NOTES SUR OPENSSL : PARTIE II

par

Ainigmatias Cruptos

Résumé. — Dans cette partie II, nous décrivons (en résumé), l'organisation générale d'un implémentation complète d'une infrastructure basée sur openSSL. Notamment l'établissement du certificat X.509 d'une autorité, d'une autorité secondaire, d'un utilisateur. Nous expliquons une utilisation finale, notamment la gestion pratique des certificats X.509 et pkcs12.

1. Utilisation élaborée de OpenSSL

1.1. Principe du fonctionnement. — La première fonctionnalité qu'on va demander au système est de pouvoir assurer l'authentification des participants. Un individu va devoir s'authentifier et authentifier une action qu'il accomplit en signant un message : par exemple signature d'un courrier électronique ou encore dans d'autres situations, signature d'un message (challenge) envoyé par le correspondant. Chaque utilisateur doit donc posséder une biclé d'un système à clé publique, et signer avec la clé privée, la clé publique servant aux correspondants à vérifier la signature. Encore faut-il que les correspondants aient une assurance que la clé publique qu'ils vont utiliser est bien la bonne et n'ait pas fait l'objet d'une attaque, auquel cas il pourrait y avoir usurpation d'identitée.

Mots clefs. — cryptographie, protocole cryptographique, SSL,TLS, signature, chiffrement, base64, certificats.

Pour assurer cette sécurité, chaque clé publique, va être encapsulée dans un certificat qui la décrit soigneusement, et qui est signé (après une soigneuse vérification de l'identité et plus généralement de toutes les informations présentes dans le certificat) par une hiérarchie d'autorités (chaque maillon de la hiérarchie signe les clés publiques des participants qui sont juste en dessous de lui). Bien entendu la question d'une autorité racine se pose. L'autorité racine possèdera un certificat contenant sa clé publique, qui sera auto-signé. Cette autorité possède une notoriété suffisante pour que la validité de sa clé publique puisse être facilement établie et ne soit pas mise en doute.

Comme on le voit, la notion de certificat est centrale dans ce processus. Nous donnons ici en syntaxe ASN1, la forme générale d'un certificat X.509.

```
Certificate ::= SIGNED SEQUENCE{
        version [0]     Version DEFAULT v1988,
        serialNumber    CertificateSerialNumber,
        signature       AlgorithmIdentifier,
        issuer          Name,
        validity        Validity,
        subject         Name,
        subjectPublicKeyInfo    SubjectPublicKeyInfo}

Version ::=     INTEGER {v1988(0)}

CertificateSerialNumber ::=     INTEGER

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

SubjectPublicKeyInfo ::=        SEQUENCE{
        algorithm               AlgorithmIdentifier,
        subjectPublicKey        BIT STRING}

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

1.2. La stratégie à établir. — Voici donc le plan que nous avons à prévoir si nous voulons mettre en place un système complet cohérent.

(1) La gestion de l'autorité.

(a) Le travail à faire par l'autorité.
(b) Le certificat de l'autorité racine.
(c) Les certificats des sous-autorités

(2) Le point de vue de l'utilisateur.
(a) Faire une requête à l'autorité
(b) Récupérer son propre certificat X.509
(c) Importer le certificat X.509 de l'autorité
(d) Importer le certificat X.509 d'un correspondant
(e) Faire un certificat pkcs12
(f) Utiliser les certificats dans les applications

2. La gestion de l'autorité

2.1. Le travail à faire par l'autorité. —

2.1.1. Au départ. — L'autorité va devoir mettre en place un environnement de travail pour pouvoir signer les certificats des autorités secondaires. Les autorités secondaires devront ensuite signer les certificats des utilisateurs (clients au sens commercial du terme). L'autorité devra gérer la numérotation des certificats attribués, la liste de ces certificats, ainsi que la liste de révocation de certains certificats

À titre d'exemple voici l'organisation à peu près standard du répertoire de l'autorité (mais évidemment cette organisation standard est très simplifiée par rapport à une gestion réelle).

On va fixer un répertoire de travail dont le nom sera attribué à la variable $dir dans les fichiers de configuration. Ce répertoire pourra être par exemple :

/home/autorite/tls

Dans ce répertoire on va créer les sous-répertoires suivantes :

certs	Répertoire où l'on conserve tous les certificats
crl	Répertoire où sont conservées les listes de révocation
newcerts	Répertoire des certificats créés sous forme numérotée (sous une autre dénomination donc que ceux de certs)
private	Répertoire où se trouvent les clés privées des autorités

On va aussi disposer d'un certain nombre de fichiers textes :

index.txt	fichier d'index de la base de donnée des certificats
serial	contient le numéro de série à attribuer au certificat suivant
crlnumber	contient le numéro de série à attribuer au prochain certificat de révocation

Au départ les répertoires sont vides, le fichier index est vide les fichiers **serial** et **crlnumber** sont vides.

Bien entendu à la mise en place du système, l'autorité racine doit créer une biclé, un certificat X.509 à son nom, auto-signé, contenant sa clé publique, tandis que sa clé privée reste dans le répertoire **private** de son environnement de travail.

Dans les exemples, nous nommerons **cacert.pem** le certificat X.509 de l'autorité racine, et **cakey.pem** la clé privée de l'autorité racine.

2.1.2. En fonctionnement normal. — L'autorité devra signer des certificats pour des autorités secondaires (pas souvent) et pour des utilisateurs, signer aussi des certificats de révocation de clés. Pour cela, l'autorité doit recevoir une requête, vérifier que cette requête est bien valide (vérifier que l'identité du demandeur est la bonne, que la signature de la requête est correcte compte tenu de la clé publique déclarée par le demandeur), et générer un certificat X.509 qu'elle signe.

Remarque 2.1. — Les instructions OpenSSL permettent diverses commandes, chacune ayant de multiples options. On peut effectivement donner toutes les options nécessaires dans la ligne de commande. Mais on peut aussi récupérer un certain nombre de ces options dans un fichier de configuration qui précise un certain nombre de choses qu'on va reproduire à chaque fois. Par exemple on peut mettre dans le fichier de configuration la taille des clés, le nombre de jours de validité, l'endroit où on va stocker telle ou telle sortie. Nous donnons un certain nombre de fichiers de configuration dans une archive [**1**].

2.2. Générons le certificat auto-signé de l'autorité. — L'instruction suivante fait à la fois une requête de certificat et aussi l'autosignature de la clé. Elle est donnée depuis le répertoire racine (dans notre cas **/home/autorite/tls**).

On remarquera l'option *-config root-ca-cert.cnf* qui permet l'usage du fichier (donné en annexe et présent dans l'archive [**1**]).

```
$ openssl req -x509 -config root-ca-cert.cnf
 -new -out certs/cacert.pem -days 3650
```

On remarquera aussi l'option *-days 3650*. Elle permet de créer une clé dont la durée de vie est 10 ans. La réponse interactive est la suivante.

```
Generating a 2048 bit RSA private key
.....................................................................................
..+++
writing new private key to 'private/cakey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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.
-----
Pays [FR]:
Etat, province ou departement [PACA]:
Ville [Marseille]:
Organisation [ACrypTA]:
Unite organisationnelle [Certification]:
Nom commun [acrypta_ca]:
Adresse mail []:acrypta@acrypta.fr
```

Le certificat de l'autorité est créé sous le nom **cacert.pem** dans le répertoire **certs**. La clé est une clé de 2048 bits ainsi que spécifié dans le fichier de configuration (ce qui fait que le système à clé publique choisi est automatiquement RSA, du fait que DSA est limité à 1024 bits. On aurait pu préciser autre chose avec par exemple l'option *-newkey rsa :1024* à la place de *-newkey*). La clé privée est sauvée comme précisé dans le fichier dde configuration dans le répertoire **private**, sous le nom de **cakey.pem**.

Remarquons que pour le moment le numéro de série (fichier **serial** n'a pas bougé.

Si on veut voir ce certificat on peut lancer la commande :

```
$ openssl x509 -text -in cacert.pem -noout
```

et on obtient pour réponse :

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            ae:8a:dd:5a:1b:41:0c:c7
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=FR, ST=PACA, L=Marseille, O=ACrypTA,
                OU=Certification,
                CN=acrypta_ca/emailAddress=acrypta@acrypta.fr
        Validity
            Not Before: Jan 31 17:11:00 2008 GMT
            Not After : Jan 28 17:11:00 2018 GMT
        Subject: C=FR, ST=PACA, L=Marseille, O=ACrypTA,
                 OU=Certification,
                 CN=acrypta_ca/emailAddress=acrypta@acrypta.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
              Modulus (2048 bit):
                00:ec:7e:d6:e1:73:7f:7a:b3:70:67:3f:44:69:8d:
                57:25:c9:f9:59:d8:81:71:03:b8:43:45:9d:e6:87:
                77:57:aa:bd:d8:bf:64:80:2b:c7:1e:2d:1b:ef:3a:
```

```
            42:58:91:de:33:02:bd:6a:cf:b0:af:44:0e:7e:28:
            bb:15:7c:47:c9:62:91:05:ef:76:5d:77:2d:fa:3d:
            85:94:ad:c7:7f:3f:c9:0c:ac:6c:d3:b5:5d:e2:29:
            dd:36:2f:1b:d9:f9:62:37:33:94:03:83:d7:0b:77:
            c8:e2:c7:22:74:41:60:1b:d1:91:c1:27:a8:97:39:
            99:ae:8e:a5:08:18:3b:ab:2a:f7:9f:25:5e:ac:f5:
            ab:d0:98:d8:0c:b9:7e:a1:60:c7:68:17:e3:69:f0:
            f6:a9:da:27:83:4d:2b:8c:c1:0c:17:13:92:a5:aa:
            c8:16:07:d0:87:0b:ec:39:64:9b:91:ec:48:59:f3:
            cb:0c:20:af:40:c8:bb:2f:ab:bf:32:1b:8b:16:30:
            15:0d:a8:fc:71:e3:8f:20:fd:a9:7b:46:72:b4:f0:
            b5:39:4d:67:a2:f5:4d:74:21:bd:bb:5f:3f:71:66:
            81:0a:db:89:1c:23:c9:b6:f4:c4:59:55:9b:27:c8:
            56:f2:72:b7:f4:b2:53:99:f1:bf:ec:63:12:49:9e:
            1c:e7
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints: critical
            CA:TRUE
        X509v3 Subject Key Identifier:
            BE:4D:DF:C8:1F:8E:2B:62:59:A6:
            FE:6A:22:54:B0:AA:6C:7D:93:D2
        X509v3 Key Usage: critical
            Certificate Sign, CRL Sign
        X509v3 Authority Key Identifier:
            keyid:BE:4D:DF:C8:1F:8E:2B:62:59:
            A6:FE:6A:22:54:B0:AA:6C:7D:93:D2
            DirName:/C=FR/ST=PACA/L=Marseille/O=ACrypTA/
           OU=Certification/
           CN=acrypta_ca/emailAddress=acrypta@acrypta.fr
            serial:AE:8A:DD:5A:1B:41:0C:C7

        Netscape Cert Type:
            SSL CA, S/MIME CA, Object Signing CA
        Netscape Comment:
            Certificat Racine. Genere par OpenSSL
Signature Algorithm: sha256WithRSAEncryption
    83:f4:f4:9f:ff:9f:14:98:b4:63:7c:d7:ee:3c:dc:cc:40:4b:
```

```
36:3e:92:49:cd:bf:80:e2:36:5a:16:e5:b6:94:ce:22:30:63:
c5:7a:f7:7b:55:e7:f0:ef:77:d3:8c:55:d7:9d:4b:64:6b:22:
f6:3f:44:00:85:da:1c:7b:ff:90:f4:98:c9:22:3d:2d:b6:48:
0a:82:32:38:32:c8:82:03:ed:5e:6b:65:a8:78:2c:26:b9:a7:
87:cc:16:18:d0:50:b1:cf:16:3d:6d:65:c4:99:af:2a:25:6e:
b5:2e:0b:45:88:b1:44:4d:78:45:d0:b8:70:29:c8:c9:38:76:
17:4e:93:a3:fd:c1:ea:3d:53:af:ba:1a:3a:ed:86:07:8a:e2:
08:d4:4b:69:f1:a4:8b:76:15:44:e3:45:57:30:3d:bb:72:19:
62:71:8e:43:de:20:a2:d2:37:7a:6c:31:eb:81:69:01:71:c5:
60:93:02:6d:3e:21:23:e9:13:2d:3b:c7:bd:c5:59:43:8e:fb:
d6:e3:cd:28:8e:37:8e:d9:4c:fc:7f:39:e5:bd:72:19:c1:e1:
db:36:41:81:0c:de:09:6d:89:80:db:db:c5:aa:19:fd:06:d3:
d7:7f:c1:7c:20:ec:6f:4c:ab:c4:e9:f5:a9:70:6c:17:4c:32:
fe:8f:21:63
```

On y voit les diverses informations qu'on a données lors de la création, on y voit aussi le module et l'exposant public (ce qui constitue la clé publique). On voit également la signature, dont on apprend qu'elle a été obtenue par hachage SHA256 (précisé dans le fichier de configuration par default_md=sha256) et par chiffrement RSA.

2.3. Génération du certificat d'une autorité secondaire. — Pour générer le certificat d'une autorité secondaire il faut tout d'abord faire une requête de certificat qui se soldera par la création de la clé privée de l'autorité secondaire qu'on notera **subcakey.pem**, et qui va être stockée dans le répertoire **private**. En outre un certificat provisoire est produit noté **tmpsubcacert.pem**, qui contient la clé publique et qui est signée avec la clé privée qui vient d'être construite **subcakey.pem**.

C'est la commande qui suit qui fait ce travail :

```
$ openssl req -config req-subca-cert.cnf
  -new -out tmpsubcacert.pem
```

La réponse est :

```
Generating a 2048 bit RSA private key
.......................+++
......................+++
```

```
writing new private key to 'private/subcakey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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.
-----
Pays [FR]:
Etat, province ou departement [PACA]:
Ville [Marseille]:
Organisation [ACrypTA]:
Unite organisationnelle [Certification]:
Nom commun [SubCA]:
Adresse mail []:a.cruptos@acrypta.fr
```

Il faut maintenant que ce fichier temporaire soit signé par l'autorité de certification pour obtenir enfin le certificat X.509 convoité.

```
$ openssl ca -config ca-subca-cert.cnf
  -in tmpsubcacert.pem -notext -out certs/subcacert.pem
```

La réponse interactive est :

```
Using configuration from ca-subca-cert.cnf
Enter pass phrase for /home/autorite/tls/private/cakey.pem:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :PRINTABLE:'PACA'
localityName          :PRINTABLE:'Marseille'
organizationName      :PRINTABLE:'ACrypTA'
```

```
organizationalUnitName:PRINTABLE:'Certification'
commonName             :PRINTABLE:'SubCA'
emailAddress           :IA5STRING:'a.cruptos@acrypta.fr'
Certificate is to be certified until
Jan  4 17:43:50 2013 GMT (1800 days)
Sign the certificate? [y/n]:y

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

On constate que le certificat est créé dans **certs** sous le nom de **subcacert.pem**, il est aussi à l'identique dans **newcerts**, sous le nom **01.pem**. Voyons quelle forme a ce certificat en le traduisant sous forme compréhensible par la commande :

```
$ openssl x509 -text -in subcacert.pem -noout
```

Le résultat est ce certificat :

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=FR, ST=PACA, L=Marseille,
                O=ACrypTA, OU=Certification,
                CN=acrypta_ca/emailAddress=acrypta@acrypta.fr
        Validity
            Not Before: Jan 31 17:43:50 2008 GMT
            Not After : Jan  4 17:43:50 2013 GMT
        Subject: C=FR, ST=PACA, L=Marseille,
                 O=ACrypTA, OU=Certification,
                 CN=SubCA/emailAddress=a.cruptos@acrypta.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
              Modulus (2048 bit):
                00:bb:c8:c4:68:88:80:42:8b:8d:49:b2:e0:66:18:
```

```
            8c:7a:3b:8b:e6:df:09:37:49:dc:8f:f9:3c:28:01:
            97:75:1c:ac:ee:97:da:f4:98:4f:7b:14:41:e1:7f:
            52:76:17:43:06:1e:94:a2:db:23:f3:a1:88:ca:9d:
            d5:39:f7:fa:db:5c:30:3b:2c:f3:5e:e2:66:ef:55:
            b2:ce:ee:54:59:68:b1:b9:2c:5f:84:3d:fe:26:31:
            74:86:da:30:6b:47:c4:80:93:d1:5b:4e:1f:45:31:
            a8:60:0e:80:e2:db:d0:fb:54:10:f7:5b:2a:31:46:
            fa:00:b5:94:b4:d5:d7:1a:b0:0b:f5:68:f2:68:67:
            89:f8:8e:ab:6d:a2:15:7d:bc:0d:3b:c5:9d:42:25:
            8f:6b:5c:b0:83:c5:23:b0:8c:17:82:d7:53:3c:6e:
            27:58:47:f6:c7:65:c5:99:d1:03:84:e8:df:74:78:
            8c:36:39:1d:a2:fc:cd:01:9e:e9:bf:e9:94:07:5e:
            07:3b:c0:67:3d:9f:72:47:51:d1:c8:0c:e7:9c:a5:
            26:73:8b:2c:61:7f:59:2c:46:99:fd:4f:45:80:a7:
            1c:92:81:89:7a:db:01:0e:08:b8:30:d8:47:fa:e8:
            99:69:32:6c:d0:3e:94:b2:12:2c:2a:84:36:20:95:
            f3:7f
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints: critical
            CA:TRUE
        X509v3 Authority Key Identifier:
            keyid:BE:4D:DF:C8:1F:8E:2B:62:59:A6:
                  FE:6A:22:54:B0:AA:6C:7D:93:D2
            DirName:/C=FR/ST=PACA/L=Marseille/
                  O=ACrypTA/OU=Certification/
                 CN=acrypta_ca/emailAddress=acrypta@acrypta.fr
            serial:AE:8A:DD:5A:1B:41:0C:C7

        X509v3 Subject Key Identifier:
            DF:8F:19:0A:66:CB:72:94:FD:
            62:FE:C2:35:53:A6:88:1F:FB:1F:61
        X509v3 Key Usage: critical
            Certificate Sign, CRL Sign
        Netscape Comment:
            Genere par OpenSSL
Signature Algorithm: sha256WithRSAEncryption
    05:da:73:f7:e2:62:fb:f1:bf:8a:23:2d:c7:82:91:e1:c3:26:
```

```
46:d8:b0:e0:b4:35:84:42:b1:0e:49:fe:91:d2:46:6c:8a:be:
48:c3:a1:4b:3e:9e:79:16:41:cc:82:28:32:d7:f5:d9:b4:6c:
4b:a4:0d:50:35:01:5a:f3:d2:c4:97:d0:56:b9:c4:38:83:32:
4d:ab:15:54:6f:f2:88:26:6f:44:b1:df:d0:aa:97:1e:40:14:
68:37:c2:2a:cb:9b:03:a0:74:05:b2:94:52:c8:c9:79:e2:59:
05:5a:5c:56:b5:cf:55:8f:57:b5:d0:25:f9:e0:66:39:29:95:
6a:4d:e4:ee:61:38:a3:8e:31:80:2b:b9:b7:c7:93:bd:65:f9:
7b:9f:e3:de:a0:76:bf:30:01:88:5d:80:f1:1c:9f:3c:60:a4:
55:8f:2e:e2:44:92:7a:b1:5c:b0:1c:0a:fe:b3:10:84:41:fd:
29:ac:91:c9:b5:e2:26:7a:c4:e5:94:76:61:4e:50:18:c2:4f:
9b:e2:32:c1:b3:27:b0:97:bf:ba:f8:b3:29:dc:2b:df:1c:1d:
0b:43:1c:b2:e0:57:3a:17:44:a8:ae:c4:af:08:5a:a1:30:1d:
ec:bb:bd:34:f7:f1:d7:6f:db:b4:16:07:4a:14:f2:f4:c8:f6:
7b:f2:56:2c
```

On remarquera que maintenant, le compteur serial a bougé. Il est à 02. La clé secrète de l'autorité secondaire a été stockée dans le répertoire **private**. Le fichier temporaire **tmpsubcacert.pem** peut maintenant être supprimé, ou stocké si on veut dans un historique.

3. Le point de vue de l'utilisateur

La création du certificat d'un utilisateur ressemble à celle du certificat d'une autorité secondaire.

3.1. La requête. — La requête se prépare localement, **sur la machine de l'utilisateur** (c'est ce qui devrait se faire en principe). L'autorité ne devrait pas disposer de la clé privée de l'utilisateur. Faisons donc cette opération dans les règles.

L'utilisateur dispose dans son répertoire de travail du fichier de configuration **req-user-cert.cnf**. Il a un sous-répertoire local qui s'appelle **private** (ce n'est évidemment pas le même répertoire que celui dont on a parlé lors de l'organisation du répertoire racine de l'autorité, et qui lui se trouve sur la machine de l'autorité). Il lance la commande :

```
$ openssl req -config req-user-cert.cnf
  -new -out tmpusercert.pem
```

La requête **tmpusercert.pem** est créé dans le répertoire de travail, alors que la clé privée **user.key** est créée dans le sous-répertoire **private**.

L'utiateur doit maintenant confier la requête, qui a, à peu près, la forme du certificat définitif, à part que pour le moment la signature est celle de l'utilisateur lui même avec la clé privée qu'il vient de créer. Voici cette requête obtenue avec la commande :

```
$ openssl req -text -in tmpusercert.pem -noout
```

```
Certificate Request:
   Data:
       Version: 0 (0x0)
       Subject: C=FR, ST=PACA, L=Marseille, O=Moi, OU=Moi,
                CN=Moi/emailAddress=Moi@Moi.fr
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (2048 bit)
               Modulus (2048 bit):
                   00:d0:df:89:c9:19:a9:5d:2d:cb:27:25:98:8d:c3:
                   42:5c:59:69:d7:8e:ab:b0:a0:96:af:3f:88:15:c7:
                   ea:26:2d:cc:86:89:16:d2:51:88:58:23:9f:f7:3b:
                   26:9c:ce:d9:d8:b6:42:a9:33:a1:f0:ba:ff:d2:82:
                   74:25:94:ff:fe:e9:83:b0:ad:e0:c7:6c:8c:9e:99:
                   98:b3:df:aa:6c:ed:da:57:06:a4:62:24:a2:78:4d:
                   da:c4:5f:64:ad:65:dd:8d:89:cd:f9:4e:86:24:d3:
                   f5:b1:2c:7a:92:92:0c:2b:bd:95:34:c9:e3:f0:02:
                   b9:55:c5:8a:68:92:48:5c:b9:9f:b5:bf:ec:f4:6d:
                   0d:54:6c:33:29:1a:b0:b6:a2:33:cc:9d:79:f3:2f:
                   1f:78:9c:0b:78:50:eb:dc:04:4b:e4:4d:5c:ed:a7:
                   b9:a9:94:78:b3:a0:4f:d5:8b:80:a6:0d:cc:6b:13:
                   e4:a0:ea:c7:32:dd:23:12:3c:0a:3c:c0:6a:b4:a0:
                   aa:31:06:86:76:ed:ef:9e:76:70:6a:b3:9d:dc:91:
                   9d:d3:8b:30:3e:6b:76:d5:e3:47:94:34:c4:8a:f1:
                   90:36:80:9d:2c:da:6e:f3:df:fb:bb:9d:c1:4c:01:
                   d5:ae:1d:cf:7d:09:e9:66:64:05:0d:a1:36:0b:1a:
                   2e:b9
               Exponent: 65537 (0x10001)
       Attributes:
```

```
        a0:00
    Signature Algorithm: sha256WithRSAEncryption
        28:e2:de:f1:96:8e:47:3e:63:a3:54:9b:cf:38:7f:85:ac:5c:
        c2:1d:1e:f2:7d:af:0f:93:f5:21:01:0b:03:3e:c2:63:24:74:
        eb:2c:1b:c5:d6:1c:90:54:30:cd:b4:09:d3:37:1a:38:62:04:
        e3:71:fe:12:73:53:26:b3:c4:a1:97:02:8a:fa:a1:b3:92:5e:
        f6:41:23:17:a5:63:88:4d:e2:cc:11:1c:56:0a:f1:1d:26:8a:
        27:60:f4:e9:28:08:04:0c:fc:45:97:a5:82:13:4d:5d:a3:be:
        fb:e7:a6:cc:0c:fa:74:25:82:4f:4c:62:1f:e3:30:e9:b8:0c:
        30:15:cc:03:3f:1f:b1:45:fa:0f:00:5e:1d:f1:06:cd:61:bd:
        e5:16:b2:b6:17:6f:1a:78:23:64:61:f4:27:4f:a7:85:01:f4:
        23:7f:9e:2c:8b:03:96:b2:00:4e:c7:1a:1e:50:ec:bf:db:31:
        54:f0:9f:f9:a1:43:0f:3d:e0:3f:5d:de:0c:73:f4:e4:91:14:
        8c:e3:70:5f:12:7f:db:e0:74:09:ed:21:16:a8:71:c9:28:5e:
        04:e6:82:76:97:2b:3d:47:e2:ab:9b:07:dc:9a:d4:51:5c:07:
        36:4d:ee:23:89:e3:73:6b:a7:64:f1:10:4b:5b:cf:c5:0f:33:
        e1:e8:4d:af
```

L'utilisateur fait donc parvenir de manière sûre ce certificat à l'autorité de certification (pas la clé privée).

L'autorité de certification s'assure de l'identité de l'utilisateur et signe la clé publique. Ici nous ferons signer cette clé par l'autorité secondaire.

```
$ openssl ca -config subca-user-cert.cnf
  -in tmpusercert.pem -notext -out certs/NomUsercert.pem
```

Le certificat signé **NomUsercert.pem** se retrouve donc dans **certs** et aussi sous le nom **02.pem** dans **newcerts**. L'index **serial** a avancé de 1.

Le certificat **NomUsercert.pem** doit être retransmis de manière sûre à l'utilisateur ainsi que les certificats des autorités habilitées à signer.

4. Un exemple d'application : le courrier électronique

4.1. Présentation. — Nous utiliserons Mozilla-Thunderbird comme client de messagerie. Mais le principe est toujours le même, et les autres clients doivent fonctionner à peu près pareil.

Dans Edit/Preference/advanced/certificates/view certificates, on a le choix de plusieurs rubriques : Your certificates, Other people's, Web sites, Authorities. Il va falloir commencer par importer les certificats des autorités.

4.2. Les certificats des autorités. — Choisir :
Edit/Preference/advanced/certificates/view certificates/Authorities,
demander l'importation d'un certificat, importer le fichier (que l'utilisateur doit avoir) **cacert.pem**. Faire pareil pour **subcacert.pem**. Désormais les autorités sont connues.

4.3. Le certificat pkcs12 de l'utilisateur. — Il faut maintenant importer la clé privée et la clé publique de l'utilisateur. Pour cela il ne suffit pas du certificat **NomUsercert.pem** qui ne contient que la clé publique signée. On doit mixer ce certificat avec la clé privée de l'utilisateur **user.key**. Ceci est réalisé sur la machine de l'utilisateur dans son répertoire de travail, par l'obtention d'un certificat pkcs12, qu'on crée par :

```
$ openssl pkcs12 -export -in NomUsercert.pem
  -inkey private/user.key -out  NomUser.p12 -name "NomUser"
```

Remarquons que lors de la commande le mot de passe relatif à la clé privée, qui a été donné lors de la création de celle-ci est demandé. Un mot de passe (qui peut être éventuellement le même) est demandé pour protéger le certificat pkcs12, qui va contenir cette clé privée sous forme chiffrée.

Une fois qu'on a le fichier **NomUser.p12**, on l'importe dans thunderbird en choisissant :
Edit/Preference/advanced/certificates/view certificates/Your certificates
et le bouton « import ».

4.4. Les certificats des correspondants. — Les certificats (clés publiques signées) des correspondants doivent aussi être importés, si on veut chiffrer des messages pour eux. La tehnique d'importation est la même.

5. Annexe

Les fichiers qui suivent peuvent être télécharger sur le site : http ://www.acrypta.fr. Ils doivent être adaptés (au moins la définition de la variable $dir). Ce sont des fichiers standards qu'on peut trouver un peu partout sur le web.

5.1. Fichier root-ca-cert.cnf. —

```
# pour generer un certificat root CA

[ req ]

default_bits        = 2048

default_keyfile     = private/cakey.pem

default_md          = sha256

distinguished_name  = req_distinguished_name

x509_extensions     = rootca_cert

[ req_distinguished_name ]

countryName             = Pays

countryName_default     = FR

countryName_min         = 2

countryName_max         = 2

stateOrProvinceName         = Etat, province ou departement

stateOrProvinceName_default = PACA

localityName         = Ville

localityName_default = Marseille

organizationName         = Organisation

organizationName_default = ACrypTA

organizationalUnitName         = Unite organisationnelle

organizationalUnitName_default = Certification
```

```
commonName          = Nom commun

commonName_default = acrypta_ca

commonName_max      = 64

emailAddress      = Adresse mail

emailAddress_max = 64


[ rootca_cert ]

# la section ci-dessous decrit les extensions a inclure dans un certificat rootCA

basicConstraints       = critical, CA:true

subjectKeyIdentifier   = hash

keyUsage               = critical, keyCertSign, cRLSign

authorityKeyIdentifier = keyid:always,issuer:always

nsCertType             = sslCA, emailCA, objCA

nsComment              = "Certificat Racine. Genere par OpenSSL"

# subjectAltName       = email:copy
```

5.2. Fichier req-subca-cert.cnf. —

```
# pour generer une requete de certificat CA intermediaire

[ req ]

default_bits         = 2048

default_keyfile      = private/subcakey.pem

default_md           = sha256

distinguished_name   = req_distinguished_name

x509_extensions      = subca_req

string_mask          = nombstr
```

```
[ req_distinguished_name ]

countryName         = Pays

countryName_default = FR

countryName_min     = 2

countryName_max     = 2


stateOrProvinceName         = Etat, province ou departement

stateOrProvinceName_default = PACA


localityName         = Ville

localityName_default = Marseille


organizationName         = Organisation

organizationName_default = ACrypTA


organizationalUnitName         = Unite organisationnelle

organizationalUnitName_default = Certification


commonName         = Nom commun

commonName_default = SubCA

commonName_max     = 64


emailAddress     = Adresse mail

emailAddress_max = 64
```

```
[ subca_req ]

basicConstraints        = critical, CA:true

subjectKeyIdentifier    = hash

authorityKeyIdentifier  = keyid, issuer:always

keyUsage                = critical, keyCertSign, cRLSign

# nsCertType            = sslCA, emailCA, objCA

# nsComment             = "Requete de signature de certificat"

# subjectAltName        = email:copy
```

5.3. Fichier req-user-cert.cnf. —

```
# poour generer une requete de certificat utilisateur

[ req ]

default_bits        = 2048

default_keyfile     = private/user.key

default_md          = sha256

distinguished_name  = req_distinguished_name

x509_extensions     = user_req

string_mask         = nombstr


[ req_distinguished_name ]

countryName         = Pays

countryName_min     = 2

countryName_max     = 2
```

```
stateOrProvinceName     = Etat, province ou departement

localityName            = Ville

organizationName        = Organisation

organizationalUnitName  = Unite organisationnelle

commonName              = Nom commun

commonName_max          = 64


emailAddress      = Adresse mail

emailAddress_max  = 64




[ user_req ]

basicConstraints         = critical, CA:false

subjectKeyIdentifier     = hash

keyUsage                 = digitalSignature, nonRepudiation, keyEncipherment

extendedKeyUsage         = clientAuth, emailProtection

nsCertType               = client, email

# nsComment              = "Requete de signature de certificat"


subjectAltName  = email:copy

#issuerAltName  =issuer:copy


#nsCaRevocationUrl =
```

```
#nsBaseUrl

#nsRevocationUrl =

#nsRenewalUrl

#nsCaPolicyUrl

#nsSslServerName
```

5.4. Fichier ca-subca-cert.cnf. —

```
# ca-subca-cert.cnf
[ ca ]

default_ca     = CA_default           # The default ca section


[ CA_default ]

dir            = /home/autorite/tls         # Where everything is kept

certs          = $dir/certs           # Where the issued certs are kept

crl_dir        = $dir/crl             # Where the issued crl are kept

database       = $dir/index.txt       # database index file.

new_certs_dir  = $dir/newcerts        # default place for new certs.


certificate    = $dir/certs/cacert.pem          # The CA certificate

serial         = $dir/serial          # The current serial number

crl            = $dir/crl.pem          # The current CRL

private_key    = $dir/private/cakey.pem  # The private key


RANDFILE       = $dir/private/.rand   # private random number file



default_days      = 1800     # how long to certify for
```

```
default_crl_days  = 30       # how long before next CRL

default_md        = sha256      # which md to use.

Preserve          = no       # keep passed DN ordering



x509_extensions   = subca_cert

copy_extensions   = none

policy            = policy_match



[ subca_cert ]

basicConstraints        = critical, CA:true

authorityKeyIdentifier  = keyid:always, issuer:always

subjectKeyIdentifier    = hash

keyUsage                = critical, keyCertSign, cRLSign

# nsCertType            = sslCA, emailCA, objCA

nsComment               = "Genere par OpenSSL"

# subjectAltName        = email:copy



[ policy_match ]

countryName             = match

stateOrProvinceName     = optional

localityName            = optional
```

```
organizationName        = supplied

organizationalUnitName  = optional

commonName              = supplied

emailAddress            = optional
```

5.5. Fichier ca-user-cert.cnf. —

```
#ca-user-cert.cnf

# pour signer un certificat utilisateur

[ ca ]

default_ca     = CA_default           # The default ca section


[ CA_default ]

dir            = /home/autorite/tls         # Where everything is kept

certs          = $dir/certs           # Where the issued certs are kept

crl_dir        = $dir/crl             # Where the issued crl are kept

database       = $dir/index.txt       # database index file.

new_certs_dir  = $dir/newcerts        # default place for new certs.


certificate    = $dir/certs/cacert.pem          # The CA certificate

serial         = $dir/serial          # The current serial number

crl            = $dir/crl.pem          # The current CRL

private_key    = $dir/private/cakey.pem  # The private key


RANDFILE       = $dir/private/.rand   # private random number file
```

```
default_days     = 730     # how long to certify for

default_crl_days = 30      # how long before next CRL

default_md       = sha256     # which md to use.

Preserve         = no      # keep passed DN ordering


x509_extensions  = user_cert

copy_extensions  = none

policy           = policy_anything


[ user_cert ]

basicConstraints        = critical, CA:false

authorityKeyIdentifier  = keyid:always

subjectKeyIdentifier    = hash

keyUsage                = digitalSignature, nonRepudiation, keyEncipherment

extendedKeyUsage        = clientAuth, emailProtection

nsCertType              = client, email, objsign

nsComment               = "Certificat utilisateur genere par OpenSSL"


subjectAltName = email:copy

#issuerAltName = issuer:copy


#nsCaRevocationUrl =

#nsBaseUrl

#nsRevocationUrl =

#nsRenewalUrl
```

```
#nsCaPolicyUrl

#nsSslServerName



[ policy_anything ]

countryName             = optional

stateOrProvinceName     = optional

localityName            = optional

organizationName        = optional

organizationalUnitName  = optional

commonName              = supplied

emailAddress            = supplied
```

5.6. Fichier subca-user-cert.cnf. —

```
#subca-user-cert.cnf


# pour signer un certificat utilisateur par l'autorite secondaire

[ ca ]

default_ca      = SUBCA_default           # The default ca section


[ SUBCA_default ]

dir             = /home/autorite/tls         # Where everything is kept

certs           = $dir/certs           # Where the issued certs are kept

crl_dir         = $dir/crl             # Where the issued crl are kept

database        = $dir/index.txt       # database index file.

new_certs_dir   = $dir/newcerts        # default place for new certs.
```

```
certificate    = $dir/certs/subcacert.pem          # The SubCA certificate

serial         = $dir/serial          # The current serial number

crl            = $dir/crl.pem          # The current CRL

private_key    = $dir/private/subcakey.pem  # The private key


RANDFILE       = $dir/private/.rand   # private random number file


default_days     = 730     # how long to certify for

default_crl_days = 30      # how long before next CRL

default_md       = sha256     # which md to use.

Preserve         = no      # keep passed DN ordering


x509_extensions  = user_cert

copy_extensions  = none

policy           = policy_anything


[ user_cert ]

basicConstraints        = critical, CA:false

authorityKeyIdentifier  = keyid:always

subjectKeyIdentifier    = hash

keyUsage                = digitalSignature, nonRepudiation, keyEncipherment

extendedKeyUsage        = clientAuth, emailProtection

nsCertType              = client, email, objsign
```

```
nsComment               = "Certificat utilisateur genere par OpenSSL"


subjectAltName = email:copy

#issuerAltName = issuer:copy


#nsCaRevocationUrl =

#nsBaseUrl

#nsRevocationUrl =

#nsRenewalUrl

#nsCaPolicyUrl

#nsSslServerName




[ policy_anything ]

countryName             = optional

stateOrProvinceName     = optional

localityName            = optional

organizationName        = optional

organizationalUnitName  = optional

commonName              = supplied

emailAddress            = supplied
```

Références

[1] Archive OpenSSL-fichiers-cnf.zip à télécharger depuis la zone de téléchargement du site http ://www.acrypta.fr

28 février 2008

A. Cruptos, Association ACrypTA. • *E-mail :* acrypta@acrypta.fr