efbebf9b796311ac4c149b6a2b0\
d48215b81e00359c38ad7f1517a10c9f3a9a57d6c0fac8d663be6aec47add59f\
d942a2b5a46a0cefa0145b73e635c172261087dd49f342d32aa5696184649734\
50eccfa833e4b9fb661a04d0b27a940115f3dc2dbefe5a63caaf7cbbde596f3f\
0e6537ffc3ce452c74f9228a78086f25317a11a7660cce638ede7340ece793ea\
6a97d646ce54932e3919b9f45cc5c0517e3134cc3cf9c87588611ee3de8b87b4\
6da78b8b9e1ca38af4f8aca82bf47bff44382e403fc353a22ed2a0648a064008\
cf56a79f122052478a75f99515acedd6d45a9c6655ab2e5e3e4fe8a65a36875c

[LTV_TST_sid]
...

[LTV_TST_attr_signingTime_values]
attr_signingTime_value = UTCTIME:120628204038Z

[LTV_TST_attr_messageDigest]
...

[LTV_TST_attr_messageDigest_values]
messageDigest = FORMAT:HEX,OCTETSTRING:\
c8330ac86400e185465f1a5856a258d0cc434c7b

...
```

Générer le codage DER correspondant.

```
$ openssl asn1parse -genconf LongTermValidation.TimeStampToken.asn.cnf -i \
-out LongTermValidation.TimeStampToken.der
```

S'assurer que le fichier généré est identique au jeton d'origine :

```
$ openssl sha256 LongTermValidation.TimeStamp.tst \
LongTermValidation.TimeStampToken.der
SHA256(LongTermValidation.TimeStamp.tst)= 895ed9b7f0492a88a69519617fa55938b8ace5
6afb1a004c55d221880472d9de
SHA256(LongTermValidation.TimeStampToken.der)= 895ed9b7f0492a88a69519617fa55938b
8ace56afb1a004c55d221880472d9de
```

Constitution de la structure Certificate pour le certificat de l'unité d'horodatage

Copier le fichier ee-Certificate.asn.cnf, constitué dans la section 4.7, sous le nom tsa-Certificate.asn.cnf. Dans ce nouveau fichier, remplacer globalement la chaîne de caractères ee_ par la chaîne tsa_, puis effectuer les modifications mises en évidence en gras ci-dessous.

```
asn1 = SEQUENCE:tsa_certificate

[tsa_certificate]
tbsCertificate = SEQUENCE:tsa_tbsCertificate
signatureAlgorithm = SEQUENCE:tsa_signatureAlgorithm
signatureValue = FORMAT:HEX,BITSTRING:\
0f9ad3afe45665ba7c3b92caa0fe2d49378d53e379e618915176f5a9da8b48c5\
589e850fa297027754f87f0f3dc305e8e21bfad90f3df235712ada052ec7ac5d\
6c40f6ad73948677148aa4ea6b9fea005ef33d7fbade612505c638b787c4308e\
4dd44ec356a493fc743bb488450f53dc90ffeb3be8008c792ba99bccfd6dc3ec\
4c5de3b812681a1d57cdf8a219652a26f796b5734e2a2ce62a9dcd5461c7993a\
8f8564f16f68f054e20c0904afa3c9c79d4c92f8ae007ba6d4d768896ae21250\
a98d268d465ece52b59cba647e12b66e73138ca90a1104f49491d6c340c7ea6a\
ec613e015fed29d6efa873964a937b5cb63e134cb858d8bb1b52fd51edc8c99a

[tsa_tbsCertificate]
version = EXPLICIT:0,INTEGER:2
serialNumber = INTEGER:0xd3b94f04ddd1a040
signature = SEQUENCE:tsa_signature
issuer = SEQUENCE:tsa_issuer
validity = SEQUENCE:tsa_validity
subject = SEQUENCE:tsa_subject
subjectPublicKeyInfo = SEQUENCE:tsa_subjectPublicKeyInfo
extensions = EXPLICIT:3,SEQUENCE:tsa_extensions

[tsa_signatureAlgorithm]
...

[tsa_validity]
notBefore = UTCTIME:120616203401Z
```

notAfter = UTCTIME:140616203401Z

[tsa_subject]

...

[tsa_subject_O_ATV]

type = OID:organizationName

value = PRINTABLESTRING:Mon Entreprise

[tsa_subject_OU_RDN]

rdn = SEQUENCE:tsa_subject_OU_ATV

[tsa_subject_OU_ATV]

type = OID:organizationalUnitName

value = PRINTABLESTRING:0002 123456789

[tsa_subject_CN_RDN]

rdn = SEQUENCE:tsa_subject_CN_ATV

[tsa_subject_CN_ATV]

type = OID:commonName

value = PRINTABLESTRING:OpenSSL TSA

[tsa_subjectPublicKeyInfo]

...

[tsa_subjectPublicKey]

n = INTEGER:0xc890c43fb9ea0df6b6a0d811df6ba4b0294b687f61606932f2189c33cc13db774\2137124592f0fe8b5c78147507daf5f332c817b76f1b62038107b56707b54e6600960c51ad9cf26\73f7342d11f93faf15648993fa5b847f18bd22832fb9fa86fe3409fe502d8f67a7dfd97541a2036\901b21484daa1223df862d70b31dc841fe87beab733c0134f146569294b1ffdd1c4ff7f9e0276e9\2482f3221242e5e958e308a7a74d6d365d911917e367beea0a1577f859fd18fe167e80e95050b1\ a6d14300a69fb7e301e0140ad3cb7fbcd0d791d67fe4771ab4f8cb1f3bfe4ddb02bd2859d93cccb\ e01a4e66a13d3c28ecad4a4565b917a915fda61a5913a8357c15

e = INTEGER:0x010001

[tsa_extensions]

subjectKeyIdentifier=SEQUENCE:tsa_subjectKeyIdentifier

authorityKeyIdentifier=SEQUENCE:tsa_authorityKeyIdentifier

keyUsage=SEQUENCE:tsa_keyUsage

certificatePolicies=SEQUENCE:tsa_certificatePolicies

crlDistributionPoints=SEQUENCE:tsa_crlDistributionPoints

basicConstraints=SEQUENCE:tsa_basicConstraints

extendedKeyUsage=SEQUENCE:tsa_extendedKeyUsage

[tsa_subjectKeyIdentifier]

extnID = OID:subjectKeyIdentifier

extnValue = OCTWRAP,FORMAT:HEX,OCTETSTRING:\

ba0bbbafe3254656fc138692d2154062db166a4a

[tsa_authorityKeyIdentifier]

...

```
[tsa_keyUsage]
extnID = OID:keyUsage
critical = BOOLEAN:true
extnValue = OCTWRAP,FORMAT:BITLIST,BITSTRING:0
```

```
[tsa_certificatePolicies]
```

...

```
[tsa_policyIdentifier]
policyIdentifier = OID:\
1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.4
```

```
[tsa_crlDistributionPoints]
```

...

```
[tsa_extendedKeyUsage]
extnID = OID:extendedKeyUsage
critical = BOOLEAN:true
extnValue = OCTWRAP,SEQUENCE:tsa_keyPurposeIds
```

```
[tsa_keyPurposeIds]
tsa_keyPurposeId = OID:timeStamping
```

Générer le certificat correspondant.

```
$ openssl asn1parse -genconf tsa-Certificate.asn.cnf -i \
-out tsa-Certificate.der
```

Convertir le certificat existant de l'unité d'horodatage au format DER.

```
$ openssl x509 -in tsa-crt.pem -outform DER -out tsa-crt.der
```

Vérifier que les empreintes des deux fichiers DER obtenus sont identiques.

```
$ openssl sha256 tsa-Certificate.der tsa-crt.der
SHA256(tsa-Certificate.der)= 68b22f49939e398872086e8e36c83b34a4e45ff521ae86c99d1
65817322c87cb
SHA256(tsa-crt.der)= 68b22f49939e398872086e8e36c83b34a4e45ff521ae86c99d165817322
c87cb
```

La vérification peut être effectuée sans conversion du certificat à partir du fichier `data.cades-lt.certificate.tsa.sha256` généré précédemment.

```
$ od -tx1 -An data.cades-lt.certificate.tsa.sha256 | tr -d " \n\r"
68b22f49939e398872086e8e36c83b34a4e45ff521ae86c99d165817322c87cb
```

Constitution de la structure Certificate pour le certificat de l'autorité de certification

Par analogie avec la méthode de constitution de la structure Certificate pour l'unité d'horodatage, copier le fichier `ee-Certificate.asn.cnf` sous le nom `ca-`

Certificate.asn.cnf. Dans ce nouveau fichier, remplacer globalement la chaîne de caractères ee_ par la chaîne ca_, puis effectuer les modifications mises en évidence en gras ci-dessous.

```
asn1 = SEQUENCE:ca_certificate

[ca_certificate]
tbsCertificate = SEQUENCE:ca_tbsCertificate
signatureAlgorithm = SEQUENCE:ca_signatureAlgorithm
signatureValue = FORMAT:HEX,BITSTRING:\
96a482086e7a3aa7327cdb5413f10fa03edce80f4f608ec7720f68e6096c8636\
5e85ec659f9233d5db27f87397e96ba4f41fd557785dd3577e416730cb2e4a20\
cae0607d46a22d9102c70c56d30b32c1ded06d0b15f89b85964a57e891ad14f9\
441cdc1d794f1086a62f1f766c280e9e2f5c9a355960787379c11fc0446860e7\
9c870033e01b756e22dc101e1386b6e2459f55f820f875ed274d4e2ba669f2c8\
632dcc95d69f28a45c384ea0489427f9f05b9f49a777fd100089e722c83c0103\
048a43788babe2f5e8f2c01d177f69e92c6cb34e4a790b41e0b91a0ef32b978e\
6fad454a0f0ea6af1b5461ae4f66460a3c303954709105b391dfda6d7c32c1bb

[ca_tbsCertificate]
version = EXPLICIT:0,INTEGER:2
serialNumber = INTEGER:0xacaaff2f9de93c53
signature = SEQUENCE:ca_signature
issuer = SEQUENCE:ca_issuer
validity = SEQUENCE:ca_validity
subject = SEQUENCE:ca_subject
subjectPublicKeyInfo = SEQUENCE:ca_subjectPublicKeyInfo
extensions = EXPLICIT:3,SEQUENCE:ca_extensions

[ca_signatureAlgorithm]
...

[ca_validity]
notBefore = UTCTIME:120407141248Z
notAfter = UTCTIME:220407141248Z

[ca_subject]
C = SET:ca_subject_C_RDN
O = SET:ca_subject_O_RDN
OU1 = SET:ca_subject_OU1_RDN
OU2 = SET:ca_subject_OU2_RDN

[ca_subject_C_RDN]
...

[ca_subject_O_ATV]
type = OID:organizationName
value = PRINTABLESTRING:Mon Entreprise

[ca_subject_OU1_RDN]
rdn = SEQUENCE:ca_subject_OU1_ATV
```



```

[ca_subject_OU1_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:0002 123456789

[ca_subject_OU2_RDN]
rdn = SEQUENCE:ca_subject_OU2_ATV

[ca_subject_OU2_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:OpenSSL Root CA

[ca_subjectPublicKeyInfo]
algorithm = SEQUENCE:ca_rsaEncryption
subjectPublicKey = BITWRAP,SEQUENCE:ca_subjectPublicKey

[ca_rsaEncryption]
algorithm = OID:rsaEncryption
parameters = NULL

[ca_subjectPublicKey]
n = INTEGER:0x9c2dba6207f395a0b4b347310d149130744f1489e0b550f356c4e4804d76c883e\
bfd6929e9c22903ea5688fc388dd238222caefa1f42c7aa0ff7c5789a2728783d19331c63b59ae5\
63ffab84dc6e02191ea051643a735daac0c24ef06c8b84fc846e267d146cac41013e67744b920f7\
49d6d8a4569dfad9c71b425c76fdceb300fea94207dadf75a86cf19a8bc00897b9e8381ecaaaede\
388578e8ed76534da38e95c49230baf8b2fd1fb73eba1d30f392a090773d48d5f1f3de8a40cfe75\
feb8100e42e48021ddae887da0f07ae649679c2ffe00c48888ceacd7f87d85448a4fb50d36fd7b6\
8f844355b2fc5e8c2f6ecd9281cbf3b55bace0bfe3eabf3f85c1
e = INTEGER:0x010001

[ca_extensions]
subjectKeyIdentifier=SEQUENCE:ca_subjectKeyIdentifier
authorityKeyIdentifier=SEQUENCE:ca_authorityKeyIdentifier
keyUsage=SEQUENCE:ca_keyUsage
certificatePolicies=SEQUENCE:ca_certificatePolicies
# Supprimer l'extension crlDistributionPoints
basicConstraints=SEQUENCE:ca_basicConstraints

[ca_subjectKeyIdentifier]
extnID = OID:subjectKeyIdentifier
extnValue = OCTWRAP,FORMAT:HEX,OCTETSTRING:\
4c6d879382f72d2c0723a20fe0712d173f39f38f

[ca_authorityKeyIdentifier]
...

[ca_keyUsage]
extnID = OID:keyUsage
critical = BOOLEAN:true
extnValue = OCTWRAP,FORMAT:BITLIST,BITSTRING:5,6

[ca_certificatePolicies]

```

...

```
[ca_policyIdentifier]
policyIdentifier = OID:\
1.2.840.113556.1.8000.2554.47311.54169.61548.20478.40224.8393003.10972002.1.1
```

Supprimer les sections relatives à l'extension crlDistributionPoints

```
[ca_basicConstraints]
extnID = OID:basicConstraints
critical = BOOLEAN:true
extnValue = OCTWRAP,SEQUENCE:ca_basicConstraints_seq
```

```
[ca_basicConstraints_seq]
cA = BOOLEAN:true
```

Générer le certificat correspondant, convertir le certificat existant de l'autorité de certification au format DER, et vérifier que les empreintes concordent.

```
$ openssl asn1parse -genconf ca-Certificate.asn.cnf -i \
-out ca-Certificate.der
```

```
$ openssl x509 -in ca-crt.pem -outform DER -out ca-crt.der
```

```
$ openssl sha256 ca-Certificate.der ca-crt.der
SHA256(ca-Certificate.der)= 8f6bc593235515f87869afbab31199543ab779688f2b21ea19da
28606c456e39
SHA256(ca-crt.der)= 8f6bc593235515f87869afbab31199543ab779688f2b21ea19da28606c45
6e39
```

Constitution de la structure CertificateList pour la liste de certificats révoqués

Créer le fichier crl-LTV-CertificateList.asn.cnf suivant.

```
asn1 = SEQUENCE:ltv_certificateList
```

```
[ltv_certificateList]
tbsCertList = SEQUENCE:ltv_crl_tbsCertList
signatureAlgorithm = SEQUENCE:ltv_crl_signatureAlgorithm
signatureValue = FORMAT:HEX,BITSTRING:\
3e7f67f7af8ec194c84960394bd6471f4f30458a73b441863645660516e7dafe\
f4adc61bb9d335bbaa4f6df76351322cadcfac28c14e9d3eef9da66ff66f1878\
0848260cdef62d72965415a4a8e2a21498ca97e932fe92775a6de5a907eac2ab\
fe2b7e3e72210c1beb120099f0ccd3b8968296e5fca592f6fa2d36929898377f\
1264ef871b3286fdb994d9916fa897255eae8e5f202fc23305374e0aef5510b\
92315c6b7085eb532a283eb23140e4f9297cad3bdbb266b5ce0a2d88b7e5c7b3\
66d5d1b6e16b0335d7e7159aacb7d99952d628757965f6867117e10eb2d8fbca\
e112074b6b44c867a833864010cb55e5f2c9dd2801cb42acd77106c242dda08d
```

```
[ltv_crl_tbsCertList]
```

```

version = INTEGER:1
signature = SEQUENCE:ltv_crl_signature
issuer = SEQUENCE:ltv_crl_issuer
thisUpdate = UTCTIME:120627194518Z
nextUpdate = UTCTIME:120629194518Z
crlExtensions = EXPLICIT:0,SEQUENCE:ltv_crl_crlExtensions

[ltv_crl_signature]
algorithm = OID:sha256WithRSAEncryption
parameters = NULL

[ltv_crl_issuer]
C = SET:ltv_crl_issuer_C_RDN
O = SET:ltv_crl_issuer_O_RDN
OU1 = SET:ltv_crl_issuer_OU1_RDN
OU2 = SET:ltv_crl_issuer_OU2_RDN

[ltv_crl_issuer_C_RDN]
rdn = SEQUENCE:ltv_crl_issuer_C_ATV

[ltv_crl_issuer_C_ATV]
type = OID:countryName
value = PRINTABLESTRING:FR

[ltv_crl_issuer_O_RDN]
rdn = SEQUENCE:ltv_crl_issuer_O_ATV

[ltv_crl_issuer_O_ATV]
type = OID:organizationName
value = PRINTABLESTRING:Mon Entreprise

[ltv_crl_issuer_OU1_RDN]
rdn = SEQUENCE:ltv_crl_issuer_OU1_ATV

[ltv_crl_issuer_OU1_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:0002 123456789

[ltv_crl_issuer_OU2_RDN]
rdn = SEQUENCE:ltv_crl_issuer_OU2_ATV

[ltv_crl_issuer_OU2_ATV]
type = OID:organizationalUnitName
value = PRINTABLESTRING:OpenSSL Root CA

[ltv_crl_crlExtensions]
authorityKeyIdentifier = SEQUENCE:ltv_crl_authorityKeyIdentifier
crlNumber = SEQUENCE:ltv_crl_crlNumber

[ltv_crl_authorityKeyIdentifier]
extnID = OID:authorityKeyIdentifier

```

```
extnValue = OCTWRAP,SEQUENCE:ltv_crl_authorityKeyIdentifier_seq
```

```
[ltv_crl_authorityKeyIdentifier_seq]
keyIdentifier = IMPLICIT:0,FORMAT:HEX,OCTETSTRING:\
4c6d879382f72d2c0723a20fe0712d173f39f38f
```

```
[ltv_crl_crlNumber]
extnID = OID:crlNumber
extnValue = OCTWRAP,INTEGER:9
```

```
[ltv_crl_signatureAlgorithm]
algorithm = OID:sha256WithRSAEncryption
parameters = NULL
```

Il serait bien entendu possible de repartir du fichier `crl-CertificateList.asn.cnf`, en ayant pris le soin de supprimer le champ `revokedCertificates` et les sections correspondantes.

Générer la liste de certificats révoqués (LCR), et vérifier que son empreinte est identique à l'empreinte du fichier `data.cades-lt.crl.ltv.der`.

```
$ openssl asn1parse -genconf crl-LTV-CertificateList.asn.cnf -i \
-out crl-LTV-CertificateList.der
```

```
$ openssl sha256 crl-LTV-CertificateList.der data.cades-lt.crl.ltv.der
SHA256(crl-LTV-CertificateList.der)= 5e46281a55a2ab7039efe80bafbf227bd945a830284
f48e72010f07b40e899ea
SHA256(data.cades-lt.crl.ltv.der)= 5e46281a55a2ab7039efe80bafbf227bd945a830284f4
8e72010f07b40e899ea
```

Construction de l'attribut *LongTermValidation*

La structure ASN.1 à spécifier pour constituer l'attribut *LongTermValidation* est la suivante, en remplaçant le choix (CHOICE) du champ `poeValue` par l'élément `timeStamp` retenu, de même pour `extraCertificates` qui contiendra uniquement (après résolution du CHOICE) des *Certificate* et pour `extraRevocation` qui contiendra uniquement des *CertificateList* :

```
LongTermValidation ::= SEQUENCE {
    poeDate GeneralizedTime,
    timeStamp [0] EXPLICIT TimeStampToken,
    extraCertificates [0] IMPLICIT SET OF Certificate,
    extraRevocation [1] IMPLICIT SET OF CertificateList
}
```

Créer le fichier `LongTermValidation.asn.cnf`, avec le contenu suivant (la valeur du champ `poeDate` est reprise du champ `genTime` de la structure *TSTInfo* encapsulée par le jeton d'horodatage à inclure dans l'attribut *LongTermValidation*).

```
asn1 = SEQUENCE:LongTermValidation

[LongTermValidation]
poeDate = GENERALIZEDTIME:20120628204038Z
```

```
timeStamp = EXPLICIT:0,SEQUENCE:LTV_TST
extraCertificates = EXPLICIT:0,SET:extraCertificates
extraRevocation = EXPLICIT:1,SET:extraRevocation
```

```
[extraCertificates]
certificate_1 = SEQUENCE:tsa_certificate
certificate_2 = SEQUENCE:ca_certificate
```

```
[extraRevocation]
certificateList_1 = SEQUENCE:ltv_certificateList
```

Copier à la suite toutes les sections des fichiers LongTermValidation.TimestampToken.asn.cnf, tsa-Certificate.asn.cnf, ca-Certificate.asn.cnf et crl-LTV-CertificateList.asn.cnf (ne pas inclure la ligne `asn1 = ...`).

Vérifier que l'attribut LongTermValidation peut être généré correctement à partir de ce fichier.

```
$ openssl asn1parse -genconf LongTermValidation.asn.cnf -i
```

Finalisation de la structure CAdES-LT

Copier le fichier `data.cades-t.asn.cnf` sous le nom `data.cades-lt.asn.cnf`.

Modifier la section `unsignedAttrs` en ajoutant la ligne en gras ci-dessous.

```
[unsignedAttrs]
attr_signatureTimeStamp = SEQUENCE:attr_signatureTimeStamp
attr_longTermValidation = SEQUENCE:attr_longTermValidation
```

Ajouter ensuite les sections suivantes, correspondant à l'attribut non signé LongTermValidation, à la fin du fichier.

```
[attr_longTermValidation]
attrType = OID:LongTermValidation
attrValues = SET:attr_longTermValidation_values
```

```
[attr_longTermValidation_values]
attr_contentType_value = SEQUENCE:LongTermValidation
```

Inclure à présent toutes les sections du fichier `LongTermValidation.asn.cnf` (la ligne `asn1 = ...` ne doit pas être copiée).

Enfin, ajouter la ligne suivante dans le fichier `ca-des-oid.tsv`, pour déclarer l'OID de l'attribut LongTermValidation.

```
0.4.0.1733.2.2 id-aa-ets-longTermValidation LongTermValidation
```

Générer la structure CAdES-LT ainsi finalisée.

```
$ openssl asn1parse -genconf data.cades-lt.asn.cnf -i -oid ca-des-oid.tsv \
-out data.cades-lt.der
```

Vérifier que la structure générée est une signature CMS valide.

```
$ openssl cms -verify -inform DER -in data.cades-t.der -content data.txt \  
-CAfile ca-crt.pem  
texte en clairVerification successful
```

Chapitre 13 — Signature électronique avancée — XAdES

À l'image de la CAdES, la norme XAdES⁷⁶ définit un ensemble de propriétés, dites qualifiantes, qui peuvent être incluses dans une signature électronique XML Signature, et dont certaines sont signées avec les données.

La norme XAdES définit également plusieurs formats de signature électronique, pour la plupart équivalents aux formats CAdES correspondants :

- XAdES-BES (*Basic Electronic Signature*, ou signature électronique de base) inclut le certificat du signataire (ou une référence à celui-ci) dans les éléments à signer, permet d'ajouter des propriétés complémentaires (telles que le lieu de signature et l'heure de signature) aux éléments à signer, et d'intégrer une contresignature en tant que propriété non signée.
- XAdES-EPES (*Explicit Policy based Electronic Signature*, ou signature électronique avec politique explicite).
- XAdES-T (T pour *time*, ou heure).
- XAdES-C (C pour *complete validation data references*, ou références complètes aux données de validation) complète le format XAdES-T avec les certificats et informations de révocation permettant de valider la signature électronique.
- XAdES-X (*extended signature with time indication forms*, ou signature étendue avec formes d'indicateur d'heure) complète une structure XAdES contenant des références aux données de validation (typiquement une structure XAdES-C) en ajoutant aux propriétés non signées un ou plusieurs jetons d'horodatage portant soit sur la signature initiale, une éventuelle heure de signature fiable, et les références aux données de validation (XAdES-X de type 1), soit sur les références aux données de validation uniquement (XAdES-X de type 2).
- XAdES-X-L (*extended long signature with time*, ou signature longue étendue avec heure) ajoute à une structure XAdES-X les données de validation référencées.
- XAdES-A (*archival signature*, ou signature pour archivage) ajoute aux propriétés non signées d'une structure XAdES des données de validation si elles ne sont pas déjà présentes, et un ou plusieurs jetons d'horodatage portant sur la signature et ces données. Afin de permettre la validation de la signature dans le temps, une signature enrichie au niveau XAdES-A peut être réhorodatée de manière à en prolonger la durée de validité.

La conformité à la norme XAdES n'impose pas d'être en capacité de produire ou de vérifier les formats XAdES-X, XAdES-X-L et XAdES-A.

La suite de ce chapitre s'intéresse à la construction d'une signature XAdES-A minimale. Pour cela, le fichier modèle `data.dsig-tmpl.xml`, à la base de signature enveloppée créée dans la section 8.1, est complété avec les éléments nécessaires à la constitution d'une structure XAdES-BES, puis

76. <http://uri.etsi.org/01903/v1.4.1/>

celle-ci est successivement enrichie avec les données permettant d'obtenir une signature avancée au format XAdES-T, puis XAdES-A. La version 1.4.2 de XAdES est utilisée.

13.1. Constitution de la structure XAdES-BES

La structure XAdES-BES à constituer est dotée de l'arborescence suivante :

```
ds:Signature (1)
  ds:SignedInfo
    ds:CanonicalizationMethod
    ds:SignatureMethod
    ds:Reference
    ...
    ds:Reference (2)
    ...
  ds:SignatureValue
  ds:KeyInfo
  ...
ds:Object (3)
  xades:QualifyingProperties (4)
  xades:SignedProperties (5)
  ...
```

Le préfixe `ds:` représente l'espace de nommage <http://www.w3.org/2000/09/xmlsig#>, et le préfixe `xades:` représente <http://uri.etsi.org/01903/v1.3.2#>.

L'espace de nommage représenté par `xades:` fait référence à la version 1.3.2 de XAdES. Un espace de nommage complémentaire, introduit dans la version 1.4.1 de XAdES et portant le préfixe `xadesv141`, est utilisé plus loin pour les jetons d'horodatage spécifiques à XAdES-A. La version 1.4.2 de la norme corrige simplement quelques coquilles dans le schéma XML de XAdES, mais le préfixe et le nom de l'espace de nommage maintiennent la référence à la version 1.4.1.

Les modifications à apporter au modèle de signature XML Signature utilisé dans la section 8.1 pour obtenir un modèle de signature XAdES-EBES sont les suivantes :

- L'élément `ds:Signature`, au nœud 1 du schéma précédent, se voit enrichi d'un attribut `Id`, lequel est référencé par les propriétés qualifiant la signature (nœud 4).
- Les propriétés signées, contenues dans l'élément `xades:SignedProperties` (nœud 5), doivent — comme leur nom l'indique — être signées : un nouvel élément `ds:Reference` est ajouté (nœud 2) sous `ds:SignedInfo`, aux côtés de l'élément `ds:Reference` référençant les données à signer (en l'occurrence l'ensemble du document XML).
- Un élément `ds:Object` (nœud 3) est ajouté sous `ds:Signature` pour stocker les propriétés qualifiant la signature dans un élément `xades:QualifyingProperties` (nœud 4), lequel contient notamment les propriétés qualifiantes signées (nœud 5).

Créer le fichier `data.xades-bes-tmpl.xml` suivant, évolution du fichier `data.dsig-tmpl.xml` intégrant ces modifications (mises en évidence en gras).

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <data>Texte en clair</data>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="Id_Signature">
    <ds:SignedInfo>
      <ds:CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      <ds:SignatureMethod
        Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
      <ds:Reference URI="">
        <ds:Transforms>
          <ds:Transform
            Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
          <ds:Transform
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        </ds:Transforms>
        <ds:DigestMethod
          Algorithm="http://www.w3.org/2001/04/xmenc#sha256" />
        <ds:DigestValue />
      </ds:Reference>
      <ds:Reference URI="#Id_SignedProperties">
        <ds:Transforms>
          <ds:Transform
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        </ds:Transforms>
        <ds:DigestMethod
          Algorithm="http://www.w3.org/2001/04/xmenc#sha256" />
        <ds:DigestValue />
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue />
    <ds:KeyInfo>
      <ds:X509Data>
        <ds:X509Certificate />
      </ds:X509Data>
    </ds:KeyInfo>
    <ds:Object>
      <xades:QualifyingProperties Target="#Id_Signature"
        xmlns:xades="http://uri.etsi.org/01903/v1.3.2#">
        <xades:SignedProperties Id="Id_SignedProperties">
          <xades:SignedSignatureProperties>
            <xades:SigningTime>2012-06-17T15:23:00Z</xades:SigningTime>
            <xades:SignatureProductionPlace>
              <xades:CountryName>France</xades:CountryName>
            </xades:SignatureProductionPlace>
          </xades:SignedSignatureProperties>
        </xades:SignedProperties>
      </xades:QualifyingProperties>
    </ds:Object>
  </ds:Signature>
</root>
```

```

    </ds:Object>
  </ds:Signature>
</root>

```

Le choix des propriétés signées `xades:SigningTime` et `xades:SignatureProductionPlace` est arbitraire.

Validation par rapport aux schémas XML

Par précaution, il est judicieux de contrôler que les fichiers XML contenant des signatures XAdES sont conformes aux schémas XML de référence de XML Signature et de XAdES.

Télécharger les schémas XML de XML Signature⁷⁷ et de XAdES⁷⁸ dans le répertoire de `data.xml`, puis valider la conformité des éléments `ds:Signature` et `xades:QualifyingProperties` par rapport à ces schémas à l'aide de l'option `--schema` de `xmllint`.

```

$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# -t |
  -c /root/ds:Signature data.xades-bes-tmpl.xml | xmllint \
  --schema xmldsig-core-schema.xsd --noout -
- validates

$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# \
  -N xades=http://uri.etsi.org/01903/v1.3.2# -t \
  -c /root/ds:Signature/ds:Object/xades:QualifyingProperties \
  data.xades-bes-tmpl.xml | xmllint --schema XAdES.xsd --noout -
- validates

```

Ces opérations peuvent être répétées pour contrôler la validité structurelle des autres formats XAdES.

Dans un souci d'exhaustivité, créer le fichier XML Schema `data-dsig.xsd` associé aux versions signées du fichier `data.xml` dans le même répertoire que ci-dessus :

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="xmldsig-core-schema.xsd"/>
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="data"/>
        <xs:element ref="ds:Signature"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

77. <http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-schema.xsd>

78. <http://uri.etsi.org/01903/v1.3.2/XAdES.xsd>

```
</xs:element>
</xs:schema>
```

Valider le fichier `data.xml` par rapport à ce schéma :

```
$ xmllint --schema data-dsig.xsd --noout data.xades-bes-tmpl.xml
data.xades-bes-tmpl.xml validates
```

Générer la signature électronique XAdES-BES.

```
$ xmlsec --sign --privkey-pem ee-key.pem,ee-crt-authsig.pem \
  --output data.xades-bes.xml data.xades-bes-tmpl.xml
```

Dans le fichier généré, le premier élément `ds:Reference` est identique à l'élément `ds:Reference` du fichier `data.dsig.xml`, puisque dans les deux cas les données référencées, à savoir l'ensemble du document XML à l'exception du nœud `ds:Signature`, sont identiques.

Vérifier la signature :

```
$ xmlsec --verify --trusted-pem ca-crt.pem data.xades-bes.xml
OK
SignedInfo References (ok/all): 2/2
Manifests References (ok/all): 0/0
```

Ce processus valide les deux éléments `ds:Reference` contenus sous le nœud `ds:SignedInfo`.

13.2. Constitution de la structure XAdES-T

Parmi les deux possibilités offertes par la norme XAdES, celle retenue ci-après pour construire la signature XAdES est de générer un jeton d'horodatage et de l'inclure dans un élément `xades:SignatureTimeStamp` parmi les propriétés non signées de la signature.

La structure `ds:Signature` ainsi générée est schématisée ci-dessous, où le nœud 1 représente un nouvel élément, `xades:UnsignedProperties`, à créer sous `xades:QualifyingProperties` pour stocker les propriétés non signées, le nœud 2 encapsule les propriétés non signées portant sur la signature, et le nœud 3 contient le jeton d'horodatage.

```
ds:Signature
  ds:SignedInfo
  ...
  ds:Object
    xades:QualifyingProperties
      xades:SignedProperties
      ...
      xades:UnsignedProperties (1)
        xades:UnsignedSignatureProperties (2)
          xades:SignatureTimeStamp (3)
```

La première étape est de générer la requête d'horodatage.

La donnée à horodater est l'élément XML `ds:SignatureValue` sous le nœud `ds:Signature` représentant la signature électronique considérée, après canonicalisation par l'algorithme indiqué dans l'élément `xades:SignatureTimeStamp` s'il est présent, sinon l'algorithme de canonicalisation par défaut de XML Signature. Dans le cas présent, l'algorithme de canonicalisation exclusive est utilisé.

Extraire l'élément `ds:Signature` de la signature XAdES-BES générée précédemment, le canonicaliser et calculer l'empreinte SHA-256 des données obtenues.

```
$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# \
  -N xades=http://uri.etsi.org/01903/v1.3.2# -t \
  -c /root/ds:Signature/ds:SignatureValue data.xades-bes.xml \
  | xmllint --exc-c14n - | openssl sha256
(stdin)= 6ad678e8370e1eba4516e643b9f5316f7235f24d1fc5761f03d992dae754fb32
```

Générer à présent une requête d'horodatage portant sur cette empreinte en utilisant l'option `-digest` de la commande `openssl ts`.

```
$ openssl ts -query \
  -digest 6ad678e8370e1eba4516e643b9f5316f7235f24d1fc5761f03d992dae754fb32 \
  -sha256 -out SignatureTimeStamp.tsq
```

Produire la réponse à cette requête :

```
$ openssl ts -reply -config tsa-ts.cnf -section tsa \
  -queryfile SignatureTimeStamp.tsq -inkey tsa-key.pem -signer tsa-crt.pem \
  -out SignatureTimeStamp.tsr
Using configuration from tsa-ts.cnf
Response has been generated.
```

La version 1.4.1 de XAdES précise bien que le jeton d'horodatage à inclure dans la signature est le champ `timeStampToken` de la réponse, et non la réponse elle-même.

Extraire le jeton d'horodatage de cette réponse.

```
$ openssl ts -reply -in SignatureTimeStamp.tsr -token_out \
  -out SignatureTimeStamp.tst
```

Coder le jeton en Base64.

```
$ openssl base64 -in SignatureTimeStamp.tst
MIIC1AYJKoZIhvcNAQcCoIICxTCCAsECAQMxCzAJBgUrDgMCGGUAMIGOBgsqhkiG
9w0BCRABBKB/BH0wewIBAQQkKoZIhvcUAb5Ak3qC8U+DpxmD4GyBn36CuiCEgKIr
hZ3WYgEFMDEwDQYJYIZIAWUDBAIBBQAEIGrWe0g3Dh66RRbmQ7n1MW9yNfJNH8V2
HwPZktrnVPsyAgEDGA8yMDEyMDYyMjE5NDQ1OV0CCQCVo26o8wA/ZjGCAhwwggIY
AgEBMGYwWTELMakGA1UEBhMCRlIxZzFzAVBgNVBAoTDk1vbiBFbnRyZXByaXNlMRcw
FQYDVQQLEw4wMDAyIDEyMzQ1Njc4OTEYMBYGA1UECzMPT3B1b1NTTCBSb290IENB
AgkA071PBN3RoEAWCQYFKw4DAh0FAKCBjDAaBgkqhkiG9w0BCQMxDQYLKoZIhvcN
AQkQAQQwHAYJKoZIhvcNAQkFMQ8XDTEyMDYyMjE5NDQ1OVowIwYJKoZIhvcNAQkE
MRYEFAh09tLfAkFk+4leqen/ln+SYkn3MCsGCyqGSib3DQEJEAIMMRwwGjAYMBYE
FEMopucZySTSxBTCSucZUJOVYcUJMAOGCSqGSib3DQEBAQUABIIBAC6z7djc4yv8
v3LjJRSR+zjJs2MV1VbdRnZ6EZnnq0spZseQUS5kHgD0oT36C0GtDhDgH1W6HDSq
unSMWxIOG08Kq/ypM+6NsD4xld6iggB8JXpHPvTet638rZrNmMiE9lzFFqfPoJsn
```

```
xElYIJzf9xLKE+tmUopyXCa69Qwxh5NAMZYiFfkXtvBtxf8+feAuYStD7lH6AQQk
eamF8k0gULz5EuYslj+II8gJHQPfM3hTiocrh0p5T8c6qrrpZ9WAOR0mj6/VIdYx
OUEnq4ws8YrvhVi5r+E9K63drt7WJh3rwWONcyoo81zqVwK2RMtBZQG38q00uUA8
ElvsanzHS1g=
```

Enfin, copier le fichier `data.xades-bes.xml` sous le nom `data.xades-t.xml`, et ajouter dans celui-ci les éléments mis en évidence en gras ci-dessous pour obtenir la structure XAdES-T (la valeur de l'élément `xades:EncapsulatedTimeStamp` est le codage en Base64 du jeton d'horodatage obtenu ci-avant) :

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <data>Texte en clair</data>
  <ds:Signature ...>
    ...
    <ds:Object>
      <xades:QualifyingProperties ...>
        <xades:SignedProperties ...>
          ...
          </xades:SignedProperties>
          <xades:UnsignedProperties>
            <xades:UnsignedSignatureProperties>
              <xades:SignatureTimeStamp>
                <ds:CanonicalizationMethod
                  Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                <xades:EncapsulatedTimeStamp>
MIIC1AYJKoZIhvcNAQcCoIICxTCCAsECAQMxCzAJBgUrDgMCGGUAMIGOBgsqhkiG
...
ElvsanzHS1g=
              </xades:EncapsulatedTimeStamp>
            </xades:SignatureTimeStamp>
          </xades:UnsignedSignatureProperties>
        </xades:UnsignedProperties>
      </xades:QualifyingProperties>
    </ds:Object>
  </ds:Signature>
</root>
```

L'élément `xades:EncapsulatedTimeStamp` est de type `xades:EncapsulatedPKIDataType`, et contient donc par défaut le codage en Base64 de la représentation DER du jeton d'horodatage, à moins qu'un attribut `Encoding` ne précise un autre type de codage (se reporter à la section 7.1.3 de la norme XAdES pour plus d'informations).

Vérifier que la signature est toujours valide, et que l'élément `xades:QualifyingProperties` reste conforme au schéma XML de XAdES.

La structure obtenue peut être visualisée à l'aide de l'un des moyens suivants.

Avec la commande du shell de l'outil `xmllint`, en perdant les préfixes des espaces de nommage :

```
xmllint --shell data.xades-t.xml
/ > du
/
```

```

root
  data
    Signature
      SignedInfo
...
      CanonicalizationMethod
      EncapsulatedTimeStamp

```

En combinant la commande `xml el` avec une expression régulière Perl (penser à remplacer les apostrophes par des guillemets sous Windows) :

```
$ xml el data.xades-t.xml | perl -pe 's/(.*?)\// /g'
```

```

root
  data
    ds:Signature
      ds:SignedInfo
        ds:CanonicalizationMethod
        ds:SignatureMethod
        ds:Reference
          ds:Transforms
            ds:Transform
            ds:Transform
          ds:DigestMethod
          ds:DigestValue
        ds:Reference
          ds:Transforms
            ds:Transform
          ds:DigestMethod
          ds:DigestValue
        ds:SignatureValue
        ds:KeyInfo
          ds:X509Data
            ds:X509Certificate
        ds:Object
          xades:QualifyingProperties
          xades:SignedProperties
            xades:SignedSignatureProperties
              xades:SigningTime
              xades:SignatureProductionPlace
              xades:CountryName
            xades:UnsignedProperties
              xades:UnsignedSignatureProperties
                xades:SignatureTimeStamp
                ds:CanonicalizationMethod
                xades:EncapsulatedTimeStamp

```

13.3. Constitution de la structure XAdES-A

La structure XAdES-A doit inclure l'ensemble des données de certification et de révocation permettant de valider la signature.

Attention, tout comme pour CAdES-LT dans 12.3, la structure produite ci-après n'a pas été validée par un outil tiers : les *Plugtests* de l'ETSI évoqués dans cette section s'intéressent également à l'interopérabilité des formats CAdES.

Copier le fichier `data.xades-t.xml` sous le nom `data.xades-a.xml`.

Ajout du certificat de l'unité d'horodatage

L'élément `xadesv141:TimeStampValidationData` introduit dans la version 1.4.1 de XAdES permet de définir, en tant que propriété non signée, les données de certification et de révocation associées à un jeton d'horodatage.

La norme XAdES est assez ouverte quant à ce qui peut ou doit être fait de ces données : s'attendre à des comportements différents de la part des outils de validation des signatures électroniques avancées.

Ajouter cet élément au fichier `data.xades-a.xml` pour inclure le certificat de l'unité d'horodatage, comme mis en évidence ci-dessous en gras. La valeur de l'élément `xades:EncapsulatedX509Certificate` est le codage en Base64 de la représentation DER du certificat de l'unité d'horodatage (contenu au format PEM dans le fichier `tsa-crt.pem`). Ajouter par ailleurs la déclaration de l'espace de nommage associé au préfixe `xadesv141` à l'élément `xades:QualifyingProperties`.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <data>Texte en clair</data>
  <ds:Signature ...>
    ...
    <ds:Object>
      <xades:QualifyingProperties
        xmlns:xades="http://uri.etsi.org/01903/v1.3.2#"
        xmlns:xadesv141="http://uri.etsi.org/01903/v1.4.1#"
        Target="#Id_Signature">
        <xades:SignedProperties ...>
          ...
          </xades:SignedProperties>
          <xades:UnsignedProperties>
            <xades:UnsignedSignatureProperties>
              <xades:SignatureTimeStamp>
                ...
              </xades:SignatureTimeStamp>
              <xadesv141:TimeStampValidationData URI="#Id_SignatureTimeStamp">
                <xades:CertificateValues>
                  <xades:EncapsulatedX509Certificate>
MIIECTCCA vGgAwIBAgIJAN05TwTd0aBAMAOGCSqGSIb3DQEBCwUAMFkxCzAJBgNV
...
AV/tKdbvqH0WSpN7XLY+E0y4WNi7G1L9Ue3IyZo=
                  </xades:EncapsulatedX509Certificate>
                </xades:CertificateValues>
              </xadesv141:TimeStampValidationData>
            </xades:UnsignedSignatureProperties>
          </xades:UnsignedProperties>
        </xades:QualifyingProperties>
      </ds:Object>
```

```
</ds:Signature>
</root>
```

Bien que cela ne soit pas imposé par XAdES, ajouter l'attribut `URI` à l'élément `xadesv141:TimeStampValidationData` pour faire référence à un élément représentant un jeton d'horodatage, à l'exemple de `xades:SignatureTimeStamp`, identifié par son attribut `Id`, comme illustré ci-dessous, les nouveaux attributs étant en gras.

```
...
    <xades:EncapsulatedTimeStamp Id="Id_SignatureTimeStamp">
        ...
    </xades:EncapsulatedTimeStamp>
</xades:SignatureTimeStamp>
<xadesv141:TimeStampValidationData URI="#Id_SignatureTimeStamp">
    ...
</xadesv141:TimeStampValidationData>
...
```

Préciser à quel jeton d'horodatage se rapportent les données de validation et de révocation se révèle particulièrement opportun lorsque plusieurs éléments `xadesv141:TimeStampValidationData` figurent dans la signature, avec par exemple des listes de révocation émises par la même autorité de certification à des dates différentes selon la date de production du jeton d'horodatage.

Ajout du certificat de l'autorité de certification

Le certificat de l'autorité de certification émettrice du certificat du signataire et de la liste de certificats de révoqués, qui ne figure pas dans la structure XAdES-T constituée précédemment, doit être ajoutés au titre des données de certification dans une propriété non signée sous l'élément `xades:CertificateValues`.

Ajouter à la signature électronique l'élément `xades:CertificateValues`, mis en évidence en gras ci-dessous (la valeur de l'éléments `xades:EncapsulatedX509Certificate` est le codage en Base64 de la représentation DER du certificat de l'autorité de certification) :

```
...
    <xades:UnsignedProperties>
        <xades:UnsignedSignatureProperties>
            <xades:SignatureTimeStamp>
                ...
            </xades:SignatureTimeStamp>
            <xadesv141:TimeStampValidationData ...>
                ...
            </xadesv141:TimeStampValidationData>
            <xades:CertificateValues>
                <xades:EncapsulatedX509Certificate>
MIIDzTCCArWgAwIBAgIJAKyq/y+d6TxTMAOGCSqGSIb3DQEBCwUAMFkxCzAJBgNV
...
CjwwOVRwkQWzkd/abXwywbs=
                </xades:EncapsulatedX509Certificate>
            </xades:CertificateValues>
        </xades:UnsignedSignatureProperties>
    </xades:UnsignedProperties>
</xades:SignatureProperties>
</xades:Signature>
```



```

    </xades:UnsignedSignatureProperties>
  </xades:UnsignedProperties>

```

...

Ajout de la liste de certificats révoqués

Mettre à jour la liste de certificats révoqués si elle a expiré.

```

$ openssl ca -gencrl -cert ca-crt.pem -keyfile ca-key.pem -crlhours 48 \
-md sha256 -config ca-crl.cnf -name ca_crl -crlexts ca_crl_ext \
-out ca-crl.pem

```

Ajouter le codage en Base64 de la liste de certificats révoqués (le fichier généré sans les balises PEM) au fichier data.xades-a.xml, dans un élément xades:CRLValues/xades:EncapsulatedCRLValue d'une nouvelle propriété non signée xades:RevocationValues, comme ci-dessous en gras :

...

```

<xades:UnsignedProperties>
  <xades:UnsignedSignatureProperties>
    <xades:SignatureTimeStamp>
      ...
    </xades:SignatureTimeStamp>
    <xadesv141:TimeStampValidationData ...>
      ...
    </xadesv141:TimeStampValidationData>
    <xades:CertificateValues>
      ...
    </xades:CertificateValues>
    <xades:RevocationValues>
      <xades:CRLValues>
        <xades:EncapsulatedCRLValue>

```

```

MIIB0zCBvAIBATANBgkqhkiG9w0BAQsFADBZMQswCQYDVQQGEwJGUjEXMBUGA1UE
ChMOTW9uIEVudHJlcHJpc2UxZzAVBgNVBAsTDjAwMDIgaMTIzNDU2Nzg5MRgwFgYD
VQQLew9PcGVuU1NMIjFJvb3QgQ0EXDTEyMDYyMjIwMjg1NloXDTEyMDYyNDIwMjg1
NlqgLzAtMB8GA1UdIwQYMBAAFEExth5OC9y0sByOiD+BxLRc/OfOPMAoGA1UdFAQD
AgEDMAOGCSqGSIB3DQEBChUAA4IBAQB34WzWDW4A12Za/JMZ68CVpMnSpD5mzeGm
vvT/LRHH0yR6xCCfiwdorCF2rYyYrg3lynnrfsInYVrZclfQWF5TkWbFQONebY+Q
4NEcXiKfkYJuca5W44YnIUfHDEQVVD1tTqbDECxcFmG/UukqflprXmMHra1RdI8h
4SrK7aac5yJiTxc+QJhOVDybpz7Tco3lDpH4uobv30CZD0+20ZBfEzy5Kce3PUV
7h+XGW/2/fRXxvBk3DVVSyPAqq069trZAPQiM6tproFGPJAXA3FgTHEqxGTIPpEf
WXqkwe2GzFTz6bxBd4qWEPEUk0ViHDB+FHdZshL5hDH8e+gEATF+

```

```

        </xades:EncapsulatedCRLValue>

```

```

      </xades:CRLValues>

```

```

    </xades:RevocationValues>

```

```

  </xades:UnsignedSignatureProperties>

```

```

</xades:UnsignedProperties>

```

...

Sauvegarder le fichier `data.xades-a.xml` obtenu à ce stade.

Ajout d'un jeton d'horodatage d'archivage

Le jeton d'horodatage d'archivage en cours de constitution aura la forme suivante :

```
<xadesv141:ArchiveTimeStamp Id="Id_ArchiveTimeStamp">
  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
  <xades:EncapsulatedTimeStamp>
...
  </xades:EncapsulatedTimeStamp>
</xadesv141:ArchiveTimeStamp>
```

Les données à horodater référencées dans le jeton d'horodatage d'archivage sont constituées en suivant les étapes décrites dans la section 8.2.1 de la version 1.4.2 de la norme XAdES : l'algorithme de canonicalisation exclusive doit être utilisé lorsque la procédure indique que des données XML doivent être canonicalisées.

Sous Windows, ne pas oublier de systématiquement faire suivre l'opération de canonicalisation de la commande `| tr -d "\r"` (cf. section 9.3).

Le premier élément `ds:Reference` à traiter est le suivant :

```
<ds:Reference URI="">
  <ds:Transforms>
    <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
    <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
  </ds:Transforms>
  <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmldsig#sha256" />
  <ds:DigestValue>o00jEBylngTpuEsj0e+RkjwymUf5e+wiDE151+Z7Zv0=</ds:DigestValue>
</ds:Reference>
```

L'extraction et le traitement de la référence correspondante a été décrite dans la section 8.4. Les explications ne sont pas reprises, seules les commandes à effectuer sont fournies.

```
$ xsltproc enveloped-signature.xslt data.xades-a.xml | xmllint --exc-c14n - \
> data.xades-a.Reference-1.bin
```

Strictement parlant, il faudrait canonicaliser la référence extraite, mais celle-ci ayant déjà été canonicalisée dans le cadre des transformations à appliquer (élément `ds:Transforms`), une canonicalisation supplémentaire n'aurait aucun impact.

En profiter pour vérifier que l'empreinte du fichier produit correspond à la valeur de l'élément `ds:DigestValue`, comme vu dans la section 8.4, page 99.

```
$ openssl sha256 -binary data.xades-a.Reference-1.bin | openssl base64
o00jEBylngTpuEsj0e+RkjwymUf5e+wiDE151+Z7Zv0=
```

Le deuxième élément `ds:Reference` à traiter, correspondant aux propriétés signées de la signature XAdES, est le suivant :

```
<ds:Reference URI="#Id_SignedProperties">
  <ds:Transforms>
```

```

    <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
  </ds:Transforms>
  <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmldsig#sha256" />
  <ds:DigestValue>qhZevqabDRuLtLVoiPDSsryPHAkzf4EFUYFyvvcblCk=</ds:DigestValue>
</ds:Reference>

```

Extraire et canonicaliser le nœud XML référencé, identifié par l'attribut Id de valeur Id_SignedProperties, et correspondant donc à l'expression XPath `//*[@Id='Id_SignedProperties']`.

```

$ xml sel -t -c //*[ @Id='Id_SignedProperties'] data.xades-a.xml \
| xmllint --exc-c14n - > data.xades-a.Reference-2.bin

```

Vérifier que l'empreinte SHA-256 du fichier obtenu concorde avec la valeur de l'élément ds:DigestValue.

```

$ openssl sha256 -binary data.xades-a.Reference-2.bin | openssl base64
qhZevqabDRuLtLVoiPDSsryPHAkzf4EFUYFyvvcblCk=

```

Les éléments à hacher pour constituer les données à horodater sont extraits séparément, et seront concaténés une fois tous extraits. Il est bien entendu possible de concaténer les fichiers au fur et à mesure en utilisant la redirection double `>> data.xades-a.ArchiveTimeStamp.data_to_hash.bin...` dont la contrepartie est d'être plus compliquée à démêler en cas d'erreur.

Ensuite, extraire et canonicaliser les éléments ds:SignedInfo, ds:SignatureValue et ds:KeyInfo.

```

$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# -t \
-c /root/ds:Signature/ds:SignedInfo data.xades-a.xml | xmllint --exc-c14n - \
> data.xades-a.SignedInfo.bin

```

```

$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# -t \
-c /root/ds:Signature/ds:SignatureValue data.xades-a.xml \
| xmllint --exc-c14n - > data.xades-a.SignatureValue.bin

```

```

$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# -t \
-c /root/ds:Signature/ds:KeyInfo data.xades-a.xml | xmllint --exc-c14n - \
> data.xades-a.KeyInfo.bin

```

Extraire et canonicaliser chacune des propriétés non signées (les chemins XPath sont à saisir sur une seule ligne).

Pour référence les propriétés non signées sont mises en évidence en gras ci-dessous :

```

root
  data
    ds:Signature
      ...
      ds:Object
        xades:QualifyingProperties
          ...
          xades:UnsignedProperties
            xades:UnsignedSignatureProperties
              xades:SignatureTimeStamp
              ds:CanonicalizationMethod

```

```

xades:EncapsulatedTimeStamp
xadesv141:TimeStampValidationData
xades:CertificateValues
  xades:EncapsulatedX509Certificate
xades:CertificateValues
  xades:EncapsulatedX509Certificate
xades:RevocationValues
  xades:CRLValues
  xades:EncapsulatedCRLValue

```

```

$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# \
-N xades=http://uri.etsi.org/01903/v1.3.2# -t \
-c /root/ds:Signature/ds:Object/xades:QualifyingProperties/xades:UnsignedPrope
rties/xades:UnsignedSignatureProperties/xades:SignatureTimeStamp \
data.xades-a.xml | xmllint --exc-c14n - | tr -d "\r" \
> data.xades-a.SignatureTimeStamp.bin

```

Avant d'extraire la propriété non signée suivante, `xadesv141:TimeStampValidationData`, une remarque s'impose : la commande `xml sel` supporte un maximum de deux options globales... or ce sont les options globales qui permettent de définir les espaces de nommage et leur préfixe, et les propriétés non signées d'une signature XAdES-A mettent en jeu trois espaces de nommage (représentés par les préfixes `ds`, `xades` et `xadesv141`).

La ligne en cause se situe dans le fichier `src/xml_select.c` du code source XMLStarlet :

```
#define TEMPLATE_OPT_MAX_ARGS 2
```

Une solution possible est de générer un fichier XSLT, inspiré par exemple de celle générée par la commande `xml sel -C`, pour chaque chemin XPath à extraire. Une méthode plus pratique serait de pouvoir passer le chemin XPath en tant que paramètre d'une transformation XSLT, mais cela n'est pas possible avec la version 1.0 de XSLT, qui est celle implémentée par `xsltproc`. Heureusement, `xsltproc` supporte des extensions EXSLT, au nombre desquelles la fonction `dyn:evaluate`⁷⁹ (`dyn` est le préfixe pour l'espace de nommage `http://exslt.org/dynamic`), qui évalue une chaîne de caractères en tant qu'expression XPath.

Créer le fichier XSLT `extractxpath.xslt` suivant pour exploiter cette fonctionnalité.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xades="http://uri.etsi.org/01903/v1.3.2#"
  xmlns:xadesv141="http://uri.etsi.org/01903/v1.4.1#"
  xmlns:dyn="http://exslt.org/dynamic"
  extension-element-prefixes="dyn">

  <xsl:output indent="no" omit-xml-declaration="yes"/>

  <xsl:param name="xpath"/>

  <xsl:template match="/">

```

79. <http://www.exslt.org/dyn/functions/evaluate/index.html>

```

    <xsl:call-template name="extractXPath"/>
</xsl:template>

<xsl:template name="extractXPath">
  <xsl:copy-of select="dyn:evaluate($xpath)"/>
</xsl:template>
</xsl:stylesheet>

```

Utiliser ce fichier XSLT pour extraire la propriété non signée `xadesv141:TimeStampValidationData`, puis canonicaliser le résultat.

```

$ xsltproc --stringparam xpath \
  /root/ds:Signature/ds:Object/xades:QualifyingProperties/xades:UnsignedProperti
es/xades:UnsignedSignatureProperties/xadesv141:TimeStampValidationData[1] \
  extractxpath.xslt data.xades-a.xml | xmllint --exc-c14n - \
  > data.xades-a.TimeStampValidationData.bin

```

Noter que l'index [1] de l'élément `xadesv141:TimeStampValidationData` est précisé, en prévision du fait que plusieurs de ces éléments peuvent figurer dans une signature au format XAdES-A.

Extraire les propriétés non signées suivantes.

```

$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# \
  -N xades=http://uri.etsi.org/01903/v1.3.2# -t -c \
  /root/ds:Signature/ds:Object/xades:QualifyingProperties/xades:UnsignedProperti
es/xades:UnsignedSignatureProperties/xades:CertificateValues data.xades-a.xml \
  | xmllint --exc-c14n - > data.xades-a.CertificateValues.bin

$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# \
  -N xades=http://uri.etsi.org/01903/v1.3.2# -t -c \
  /root/ds:Signature/ds:Object/xades:QualifyingProperties/xades:UnsignedProperti
es/xades:UnsignedSignatureProperties/xades:RevocationValues data.xades-a.xml \
  | xmllint --exc-c14n - > data.xades-a.RevocationValues.bin

```

La syntaxe alternative suivante peut évidemment être utilisée, avec le même résultat (comparer les empreintes pour s'en assurer) :

```

$ xsltproc --stringparam xpath \
  /root/ds:Signature/ds:Object/xades:QualifyingProperties/xades:UnsignedProperti
es/xades:UnsignedSignatureProperties/xades:CertificateValues extractxpath.xslt \
  data.xades-a.xml | xmllint --exc-c14n - > data.xades-a.CertificateValues.bin

$ xsltproc --stringparam xpath \
  /root/ds:Signature/ds:Object/xades:QualifyingProperties/xades:UnsignedProperti
es/xades:UnsignedSignatureProperties/xades:RevocationValues extractxpath.xslt \
  data.xades-a.xml | xmllint --exc-c14n - > data.xades-a.RevocationValues.bin

```

Si le lecteur utilise les mêmes données que celles fournies dans le présent document, alors il pourra vérifier qu'il obtient les mêmes nœuds extraits et canonicalisés en vérifiant que les empreintes des fichiers `.bin` sont identiques aux valeurs ci-dessous.

```

$ openssl sha256 data.xades-a.*.bin
SHA256(data.xades-a.CertificateValues.bin)= fc7d08ddd1b8608a31b01fa7efc93cf384f3
13c3fa64160738ad88e57b2164fa
SHA256(data.xades-a.KeyInfo.bin)= 86cec5e8080b1c7846f40e13d9f6266f511da2ffad0613

```

```
3d3a25262c6ae03b3c
SHA256(data.xades-a.Reference-1.bin)= a34d23101ca59e04e9b84b2339ef91923c329947f9
7bec220c4d79d7e67b66fd
SHA256(data.xades-a.Reference-2.bin)= aa165ebea69b0d1b8bb4b56888f0d2b2bc8f1c0933
7f8105518172bef71b2c29
SHA256(data.xades-a.RevocationValues.bin)= 4708fc3405af5249ab7238a94e47e701357f5
eb53ffc0d4ed2f3890f60e5fe2e
SHA256(data.xades-a.SignatureTimeStamp.bin)= 3207e444d25ad1566d0bc49c0580849f04c
927758bab1824cad759aee7341da6
SHA256(data.xades-a.SignatureValue.bin)= 297fbd03f006ed8f7651834cdf5b64a28529ad7
2097ff38d3e11c628e092b7e5
SHA256(data.xades-a.SignedInfo.bin)= e1c0377e51d7e438f47bec23aad4e8427113cee0975
dc7292be99720cd2ca64e
SHA256(data.xades-a.TimeStampValidationData.bin)= 092a5627111abf37cad6680744ec6a
61a9d616fec0bf8cc35772c8731f6f199c
```

Concaténer dans l'ordre les fichiers obtenus ci-dessus.

```
$ cat \
  data.xades-a.Reference-1.bin \
  data.xades-a.Reference-2.bin \
  data.xades-a.SignedInfo.bin \
  data.xades-a.SignatureValue.bin \
  data.xades-a.KeyInfo.bin \
  data.xades-a.SignatureTimeStamp.bin \
  data.xades-a.TimeStampValidationData.bin \
  data.xades-a.CertificateValues.bin \
  data.xades-a.RevocationValues.bin \
  > data.xades-a.ArchiveTimeStamp.data_to_hash
```

Le fichier `data.xades-a.ArchiveTimeStamp.data_to_hash` ainsi produit représente les données à horodater pour obtenir le jeton d'horodatage d'archivage.

L'empreinte de ce fichier est la suivante :

```
$ openssl sha256 data.xades-a.ArchiveTimeStamp.data_to_hash
SHA256(data.xades-a.ArchiveTimeStamp.data_to_hash)= d0b28d8d2f6c68038b904f82720b
8db139a7ecdd319060e502447504e7512ac9
```

Générer la requête d'horodatage associée à ce fichier.

```
$ openssl ts -query -data data.xades-a.ArchiveTimeStamp.data_to_hash \
  -sha256 -out ArchiveTimeStamp.tsq
```

Générer la réponse d'horodatage.

```
$ openssl ts -reply -config tsa-ts.cnf -section tsa \
  -queryfile ArchiveTimeStamp.tsq -inkey tsa-key.pem -signer tsa-crt.pem \
  -out ArchiveTimeStamp.tsr
```

Extraire le jeton d'horodatage de cette réponse.

```
$ openssl ts -reply -in ArchiveTimeStamp.tsr -token_out \
  -out ArchiveTimeStamp.tst
```

Coder le jeton en Base64.

```
$ openssl base64 -in ArchiveTimeStamp.tst
```

```
MIIC1AYJKoZIhvcNAQcCoIICxTCCAsECAQMxCzAJBgUrDgMCGGUAMIGOBgsqhkiG
9w0BCRABBKB/BH0wewIBAqYkKoZIhvcUAb5Ak3qC8U+DpxmD4GyBn36CuiCEgKIr
hZ3WYgEFMDEwDQYJYIZIAWUDBAIBBQAEINCyY0vbGgDi5BPgnILjbE5p+zdMZBg
5QJEdQTnUSrJAgEEGA8yMDEyMDYyNDEOMTQOMFoCCQCUR+EKI6ZSfjGCAhwwggIY
AgEBMGYwWTELMakGA1UEBhMCRlIxFzAVBgNVBAoTDk1vbiBFbnRyZXByaXNlMRcw
FQYDVQQLew4wMDAyIDEyMzQ1Njc4OTEYMBYGA1UECzMPT3B1b1NTTCBSb290IENB
AgkA071PBN3RoEAWCQYFKw4DAh0FAKCBjDAaBgkqhkiG9w0BCQMxDQYLKoZIhvcN
AQkQAQQwHAYJKoZIhvcNAQkFMQ8XDTEyMDYyNDEOMTQOMFowIwYJKoZIhvcNAQkE
MRYEFM0ESu4Vi8DU46XrB799sbNVJcUcMCsGCyqGSib3DQEJEAIMMRwwGjAYMBYE
FEMopucZySTSxBTCSucZUJOVYcUJMAOGCSqGSib3DQEBAQUABIIBALbda1Pt/26K
RRkA1ZKzndxhjp3620N7NpSDpe9IEr6gdtiqLXhd5ktOjUKWSQnXdHaEsMaTOP08
0Ac8X+YguiQSTqdkLnNLelyKPf++zM140rvUKvM/nfCwSfk1vVk0C6h6h4JbYTka
8D/GIub3HopG/tWzCxe/UoTvcv2GhIUsHDw11gHlXYcCzIS8UILI3n89as3eW0d4
Q4PCUEKA15RdSForXhP7DQ6B/cR/PjGnpqUy2BDjDqS0ujj57ToAngMqmKMwV4pi
88zB56NGqvcPmKERgAIR1+iD+10vAOSJUfJl7lUMmlJMcwYNIYp78RaZx+hx6+H
jLo+W++5i7g=
```

Ajouter l'élément `xadesv141:ArchiveTimeStamp` suivant dans la signature en tant que propriété non signée, sous l'élément `xades:UnsignedProperties` dans le fichier `data.xades-a.xml`, en reprenant la valeur en Base64 obtenue ci-dessus.

```
<xadesv141:ArchiveTimeStamp Id="Id_ArchiveTimeStamp">
  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
  <xades:EncapsulatedTimeStamp>
MIIC1AYJKoZIhvcNAQcCoIICxTCCAsECAQMxCzAJBgUrDgMCGGUAMIGOBgsqhkiG
...
jLo+W++5i7g=
  </xades:EncapsulatedTimeStamp>
</xadesv141:ArchiveTimeStamp>
```

Vérifier que la signature est toujours valide.

Afin de vérifier que l'élément `xades:QualifyingProperties` est conforme au schéma XML de la version 1.4.2 de XAdES, télécharger d'abord le fichier du schéma XML correspondant⁸⁰, puis lancer la commande suivante.

```
$ xml sel -N ds=http://www.w3.org/2000/09/xmldsig# \
-N xades=http://uri.etsi.org/01903/v1.3.2# -t \
-c /root/ds:Signature/ds:Object/xades:QualifyingProperties data.xades-a.xml \
| xmllint --schema XAdESv141.xsd --noout -
- validates
```

80. <http://uri.etsi.org/01903/v1.4.1/XAdESv141.xsd>

Ajout des données de révocation du jeton d'horodatage d'archivage

Afin de permettre de vérifier que le certificat de l'unité d'horodatage est valide de manière autonome à partir de la signature, il est judicieux d'inclure la liste de certificats révoqués émise par son autorité de certification dans celle-ci.

Mettre à jour la liste de certificats révoqués.

```
$ openssl ca -gencrl -cert ca-crt.pem -keyfile ca-key.pem -crlhours 48 \
  -md sha256 -config ca-crl.cnf -name ca_crl -crlexts ca_crl_ext \
  -out ca-crl.pem
```

Ajouter le codage en Base64 de la liste de certificats révoqués dans une nouvelle propriété non signée `xadesv141:TimeStampValidationData`, après le jeton d'horodatage d'archivage, et référant celui-ci à l'aide de l'attribut `URI` (dont la valeur reprend l'attribut `Id` de l'élément `xadesv141:ArchiveTimeStamp`), comme ci-dessous en gras :

```
...
  <xades:UnsignedProperties>
    <xades:UnsignedSignatureProperties>
      <xades:SignatureTimeStamp>
        ...
      </xades:SignatureTimeStamp>
      <xadesv141:TimeStampValidationData URI="#Id_SignatureTimeStamp">
        ...
      </xadesv141:TimeStampValidationData>
      <xades:CertificateValues>
        ...
      </xades:CertificateValues>
      <xades:RevocationValues>
        ...
      </xades:RevocationValues>
      <xadesv141:ArchiveTimeStamp Id="Id_ArchiveTimeStamp">
        ...
      </xadesv141:ArchiveTimeStamp>
      <xadesv141:TimeStampValidationData URI="#Id_ArchiveTimeStamp">
        <xades:RevocationValues>
          <xades:CRLValues>
            <xades:EncapsulatedCRLValue>
MIIB0zCBvAIBATANBgkqhkiG9w0BAQsFADBZMQswCQYDVQQGEwJGUjEXMBUGA1UE
ChMOTW9uIEVudHJlcHJpc2UxFzAVBgNVBAsTDjAwMDIgMTIzNDU2Nzg5MRgwFgYD
VQQLew9PcGVuU1NMFJvbn3QgQ0EXDTEyMDYyNDEOMzQxM1oXDTEyMDYyNjE0MzQx
M1qgLzAtMB8GA1UdIwQYMBaAFAExth5OC9y0sByOiD+BxLRc/OfOPMAoGA1UdFAQD
AgEEMAOGCSqGSIb3DQEBCwUAA4IBAQCZCReBWo/EW9ItxK9A8gp4hj5YiV11cODI
WE8D5ubrPzUJufuSU9VgiuLvOGWrq6SdPDJ9umq1SWjVhJ2HFKSokQJa2wHvyfC
xtqDWHOtCBasbTbMmVOGB8w4dZWjlaHuBkeRjffU92SbOK1e397cu5rH/EA9fcYfx
K1mivI3xCn008fA8n49MfigG4dws+Nz4YveCw76AuI6CEwcZQa5y6BJLBAFweul2
MpU1A75kEdqYHoJgW1KbVEjXFYVnTvH0UTPCXKIFf4e4MjyIrH2fzI9drZqZ9ypr
tdGvi51hMEnvFa0xb8yCfzdkFf6H7Sm9CtY3+Ag5cxs8mHnRf09m
            </xades:EncapsulatedCRLValue>
          </xades:CRLValues>
        </xades:RevocationValues>
      </xadesv141:TimeStampValidationData>
    </xades:UnsignedSignatureProperties>
  </xades:UnsignedProperties>
...

```



```
        </xades:RevocationValues>
      </xadesv141:TimeStampValidationData>
    </xades:UnsignedSignatureProperties>
  </xades:UnsignedProperties>
```

...

Sauvegarder le fichier `data.xades-a.xml`, ce qui achève la constitution de la signature électronique au format XAdES-A.

Annexe A — Représentation et codage des données

A.1. Base64

Un des codages fréquemment employé pour représenter des données binaires, notamment dans le domaine de la confiance électronique, est Base64. Le codage Base64 code des groupes de trois octets (soit 24 bits) sous la forme de quatre caractères, chaque caractère pouvant prendre 64 valeurs (6 bits) possibles (24 bits = 4 caractères × 6 bits). Ces caractères sont les chiffres, les majuscules et minuscules ASCII, ainsi que les caractères « + », « / » et (pour le *padding* dans le cas où le dernier groupe d'octets à coder comporte moins de trois octets) « = ». La commande `openssl base64` permet de coder et (avec l'option `-d`) de décoder des données en Base64. Par exemple, l'empreinte SHA-256 d'un fichier, codée en Base64, peut être obtenue ainsi :

```
$ openssl sha256 -binary data.txt | openssl base64
ib2SKG1sgBTAYDCyX4tAzB1WVtSzt7SDGHT1DW9VV/M=
```

Exercice — Sachant qu'une empreinte SHA-256 a une taille de 256 bits, quelle est la longueur des chaînes de caractères de ses représentations hexadécimale et Base64 ?

A.2. PEM

Le codage Base64, classiquement utilisé pour limiter les problèmes de support des encodages lors du transport des données binaire, est parfois complété par des balises PEM (*privacy enhanced mail*, du nom de la RFC 1421 qui les définit) : `-----BEGIN type-----` avant les données et `-----END type-----` après, où *type* désigne le type de données (ex. : `CERTIFICATE`, `X509 CRL`), les noms des types étant plutôt des conventions que des standard. Le nom de codage PEM est parfois donné au codage Base64 (qu'il soit complété par les balises PEM, ou — par abus de langage — non).

A.3. Distinguished Name

De manière simplifiée par rapport à la norme X.501 (mais réaliste), la syntaxe ASN.1 d'un DN (*Distinguished Name*, ou nom distingué s'il faut vraiment traduire ce terme, qui désigne simplement l'identifiant unique d'une entrée d'annuaire) est la suivante (issue de la RFC 4514) :

```
DistinguishedName ::= RDNSequence
```

```
RDNSequence ::= SEQUENCE OF RelativeDistinguishedName
```

```
RelativeDistinguishedName ::= SET SIZE (1..MAX) OF
```

AttributeTypeAndValue

```
AttributeTypeAndValue ::= SEQUENCE {  
    type AttributeType,  
    value AttributeValue }
```

L'affichage de l'ordre des RDN (*Relative Distinguished Name* ou noms distingués relatifs, les composants du DN) diffère selon l'outil utilisé. Par défaut, OpenSSL affiche les RDN d'un DN dans l'ordre dans lequel ils figurent dans la structure ASN.1, correspondant à la succession de nœuds X.501 décrivant le nœud cible en partant du nœud racine (le séparateur étant « \ », soit encore une description partant du général vers le particulier, où l'identité de l'entité est dans le dernier RDN affiché. Ainsi, pour la CSR de l'autorité de certification, l'affichage par défaut d'OpenSSL est :

```
$ openssl req -in ca-req.pem -noout -subject  
subject=/C=FR/O=Mon Entreprise/OU=0002 123456789/OU=OpenSSL Root CA
```

Or l'affichage défini par, entre autres, les RFC 2253 puis 4514 sur la représentation des DN de LDAP, prescrit de partir du dernier RDN et de remonter vers le premier (avec le séparateur « , »), du particulier vers le général, à l'image d'une adresse postale « classique » (hors adresses postales est-asiatiques, s'entend). C'est notamment l'affichage retenu par le magasin de certificats de Windows. OpenSSL peut afficher les DN dans l'ordre prévu par les RFC en utilisant l'option `-nameopt RFC2253` des commandes qui la supportent (`req` et `x509`) :

```
$ openssl req -in ca-req.pem -noout -subject -nameopt RFC2253  
subject=OU=OpenSSL Root CA,OU=0002 123456789,O=Mon Entreprise,C=FR
```

A.4. Codage des caractères

Le problème de codage des caractères est théoriquement simple à résoudre (il « suffit » que tout le monde utilise, par exemple, le codage UTF-8 des caractères Unicode), mais est en pratique un nid de guêpes, compte tenu de la diversité des codages de caractères par défaut mis en œuvre dans les systèmes exploitation et applications. Pour faire simple, ASCII, la *lingua franca* originelle des normes de codage de caractères, supporte les minuscules et majuscules non accentuées, les chiffres et une trentaine de symboles courants, et a été étendue par des éditeurs de logiciels et autres organismes pour supporter des caractères supplémentaires. L'utilisateur francophone/francophile creusant la problématique du codage des caractères est rapidement confronté aux codages suivants :

- ISO-8859-1, ou Latin-1, ou encore *Western European* (européen occidental), qui supporte entre autres les caractères accentués français, et qui est particulièrement utilisé pour les pages web et systèmes d'exploitation (francophones ou non, d'ailleurs), ainsi que pour les fichiers .properties de Java.
- ISO-8859-15, ou Latin-9, qui enrichit Latin-1 notamment du symbole « € », des ligatures « œ » et « Œ » et la lettre « Ÿ », pour une couverture exhaustive des caractères de la langue française. Il remplace progressivement ISO-8859-1 pour ceux qui ne migrent pas vers UTF-8, mais les soucis résiduels d'affichage des symboles « € » et des « œ » copiés/collés depuis les logiciels de

bureautique dans certains formulaires web laissent penser que la transition est plus longue que prévu.

- Windows-1252, ou CP1252 (parfois appelé ANSI par abus de langage), le codage par défaut en environnement Windows, qui est fonctionnellement équivalent à ISO-8859-15, mais techniquement incompatible (en particulier pour les caractères ajoutés dans ISO-8859-15).
- OEM 850, ou *code page* 850 (page de code 850) ou encore MS-DOS Latin-1, codage par défaut sous DOS (dont l'invite de commandes Windows), qui est fonctionnellement équivalent à ISO-8859-1, mais techniquement incompatible, en particulier pour tous les caractères accentués français.
- UTF-8, qui permet de coder, sur un (pour les caractères ASCII) ou plusieurs octets (pour tous les autres caractères), l'ensemble des caractères du jeu de caractères Unicode (plus d'un million de caractères). Malgré quelques inconvénients liés au codage des caractères non ASCII sur plusieurs octets, UTF-8 a pour vocation d'être la meilleure pratique pour le codage des caractères.

Pour se faire une idée du type de problème lié au codage des caractères, sous Windows, essayer d'afficher dans une invite de commandes Windows un fichier texte français saisi dans le bloc-notes : les caractères accentués s'affichent incorrectement car leur codage est différent entre Windows-1252 (codage par défaut du bloc-notes) et OEM 850 (DOS). Une solution possible est de sauvegarder le fichier au format UTF-8, et de changer de page de code sous DOS : la page de code par défaut porte le numéro 850 (utiliser la commande `chcp` seule pour obtenir la page de code courante), et la page de code Microsoft correspondant à UTF-8 est 65001 (saisir `chcp 65001` pour passer à la page de code 65001).

Autre expérience sous Windows : par défaut l'accès distant (telnet ou SSH) à l'aide du client PuTTY prévoit que les données reçues sont codées en ISO-8859-1, d'où des comportements inhabituels à prévoir si l'environnement cible est en UTF-8 (utiliser la commande `locale` pour connaître le codage en cours) et s'il faut par exemple saisir le caractère « € ». L'option du PuTTY à régler dans ce cas est dans Window > Translation > Received data assumed to be in which character set > UTF-8.

L'exigence initiale de la RFC 3280 de coder systématiquement les `DirectoryString` sous la forme d'`UTF8String` à partir de décembre 2003 a été assouplie dans la RFC 4630, qui – suite aux premières années de retour d'expérience – lève l'exigence et semble inciter à demi-mot, pour éviter les problèmes, à utiliser les `UTF8String` uniquement quand il n'est pas possible de faire autrement.

Malheureusement, il n'existe aucune solution miracle : sauf cas où le codage est imposé (et encore...), vérifier le codage mis en œuvre (que ce soit dans une invite de commande, un fichier texte, un formulaire web, etc.) avant d'utiliser un caractère non ASCII ne garantit pas qu'une transformation ultérieure ne corrompra pas le codage, et quand elles existent, les solutions pour y remédier sont spécifiques à chaque cas rencontré. À toutes fins utiles, pour des fichiers texte, il est recommandé de connaître l'outil `iconv`, qui permet de convertir du texte d'un codage à un autre. Cet outil est disponible pour les systèmes UNIX et GNU/Linux, et la version GNU d'`iconv` a été portée⁸¹ sous Windows dans la suite GnuWin32 (elle se compile par ailleurs très facilement avec MinGW et MSYS⁸²).

81. <http://gnuwin32.sourceforge.net/packages/libiconv.htm>

82. <http://www.mingw.org/>

Annexe B — ASN.1

B.1. Distinguished Encoding Rules

DER (*Distinguished Encoding Rules*) désigne un ensemble de règles de représentation binaire, qui est notamment utilisé pour coder la plupart des structures de données binaires (par opposition à XML) dans le domaine de la confiance électronique (ex. : certificats, listes de certificats révoqués, les structures de données définies par les PKCS#). Le « schéma » (par analogie à XML Schema dans le monde XML) de ces structures de données est défini en ASN.1 (*Abstract Syntax Notation One*). La passion pour ASN.1 nourrit une population à vrai dire plutôt confidentielle, y compris dans le domaine de la confiance électronique où ASN.1 constitue pourtant un socle technique fondamental aux côtés de XML. Le lecteur souhaitant aiguïser son appétit pourra lire la section ci-après intitulée MII... avant de se plonger dans les normes et les quelques ouvrages⁸³ sur le sujet.

MI I...

Le lecteur attentif notera que le codage Base64 des petites structures de données binaires élémentaires (ex. : certificats, listes de certificats révoqués de test, signatures PKCS#7 détachées) commence quasi systématiquement par les caractères MI I..., c'est-à-dire les valeurs décimales 12 08 08... du tableau de correspondance Base64, soit encore les valeurs binaires sur 6 bits 0b001100 0b001000 0b001000.... Le décodage Base64 donne, en regroupant ces bits par groupes de huit, 0b00110000 0b10000010 0b00..., soit 0x30 0x82 n (où n est un nombre inférieur à 0x40). Dans le codage DER, ces octets représentent :

- Premier octet (0x30) : le type de données. Il s'agit d'une SEQUENCE ASN.1 (équivalent à une sequence de XML Schema).
- Deuxième octet (0x82) : la longueur des données constituant le contenu de la structure (en général si elle est inférieure ou égale à 127 octets) ou la longueur de la longueur du contenu si elle est supérieure à 127 octets (bit de poids fort égal à 1 et longueur de la longueur sur les 7 bits de poids faible). Ici, la longueur du contenu est supérieure à 127 octets (le bit de poids fort 0b1...) et doit donc être représentée sur deux octets (les 7 bits 0b0000010).
- Les deux octets suivants (0b00... 0b...) représentent la longueur du contenu, c'est-à-dire un nombre inférieur à 0x4000 (16 384 en décimal).

En pratique, les petites structures élémentaires sont effectivement des séquences de taille comprise entre 128 octets et 16 Ko, d'où le MI I... classique des fichiers PEM.

83. <http://www.oss.com/asn1/resources/books-whitepapers-pubs/asn1-books.html>

B.2. Génération d'un fichier de configuration ASN.1 pour OpenSSL

Il est envisageable d'automatiser la production des fichiers interprétables par la commande `openssl asn1parse -genconf` à partir d'un fichier DER, en utilisant un outil de décodage ASN.1 pour obtenir la structure des données puis en générant les sections de fichier attendues en parcourant cette structure.

Voici quelques pistes gratuites à envisager :

- le module Perl `Convert::ASN1`⁸⁴, dont le script d'exemple `x509decode` dans le paquetage source est un bon point de départ,
- la bibliothèque Python `pyasn1`⁸⁵, qui propose des modules prédéfinis⁸⁶ pour les structures usuelles du domaine de la confiance numérique,
- la bibliothèque C GNU `libtasn1`⁸⁷, utilisée notamment par `GnuTLS`⁸⁸, l'implémentation GNU du protocole TLS,
- le paquetage Java `CODEC`⁸⁹, qui propose des paquetages correspondant aux structures définies par les normes PKCS et X.500,
- la combinaison de `asn1c`⁹⁰ pour convertir la structure DER en XER, et d'une transformation XSLT.
- les versions Java et C# de la bibliothèque cryptographique `BouncyCastle`⁹¹ proposent des objets permettant de gérer des structures ASN.1.
- les fonctions ASN.1 de la bibliothèque `libcrypto` d'OpenSSL.

Les solutions permettant de compiler des modules ASN.1 sont susceptibles de produire les fichiers les plus lisibles, car elles peuvent faire correspondre une représentation DER à la structure ASN.1, et ainsi de générer des noms de sections représentatifs (tels que `[certificate]`). Malheureusement, les compilateurs ASN.1 implémentés par la plupart des outils sont imparfaits, et il faut donc adapter les modules normatifs pour qu'ils soient supportés par ces compilateurs, ce qui représente un travail fastidieux.

Une solution plus grossière mais plus rapide consiste à utiliser l'outil en ligne de commande `unber` d'`asn1c` (cf. annexe B.4) pour produire une représentation XML générique de la structure, qui peut ensuite servir de point d'entrée pour obtenir un fichier de configuration ASN.1 (peu lisible mais utilisable).

84. <http://search.cpan.org/~gbarr/Convert-ASN1/>

85. <http://pyasn1.sourceforge.net>

86. <http://sourceforge.net/projects/pyasn1/files/pyasn1-modules/>

87. <http://www.gnu.org/software/libtasn1/>

88. <https://www.gnu.org/software/gnutls/>

89. <http://codec.sourceforge.net/>

90. <http://lionet.info/asn1c/blog/>

91. <http://www.bouncycastle.org>

Le résultat de la commande `unber` produit un fichier ressemblant à du XML à première vue, mais il présente trois caractéristiques qui l'empêchent d'être document XML valide, et qui imposent un nettoyage (l'outil `sed`) avant de pouvoir être interprété par les outils XML usuels :

- Les valeurs hexadécimales sont représentées par des suites d'entités XML (par exemple `` pour la valeur `0x02`). Or⁹², les caractères de contrôle ne sont pas autorisés dans un document XML. Il a donc été choisi d'échapper les valeurs hexadécimales des entités à l'aide du préfixe `\x` (par exemple, `` est remplacé par `\x02`), en utilisant l'expression régulière compatible `sed` suivante : `s/&#x\(. . \); /\x\1/g`... mais cette solution est incomplète, car dans le cas d'une chaîne mélangeant caractères ASCII et entités XML, le caractère `\` peut être ambigu. Il convient donc de précéder l'expression régulière précédente de l'expression régulière `s/\\/\\\\/g` pour échapper les caractères `\` et les remplacer par `\\`. Des *templates* dans le fichier XSLT gèrent les conversions inverses.
- Certaines balises (celles « mises en forme » d'après la documentation) contiennent un attribut orphelin `F` sans valeur juste avant le caractère « `>` » de fin de balise (par exemple : `<P O="10" T="[UNIVERSAL 2]" TL="2" V="1" A="INTEGER" F>2</P>`, ce qui enfreint⁹³ les règles de la grammaire de XML. Cet attribut peut être supprimé par l'expression régulière `s/<\(. *\)\ F>/<\1>/g`. À noter que les quantificateurs de `sed` sont obligatoirement gourmands, mais cela ne pose pas de problème dans la présente situation car une ligne contient une seule balise, donc le `\(. *\)` ne risque pas d'englober plusieurs balises par mégarde.
- Les balises fermantes incluent les mêmes attributs que les balises ouvrantes, ce qui est interdit dans un document XML⁹⁴. L'expression régulière `s/<\/\([^]\) .*>/<\/\1>/g` conserve uniquement le nom de la balise dans les balises fermantes.

Créer le fichier `unberclean.sed` suivant, reprenant les expressions régulières ci-dessus :

```
s/\\/\\\\/g
s/&#x\(. . \); /\x\1/g
s/<\(. *\)\ F>/<\1>/g
s/<\/\([^ ]\) .*>/<\/\1>/g
```

Décoder le fichier d'entrée `ee-Certificate.der` généré en fin de section 4.7 avec `unber` et nettoyer le résultat, à l'aide de la ligne de commande suivante :

```
$ unber ee-Certificate.der | sed -f unberclean.sed > ee-Certificate.xml
```

Créer le fichier XSLT `unber2asnconf.xslt` ci-après.

■ Quelques commentaires ont été inclus pour expliquer les transformations effectuées.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" encoding="UTF-8" indent="no"/>

  <xsl:param name="prefix"/>
```

92. <http://www.w3.org/TR/2004/REC-xml-20040204/#NT-Char>

93. <http://www.w3.org/TR/2004/REC-xml-20040204/#NT-Eq>

94. <http://www.w3.org/TR/2004/REC-xml-20040204/#NT-ETag>

```

<!-- Top-level template: creates the initial "asn1 = ..." line, and
      then processes the C element if it exists -->
<xsl:template match="/">
  <xsl:text>asn1 = </xsl:text>
  <xsl:apply-templates select="." mode="field_value"/>
  <xsl:text>&#x0a;</xsl:text>
  <xsl:if test="C">
    <xsl:text>&#x0a;</xsl:text>
    <xsl:apply-templates select="C" mode="section"/>
  </xsl:if>
</xsl:template>

<!--
  Templates for ASN.1 primitives
  =====
-->

<!-- Renders "field_name = field_value" from a P node. -->
<xsl:template match="P">
  <xsl:apply-templates select="." mode="field_name"/>
  <xsl:text> = </xsl:text>
  <xsl:apply-templates select="." mode="field_value"/>
  <xsl:text>&#x0a;</xsl:text>
</xsl:template>

<!-- Generates a field name from a P node, using @T to determine
      the type of field and @O to give it a unique number.
      If the global $prefix parameter is defined, then it is prepended
      with an '_' to the field name. -->
<xsl:template match="P" mode="field_name">
  <xsl:if test="$prefix">
    <xsl:value-of select="$prefix"/>
    <xsl:text>_</xsl:text>
  </xsl:if>

  <xsl:choose>
    <!-- INTEGER -->
    <xsl:when test="@T=' [UNIVERSAL 2] '">
      <xsl:text>int</xsl:text>
    </xsl:when>

    <!-- ENUMERATED -->
    <xsl:when test="@T=' [UNIVERSAL 10] '">
      <xsl:text>enum</xsl:text>
    </xsl:when>

    <!-- OBJECT IDENTIFIER -->
    <xsl:when test="@T=' [UNIVERSAL 6] '">
      <xsl:text>oid</xsl:text>
    </xsl:when>
  </xsl:choose>
</xsl:template>

```

```

<!-- NULL -->
<xsl:when test="@T=' [UNIVERSAL 5] '">
  <xsl:text>null</xsl:text>
</xsl:when>

<!-- BIT STRING -->
<xsl:when test="@T=' [UNIVERSAL 3] '">
  <xsl:text>bitstring</xsl:text>
</xsl:when>

<!-- OCTET STRING -->
<xsl:when test="@T=' [UNIVERSAL 4] '">
  <xsl:text>octstring</xsl:text>
</xsl:when>

<!-- PRINTABLE STRING -->
<xsl:when test="@T=' [UNIVERSAL 19] '">
  <xsl:text>printablestring</xsl:text>
</xsl:when>

<!-- UTF8 STRING -->
<xsl:when test="@T=' [UNIVERSAL 12] '">
  <xsl:text>utf8string</xsl:text>
</xsl:when>

<!-- UTCTIME -->
<xsl:when test="@T=' [UNIVERSAL 23] '">
  <xsl:text>utctime</xsl:text>
</xsl:when>

<!-- GENERALIZEDTIME -->
<xsl:when test="@T=' [UNIVERSAL 24] '">
  <xsl:text>gentime</xsl:text>
</xsl:when>

<!-- BOOLEAN -->
<xsl:when test="@T=' [UNIVERSAL 1] '">
  <xsl:text>bool</xsl:text>
</xsl:when>

<!-- IMPLICIT -->
<xsl:when test
  ="string(number(substring-before(substring-after(@T, '['), ']')) != 'NaN')">
  <xsl:text>implicit</xsl:text>
</xsl:when>

<xsl:otherwise>
  <xsl:text>unknown</xsl:text>
</xsl:otherwise>
</xsl:choose>

```

```

    <xsl:text>_</xsl:text>
    <xsl:value-of select="@0"/>
</xsl:template>

<!-- Generates a field value from a P node, using @T to determine
the type of field. -->
<xsl:template match="P" mode="field_value">
  <xsl:choose>
    <!-- INTEGER and ENUMERATED-->
    <xsl:when test="@T=' [UNIVERSAL 2] ' or @T=' [UNIVERSAL 10] '">
      <xsl:choose>
        <xsl:when test="@T=' [UNIVERSAL 2] '">
          <xsl:text>INTEGER:</xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>ENUMERATED:</xsl:text>
        </xsl:otherwise>
      </xsl:choose>
    <xsl:choose>
      <xsl:when test="starts-with(., '\x')">
        <!-- if the string begins with "\x" then it is a
        hexstring and needs to be unescaped and prefixed -->
        <xsl:text>0x</xsl:text>
        <xsl:call-template name="xstring_to_hexstring">
          <xsl:with-param name="string" select="."/>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="."/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:when>

  <!-- OBJECT IDENTIFIER -->
  <xsl:when test="@T=' [UNIVERSAL 6] '">
    <xsl:text>OID:</xsl:text>
    <xsl:value-of select="."/>
  </xsl:when>

  <!-- NULL -->
  <xsl:when test="@T=' [UNIVERSAL 5] '">
    <xsl:text>NULL</xsl:text>
  </xsl:when>

  <!-- BIT STRING -->
  <xsl:when test="@T=' [UNIVERSAL 3] '">
    <!-- Two cases are considered depending on the first byte (unused
    bits) -->
    <xsl:choose>
      <xsl:when test="starts-with(., '\x00')">
        <!-- 1) the number of unused bits is 0: discard first byte

```

```

        and render as hex-string -->
<xsl:text>FORMAT:HEX,BITSTRING:</xsl:text>
<xsl:call-template name="fold_long_line">
  <xsl:with-param name="line">
    <xsl:call-template name="xstring_to_hexstring">
      <xsl:with-param name="string" select="substring-after(.,'\x00')"/>
    </xsl:call-template>
  </xsl:with-param>
</xsl:call-template>
</xsl:when>
<xsl:otherwise>
  <!-- 2) the number of unused bits is not zero: render as
    bitlist -->
  <xsl:text>FORMAT:BITLIST,BITSTRING:</xsl:text>
  <xsl:call-template name="bitstring_to_bitlist">
    <!-- the number of unused bits is in the range 1-7
      (i.e. \x01-\x04), so extract fourth character for
      unused_bits parameter -->
    <xsl:with-param name="unused_bits" select="substring(.,4,1)"/>
    <xsl:with-param name="bits">
      <xsl:call-template name="xstring_to_binary">
        <!-- convert all but first byte to binary -->
        <xsl:with-param name="string" select="substring(.,5)"/>
      </xsl:call-template>
    </xsl:with-param>
  </xsl:call-template>
</xsl:otherwise>
</xsl:choose>
</xsl:when>

<!-- OCTET STRING -->
<xsl:when test="@T=' [UNIVERSAL 4] '">
  <xsl:text>FORMAT:HEX,OCTETSTRING:</xsl:text>
  <xsl:call-template name="fold_long_line">
    <xsl:with-param name="line">
      <xsl:call-template name="xstring_to_hexstring">
        <xsl:with-param name="string" select="."/>
      </xsl:call-template>
    </xsl:with-param>
  </xsl:call-template>
</xsl:when>

<!-- PRINTABLE STRING -->
<xsl:when test="@T=' [UNIVERSAL 19] '">
  <xsl:text>PRINTABLESTRING:</xsl:text>
  <xsl:value-of select="."/>
</xsl:when>

<!-- UTF8 STRING -->
<xsl:when test="@T=' [UNIVERSAL 12] '">
  <xsl:text>FORMAT:UTF8,UTF8String:</xsl:text>

```

```

    <xsl:value-of select="."/>
</xsl:when>

<!-- IA5STRING -->
<xsl:when test="@T=' [UNIVERSAL 19] '">
    <xsl:text>IA5STRING:</xsl:text>
    <xsl:value-of select="."/>
</xsl:when>

<!-- UTCTIME -->
<xsl:when test="@T=' [UNIVERSAL 23] '">
    <xsl:text>UTCTIME:</xsl:text>
    <xsl:value-of select="."/>
</xsl:when>

<!-- GENERALIZEDTIME -->
<xsl:when test="@T=' [UNIVERSAL 24] '">
    <xsl:text>GENERALIZEDTIME:</xsl:text>
    <xsl:value-of select="."/>
</xsl:when>

<!-- BOOLEAN -->
<xsl:when test="@T=' [UNIVERSAL 1] '">
    <xsl:text>BOOLEAN:</xsl:text>
    <xsl:choose>
        <xsl:when test=". = true">
            <xsl:text>>true</xsl:text>
        </xsl:when>
        <xsl:otherwise>
            <xsl:text>>false</xsl:text>
        </xsl:otherwise>
    </xsl:choose>
</xsl:when>

<!-- IMPLICIT -->
<!-- The test below checks whether the string between the
brackets - which defines the tag type - is a number, by
checking that it is not NaN (NaN: not a number) -->
<xsl:when test="string(number(substring-before(substring-after(@T, '['), ']')) != 'NaN')">
    <xsl:text>IMPLICIT:</xsl:text>
    <xsl:value-of select="substring-before(substring-after(@T, '['), ')]')"/>
    <!-- IMPLICIT-ly tagged types may be rendered by unber as
entities only or as a hybrid (ASCII+entities) string -->

    <!-- get number of \x-prefixed hex values -->
    <xsl:variable name="number_of_hexvalues">
        <xsl:call-template name="count_hexvalues">
            <xsl:with-param name="string" select="."/>
        </xsl:call-template>
    </xsl:variable>

```

```

<xsl:choose>
  <!-- if the string only contains hexvalues then render it
        as an OCTET STRING -->
  <xsl:when test="$number_of_hexvalues * 4 = string-length(.)">
    <xsl:text>,FORMAT:HEX,OCTETSTRING:</xsl:text>
    <xsl:call-template name="fold_long_line">
      <xsl:with-param name="line">
        <xsl:call-template name="xstring_to_hexstring">
          <xsl:with-param name="string" select="."/>
        </xsl:call-template>
      </xsl:with-param>
    </xsl:call-template>
  </xsl:when>

  <xsl:otherwise>
    <!-- otherwise render it as a UTF8STRING, leaving the
          hex values escaped -->
    <xsl:text>,FORMAT:UTF8,UTF8String:</xsl:text>
    <xsl:value-of select="."/>
  </xsl:otherwise>
</xsl:choose>
<!-- unber always renders IMPLICIT-ly tagged primitive types as
      hexadecimal strings, so an OCTET STRING type is used -->
</xsl:when>

<!-- Fallback: if this happens then this XSLT document needs to
      be extended to cover the unexpected case -->
<xsl:otherwise>
  <xsl:value-of select="@T"/>
  <xsl:text>:</xsl:text>
  <xsl:value-of select="."/>
</xsl:otherwise>

</xsl:choose>
</xsl:template>

<!--
  Templates for ASN.1 constructed types
  =====
-->

<!-- Generates a section, corresponding to e.g. a SEQUENCE or a
      SET -->
<xsl:template match="C" mode="section">
  <!-- section name -->
  <xsl:text>[</xsl:text>
  <xsl:apply-templates select="." mode="field_name"/>
  <xsl:text>]</xsl:text>
  <xsl:text>&#x0a;</xsl:text>

  <!-- section contents -->

```

```

<xsl:apply-templates select="C|P"/>
<xsl:text>&#x0a;</xsl:text>

<!-- recurse to create the sections required by the fields in
the current section -->
<xsl:for-each select="C">
  <xsl:apply-templates select="." mode="section"/>
</xsl:for-each>
</xsl:template>

<!-- Renders "field_name = field_value" from a C node. -->
<xsl:template match="C">
  <xsl:apply-templates select="." mode="field_name"/>
  <xsl:text> = </xsl:text>
  <xsl:apply-templates select="." mode="field_value"/>
  <xsl:text>&#x0a;</xsl:text>
</xsl:template>

<!-- Generates a field value from a C node, using @T to determine
the type of field. -->
<xsl:template match="C" mode="field_value">
  <xsl:choose>
    <!-- SEQUENCE -->
    <xsl:when test="@T=' [UNIVERSAL 16] '">
      <xsl:text>SEQUENCE:</xsl:text>
      <xsl:apply-templates select="." mode="field_name"/>
    </xsl:when>

    <!-- SET -->
    <xsl:when test="@T=' [UNIVERSAL 17] '">
      <xsl:text>SET:</xsl:text>
      <xsl:apply-templates select="." mode="field_name"/>
    </xsl:when>

    <!-- [CONTEXT n] EXPLICIT/IMPLICIT -->
    <!-- The test below checks whether the string between the
brackets - which defines the tag type - is a number, by
checking that it is not NaN (NaN: not a number) -->
    <xsl:when
      test="not (number(substring-before(substring-after(@T, '['), ']')) = NaN)">

      <!-- Explicitly and implicitly tagged types are both decoded
as [n] by unber.
Noting that [n] IMPLICIT SEQUENCE ... is equivalent (after
encoding) to [n] EXPLICIT ..., choosing either representation
is a matter of convenience.
Two cases are handled here:
1) There is only one node under the current node: in that
case, the [n] EXPLICIT ... syntax is assumed. -->
      <xsl:choose>
        <xsl:when test="count(C|P) = 1">

```



```

    <xsl:text>EXPLICIT:</xsl:text>
    <xsl:value-of select="substring-after(substring-before(@T,']'), '[')"/>
    <xsl:text>,</xsl:text>
    <xsl:apply-templates select="C|P" mode="field_value"/>
  </xsl:when>

  <!--
  2) There are several nodes under the current node: in that
     case, the [n] IMPLICIT SEQUENCE ... syntax is used. -->
  <xsl:otherwise>
    <xsl:text>IMPLICIT:</xsl:text>
    <xsl:value-of select="substring-after(substring-before(@T,']'), '[')"/>
    <xsl:text>,</xsl:text>SEQUENCE:</xsl:text>
    <xsl:apply-templates select="." mode="field_name"/>
  </xsl:otherwise>
</xsl:choose>
</xsl:when>

<!-- Fallback: if this happens then this XSLT document needs to
     be extended to cover the unexpected case -->
<xsl:otherwise>
  <xsl:value-of select="@T"/>
  <xsl:text>:</xsl:text>
  <xsl:apply-templates select="." mode="field_name"/>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

<!-- Generates a field name from a C node, using @T to determine
     the type of field and @0 to give it a unique number.
     If the global $prefix parameter is defined, then it is prepended
     with an '_' to the field name. -->
<xsl:template match="C" mode="field_name">
  <xsl:if test="$prefix">
    <xsl:value-of select="$prefix"/>
    <xsl:text>_</xsl:text>
  </xsl:if>

  <xsl:choose>
    <!-- SEQUENCE -->
    <xsl:when test="@T=' [UNIVERSAL 16] '">
      <xsl:text>seq_</xsl:text>
    </xsl:when>

    <!-- SET -->
    <xsl:when test="@T=' [UNIVERSAL 17] '">
      <xsl:text>set_</xsl:text>
    </xsl:when>

    <!-- [CONTEXT n] EXPLICIT/IMPLICIT -->
    <xsl:otherwise>

```

```

        <xsl:text>context_</xsl:text>
    </xsl:otherwise>
</xsl:choose>

    <xsl:value-of select="@0"/>
</xsl:template>

<!--
    Auxiliary templates
    =====
-->

<!-- folds a long line (in parameter $line) into '\'-appended
    64-character lines -->
<xsl:template name="fold_long_line">
    <xsl:param name="line"/>
    <xsl:text>\&#x0a;</xsl:text>
    <xsl:choose>
        <xsl:when test="string-length($line) < 65">
            <xsl:value-of select="$line"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="substring($line, 1, 64)"/>
            <xsl:call-template name="fold_long_line">
                <xsl:with-param name="line" select="substring($line, 65)"/>
            </xsl:call-template>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>

<!-- removes the '\x' from a string (in parameter $string) -->
<xsl:template name="xstring_to_hexstring">
    <xsl:param name="string"/>
    <xsl:call-template name="search_and_replace">
        <xsl:with-param name="replace" select="'\x'"/>
        <xsl:with-param name="in" select="$string"/>
        <xsl:with-param name="with"/>
    </xsl:call-template>
</xsl:template>

<!-- converts a bitstring, specified by a number of unused bits
    (in $unused_bits) and the string itself ($bits) to a BITLIST
    e.g. $unused_bits=6, $bits=11000000 => 0,1 -->
<xsl:template name="bitstring_to_bitlist">
    <xsl:param name="unused_bits"/>
    <xsl:param name="bits"/>
    <xsl:call-template name="indices_of_set_bits_in_bitstring">
        <xsl:with-param name="offset" select="0"/>
        <xsl:with-param name="string"
            select="substring($bits,1,string-length($bits)-$unused_bits)"/>
    </xsl:call-template>

```

```

</xsl:template>

<!-- returns the comma-separated indices of set bits (i.e. 1's) in
a bitstring in parameter $string, adding offset $offset
(e.g.: 1010 would return 0,2 if $offset == 0 and 1,3 if
$offset == 1)-->
<xsl:template name="indices_of_set_bits_in_bitstring">
  <xsl:param name="offset"/>
  <xsl:param name="string"/>
  <xsl:if test="contains($string, '1')">
    <xsl:variable name="set_bit_position"
      select="string-length(substring-before($string, '1')) + $offset"/>
    <xsl:variable name="bitstring_after_set_bit"
      select="substring-after($string, '1')"/>
    <xsl:value-of select="$set_bit_position"/>
    <xsl:if test="contains($bitstring_after_set_bit, '1')">
      <xsl:text>,</xsl:text>
      <xsl:call-template name="indices_of_set_bits_in_bitstring">
        <xsl:with-param name="offset" select="$set_bit_position + 1"/>
        <xsl:with-param name="string" select="$bitstring_after_set_bit"/>
      </xsl:call-template>
    </xsl:if>
  </xsl:if>
</xsl:template>

<!-- converts a \x-prefixed string (in parameter $string) to a binary
string -->
<xsl:template name="xstring_to_binary">
  <xsl:param name="string"/>
  <xsl:call-template name="hexstring_to_binary">
    <xsl:with-param name="string">
      <xsl:call-template name="xstring_to_hexstring">
        <xsl:with-param name="string" select="$string"/>
      </xsl:call-template>
    </xsl:with-param>
  </xsl:call-template>
</xsl:template>

<!-- converts a hexstring (in parameter $string) to a binary string -->
<xsl:template name="hexstring_to_binary">
  <xsl:param name="string"/>
  <xsl:if test="string-length($string) != 0">
    <xsl:call-template name="hexdigit_to_binary">
      <xsl:with-param name="digit" select="substring($string,1,1)"/>
    </xsl:call-template>
    <xsl:call-template name="hexstring_to_binary">
      <xsl:with-param name="string" select="substring($string,2)"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>

```

```

<!-- converts a hexadecimal digit ([0-9a-f]) in parameter $digit
to its binary representation -->
<xsl:template name="hexdigit_to_binary">
  <xsl:param name="digit"/>
  <xsl:choose>
    <xsl:when test="$digit = '0'">0000</xsl:when>
    <xsl:when test="$digit = '1'">0001</xsl:when>
    <xsl:when test="$digit = '2'">0010</xsl:when>
    <xsl:when test="$digit = '3'">0011</xsl:when>
    <xsl:when test="$digit = '4'">0100</xsl:when>
    <xsl:when test="$digit = '5'">0101</xsl:when>
    <xsl:when test="$digit = '6'">0110</xsl:when>
    <xsl:when test="$digit = '7'">0111</xsl:when>
    <xsl:when test="$digit = '8'">1000</xsl:when>
    <xsl:when test="$digit = '9'">1001</xsl:when>
    <xsl:when test="$digit = 'a' or $digit = 'A'">1010</xsl:when>
    <xsl:when test="$digit = 'b' or $digit = 'B'">1011</xsl:when>
    <xsl:when test="$digit = 'c' or $digit = 'C'">1100</xsl:when>
    <xsl:when test="$digit = 'd' or $digit = 'D'">1101</xsl:when>
    <xsl:when test="$digit = 'e' or $digit = 'E'">1110</xsl:when>
    <xsl:when test="$digit = 'f' or $digit = 'F'">1111</xsl:when>
  </xsl:choose>
</xsl:template>

<!-- global search and replace function, which replaces the content
of $replace in $in with $with -->
<xsl:template name="search_and_replace">
  <xsl:param name="replace"/>
  <xsl:param name="in"/>
  <xsl:param name="with"/>

  <xsl:choose>
    <xsl:when test="contains($in, $replace)">
      <xsl:value-of select="substring-before($in, $replace)"/>
      <xsl:value-of select="$with"/>
      <xsl:call-template name="search_and_replace">
        <xsl:with-param name="replace" select="$replace" />
        <xsl:with-param name="in" select="substring-after($in, $replace)"/>
        <xsl:with-param name="with" select="$with" />
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$in"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- count number of \x's in a string ($string), thus counting
hexvalues -->
<xsl:template name="count_hexvalues">
  <xsl:param name="string"/>

```

```

<xsl:choose>
  <xsl:when test="contains($string, '\x')">
    <xsl:variable name="count_hexvalues_after_prefix">
      <xsl:call-template name="count_hexvalues">
        <xsl:with-param name="string"
          select="substring-after($string, '\x')"/>
      </xsl:call-template>
    </xsl:variable>
    <xsl:value-of select="1+$count_hexvalues_after_prefix"/>
  </xsl:when>
  <xsl:otherwise>0</xsl:otherwise>
</xsl:choose>
</xsl:template>

</xsl:stylesheet>

```

Ce fichier comporte des limitations connues, évoquées ci-dessous.

Tout d'abord, la syntaxe ASN.1 n'a pas été reprise intégralement, en particulier : seules les chaînes de caractères de type PrintableString et UTF8String sont gérées, tous les types universels n'ont pas été traités (par exemple le type REAL, qui n'est pas utilisé dans les structures usuelles de la confiance électronique), et les types PRIVATE et APPLICATION ne sont pas pris en compte.

Tous ces éléments peuvent être ajoutés si nécessaire en ajoutant de nouveaux cas (<xsl:when test="...">...</xsl:when>) dans les *templates* <xsl:template match="P" mode="field_name"> et <xsl:template match="P" mode="field_value">.

Ensuite, dans le cas où un type IMPLICIT primitif contient des caractères ASCII et des caractères non ASCII, ces derniers sont représentés sous la forme \xhh, où hh est le codage hexadécimal de l'octet considéré.

Pour illustrer cette limitation, créer par exemple le fichier accents.asn.cnf suivant (à enregistrer avec le codage UTF-8 sans BOM) :

```
asn1 = IMPLICIT:0,FORMAT:UTF8,UTF8String:voici quelques caractères accentués
```

Générer le codage DER puis le décoder à l'aide de sed et du fichier XSLT :

```

$ openssl asn1parse -genconf accents.cnf -out accents.der
  0:d=0  hl=2 l= 37 prim: cont [ 0 ]

$ unber accents.der | sed -f unberclean.sed | xsltproc unber2asnconf.xslt -
asn1 = IMPLICIT:0,FORMAT:UTF8,UTF8String:\
voici quelques caract\xc3\xa8res accentu\xc3\xa9s

```

Les codes \xhh peuvent être décodés en « filtrant » le résultat de la transformation XSLT dans la commande Perl suivante (remplacer les apostrophes par des guillemets sous Windows) :

```
... | perl -pe 's/\\x(.{2})/chr(hex($1))/eg'
```

Pour éviter de retenir cette expression régulière, créer le fichier unescape_hex.pl contenant la ligne suivante :

```
s/\\x(.{2})/chr(hex($1))/eg
```

et utiliser la syntaxe suivante à la place de celle proposée ci-dessus :

```
... | perl -p unescape_hex.pl
```

Par exemple (avec une console shell en UTF-8) :

```
$ unber accents.der | sed -f unberclean.sed | xsltproc unber2asnconf.xslt - \
  | perl -p unescape_hex.pl
asn1 = IMPLICIT:0,FORMAT:UTF8,UTF8String:voici quelques caractères accentués
```

Enfin, le cas des types IMPLICIT et EXPLICIT pouvant donner lieu à des décodages ambigus en l'absence d'un fichier de définition, des arbitrages (décrits dans les commentaires) ont été faits pour déterminer si une balise IMPLICIT SEQUENCE ou EXPLICIT doit être utilisée. La représentation DER est identique, mais la syntaxe obtenue pourra différer par rapport à la syntaxe originelle.

Pour illustrer les différences possibles, soit le fichier d'origine suivant, nommé `taggedtypes.asn.cnf` :

```
asn1 = SEQUENCE:taggedtypes_seq

[taggedtypes_seq]
expl_integer      = EXPLICIT:0,INTEGER:0
expl_seq_1_item   = EXPLICIT:0,SEQUENCE:expl_seq_1_item
expl_seq_n_items  = EXPLICIT:0,SEQUENCE:expl_seq_n_items
impl_integer      = IMPLICIT:0,INTEGER:4
impl_seq_1_item   = IMPLICIT:0,SEQUENCE:impl_seq_1_item
impl_seq_n_items  = IMPLICIT:0,SEQUENCE:impl_seq_n_items

[expl_seq_1_item]
a = INTEGER:1

[expl_seq_n_items]
a = INTEGER:2
b = INTEGER:3

[impl_seq_1_item]
a = INTEGER:5

[impl_seq_n_items]
a = INTEGER:6
b = INTEGER:7
```

Le codage DER obtenu à partir de ce fichier est le suivant :

```
$ openssl asn1parse -genconf taggedtypes.asn.cnf -i -out taggedtypes.der
 0:d=0  hl=2 l= 38 cons: SEQUENCE
 2:d=1  hl=2 l=  3 cons: cont [ 0 ]
 4:d=2  hl=2 l=  1 prim:  INTEGER           :00
 7:d=1  hl=2 l=  5 cons: cont [ 0 ]
 9:d=2  hl=2 l=  3 cons:  SEQUENCE
11:d=3  hl=2 l=  1 prim:  INTEGER           :01
14:d=1  hl=2 l=  8 cons: cont [ 0 ]
16:d=2  hl=2 l=  6 cons:  SEQUENCE
18:d=3  hl=2 l=  1 prim:  INTEGER           :02
21:d=3  hl=2 l=  1 prim:  INTEGER           :03
24:d=1  hl=2 l=  1 prim: cont [ 0 ]
27:d=1  hl=2 l=  3 cons: cont [ 0 ]
29:d=2  hl=2 l=  1 prim:  INTEGER           :05
32:d=1  hl=2 l=  6 cons: cont [ 0 ]
```

```

34:d=2 hl=2 l= 1 prim:  INTEGER          :06
37:d=2 hl=2 l= 1 prim:  INTEGER          :07

```

Le décodage par `unber` suivi de l'application de la transformation XSLT précédente produit le résultat suivant :

```

$ unber taggedtypes.der | sed -f unberclean.sed | xsltproc unber2asnconf.xslt -
asn1 = SEQUENCE:seq_0

```

```

[seq_0]
context_2 = EXPLICIT:0,INTEGER:0
context_7 = EXPLICIT:0,SEQUENCE:seq_9
context_14 = EXPLICIT:0,SEQUENCE:seq_16
implicit_24 = IMPLICIT:0,FORMAT:HEX,OCTETSTRING:\
04
context_27 = EXPLICIT:0,INTEGER:5
context_32 = IMPLICIT:0,SEQUENCE:context_32

```

```

[context_2]
int_4 = INTEGER:0

```

```

[context_7]
seq_9 = SEQUENCE:seq_9

```

```

[seq_9]
int_11 = INTEGER:1

```

```

[context_14]
seq_16 = SEQUENCE:seq_16

```

```

[seq_16]
int_18 = INTEGER:2
int_21 = INTEGER:3

```

```

[context_27]
int_29 = INTEGER:5

```

```

[context_32]
int_34 = INTEGER:6
int_37 = INTEGER:7

```

Noter que l'élément initialement nommé `impl_seq_1_item` admet une syntaxe finale différente, même si la représentation DER est identique. Noter également que les types `EXPLICIT` portant sur une structure à un seul élément donnent lieu à la création d'une section `[context_...]` inutilisée (`context_2`, `context_7` et `context_27` dans le résultat obtenu ci-avant) : cet artéfact sans importance évite des tests et récursions complémentaires qui auraient alourdi le code XSLT.

Générer le fichier de configuration ASN.1 en appliquant cette transformation XSLT au fichier XML généré précédemment :

```

$ xsltproc unber2asnconf.xslt ee-Certificate.xml \
> ee-Certificate.asn-autogen.cnf

```

Dans le fichier généré, les noms des champs ont la forme `type_position` (par exemple `int_13 = INTEGER:0x00dcd21ee5a2b7dfc7`), où `type` est le type ASN.1 d'origine, et `position` est la position (en octets) de la structure dans le fichier DER initial. Pour rendre les champs uniques et ainsi permettre leur

coexistence dans un même fichier de configuration, le fichier XSLT admet un paramètre optionnel, `prefix`, qui sera utilisé (avec le caractère « `_` ») pour préfixer les noms des champs. Par exemple :

```
$ xsltproc --stringparam prefix crt unber2asnconf.xslt ee-crt.xml
asn1 = SEQUENCE:crt_seq_0

[crt_seq_0]
crt_seq_4 = SEQUENCE:crt_seq_4
crt_seq_734 = SEQUENCE:crt_seq_734
...
```

Vérifier que le fichier de configuration ainsi généré produit, après injection dans la commande `openssl asn1parse -genconf ...`, un fichier identique au fichier `ee-Certificate.der` d'origine, par exemple en vérifiant que leurs empreintes sont égales.

B.3. dumpasn1

L'outil `dumpasn1`⁹⁵ de Peter Gutmann propose un affichage agréable et détaillé d'une structure ASN.1 codée en DER.

Cet outil a notamment été utilisé pour représenter le contenu des certificats et de la liste de certificats révoqués dans l'annexe C de la RFC 5280.

L'outil `dumpasn1` n'étant pas livré sous forme d'exécutable, il doit être compilé. Télécharger le fichier source `dumpasn1.c` et le fichier de configuration `dumpasn1.cfg`. Compiler le code source avec le compilateur `gcc` en utilisant la commande `gcc -o dumpasn1 dumpasn1.c`, ou avec le compilateur `cl` de Visual C++ avec la commande `cl /MD dumpasn1.c`.

Pour éviter de compiler `dumpasn1` sous Windows, il est possible d'utiliser l'outil `GUIDumpASN` référencé sur la page de `dumpasn1`, qui propose une interface graphique à l'outil `dumpasn1` et à ses options en ligne de commande.

L'outil d'analyse ASN.1 en ligne⁹⁶ référencé sur la page de `dumpasn1` présente l'intérêt de mettre en évidence par un code couleur les composants du codage DER des éléments ASN.1 (libellé, longueur, valeur), et d'afficher la correspondance entre structure et valeur d'un élément un survolant celui-ci avec le curseur de la souris.

B.4. Initiation au compilateur `asn1c`

À partir d'un module ASN.1, le compilateur ASN.1 `asn1c`⁹⁷ génère des structures C correspondant aux structures ASN.1, et les fonctions permettant de coder et de décoder les structures C vers ou à partir d'une représentation binaire (BER/DER) et XML (XER ou *XML Encoding Rules*). Il est ainsi possible de décrire un objet ASN.1 en XML et de le convertir en binaire, ce qui fait l'objet de cette

95. <http://www.cs.auckland.ac.nz/~pgut001/>

96. <http://lapo.it/asn1js/>

97. <http://lionet.info/asn1c/>

section, qui construit la représentation BER d'une clé publique RSA à partir de sa description XML. Le compilateur C gcc est utilisé ci-après.

Télécharger et installer `asn1c` : la compilation en environnement UNIX/Linux s'effectue sans difficulté en suivant les instructions fournies dans le fichier `INSTALL` du paquetage source, et pour Windows il est fortement recommandé d'installer la version pré-compilée proposée sur le site web.

Créer le fichier `RSAPublicKey.asn1` suivant :

```
RSAPublicKey DEFINITIONS ::=
BEGIN

-- From RFC2313/PKCS#1v1.5
RSAPublicKey ::= SEQUENCE {
    modulus INTEGER,
    publicExponent INTEGER
}

END
```

Compiler le module, avec les options `-S` et `-fskeletons-copy` ci-dessous sous Windows (non nécessaires pour les environnements Linux/UNIX) :

```
> <chemin_vers>asn1c -S<chemin_vers>skeletons \
    -fskeletons-copy RSAPublicKey.asn1
```

La compilation du module inclut un outil nommé `converter-sample` qui permet d'effectuer des conversions de format en ligne de commande.

Compilation sous Windows des fichiers générés par `asn1c`

Le résultat d'une compilation d'`asn1c` sous Windows nécessite quelques modifications pour pouvoir fonctionner, comme décrit ci-après.

Obtenir un fichier `sysexits.h` (provenant d'un système UNIX/Linux par exemple, ou du code source de la GNU C Library [<http://sourceware.org/git/?p=glibc.git>] dans le répertoire `include`) et le placer dans le répertoire de compilation du module ASN.1.

Le code source `converter-sample.c` doit être légèrement modifié pour éviter d'interpréter la valeur `0x0a` comme étant un retour chariot Windows/DOS à convertir en `0x0d 0x0a` : les fichiers lus et l'entrée et la sortie standard doivent être déclarés comme étant en mode binaire.

Changer le mode de lecture sur `stdin` et d'écriture sur `stdout`, en ajoutant cette ligne au début du fichier source :

```
#include <fcntl.h>
```

et en ajoutant les deux lignes suivantes dans la fonction `main()`, après la déclaration initiale des variables :

```
setmode(fileno(stdout), O_BINARY);
setmode(fileno(stdin), O_BINARY);
```

Activer le mode de lecture binaire du fichier d'entrée en remplaçant "r" par "rb" dans le second paramètre de l'appel à `fopen()` dans la fonction `argument_to_file()`. La ligne modifiée est :

```
: fopen(av[idx], "rb");
```

Compiler le convertisseur :

```
$ gcc -o converter-sample.exe -I. -DPDU=RSAPublicKey *.c
```

Convertir la clé publique existante en structure `RSAPublicKey` au format DER.

```
$ openssl rsa -in ee-key.pem -inform PEM -RSAPublicKey_out -outform DER \
-out ee-RSAPublicKey.der
```

Convertir cette structure DER au format XER :

```
$ converter-sample -iber -oxer ee-RSAPublicKey.der
<RSAPublicKey>
  <modulus>00:B8:6F:48:F9:99:F1:99:71:C6:6F:80:64:D1:CA:0C:1A:6E:C8:8A:F3:B9:3
9:FD:07:08:8D:97:B2:BE:1E:27:95:BD:1E:86:88:FF:0E:61:72:B7:3D:37:A5:B8:19:35:C7:
C3:AE:57:A2:7E:5D:46:F3:83:83:08:9C:44:10:ED:A9:5D:1E:FA:99:C4:93:15:86:CE:57:49
:00:60:00:39:02:03:4A:35:8E:07:F9:0F:F2:D3:47:34:2E:6B:F9:51:39:E0:6E:63:F3:C9:9
7:87:4F:4B:35:E8:DA:3A:87:F4:F5:18:8A:86:74:C1:B1:1B:A0:F3:29:FC:5C:2E:B6:CD:26:
F4:65:75:1C:37:89:C6:B9:7E:39:63:36:8A:A1:88:19:10:26:2B:A9:D8:FE:E7:9D:34:34:E7
:22:A2:30:2D:F4:E8:2D:39:79:35:5B:62:54:D0:32:A3:A5:2C:31:7F:A5:D2:98:40:36:93:A
6:C3:A1:66:5F:3F:F6:FC:ED:45:4A:06:45:97:F5:EF:41:7E:68:BB:7F:D2:D3:89:2B:4F:04:
CD:35:9F:2B:01:33:46:00:12:BE:7B:52:82:DB:23:AA:D2:3F:FB:37:64:0F:65:5F:98:F3:80
:6F:10:D9:B4:BF:75:2F:41:A7:E5:BD:17:3A:B8:34:5B:81:FD:5C:8D:D1</modulus>
  <publicExponent>65537</publicExponent>
</RSAPublicKey>
```

Les entiers (INTEGER) supérieurs à $2^{31}-1$ sont représentés par des octets hexadécimaux séparés par des « : ».

Le résultat obtenu permet de se familiariser avec le schéma XML correspondant au module ASN.1.

Une `RSAPublicKey` décrite en XER peut être convertie au format DER en utilisant les options `-ixer` et `-oder`. Ainsi, à partir du résultat ci-dessus préalablement stocké dans le fichier `ee-RSAPublicKey.xer` :

```
$ converter-sample -ixer -oder ee-RSAPublicKey.xer \
| openssl asn1parse -inform DER -i
0:d=0 hl=4 l= 266 cons: SEQUENCE
4:d=1 hl=4 l= 257 prim:  INTEGER               :B86F48F999F19971C66F8064D1CA0C1
```

```
A6EC88AF3B939FD07088D97B2BE1E2795BD1E8688FF0E6172B73D37A5B81935C7C3AE57A27E5D46F
38383089C4410EDA95D1EFA99C4931586CE57490060003902034A358E07F90FF2D347342E6BF9513
9E06E63F3C997874F4B35E8DA3A87F4F5188A8674C1B11BA0F329FC5C2EB6CD26F465751C3789C6B
97E3963368AA1881910262BA9D8FEE79D3434E722A2302DF4E82D3979355B6254D032A3A52C317FA
5D298403693A6C3A1665F3FF6FCED454A064597F5EF417E68BB7FD2D3892B4F04CD359F2B0133460
012BE7B5282DB23AAD23FFB37640F655F98F3806F10D9B4BF752F41A7E5BD173AB8345B81FD5C8DD
1
```

```
265:d=1 hl=2 l= 3 prim: INTEGER :010001
```

Avec « un peu » de patience pour obtenir et adapter les modules, il est possible d'adapter et de compiler l'ensemble des modules ASN.1 correspondant aux certificats X.509, aux jetons TSP, aux capsules CMS et CAdES etc., de manière à pouvoir travailler sur les représentations XER des structures ASN.1 correspondantes, plus faciles à manipuler et modifier que les codages binaires.

Annexe C — Compilation et installation d'OpenSSL

Les instructions de compilation et installation décrites dans le fichier `INSTALL` (pour UNIX/Linux) du code source d'OpenSSL fonctionnent parfaitement. Celles pour Windows (dans `INSTALL.W32`) fonctionnent de manière variable en fonction du compilateur et des versions successives d'OpenSSL. Cette annexe donne quelques conseils pour installer OpenSSL sous ces environnements.

C.1. Linux

Les distributions Linux incluent le plus souvent une version d'OpenSSL, qui est rarement la dernière parue. Les instructions ci-après permettent de faire coexister la dernière version d'OpenSSL avec la version du système (dont les bibliothèques sont souvent nécessaires à l'utilisation d'autres applications ou services, et dont la modification peut donc avoir des effets de bord non souhaitables). Les pré-requis sont l'installation de Perl 5 et gcc.

Télécharger la dernière version d'OpenSSL (la version 1.0.1a à la date de rédaction — se reporter au site d'OpenSSL⁹⁸ pour connaître la dernière version à la date de lecture) dans un répertoire (pour fixer les idées, ce répertoire est supposé être `~/src`) :

```
$ wget http://www.openssl.org/source/openssl-1.0.1a.tar.gz
```

ou

```
$ curl -O http://www.openssl.org/source/openssl-1.0.1a.tar.gz
```

ou toute commande équivalente (ex. : copie FTP/SCP depuis une autre machine ayant accès à Internet), en fonction des possibilités de la machine.

Décompresser l'archive :

```
$ tar xvzf ../openssl-1.0.1a.tar.gz
$ cd openssl-1.0.1a
```

Configurer la compilation. Pour une installation transverse, si l'accès à un shell `root` sera possible lors de la phase d'installation, alors utiliser la commande suivante :

```
$ ./config --prefix /opt/openssl-1.0.1a shared
```

Si l'accès `root` ne sera pas disponible, alors utiliser :

```
$ ./config --prefix $HOME/openssl-1.0.1a shared
```

98. <http://www.openssl.org/>

Pour éviter des conflits de fichiers, remplacer `$HOME/openssl-1.0.1a` si le code source a été décompressé dans ce répertoire (au lieu par exemple de `~/src` comme proposé ci-avant).

Si la configuration n'a soulevé aucune erreur, alors démarrer la compilation.

```
$ make
```

Cette opération dure de quelques minutes à quelques dizaines de minutes selon la puissance de la machine. (Si elle dure significativement plus longtemps, alors la machine n'est peut-être pas la plus adéquate pour effectuer des calculs cryptographiques complexes !)

Si la compilation ne génère aucune erreur, alors effectuer optionnellement les tests (`$ make test`) puis procéder à l'installation. S'il s'agit d'une installation transverse à la machine (dans `/opt/openssl-1.0.1a`), se connecter à la machine avec le compte `root` (ou employer `sudo`) et se rendre dans le répertoire de compilation (`~utilisateur_compilateur/src/openssl-1.0.1a`) au préalable.

```
$ make install
```

ou (cas `root`)

```
# make install
```

Le compte `root` n'est plus nécessaire pour la suite.

Créer s'il n'existe pas le répertoire `~/bin`. Ajouter, s'il n'y est pas déjà, le répertoire `$HOME/bin` à la fin de la variable d'environnement `$PATH` du profil de l'utilisateur (par exemple sous `bash` en ajoutant la ligne `PATH=$PATH:$HOME/bin` en fin de fichier `~/.bash_profile`). Enfin, créer dans `~/bin` un lien symbolique vers l'exécutable `openssl` :

```
$ ln -s /opt/openssl-1.0.1a/bin/openssl ~/bin/openssl-1.0.1a
```

dans le cas d'une installation commune, ou (dans le cas d'une installation pour l'utilisateur uniquement) :

```
$ ln -s ~/openssl-1.0.1a/bin/openssl ~/bin/openssl-1.0.1a
```

Les utilisateurs souhaitant utiliser la commande `openssl` au lieu de `openssl-1.0.1a` et conscients des risques de sécurité sous-jacents, peuvent nommer le lien symbolique `openssl` et ajouter `$HOME/bin` au début de la variable d'environnement `$PATH` (donc `PATH=$HOME/bin:$PATH`).

Redémarrer la session si la variable `$PATH` a été modifiée dans le profil, et vérifier que tout fonctionne comme attendu :

```
$ openssl-1.0.1a version
```

```
OpenSSL 1.0.1a 19 Apr 2012
```

Le répertoire de compilation d'OpenSSL n'est plus nécessaire et peut être supprimé. Pour supprimer l'installation d'OpenSSL, supprimer simplement le répertoire d'installation (ex. : `$ rm -Rf ~/openssl-x.x.x` ou `# rm -Rf /opt/openssl-x.x.x`, avec les précautions d'usage applicables à l'utilisation de `rm -Rf` !).

C.2. Windows

Cette section propose un mode opératoire pour compiler OpenSSL sous Windows de manière à pouvoir utiliser le moteur cryptographique `cap`. D'autres méthodes sont possibles (et sont décrites dans le fichier `INSTALL.W32` à la racine du paquetage source d'OpenSSL) mais présentent certaines limitations en l'absence de modifications.

La compilation d'OpenSSL sous Windows suppose que les logiciels suivants sont installés :

- Perl : ActivePerl Community Edition⁹⁹ et Strawberry Perl¹⁰⁰ sont des versions gratuites de Perl pour Windows.
- L'assembleur `nasm`¹⁰¹, optionnellement utilisé pour tirer parti de code accéléré pour certaines fonctions de bas niveau (opérations sur les grands nombres et calculs cryptographiques).
- Un compilateur C : celui utilisé ci-après est le compilateur inclus dans Visual C++ Express Edition 2010 (l'utilisation des versions précédentes, 2005 et 2008, devrait également être possible) de Microsoft, disponible gratuitement au téléchargement sur le site de Visual Studio¹⁰².

Télécharger le code source d'OpenSSL à partir du site www.openssl.org¹⁰³. Le code source se présente sous la forme d'une archive compressée, portant un nom du type `openssl-x.x.x.tar.gz`. Décompresser cette archive.

Pour les versions 1.0.1 à (au moins) 1.0.1b d'OpenSSL, modifier le code du script `util/mk1mf.pl` comme décrit sur cette page¹⁰⁴, en ajoutant la ligne `s/\r$//`; après la ligne `chop` ;.

Depuis le répertoire source d'OpenSSL, configurer la compilation :

```
> perl Configure VC-WIN32 --prefix=c:\openssl\openssl-1.0.1b
> ms\do_nasm
```

Ouvrir ensuite une invite de commande de Visual C++ 2010, et démarrer la compilation :

```
> nmake -f ms\ntdll.mak
```

Dans cette même invite de commande, lancer optionnellement les tests :

```
> nmake -f ms\ntdll.mak test
```

Installer enfin OpenSSL :

```
> nmake -f ms\ntdll.mak install
```

99. <http://www.activestate.com/activeperl/downloads>

100. <http://strawberryperl.com/>

101. <http://nasm.sourceforge.net/>

102. <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/express>

103. <http://www.openssl.org>

104. http://groups.google.com/group/mailling.openssl.users/browse_thread/thread/42a8f226f1fc279f

Annexe D — Compilation de xmlsec sous Windows

La compilation de xmlsec repose sur la compilation préalable de plusieurs bibliothèques. Télécharger et décompresser le code source de libxml2¹⁰⁵, libxslt¹⁰⁶, xmlsec¹⁰⁷, et compiler OpenSSL comme décrit dans l'annexe consacrée à ce sujet.

Les bibliothèques libxml2, libxslt et xmlsec peuvent s'appuyer sur iconv, mais cette option n'est pas retenue car la compilation d'iconv supporte uniquement MinGW, contrairement aux autres bibliothèques qui privilégient ou supportent uniquement Visual C++, et les incompatibilités entre les bibliothèques produites par les deux compilateurs compliquent la procédure. De même, les bibliothèques peuvent s'appuyer sur zlib, mais cela n'apportant rien dans le cadre des manipulations proposées dans ce document, la compilation de zlib est omise.

L'environnement de compilation est Visual C++ 2010 Express. Le répertoire d'installation de xmlsec est noté *XMLSEC_DIR*.

D.1. Compilation de libxml2

Effectuer les modifications suivantes dans le fichier `win32/Makefile.msvc` du répertoire source de libxml2 :

- Supprimer les symboles « + » au début de chacune des trois lignes référencées dans ce patch¹⁰⁸.
- Effacer l'option de compilation `/OPT:NOWIN98` (obsolète sous Visual Studio 2008 et supprimée sous Visual Studio 2010).

Ouvrir une invite de commande Visual C++ 2010 Express dans le sous-répertoire `win32` du répertoire source, et configurer la compilation.

```
> cscript configure.js iconv=no prefix=XMLSEC_DIR
```

Démarrer la compilation et l'installation, depuis le même répertoire.

```
> nmake /f Makefile.msvc clean all install
```

Déplacer `libxml2.dll` du répertoire `XMLSEC_DIR/lib` vers `XMLSEC_DIR/bin`.

Pour vérifier que la compilation et l'installation se sont effectuées correctement, ouvrir une invite de commande standard dans `XMLSEC_DIR/bin`, réinitialiser la variable d'environnement `PATH` (via `set PATH=` par exemple) pour éviter d'éventuels conflits avec des bibliothèques pré-installées dans un des

105. <http://xmlsoft.org/>

106. <http://xmlsoft.org/>

107. <http://www.aleksey.com/xmlsec/>

108. <http://git.gnome.org/browse/libxml2/commit/?id=364e3d2b054656f2cf97594365d15b2ddb72a9ed>

répertoires de PATH, copier les sous-répertoires `test` et `result` du répertoire source de `libxml2` dans `XMLSEC_DIR/bin`, et exécuter `runtest.exe`. Quelques milliers de tests devraient se dérouler, avec une poignée d'erreurs tout au plus.

D.2. Compilation de libxslt

Dans le fichier `win32/Makefile.msvc` du répertoire source de `libxslt`, effacer l'option de compilation `/OPT:NOWIN98`.

Ouvrir une invite de commande Visual C++ 2010 Express dans le sous-répertoire `win32` du répertoire source, et configurer la compilation.

```
> cscript configure.js iconv=no prefix=XMLSEC_DIR \
  lib=$(PREFIX)\lib include=$(PREFIX)\include
```

Démarrer la compilation et l'installation, depuis le même répertoire.

```
> nmake /f Makefile.msvc clean all install
```

Déplacer `libxslt.dll` et `libexslt.dll` du répertoire `XMLSEC_DIR/lib` vers `XMLSEC_DIR/bin`.

D.3. Compilation et installation de xmlsec

Avant tout, copier les sous-répertoires `bin`, `include` et `lib` du répertoire d'installation d'OpenSSL dans le répertoire `XMLSEC_DIR` (fusionnant ainsi avec les sous-répertoires équivalents).

Optionnellement, dans le fichier `win32/configure.js`, remplacer `"098"` par `"101"` dans la ligne `withOpenSSLVersion = "098"; /* default */`.

Dans le fichier `win32/Makefile.msvc` du répertoire source de `xmlsec`, effacer l'option de compilation `/OPT:NOWIN98`.

Ouvrir une invite de commande dans le sous-répertoire `win32` du répertoire source, et configurer la compilation.

```
> cscript configure.js iconv=no prefix=XMLSEC_DIR \
  lib=$(PREFIX)\lib include=$(PREFIX)\include static=no
```

Démarrer la compilation et l'installation, depuis le même répertoire.

```
> nmake /f Makefile.msvc clean all install
```

Déplacer `libxmlsec.dll` et `libxmlsec-openssl.dll` du répertoire `XMLSEC_DIR/lib` vers `XMLSEC_DIR/bin`.

L'exécutable résultant de la compilation, `xmlsec.exe`, est compilé avec les bibliothèques `libxml`, `libxslt` et `libxmlsec` de manière dynamique (et charge la bibliothèque `libxmlsec-openssl` de manière dynamique à l'exécution), c'est-à-dire que pour pouvoir exécuter `xmlsec.exe`, les fichiers DLL `libxml2.dll`, `libxslt.dll`, `libxmlsec.dll` et `libxmlsec-openssl.dll` doivent être dans le

répertoire courant ou dans un répertoire inclus dans la variable d'environnement PATH. Après l'installation initiale, pour générer la version compilée de manière statique (plus grande, mais ne dépendant pas des bibliothèques dynamiques), configurer la compilation avec l'option `with-dl=no` (`static=yes` étant implicite) :

```
> cscript configure.js iconv=no prefix=XMLSEC_DIR \  
  lib=$(PREFIX)\lib include=$(PREFIX)\include with-dl=no
```

Démarrer la compilation sans installation (omettre le paramètre `install`).

```
> nmake /f Makefile.msvc clean all install
```

Copier l'exécutable `xmlseca.exe` ainsi généré du sous-répertoire `win32/binaries` du répertoire de compilation vers `XMLSEC_DIR/bin`.

■ L'exécutable reste dépendant de la bibliothèque dynamique `libeay32.dll`.

Annexe E — Spécificités de Windows

E.1. OpenSSL et CryptoAPI

OpenSSL peut s'appuyer sur des moteurs cryptographiques tiers (appelés *engines*), dont la CryptoAPI (ou CAPI) Microsoft, l'API cryptographique native de Windows. Sans configuration spécifique, OpenSSL s'attend à ce que les bibliothèques dynamiques associées aux *engines* (par exemple `capi.dll` pour la CryptoAPI) soient situées dans le sous-répertoire `lib/engines` du répertoire d'installation par défaut d'OpenSSL, lequel est compilé dans l'exécutable `openssl`. Si tel est le cas, alors la commande suivante, qui affiche les CSP (*Cryptographic Service Provider* ou fournisseur de service cryptographique) installés, ne provoque aucune erreur :

```
> openssl engine -t capi -post list_csps
(capi) CryptoAPI ENGINE
    [ available ]
Available CSPs:
0. Microsoft Base Cryptographic Provider v1.0, type 1
1. Microsoft Base DSS and Diffie-Hellman Cryptographic Provider, type 13
2. Microsoft Base DSS Cryptographic Provider, type 3
3. Microsoft Base Smart Card Crypto Provider, type 1
4. Microsoft DH SChannel Cryptographic Provider, type 18
5. Microsoft Enhanced Cryptographic Provider v1.0, type 1
6. Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider, type 13
7. Microsoft Enhanced RSA and AES Cryptographic Provider, type 24
8. Microsoft Exchange Cryptographic Provider v1.0, type 5
9. Microsoft RSA SChannel Cryptographic Provider, type 12
10. Microsoft Strong Cryptographic Provider, type 1
[Success]: list_csps
```

Dans le cas contraire, une ligne telle que celle-ci fait savoir que la bibliothèque ne se trouve pas dans le répertoire prévu :

```
2068:error:25078067:DSO support routines:WIN32_LOAD:could not load the shared li
brary:.\crypto\dso\dso_win32.c:180:filename(c:\openssl\openssl-1.0.1\lib\engines
\capi.dll)
```

La commande `openssl engine` permet de définir les paramètres de chargement d'un moteur cryptographique en ligne de commande, notamment le chemin de la bibliothèque (`-pre SO_PATH:...\\lib\\engines\\capi.dll` pour la CryptoAPI), et l'identifiant de l'*engine* tel que défini par OpenSSL (`-pre ID:capi`). Pour une raison que ne comprend pas l'auteur (*bug* ?), cet usage permet uniquement de charger l'*engine* (via `-pre LOAD`), mais ignore l'exécution de commandes (ci-dessous la commande `list_csps` n'est pas exécutée). L'auteur n'a pas creusé ce point, car il permettrait de résoudre le problème pour la commande `openssl engine` uniquement, et non pour les autres commandes, qui n'acceptent pas les options `-pre`.

```
> openssl engine -t dynamic -pre SO_PATH:...\\lib\\engines\\capi.dll -pre ID:capi \
-pre LOAD -post list_csps
```

```
(dynamic) Dynamic engine loading support
[Success]: SO_PATH:...\\lib\\engines\\capi.dll
[Success]: ID:capi
[Success]: LOAD
Loaded: (capi) CryptoAPI ENGINE
      [ available ]
```

Il est possible de définir le chemin des bibliothèques des moteurs cryptographiques dans le fichier de configuration `openssl.cnf`, qu'OpenSSL s'attend par défaut à trouver dans le sous-répertoire `ssl` du répertoire d'installation. L'absence de ce fichier de configuration se traduit par l'avertissement `WARNING: can't open config file:...` au lancement de toutes les commandes `openssl`. Si certaines commandes `openssl` permettent de définir l'emplacement d'un fichier de configuration alternatif (option `-config`), ce n'est pas le cas de toutes. Heureusement, une fonctionnalité parfois méconnue (mentionnée ici¹⁰⁹ ou encore ici¹¹⁰) permet de modifier l'emplacement du fichier de configuration consulté par OpenSSL, en affectant à la variable d'environnement `OPENSSL_CONF` le chemin du fichier.

Créer le fichier de configuration ci-dessous — inspiré de cet échange¹¹¹ — sous le nom `CONF_DIR\\capi.cnf`, où `CONF_DIR` est un répertoire arbitraire. Dans ce fichier, `OPENSSL_DIR` est à remplacer par le chemin du répertoire d'installation d'OpenSSL, en veillant à échapper le caractère « `\` » avec « `\\` » (en d'autres termes, utiliser « `\\` » comme séparateur de sous-répertoires).

```
openssl_conf = openssl_init

[openssl_init]
engines = engine_section

[engine_section]
capi = capi_config

[capi_config]
engine_id = capi
dynamic_path = OPENSSL_DIR\\lib\\engines\\capi.dll
init=1
```

Utiliser la variable d'environnement `OPENSSL_CONF` pour définir l'emplacement du fichier de configuration, et tenter à nouveau d'afficher les CSP :

```
> set OPENSSL_CONF=CONF_DIR\\capi.cnf
> openssl engine -t capi -post list_csps
```

La liste des commandes supportées peut être affichée via :

```
> openssl engine -vvvv -t capi
(capi) CryptoAPI ENGINE
      [ available ]
      list_certs: List all certificates in store
                  (input flags): NO_INPUT
      lookup_cert: Lookup and output certificates
```

109. http://www.openssl.org/docs/crypto/OPENSSL_config.html

110. <http://www.openssl.org/docs/apps/req.html>

111. <http://www.mail-archive.com/openssl-users@openssl.org/msg62249.html>

```

        (input flags): STRING
debug_level: debug level (1=errors, 2=trace)
        (input flags): NUMERIC
debug_file: debugging filename)
        (input flags): STRING
key_type: Key type: 1=AT_KEYEXCHANGE (default), 2=AT_SIGNATURE
        (input flags): NUMERIC
list_csps: List all CSPs
        (input flags): NO_INPUT
csp_idx: Set CSP by index
        (input flags): NUMERIC
csp_name: Set CSP name, (default CSP used if not specified)
        (input flags): STRING
csp_type: Set CSP type, (default RSA_PROV_FULL)
        (input flags): NUMERIC
list_containers: list container names
        (input flags): NO_INPUT
list_options: Set list options (1=summary,2=friendly name, 4=full printout,
8=PEM output, 16=XXX, 32=private key info)
        (input flags): NUMERIC
lookup_method: Set key lookup method (1=substring, 2=friendlyname, 3=container name)
        (input flags): NUMERIC
store_name: certificate store name, default "MY"
        (input flags): STRING
store_flags: Certificate store flags: 1 = system store
        (input flags): NUMERIC

```

Pour déboguer l'utilisation du moteur capi, ajouter les deux lignes suivantes après la ligne `init=1` du fichier de configuration ci-dessus :

```

debug_level = 2
debug_file = capi.log

```

Chaque utilisation du moteur capi ajoutera des lignes de trace à la fin du fichier `capi.log` (créé au besoin).

E.2. Analyse du magasin de certificats de Windows

Les certificats logiciels de l'utilisateur sont généralement situés dans l'un des répertoires suivants :

- Sous Windows XP : `C:\Documents and Settings\<utilisateur>\Application Data\Microsoft\SystemCertificates\My\Certificates.`
- Sous Windows Vista/7 : `C:\Users\<utilisateur>\AppData\Roaming\Microsoft\SystemCertificates\My\Certificates.`

La liste complète des emplacements où sont stockés les certificats est disponible sur cette page¹¹² relative à la migration de magasin de certificats.

Le nom du fichier correspondant à un certificat donné est l'empreinte SHA-1 du codage DER du certificat, dont l'obtention a été décrite dans la section 4.6.

Si elle existe, alors la clé privée associée à un certificat du magasin de certificats de Windows a un identifiant de conteneur unique de type GUID (*Globally Unique Identifier*), affichable par exemple de la manière suivante à partir d'une recherche sur le nom convivial du certificat :

```
> openssl engine -t capi -post lookup_method:2 -post list_options:32 \
  -post lookup_cert:"OpenSSL EE"
(capi) CryptoAPI ENGINE
  [ available ]
[Success]: lookup_method:2
[Success]: list_options:32
  Private Key Info:
    Provider Name:  Microsoft Enhanced Cryptographic Provider v1.0, Provider Typ
e 1
    Container Name: {B05E3F2B-27AE-48B6-A33E-3CB4A6BDF046}, Key Type 1
[Success]: lookup_cert:OpenSSL EE
```

Le GUID B05E3F2B-27AE-48B6-A33E-3CB4A6BDF046 est ici l'identifiant recherché.

Un outil C# est proposé pour obtenir le GUID du conteneur de la clé privée à partir d'un certificat sélectionné dans une boîte de dialogue Windows. Créer le fichier `GetContainerNameFromCertPicker.cs` suivant :

```
using System;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;

class CertSelect {
    static void Main() {
        try {
            X509Store store = new X509Store("MY", StoreLocation.CurrentUser);
            store.Open(OpenFlags.ReadOnly | OpenFlags.OpenExistingOnly);
            X509Certificate2Collection collection =
                X509Certificate2UI.SelectFromCollection(
                    (X509Certificate2Collection)store.Certificates,
                    "Certificate selection",
                    "Select a certificate to obtain the container name from",
                    X509SelectionFlag.SingleSelection);

            if (collection.Count == 1) {
                X509Certificate2 x509 = collection[0] ;
                Console.WriteLine("Subject: {0}", x509.Subject) ;
                Console.WriteLine("Friendly name: {0}", x509.FriendlyName) ;
                if (x509.PrivateKey != null) {
                    ICspAsymmetricAlgorithm pkey = x509.PrivateKey
                        as ICspAsymmetricAlgorithm ;
```

112. <http://msdn.microsoft.com/en-us/library/windows/desktop/bb204781%28v=vs.85%29.aspx>


```

        Console.WriteLine("Key container name: {0}",
            pkey.CspKeyContainerInfo.KeyContainerName);
    }
    x509.Reset();
}
store.Close();
}
catch (Exception e) {
    Console.WriteLine(e.ToString());
}
}
}

```

Compiler le code à l'aide de la commande suivante :

```
> <chemin_vers>\csc.exe GetContainerNameFromCertPicker.cs
```

Le *framework* .NET de Microsoft doit être installé pour disposer du compilateur C# `csc.exe`. Celui-ci est alors situé dans le répertoire `C:\WINDOWS\Microsoft.NET\Framework\<version>\`.

Exécuter `GetContainerNameFromCertPicker.exe`, et sélectionner un certificat dans la boîte de dialogue.

```
> GetContainerNameFromCertPicker
```

```
Subject: CN=Entité Finale, OU=0002 963852741, O=Mon Organisation, C=FR
```

```
Friendly name: OpenSSL EE2
```

```
Key container name: {554D37DB-5F0B-4050-ABE1-379D3FD39533}
```

Pour les utilisateurs, les clés privées RSA sont généralement stockées dans des fichiers situés dans l'un des répertoires suivants :

- Sous Windows XP : `C:\Documents and Settings\<utilisateur>\Application Data\Microsoft\Crypto\RSA\S-1-5-21-...`, où `S-1-5-21-...` est le `SID`¹¹³ (*Security Identifier*, ou identifiant de sécurité) de l'utilisateur, identifiant celui-ci de manière unique.
- Sous Windows Vista/7 : `C:\Users\<utilisateur>\AppData\Roaming\Microsoft\Crypto\RSA\S-1-5-21-...`

Cette page¹¹⁴ donne la liste exhaustive des cas de figure rencontrés.

113. <http://technet.microsoft.com/en-us/sysinternals/bb897417>

114. <http://msdn.microsoft.com/en-us/library/bb204778%28v=vs.85%29.aspx>

La méthode la plus simple pour déterminer le fichier contenant une clé privée donnée consiste à faire correspondre la date et l'heure de génération ou d'importation de la clé privée avec la date des fichiers dans le répertoire des clés privées RSA. À défaut d'informations sur la date et l'heure de génération, il est possible d'utiliser l'algorithme décrit par Steve Johnson¹¹⁵ pour déterminer le nom du fichier contenant la clé privée à partir d'un GUID donné. Cet algorithme peut être automatisé à l'aide d'une application C# : télécharger l'archive `RsaUtil.zip` de la page web référencée, extraire le fichier `RsaUtil.cs` dans un répertoire arbitraire, et créer le fichier `GetRsaKeyContainerFilename.cs` suivant dans le même répertoire :

```
using System;

public class GetRsaKeyContainerFilename {
    static void Main(string[] args) {
        if (args.Length == 0) {
            Console.WriteLine(
                "Syntax: GetRsaKeyContainerFilename <private key GUID>"
            );
        }
        else {
            Console.WriteLine(
                RsaUtil.RsaKeyFileUtil.GetRsaKeyContainerFilename(
                    args[0], false
                )
            );
        }
    }
}
```

Compiler les deux fichiers source à l'aide de la commande suivante :

```
> <chemin_vers>\csc.exe RsaUtil.cs GetRsaKeyContainerFilename.cs
```

Exécuter le fichier `GetRsaKeyContainerFilename.exe` résultant, en passant le GUID en paramètre, par exemple :

```
> GetRsaKeyContainerFilename.exe {554D37DB-5F0B-4050-ABE1-379D3FD39533}
C:\Documents and Settings\<utilisateur>\Application Data\Microsoft\Crypto\RSA\S-
1-5-21-xxxxxxxx-xxxxxxxx-xxxxxxxx-xxxxx\df499da8fc142b55c15a9d7e1a3fa6b9_665d
02d4-e128-4978-8058-4969c9695bf6
```

La structure du fichier dont le chemin est retourné n'est pas documenté, mais un peu de travail de reconnaissance de motifs permet de distinguer quatre parties :

- Une en-tête référençant notamment le GUID de la clé privée.
- La clé publique.
- La clé privée, protégée.
- Le drapeau d'exportation de la clé privée, protégé.

115. <http://www.stevestechspot.com/SSLCertificatesWrapup.aspx>

Les positions relatives de ces blocs de données varient en fonction du type et de la taille du bi-clé, mais devraient être les suivantes pour un bi-clé RSA de 2048 bits, à adapter au besoin.

Le bloc d'en-tête pour la clé privée considérée est le suivant (octets 0x0000 à 0x0062 du fichier d'entrée). Noter la chaîne de caractères du GUID de la clé privée.

```
> od -tx1z -Ax -N0x73 <fichier clé privée>
000000 02 00 00 00 01 00 00 00 27 00 00 00 00 00 00 00 >.....'.....<
000010 00 00 00 00 1c 01 00 00 f2 05 00 00 14 00 00 00 >.....=<
000020 00 00 00 00 a8 00 00 00 7b 35 35 34 44 33 37 44 >....j...{554D37D<
000030 42 2d 35 46 30 42 2d 34 30 35 30 2d 41 42 45 31 >B-5FOB-4050-ABE1<
000040 2d 33 37 39 44 33 46 44 33 39 35 33 33 7d 00 00 >-379D3FD39533}...<
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 >.....<
000060 00 00 00 >...<
000063
```

La clé publique est représentée par le bloc suivant (octets 0x0063 à 0x017e du fichier d'entrée, soit une longueur de 0x011C octets).

```
> od -tx1z -Ax -j0x63 -N0x011C <fichier clé privée>
000063 52 53 41 31 08 01 00 00 00 08 00 00 ff 00 00 00 >RSA1.....<
000073 01 00 01 00 ed 6b 32 2b a8 7b 67 78 50 45 d9 af >...Ÿk2+;{gxPEJ»<
000083 ae bb 35 06 29 24 09 4f ff 54 f1 d6 75 53 61 89 >«Ÿ5.)$ 0 T±ÍuSaë<
000093 a6 b8 4a a7 87 dc af 2c 21 3a 7d 82 92 de 56 15 >ª©J°g»,!;}éÆIV.<
0000a3 ac d8 59 18 ea 79 03 ea c5 19 21 57 40 7c 3f e8 >%İY.Ûy.Û+.!W@|?þ<
0000b3 4d 73 08 09 78 49 82 df 39 35 5b a3 59 d5 b6 34 >Ms. xIé■95[úY1Â4<
0000c3 27 7a 90 65 12 15 95 8b 1a c8 7e dc 8e 37 47 48 >'z.e..ði.Ł~■Ä7GH<
0000d3 71 6b c8 6d 58 3e af b8 b2 84 69 7a cf 95 51 c0 >qkŁmX>»©■äiz#ôQŁ<
0000e3 40 b8 2e 03 98 d8 ea 69 96 0d 71 b1 67 e3 d9 93 >©©...İÛiû.q■gÛJ ô<
0000f3 6b e3 a9 a6 40 2e 8c b9 08 06 95 35 6f 2f a6 11 >kÛªª@.i¶||..ð5o/ª.<
000103 18 2d 43 00 a3 36 0a 40 b2 1d 44 1a 24 60 e4 da >.-C.ú6.©■.D.$`ðŸ<
000113 a5 56 18 94 ee 26 5e 60 7d e4 a6 0d 87 a3 99 4f >ŒV.ö- & ^`ðª.çú.0<
000123 ce 31 73 08 7d e8 9f fe 80 0f d4 63 00 81 c7 a7 >¶1s.}þf■..Èc..Ä°<
000133 46 8a ab 4a 10 12 e5 e4 a6 2f 05 10 56 12 b3 c8 >Fè½J..Ûðª/..V. | Ł<
000143 9f cc 3d ae 95 4c da 4c 6a e1 3a 15 fd 72 23 db >f ¶=«òL ŸLjß:..²r#■<
000153 a7 a2 48 3a f4 37 e8 7a a3 74 5f c1 97 f3 98 08 >°óH:¶7þzût_Łû¾...<
000163 3e 7f 3f bf 76 37 58 fc 29 69 37 ae f0 dd 20 de >>..?Ÿv7X³)i7«! İ<
000173 ec be 89 bc 00 00 00 00 00 00 00 00 00 00 >ýŸë||.....<
00017f
```

Ce bloc ressemble à une structure `PUBLICKEYBLOB`¹¹⁶, avec des éléments supplémentaires. Noter la chaîne « RSA1 » indiquant qu'il s'agit d'une clé publique RSA, la taille de la clé (à l'octet 0x007b pour une longueur de quatre octets, 00 08 00 00 est la représentation *little-endian* — ou petit-boutiste, c'est-à-dire que les octets sont ordonnés par poids croissants, donc la représentation « humaine » est 0x00000800 — de 2048), l'exposant public (01 00 01 00, à l'octet 0x0073, représente 65 537) et le module (à partir de l'octet 0x0077 sur une longueur de 256 octets, toujours en représentation *little-endian*, l'octet de poids fort étant à la position 0x0176, ce qui permet de rétablir l'ordre *big-endian* 0xbc89bc... affiché par OpenSSL).

116. [http://msdn.microsoft.com/en-us/library/windows/desktop/aa387459\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/aa387459(v=vs.85).aspx)

Les deux blocs suivants sont protégés par l'API de protection DPAPI¹¹⁷ de Microsoft. Cette API propose deux fonctions, `CryptProtectData()` et `CryptUnprotectData()`, la première pour protéger un ensemble de données (de type `DATA_BLOB`, où `blob` signifie *binary lump of bits*, *binary large object* etc., c'est-à-dire des données binaires), la seconde pour déprotéger un `DATA_BLOB`.

La DPAPI s'appuie en particulier sur le mot de passe de l'utilisateur pour déprotéger un bloc de données. Lorsque la session de l'utilisateur est ouverte, le mot de passe est conservé en cache et n'est pas demandé à l'utilisateur pour déprotéger les données. Lorsque la session de l'utilisateur est fermée, l'inaccessibilité du mot de passe garantit la protection des données (à concurrence de la sécurité du mot de passe).

Le premier bloc protégé (de l'octet `0x017f` à l'octet `0x0771`, pour une longueur de `0x05F2` ou 1522 octets) est la clé privée de l'utilisateur.

```
> od -tx1z -Ax -j0x17f -N0x05f2 <fichier clé privée>
00017f 01 00 00 00 d0 8c 9d df 01 15 d1 11 8c 7a 00 c0 >...ðî.■...Ð.îz. L<
00018f 4f c2 97 eb 01 00 00 00 e9 75 b6 d6 59 ac cb 40 >0TùÛ...ÛuÃÍY¼T@<
00019f a9 75 1a 04 f0 e0 f6 26 03 00 00 00 2a 00 00 00 >@u..Û÷&....*...<
0001af 43 00 6c 00 e9 00 20 00 70 00 72 00 69 00 76 00 >C.l.Û. .p.r.i.v.<
0001bf e9 00 65 00 20 00 43 00 72 00 79 00 70 00 74 00 >Û.e. .C.r.y.p.t.<
0001cf 6f 00 41 00 50 00 49 00 00 00 03 66 00 00 a8 00 >o.A.P.I....f..j.<
0001df 00 00 10 00 00 00 ec e8 57 96 42 4a c4 da 76 f7 >.....ýPwûBJ—rv,<
0001ef bf 87 b5 af d0 eb 00 00 00 04 80 00 00 a0 00 >γςÁ»ðÛ.....á.<
0001ff 00 00 10 00 00 00 13 6b 19 40 d3 bd 0a 78 2b fa >.....k.®Ëç.x+<
00020f b2 a8 e6 37 2b 0d 40 05 00 00 05 40 25 a7 62 34 >■µ7+.@....@%°b4<
...
00075f 0b 4a d3 5a cf cb c5 a7 a9 cc 05 a4 dd 55 34 72 >.JËZ¼T+°@||.ñ!U4r<
00076f 5b 12 >[.<
000771
```

L'ingénierie inverse d'un blob protégé par la DPAPI a été décrite par Elie Bursztein et Jean-Michel Picod dans la section 3.3.1 de leur livre blanc¹¹⁸, et n'est pas reprise ici. Noter simplement la chaîne « Clé privé CryptoAPI\0 » dans le champ description (codé en UTF-16 *little-endian*).

Le second bloc protégé (couvrant les octets `0x0772` à `0x0818`, pour une longueur de `0x00a8`) a une structure analogue à la précédente. Y noter la description « Export flag ».

```
> od -tx1z -Ax -j0x0772 -N0xa8 <fichier clé privée>
000772 00 00 00 d0 8c 9d df 01 15 d1 11 8c 7a 00 c0 4f >...ðî.■...Ð.îz. L0<
000782 c2 97 eb 01 00 00 00 e9 75 b6 d6 59 ac cb 40 a9 >TùÛ...ÛuÃÍY¼T@&<
000792 75 1a 04 f0 e0 f6 26 00 00 00 00 18 00 00 00 45 >u..Û÷&.....E<
0007a2 00 78 00 70 00 6f 00 72 00 74 00 20 00 46 00 6c >.x.p.o.r.t. .F.l<
0007b2 00 61 00 67 00 00 00 03 66 00 00 a8 00 00 00 10 >.a.g....f..j....<
0007c2 00 00 00 68 33 ee a9 d9 93 7b 29 82 0e fd ed 9c >...h3~@Jô{)é.²Ý&<
0007d2 03 fa e4 00 00 00 00 04 80 00 00 a0 00 00 00 10 >..ð.....á....<
0007e2 00 00 00 65 fa 8d a9 d7 1f c0 bd 56 66 b8 9b 82 >...e..®Î. LçVf©øé<
0007f2 a3 4f c2 08 00 00 00 77 15 82 93 3c a1 f3 dd 14 >ú0T...w.éo<1%!.<
000802 00 00 00 9a 95 90 10 ea ad c8 f6 56 25 f8 2c 69 >...Ûð..Ûj L÷V%°,i<
000812 f0 5b 7c 87 31 de 4c >[|ç1ÎL<
000819
```

117. <http://msdn.microsoft.com/en-us/library/ms995355.aspx>

118. <http://dpapick.com/documentation>

Extraire le bloc DPAPI associé à la clé privée (de l'octet 0x017f ou 383, sur une longueur de ou 1552 octets) :

```
> dd if=<fichier clé privée> bs=1 skip=383 count=1522 of=pkey.blob
1522+0 enregistrements lus.
1522+0 enregistrements écrits.
1522 bytes (1,5 kB) copied, 0,0467547 seconds, -0,0 kB/s
```

Ce bloc peut être déchiffré à l'aide de la DPAPI. À titre d'exemple, coder la clé privée ci-dessus en Base64.

```
$ openssl base64 -in pkey.blob -out pkey.blob.b64
```

Créer un fichier `decrypt_privkeyblob_dpapi.cs`, y copier le contenu du code C# d'exemple publié sur ce site¹¹⁹, et y remplacer la méthode `Main` de la classe `DPAPITest` par les lignes suivantes :

```
[STAThread]
static void Main(string[] args) {
    try {
        string description;

        // private key blob
        string encrypted = @"AQAAANCMnd8BFdERjHoAwE/Cl+sBAAA6XW21lmsyOCp...
        ... reste du contenu du fichier pkey.blob.b64 ...
        kYYtLxpH/ABDbBQAAACJ9wtK01rPy8WnqcfpN1VNHJbEg==" ;

        // Call DPAPI to decrypt data.
        string decrypted = DPAPI.Decrypt(    encrypted,
                                            out description);

        Console.WriteLine(decrypted) ;
    }
    catch (Exception ex) {
        while (ex != null) {
            Console.WriteLine(ex.Message);
            ex = ex.InnerException;
        }
    }
}
```

Compiler ce code à l'aide de `csc` et exécuter la commande `decrypt_privkeyblob_dpapi.exe` pour déchiffrer la clé privée chiffrée, en notant que Windows demande l'autorisation de l'utilisateur pour accéder à la clé privée.

119. <http://www.obviex.com/samples/dpapi.aspx>

Annexe F — Code source

Cette annexe s'adresse aux lecteurs souhaitant se familiariser avec les bibliothèques sous-jacentes aux outils cryptographiques utilisés dans ce document, qui permettent tirer parti de bien plus de fonctionnalités et de paramètres que n'offrent les outils en ligne de commande.

F.1. Paramétrage de l'algorithme de signature des jetons OCSP produits par OpenSSL

Les jetons OCSP produits par `openssl ocsp` sont signés avec SHA-1 car dans le code source de cette commande (`apps/ocsp.c` du répertoire source d'OpenSSL), la fonction invoquée pour signer le jeton OCSP est :

```
OCSP_basic_sign(bs, rcert, rkey, NULL, rother, flags);
```

où `NULL` désigne l'algorithme de hachage à utiliser, soit l'algorithme de hachage par défaut associé à l'algorithme de chiffrement asymétrique (en l'occurrence RSA), défini dans `crypto/rsa/rsa_ameth.c`, c'est-à-dire SHA-1 :

```
static int rsa_pkey_ctrl(EVP_PKEY *pkey, int op, long arg1, void *arg2)
{
...
    case ASN1_PKEY_CTRL_DEFAULT_MD_NID:
        *(int *)arg2 = NID_sha1;
        return 1;
...
}
```

Cette annexe s'intéresse à la possibilité d'utiliser l'option `-<dgst>` pour spécifier l'algorithme de hachage de l'algorithme de signature.

Dans un répertoire vide, copier depuis le répertoire source d'OpenSSL les fichiers suivants :

- `apps/ocsp.c`, source de la commande `openssl ocsp`,
- `apps/apps.c` et `apps/apps.h`, source des fonctions auxiliaires des commandes `openssl` et en-tête associée,
- `e_os.h`, fichier d'en-tête définissant les spécificités liées aux environnements système.

Modifier, dans le fichier `ocsp.c` nouvellement créé, le prototype de la fonction `make_ocsp_response`, en ajoutant un dernier paramètre définissant l'algorithme de hachage (en gras ci-dessous). Cette modification est à effectuer deux fois : une fois lors de la déclaration de la fonction (ligne terminant par « ; ») et une fois lors de son implémentation (ligne suivie de « { » et du code de la fonction).

```
static int make_ocsp_response(OCSP_RESPONSE **resp, ...
    int nmin, int ndays, const EVP_MD *ocsp_id_md)
```

Modifier les paramètres d'appel de la fonction (faire une recherche sur `make_ocsp_response`) :

```
i = make_ocsp_response(&resp, ..., ndays, cert_id_md);
```

Dans le corps de la fonction, remplacer, lors de l'appel à `OCSP_basic_sign`, le paramètre `NULL` par le nouveau paramètre `ocsp_id_md` définissant l'algorithme de hachage à utiliser lors de la signature.

```
OCSP_basic_sign(bs, rcert, rkey, ocsp_id_md, rother, flags);
```

À la fin du fichier, ajouter les blocs de code suivants issus de `apps/ca.c` du répertoire source d'OpenSSL :

```
static const char *crl_reasons[] = {
    ...
};

#define NUM_REASONS (sizeof(crl_reasons) / sizeof(char *))

int unpack_revinfo(ASN1_TIME **prevtm, ..., const char *str)
{
    ...
}
```

Compiler le code source. Quelques exemples de commandes de compilation sont proposés ci-après.

Pour `gcc` de MinGW, en environnement Win32 :

```
> set OPENSSL_DIR=<répertoire d'installation d'OpenSSL>
> gcc -o ocsp.exe ocsp.c apps.c %OPENSSL_DIR%\include\openssl\applink.c \
    -I%OPENSSL_DIR%\include -L%OPENSSL_DIR%\bin -leay32 -lssl32 \
    -lwsck32
```

Pour l'invite de commande de Visual Studio Express sous Windows :

```
> cl ocsp.c apps.c %OPENSSL_DIR%\include\openssl\applink.c \
    /I %OPENSSL_DIR%\include -DOPENSSL_SYSNAME_WIN32 -DWIN32_LEAN_AND_MEAN \
    /link /LIBPATH:%OPENSSL_DIR%\lib libeay32.lib ssl32.lib wsck32.lib
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 16.00.30319.01 for 80x86
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
ocsp.c
apps.c
applink.c
```



```
Generating Code...
Microsoft (R) Incremental Linker Version 10.00.30319.01
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
/out:ocsp.exe
/LIBPATH:C:\Local\apps\openssl-1.0.1b\lib
libeay32.lib
ssleay32.lib
wsock32.lib
ocsp.obj
apps.obj
applink.obj
    Creating library ocsp.lib and object ocsp.exp
```

Pour un environnement UNIX/Linux (en utilisant le répertoire /opt/openssl-1.0.1a proposé dans l'annexe sur la compilation d'OpenSSL) :

```
$ export OPENSSL_DIR=/opt/openssl-1.0.1a
$ gcc -o ocsp ocsp.c apps.c -I$OPENSSL_DIR/include -L$OPENSSL_DIR/lib -lssl \
    -lcrypto
```

Des plaintes relatives à des références indéfinies à dl... (ex. : dlopen, dlsym) indiquent que les bibliothèques dynamiques d'OpenSSL ne sont pas disponibles (l'option shared de ./config a sans doute été oubliée à la compilation, et une recompilation d'OpenSSL est alors nécessaire).

L'exécutable généré peut être invoqué avec les mêmes options qu'openssl ocsp, auxquelles s'ajoute -<dgst> pour spécifier l'algorithme de hachage (par exemple -sha256).

En environnement Linux (des modifications analogues étant à effectuer dans les autres environnements UNIX, comme suggéré dans le fichier doc/openssl-shared.txt du répertoire source d'OpenSSL), il convient de préciser dans la variable d'environnement LD_LIBRARY_PATH le chemin du répertoire dans lequel sont installées les bibliothèques dynamiques d'OpenSSL, par exemple :

```
$ LD_LIBRARY_PATH=$OPENSSL_DIR/lib ./ocsp ...
```

S'assurer que le remplacement de openssl ocsp par le nouvel ocsp avec l'option -sha-256 retourne bien un jeton signé avec l'algorithme attendu.

F.2. Génération d'un fichier test_ev_roots.txt pour Firefox

L'outil dont le code source est proposé dans cette annexe permet de générer le contenu d'un fichier test_ev_roots.txt pour une AC racine (identifiée par son certificat) et un OID de PC EV SSL avec la syntaxe suivante :

```
$ gen_test_ev_roots tls-rootca-crt.pem \
    1.2.840.113556.1.8000.2554.29563.49294.43847.16581.44480.6974059.2493436.1.3
1_fingerprint 43:90:8F:13:8A:57:68:BD:6D:7A:84:26:4A:F5:7A:D2:54:91:D3:18
2_readable_oid 1.2.840.113556.1.8000.2554.29563.49294.43847.16581.44480.6974059.
2493436.1.3
```

F. Code source

```
3_issuer MFwxCzAJBgNVBAYTAkZSMRYwFAYDVQQKEw1Nb24gT3JnYW5pc21lMRcwFQYDVQQLEw4wMDA
yIDE0NzI1ODM2OTEcMBoGA1UECzMtT3B1b1NTTCBUTFMgUm9vdCBDQQ==
4_serial 37e0onwlt0=
```

Voici le code C du fichier source `gen_test_ev_roots.c`.

```
#include <stdio.h>
#include <openssl/bio.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/bn.h>
#include <openssl/asn1.h>
#include <openssl/x509v3.h>

#include "gen_test_ev_roots.h"

int main (int argc, char **argv) {
    int res = 1 ;
    BIO *err=NULL;
    BIO *cert=NULL;
    BIO *out=NULL;
    X509 *x=NULL;

    /* Check command-line syntax */
    if (argc != 3) {
        fprintf (stderr, "Error - Expected syntax is:\n") ;
        fprintf (stderr,
            "gen_test_ev_roots <EV OID> <PEM-encoded CA certificate file>\n") ;
        res = 1 ;
        goto end ;
    }

    /* Initialise stderr/stdout BIO */
    err = BIO_new_fp(stderr,BIO_NOCLOSE);
    out = BIO_new_fp(stdout,BIO_NOCLOSE);

    /* Initialise input file BIO */
    if ((cert=BIO_new(BIO_s_file())) == NULL) {
        ERR_print_errors(err);
        goto end;
    }

    /* Open file for reading */
    if (!BIO_read_filename(cert,argv[1])) {
        BIO_printf(err, "Error - Cannot open file %s\n", argv[1]);
        ERR_print_errors(err);
        goto end;
    }

    /* Read PEM-encoded certificate */
    x = PEM_read_bio_X509(cert, NULL, NULL, NULL);
    if (x == NULL) {
```

```

    BIO_printf(err,
        "Error - Cannot read certificate (hint: check content of file %s)\n",
        argv[1]);
    ERR_print_errors(err);
    goto end;
}

/* Output each line */
if (res = output_fingerprint (out, err, x)) goto end ;
if (res = output_OID (out, err, argv[2])) goto end ;
if (res = output_issuer (out, err, x)) goto end ;
if (res = output_serialNumber (out, err, x)) goto end ;

end:
    if (x != NULL) X509_free(x);
    if (cert != NULL) BIO_free(cert);
    return res ;
}

int output_fingerprint (BIO *out, BIO *err, X509 *x) {
    int j;
    unsigned int len;
    unsigned char md[EVP_MAX_MD_SIZE];

    if (!X509_digest(x,EVP_sha1(),md,&len)) {
        BIO_printf(err,
            "Error - Out of memory while calculating certificate fingerprint\n");
        return 1 ;
    }

    BIO_printf(out, "1_fingerprint ") ;
    for (j=0; j<len; j++) {
        BIO_printf(out,"%02X%c",md[j], (j+1 == (int)len)?'\n':':');
    }

    return 0 ;
}

int output_OID (BIO *out, BIO *err, char *oid) {
    if (OBJ_txt2obj(oid, 1) == NULL) {
        BIO_printf(err,"Error - Invalid OID %s\n", oid);
        return 1 ;
    }
    BIO_printf(out, "2_readable_oid %s\n", oid) ;
    return 0 ;
}

int output_issuer (BIO *out, BIO *err, X509 *x) {
    X509_NAME *issuer ;
    BIO *b64 ;

```

F. Code source

```
if ((b64 = BIO_push(BIO_new(BIO_f_base64()), out)) == NULL) {
    BIO_printf(err,
        "Error - Cannot initialise Base64 output filter for issuer\n");
    return 1 ;
}
BIO_set_flags(b64, BIO_FLAGS_BASE64_NO_NL) ;

BIO_printf(out, "3_issuer ") ;

issuer = X509_get_issuer_name(x) ;
ASN1_item_i2d_bio(ASN1_ITEM_rptr(X509_NAME), b64, issuer);
BIO_flush(b64);

BIO_printf(out, "\n") ;
return 0 ;
}

int output_serialNumber (BIO *out, BIO *err, X509 *x) {
    ASN1_INTEGER *serial ;
    BIO *b64 ;
    BIGNUM *bn = NULL ;
    unsigned char *binserial = NULL ;
    int len ;
    int res = 1 ;

    if ((b64 = BIO_push(BIO_new(BIO_f_base64()), out)) == NULL) {
        BIO_printf(err,
            "Error - Cannot initialise Base64 output filter for serial number\n");
        goto end ;
    }
    BIO_set_flags(b64, BIO_FLAGS_BASE64_NO_NL) ;

    serial = X509_get_serialNumber(x) ;
    bn = ASN1_INTEGER_to_BN (serial, NULL) ;
    len = BN_num_bytes(bn);
    binserial = malloc(len) ;
    if (BN_bn2bin(bn, binserial) != len) {
        BIO_printf(err,
            "Error - Cannot get a binary representation of serial number\n");
        goto end ;
    }

    BIO_printf(out, "4_serial ") ;
    /* Assuming that serial number is positive, prepend 0x00 if first byte
       of serial number is >= 0x80 */
    if (binserial[0] & 0x80) {
        BIO_write(b64, "\0", 1);
    }
    BIO_write(b64, binserial, len);
    BIO_flush(b64);
}
```

```

    BIO_printf(out, "\n") ;

    res = 0 ;
end:
    if (bn != NULL) BN_free (bn) ;
    if (binserial != NULL) free (binserial) ;
    return res ;
}

```

Le code ne présente pas de difficulté de compréhension particulière : il s'appuie essentiellement sur les fonctions proposées par la libcrypto d'OpenSSL. La production par la fonction `output_serialNumber()` de la ligne correspondant au numéro de série mérite toutefois quelques explications.

La valeur à inclure pour le champ `4_serial` n'est pas le numéro de série à proprement parler mais le contenu des octets de valeur du codage DER de l'INTEGER représentant le numéro de série, qui nécessite de tenir compte d'une petite particularité liée à la gestion des nombres négatifs. Si le premier octet de la représentation hexadécimale du numéro de série est supérieur ou égal à `0x80`, alors, pour être considéré comme un nombre positif, il doit être précédé, dans la représentation DER de sa valeur, par un octet `0x00`. Ainsi, la représentation DER du champ de valeur du nombre décimal 127 est `0x7F`, et celle de 128 est `0x00 0x80` (tandis que `0x80` correspond à -128).

Des exemples de codage DER d'entiers sont proposés par Burton S. Kaliski Jr. dans la section 5.7 de son document *A Layman's Guide to a Subset of ASN.1, BER, and DER*¹²⁰.

Ainsi, en supposant que le numéro de série du certificat est un entier positif (ce qui est imposé par la RFC 5280), un octet `0x00` est ajouté en début de numéro de série.

Créer le fichier d'en-tête `gen_test_ev_roots.h` suivant.

```

#include <openssl/bio.h>
#include <openssl/x509.h>

int output_fingerprint (BIO *out, BIO *err, X509 *x) ;
int output_OID (BIO *out, BIO *err, char *oid) ;
int output_issuer (BIO *out, BIO *err, X509 *x) ;
int output_serialNumber (BIO *out, BIO *err, X509 *x) ;

```

Compiler le code à l'aide de l'une des lignes de commande suivantes.

Pour gcc de MinGW, en environnement Win32 :

```

> set OPENSSL_DIR=<répertoire d'installation d'OpenSSL>
> gcc -o gen_test_ev_roots.exe gen_test_ev_roots.c -I%OPENSSL_DIR%\include \
  -L%OPENSSL_DIR%\bin -leay32

```

Pour l'invite de commande de Visual Studio Express sous Windows :

```

> set OPENSSL_DIR=<répertoire d'installation d'OpenSSL>
> cl gen_test_ev_roots.c /I %OPENSSL_DIR%\include /link \
  /LIBPATH:%OPENSSL_DIR%\lib libeay32.lib

```

120. <http://luca.ntop.org/Teaching/Appunti/asn1.html>

Pour un environnement UNIX/Linux (en utilisant le répertoire `/opt/openssl-1.0.1a` proposé dans l'annexe sur la compilation d'OpenSSL) :

```
$ export OPENSSL_DIR=/opt/openssl-1.0.1a
$ gcc -o gen_test_ev_roots gen_test_ev_roots.c -I$OPENSSL_DIR/include \
  -L$OPENSSL_DIR/lib -lcrypto
```

Se reporter à la fin de l'annexe F.1 pour des explications sur l'exécution de fichier résultant avec une bibliothèque dynamique située dans un répertoire non standard.

Annexe G — Considérations légales

G.1. Conditions d'utilisation de ce document

Le texte de ce document est disponible sous licence *Creative Commons* paternité partage à l'identique¹²¹, à l'exception des éléments recensés ci-dessous, soumis à des conditions d'utilisation spécifiques non compatibles avec cette licence.

Les captures d'écran des produits Microsoft sont utilisées avec l'autorisation de Microsoft¹²².

Les extraits de documents normatifs sont cités conformément à l'article 10.1 de la convention de Berne¹²³.

L'utilisation de code source dérivé du code source d'OpenSSL est soumis aux exigences ci-dessous.

Copyright (c) 1999 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:
"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.OpenSSL.org/>)"
4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact licensing@OpenSSL.org.
5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following

121. <http://creativecommons.org/licenses/by-sa/3.0/deed.fr>

122. <http://www.microsoft.com/About/Legal/EN/US/IntellectualProperty/Permissions/Default.aspx>

123. http://www.wipo.int/treaties/fr/ip/berne/trtdocs_wo001.html

acknowledgment:

"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.OpenSSL.org/>)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

G.2. Droit des marques

Toutes les marques déposées sont la propriété de leurs détenteurs respectifs, en particulier :

- Microsoft, Windows et Internet Explorer sont des marques déposées de Microsoft Corporation¹²⁴ aux États-Unis et dans d'autres pays.
- UNIX est une marque déposée de The Open Group¹²⁵.
- Mac OS et Apple sont des marques déposées d'Apple Computer Inc¹²⁶.
- Adobe et Reader sont des marques déposées d'Adobe System Incorporated¹²⁷.
- RSA Security est une marques déposée d'EMC Corporation¹²⁸.

Les symboles « TM » et « [®] » n'ayant aucune portée juridique en France¹²⁹, ils ont été omis des noms de marque ci-dessus. Ils sont à restituer pour toute adaptation de ce document dans un pays où ils ont cours (en particulier dans les pays soumis à la *common law*).

G.3. Usage de la cryptographie

Les lois régissant les conditions d'utilisation de la cryptographie variant en fonction des pays, l'auteur recommande vivement au lecteur de se renseigner sur les lois qui lui sont applicables (le panorama des lois cryptographiques¹³⁰ de Bert-Jaap Koops est un bon point de départ), et ne pourra nullement être tenu responsable de toute infraction commise par le lecteur dans ce domaine.

124. <http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx>

125. <http://www.unix.org/trademark.html>

126. <http://www.apple.com/legal/trademark/appletm.html>

127. <http://www.adobe.com/misc/trade.html>

128. <http://www.emc.com/legal/emc-corporation-trademarks.htm#rsa>

129. www.inpi.fr/fr/questions-faq/liste-des-questions/faq_categorie/autres-questions415.html

130. <http://rechten.uvt.nl/koops/cryptolaw/>

G.4. Attributions

Ce document s'appuie sur du code et des applications cryptographiques écrits par Eric A. Young (eay@cryptsoft.com) et Tim J. Hudson (tjh@cryptsoft.com).

This product includes cryptographic software written by Eric A. Young (eay@cryptsoft.com). This product includes software written by Tim J. Hudson (tjh@cryptsoft.com).

Annexe H — Acronymes et sigles

3DES

Triple DES

3TDEA

Triple Data Encryption Algorithm, keying option 1

AC

autorité de certification

AES

Advanced Encryption Standard

ANSI

American National Standards Institute

API

Application Programming Interface

ASCII

American Standard Code for Information Interchange

ASN.1

Abstract Syntax Notation One

BER

Basic Encoding Rules

BES

Basic Electronic Signature

BOM

byte order mark

CA

certificate Authority

CAdES

CMS Advanced Electronic Signature

CBC

cipher-block chaining

H. Acronymes et sigles

CMS

Cryptographic Message Syntax

CN

Common Name

CRL

certificate revocation list

CSP

cryptographic service provider

CSR

certificate signature request

DER

Distinguished Encoding Rules

DES

Data Encryption Standard

DN

Distinguished Name

DPAPI

Data Protection API

DSA

Digital Signature Algorithm

ERS

Evidence Record Syntax

EV

Extended Validation

EXSLT

extensions to XSLT (non officiel)

FAI

fournisseur d'accès à Internet

FAQ

Frequently Asked Questions ou foire aux questions

GOST

standards d'État, du russe *ГОСТ* pour *Государственный стандарт*

GUID

Globally Unique IDentifier

HTTPS

Hypertext Transfer Protocol Secure

ICP

infrastructure à clés publiques

IETF

Internet Engineering Task Force

IGC

infrastructure de gestion de clés

IIS

Internet Information Services

IP

Internet Protocol

ITU

International Telecommunication Union

LCR

liste de certificats révoqués

MIME

Multipurpose Internet Mail Extensions

OCSP

Online Certificate Status Protocol

OID

Object IDentifier

OU

Organizational Unit

PAdES

PDF Advanced Electronic Signature

PC

politique de certification

PEM

Privacy Enhanced Mail

H. Acronymes et sigles

PKCS

public-key cryptography standard

PKI

Public Key Infrastructure

RDN

Relative Distinguished Name

RFC

Request For Comment

RGS

référentiel général de sécurité

RSA

Rivest, Shamir, Adleman

SHA

Secure Hash Algorithm

SMIME

Secure/Multipurpose Internet Mail Extensions

SSL

Secure Sockets Layer

SVN

Apache Subversion

TLS

Transport Layer Security

TSP

Time-Stamp Protocol

URL

uniform resource locator

UTC

Temps universel coordonné

UTF-8

Universal Character Set (UCS) transformation format 8 bits

UUID

universally unique identifier

WYSIWYS

What You See Is What You Sign

XAdES

XML Advanced Electronic Signature

XER

XML Encoding Rules

XML

Extensible Markup Language

XSLT

eXtensible Stylesheet Language Transformations

Colophon

Consultant puis chef de projet, Sébastien Pujadas a découvert la confiance électronique au début des années 2000. Il a également été traducteur, auteur et relecteur pour les éditions O'Reilly France. Son profil professionnel complet est publié sur LinkedIn : <http://www.linkedin.com/in/spujadas>

La source de ce document est en XHTML¹³¹, et la mise en forme est assurée à l'aide d'une CSS¹³² initialement construite à partir d'un fichier d'exemple¹³³ d'utilisation de Prince.

La cohérence des renvois de section et de page est vérifiée à l'aide d'un schéma Schematron¹³⁴ (`validatedoc.sch`), dont le résultat est mis en forme par une transformation XSLT (fichier `validatedoc.xslt`), à l'aide de la ligne de commande suivante (les autres fichiers sont ceux de Schematron) :

```
$ xsltproc schematron/iso_dsdl_include.xsl validatedoc.sch \
  | xsltproc schematron/iso_abstract_expand.xsl - \
  | xsltproc schematron/iso_svrl_for_xslt1.xsl - \
  | xsltproc --nonet - tp-confiance.xhtml \
  | xsltproc validatedoc.xslt -
```

La version PDF de ce document est générée à partir des fichiers XHTML et CSS en utilisant l'outil Prince¹³⁵ et la ligne de commande suivante :

```
$ prince --no-author-style --javascript -s doc.css tp-confiance.xhtml \
  -o tp-confiance.pdf
```

La production du fichier PDF par Prince inclut la génération automatisée de la table des matières à l'aide d'un script JavaScript (`toc.js`) adapté d'un message¹³⁶ posté sur le forum de Prince.

Le texte courant est composé en Gentium¹³⁷, le code est composé en Latin Modern¹³⁸.

Ce document a été rédigé de mars à juillet 2012, puis finalisé et publié en octobre 2013 par Sébastien Pujadas.

131. <http://www.w3.org/TR/xhtml1/>

132. <http://www.w3.org/TR/CSS2/>

133. <http://www.princexml.com/howcome/2008/wikipedia/wiki2.css>

134. <http://www.schematron.com/>

135. <http://www.princexml.com/>

136. <http://www.princexml.com/bb/viewtopic.php?f=5&t=14898>

137. <http://scripts.sil.org/gentium>

138. http://www.gust.org.pl/projects/e-foundry/latin-modern/index_html