TP : RSA et certificats avec **openSSL**

1 Presentation de **openSSL**

1.1 Protocole SSL

Le protocole SSL (Secure Socket Layer) a été développé par la société Netscape Communications Corporation pour permettre aux applications client/serveur de communiquer de façon sécurisée. TLS (Transport Layer Security) est une évolution de SSL réalisée par l'IETF.

La version 3 de SSL est utilisée par les navigateurs tels Netscape et Microsoft Internet Explorer depuis leur version 4.

SSL est un protocole qui s'intercale entre TCP/IP et les applications qui s'appuient sur TCP. Une session SSL se déroule en deux temps

1. une phase de *poignée de mains* (handshake) durant laquelle le client et le serveur s'identifient, conviennent du système de chiffrement et d'une clé qu'ils utiliseront par la suite.
2. la phase de communication proprement dite durant laquelle les données échangées sont compressées, chiffrées et signées.

L'identification durant la poignée de mains est assurée à l'aide de certificats X509.

1.2 openSSL

openSSL est une boîte à outils cryptographiques implémentant les protocoles SSL et TLS qui offre

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

Pour connaître toutes les fonctionnalités de `openSSL` : `man openssl`.

La syntaxe générale de la commande `openssl` est

```
$ openssl <commande> <options>
```

(le `$` est le prompt du shell)

Dans le texte qui suit, les commandes invoquant `openssl` supposent que cette commande est dans votre PATH.

2 RSA avec **openSSL**

2.1 Génération d'une paire de clés

On peut générer une paire de clés RSA avec la commande `genrsa` de `openSSL`.

```
$ openssl genrsa -out <fichier> <taille>
```

où *fichier* est un nom de fichier de sauvegarde de la clé, et *taille* et la taille souhaitée (exprimée en bits) du modulus de la clé.

Par exemple, pour générer une paire de clés de 1024 bits, stockée dans le fichier `maCle.pem`, on tape la commande

```
$ openssl genrsa -out maCle.pem 1024
```

Le fichier obtenu est un fichier au format PEM (Privacy Enhanced Mail, format en base 64), dont voici un exemple

```
$ cat maCle.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCveVjLltevTC5kSAiTYjHMVuAR80DHMLWCp3BOVZ49eXwraXxO
7AfKWpA5g0wFZgZNERIfFYaCnvaQDQA+9BRIfsSSr3oSw0My5SD6eg15v0VmJmvP
d8LgBypJHbr6f5MXWqntvzp0Qvg6ddeNpUIrqkkh4uDfHFDWqyrkQUCvKwIDAQAB
AoGANchUrfnq28DWy0fE0R+cscvC292Z8jN8vrIBWxEk8iSlKU0om6v+a0g8wlP6
3gC6V66uxjY7xxdf7SD+/UykVl4PGFymhLtywSdGlgec3tLgBtV3ytJFilAVDBij
LzQwUegCO4zt1JWYc6vvaVdNyQSaGIIeYGsNDWEYlOtDSlkCQQDVRn9JS15G8p+H
4Z0PbU9ZQg2L1u9/SD/kELVe3Kx1fdHulxH0v8V2AgPdXA29Nhi+TxUtC+V8CMc2
KXmAvFsHAkEA0qBDmjHMDPwcGaqbQ2lymYQIGlZ5TLQFA98Dey2uE+CB6pmS/e/Z
ilu1IaasuE3vBzXfB/JU7DUkV++JQ7TtvQJBAL2s5dUch2sXqlOhjhpDP/eE7CE6
9WLAsbm2Nmd4YJRZYtQLXPfLeeSapC9BCCMHsnfGQ3H9i4mFEQ6VUi7w1Q8CQAQa
pVaS09QI8Y86eM4GdvowzWud9b0d4N8jcFDtIfA3NrDYjzmte8KraMsgEUuCET9F
uHPSL/9uRagE/dq44s0CQCMQU4PMqkMtwzCFsV8ZqLmkDPn1binIAwRLYFcsQRDt
gTi6rycz3Pk1hCVzBfyMd8zwqpwKmR5FoOXuJEv+mVg=
-----END RSA PRIVATE KEY-----
```

2.2 Visualisation des clés RSA

La commande `rsa` permet de visualiser le contenu d'un fichier au format PEM contenant une paire de clés RSA.

```
$ openssl rsa -in <fichier> -text -noout
```

L'option `-text` demande l'affichage décodé de la paire de clés. L'option `-noout` supprime la sortie normalement produite par la commande `rsa`.

Par exemple

```
$ openssl rsa -in maCle.pem -text -noout
Private-Key: (1024 bit)
modulus:
    00:af:79:58:cb:96:d7:af:4c:2e:64:48:08:93:62:
    31:cc:56:e0:11:f3:40:c7:30:b5:82:a7:70:4e:55:
    9e:3d:79:7c:2b:69:7c:4e:ec:07:ca:5a:90:39:83:
    4c:05:66:06:4d:11:12:1f:15:86:82:9e:f6:90:0d:
    00:3e:f4:14:48:7e:c4:92:af:7a:12:c3:43:32:e5:
    20:fa:7a:0d:79:bf:45:66:26:6b:cf:77:c2:e0:07:
    2a:49:1d:ba:fa:7f:93:17:5a:a9:ed:bf:3a:74:42:
    f8:3a:75:d7:8d:a5:42:2b:aa:49:21:e2:e0:df:1c:
    50:d6:ab:2a:e4:41:40:af:2b
publicExponent: 65537 (0x10001)
privateExponent:
    35:c8:54:ad:f9:ea:db:c0:d6:cb:47:c4:d1:1f:9c:
    b1:cb:c2:db:dd:99:f2:33:7c:be:b2:01:5b:11:24:
    f2:24:a5:29:4d:28:9b:ab:fe:6b:48:3c:c2:53:fa:
    de:00:ba:57:ae:ae:c6:36:3b:c7:17:5f:ed:20:fe:
    fd:4c:a4:56:5e:0f:18:5c:a6:84:bb:72:c1:27:46:
    96:07:9c:de:d2:e0:06:d5:77:ca:d2:45:8a:50:15:
    0c:18:a3:2f:34:30:51:e8:02:3b:8c:ed:d4:95:98:
    73:ab:ef:69:57:4d:c9:04:9a:18:82:1e:60:6b:0d:
    0d:61:18:94:eb:43:4a:59
prime1:
    00:d5:46:7f:49:4b:5e:46:f2:9f:87:e1:9d:0f:6d:
    4f:59:42:0d:8b:d6:ef:7f:48:3f:e4:10:b5:5e:dc:
    ac:75:7d:d1:ee:97:11:f4:bf:c5:76:02:03:dd:5c:
    0d:bd:36:18:be:4f:15:2d:0b:e5:7c:08:c7:36:29:
    79:80:bc:5b:07
```

```
prime2:
    00:d2:a0:43:9a:31:cc:0c:fc:1c:19:aa:9b:43:69:
    72:99:84:08:1a:56:79:4c:b4:05:03:df:03:7b:2d:
    ae:13:e0:81:ea:99:92:fd:ef:d9:8a:5b:b5:21:a6:
    ac:b8:4d:ef:07:35:df:07:f2:54:ec:35:24:57:ef:
    89:43:b4:ed:bd
exponent1:
    00:bd:ac:e5:d5:1c:87:6b:17:aa:53:a1:8e:1a:43:
    3f:f7:84:ec:21:3a:f5:62:c0:b1:b9:b6:36:67:78:
    60:94:59:62:d4:0b:5c:f7:cb:79:e4:9a:a4:2f:41:
    08:23:07:b2:77:c6:43:71:fd:8b:89:85:11:0e:95:
    52:2e:f0:d5:0f
exponent2:
    04:1a:a5:56:92:d3:d4:08:f1:8f:3a:78:ce:06:76:
    fa:30:cd:6b:9d:f5:bd:1d:e0:df:23:70:50:ed:21:
    f0:37:36:b0:d8:8f:39:ad:7b:c2:ab:68:cb:20:11:
    4b:82:11:3f:45:b8:73:d2:2f:ff:6e:45:a8:04:fd:
    da:b8:e2:cd
coefficient:
    23:10:53:83:cc:aa:43:2d:c3:30:85:b1:5f:19:a8:
    b9:a4:0c:f9:f5:6e:29:c8:03:04:4b:60:57:2c:41:
    10:ed:81:38:ba:af:27:33:dc:f9:35:84:25:73:05:
    fc:8c:77:cc:f0:aa:9c:0a:99:1e:45:a0:e5:ee:24:
    4b:fe:99:58
```

Les différents éléments de la clé sont affichés en hexadécimal (hormis l'exposant public). On peut distinguer le modulus, l'exposant public (qui par défaut est toujours 65537 [1]), l'exposant privé, les nombres premiers facteurs du modulus, plus trois autres nombres qui servent à optimiser l'algorithme de déchiffrement.

Exercice 1. Donnez une explication du choix de la valeur 65537 pour exposant public par défaut.

2.3 Chiffrement d'un fichier de clés RSA

Il n'est pas prudent de laisser une paire de clés en clair (surtout la partie privée). Avec la commande `rsa`, il est possible de chiffrer une paire de clés [2] . Pour cela trois options sont possibles qui précisent l'algorithme de chiffrement symétrique à utiliser : `-des`, `-des3` et `-idea`.

```
$ openssl rsa -in maCle.pem -des3 -out maCle.pem
writing RSA key
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

Une phrase de passe est demandée deux fois pour générer une clé symétrique protégeant l'accès à la clé.

Exercice 2. Avec la commande `cat` observez le contenu du fichier `maCle.pem`. Utilisez à nouveau la commande `rsa` pour visualiser le contenu de la clé.

2.4 Exportation de la partie publique

La partie publique d'une paire de clés RSA est publique, et à ce titre peut être communiquée à n'importe qui. Le fichier `maCle.pem` contient la partie privée de la clé, et ne peut donc pas être communiqué tel quel (même s'il est chiffré). Avec l'option `-pubout` on peut exporter la partie publique d'une clé.

```
$ openssl rsa -in maCle.pem -pubout -out maClePublique.pem
```

Exercice 3.

Question 1. Notez le contenu du fichier `maClePublique.pem`. Remarquez les marqueurs de début et de fin.

1. `openssl` n'autorise que deux exposants publics : 65537 (valeur par défaut) ou 3 obtenu avec 'option `-3`
2. Il est possible de chiffrer le fichier lors de sa génération. Il suffit de mettre l'une des trois options `-des`, `-des3`, `-idea` dans la ligne de commande `genrsa`.

Question 2. Avec la commande `rsa` visualisez la clé publique. Attention vous devez préciser l'option `-pubin`, puisque seule la partie publique figure dans le fichier `maClePublique.pem`.

2.5 Chiffrement/déchiffrement de données avec RSA

On peut chiffrer des données avec une clé RSA. Pour cela on utilise la commande `rsautl`

```
$ openssl rsautl -encrypt -in <fichier_entree> -inkey <cle> \
                 -out <fichier_sortie>
```

où

- *fichier_entree* est le fichier des données à chiffrer. Attention, le fichier des données à chiffrer ne doit pas avoir une taille excessive (ne doit pas dépasser 116 octets pour une clé de 1024 bits).
- *cle* est le fichier contenant la clé RSA. Si ce fichier ne contient que la parte publique de la clé, il faut rajouter l'option `-pubin`.
- *fichier_sortie* est le fichier de données chiffré.

Pour déchiffrer on remplace l'option `-encrypt` par `-decrypt`. Le fichier contenant la clé doit évidemment contenir la partie privée.

Exercice 4. Chiffrez le fichier de votre choix avec le système symétrique de votre choix. Chiffrez la clé ou le mot de passe utilisé(e) avec la clé publique de votre destinataire (demandez-lui sa clé publique si vous ne l'avez pas). Envoyez-lui le mot de passe chiffré ainsi que le fichier chiffré.

Exercice 5. Il s'agit de déchiffrer le fichier `cryptogram14`. Pour cela vous devez récupérer les fichiers suivants (attention la plupart de ces fichiers sont des fichiers binaires)

- le fichier cryptogram14 (fichier binaire) a été obtenu en chiffrant un texte avec `openssl` et la commande `enc`. Le système de chiffrement symétrique utilisé est BlowFish en mode CBC avec un vecteur d'initialisation nul (voir TP2) ;
- la clé BlowFish a été dérivée à partir d'un mot de passe dont voici la version chiffrée ;
- la clé privée RSA qui a servi à chiffrer le mot de passe ;
- le mot de passe protégeant la clé RSA, codé en base 64, est `TGlsbGUxLUZJTC1QQUM=` .

2.6 Signature de fichiers

Il n'est possible de signer que de petits documents. Pour signer un gros document on calcule d'abord une empreinte de ce document. La commande `dgst` permet de le faire.

```
$ openssl dgst <hachage>  -out <empreinte> <fichier_entree>
```

où *hachage* est une fonction de hachage. Avec `openssl`, plusieurs fonctions de hachage sont proposées dont

- MD5 (option `-md5`), qui calcule des empreintes de 128 bits,
- SHA1 (option `-sha1`), qui calcule des empreintes de 160 bits,
- RIPEMD160 (option `-ripemd160`), qui calcule des empreintes de 160 bits.

Signer un document revient à signer son empreinte. Pour cela, on utilise l'option `-sign` de la commande `rsautl`

```
$ openssl rsautl -sign   -in <empreinte> \
                 -inkey <cle> \
                 -out <signature>
```

et pour vérifier la signature

```
$ openssl rsautl -verify   -in <signature> -pubin \
                 -inkey <cle>  -out <empreinte>
```

il reste ensuite à vérifier que l'empreinte ainsi produite est la même que celle que l'on peut calculer. L'option `-pubin` indique que la clé utilisée pour la vérification est la partie publique de la clé utilisée pour la signature.

Exercice 6. Signez le fichier de votre choix, puis vérifiez la signature.

Exercice 7. Récupérez l'archive `signatures.zip` qui contient deux fichiers accompagnés d'une signature

– fichier : `cigale.txt`, signature : `signature1`
– et fichier : `QuandLaMerMonte.txt`, signature : `signature2`,

ainsi que la partie publique de la clé RSA ayant produit la signature : `uneClePublique.pem`.

De ces deux fichiers, lequel a bien été signé ?

3 Certificats

Vous allez maintenant élaborer un certificat pour votre clé publique. Puis, vous verrez comment utiliser les clés certifiées pour signer et/ou chiffrer des courriers électoniques.

3.1 Génération de la paire de clés

Exercice 8. Générez votre paire de clés RSA d'une taille de 2048 bits, protégée par un mot de passe (cf génération d'une paire de clés). Dans la suite, on suppose nommé `maCle.pem` le fichier contenant votre paire de clés RSA. Ce fichier est protégé par le mot de passe fourni lors de la génération.

Exercice 9. Créez un fichier ne contenant que la partie publique de votre clé RSA (cf Exportation de la clé publique). Dans la suite, on suppose ce fichier nommé `maClePublique.pem`.

3.2 Création d'une requête de certificats

Maintenant que vous disposez d'une clé RSA, vous allez établir une requête pour obtenir un certificat.

Outre la clé publique du sujet, un certificat contient un certain nombre d'informations concernant l'identité de son propriétaire :

– Pays (`C`),
– État ou province (`ST`)
– Ville ou localité (`L`)
– Organisation (`O`)
– Unité (`OU`)
– Nom (`CN`)
– Email

Toutes ces informations et d'autres encore sont demandées lors de la création de la requête. Un fichier de configuration (`req.cnf`) peut-être défini qui propose les informations à apporter dans le certificat avec des valeurs par défaut.

On établit une requête avec la commande `req` de `openssl`.

```
$ openssl req -config req.cnf -new -key maCle.pem -out maRequete.pem
```

Le fichier produit `maRequete.pem` est aussi au format PEM.

```
$ cat maRequete.pem
-----BEGIN CERTIFICATE REQUEST-----
MIIB/zCCAWgCAQAwgb4xCzAJBgNVBAYTAkZSMRIwEAYDVQQIEwlOb3JkICg1OSkx
GjAYBgNVBAcTEVZpbGxlbmV1dmUgZCdBc2NxMR4wHAYDVQQKExVVbml2ZXJzaXRl
IGRlIExpbGxlIDExGTAXBgNVBAsTEExpY2VuY2UgUHJvIERBMkkxGjAYBgNVBAMT
EUVyaWMgV2Vncnp5bm93c2tpMSgwJgYJKoZIhvcNAQkBFhlFcmljLldlZ3J6eW5v
d3NraUBsaWZsLmZyMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCveVjLltev
TC5kSAiTYjHMVuAR80DHMLWCp3BOVZ49eXwraXxO7AfKWpA5g0wFZgZNERIfFYaC
nvaQDQA+9BRIfsSSr3oSw0My5SD6eg15v0VmJmvPd8LgBypJHbr6f5MXWqntvzp0
Qvg6ddeNpUIrqkkh4uDfHFDWqyrkQUCvKwIDAQABoAAwDQYJKoZIhvcNAQEEBQAD
gYEAHOwGNN6A8d4EhjfXCRvC2fhIjt7i6jxfkHKBkHpm2yNBBDHQwiv+O/Y0MeNh
Ira+y8KUMjeImsSiH4731sfgA6ycm+6JoDV7n6z8tzN5QMGsw7V3ErduskayKP4T
ja+BMImEcDwlr+KuRO704rGeiAG7pvtDGcDcj2Mex68ki94=
-----END CERTIFICATE REQUEST-----
```

On peut consulter les informations contenues dans la requête avec la commande

```
$ openssl req -config req.cnf -in maRequete.pem -text -noout
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=FR, ST=Nord (59), L=Villeneuve d'Ascq, O=Universite de Lille 1, OU=Li
info, CN=Eric Wegrzynowski/emailAddress=Eric.Wegrzynowski@lifl.fr
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:b4:f3:e0:65:50:12:f6:51:c2:96:90:77:5b:c3:
                    25:46:17:1f:3b:0e:b9:a0:5a:fa:27:14:2c:45:96:
                    88:68:9c:60:b1:8a:94:2c:22:76:f0:de:7e:02:1b:
                    1d:1d:57:b6:b8:8d:55:7e:15:a4:5d:a1:ce:02:71:
                    cd:56:30:dc:67:64:fb:cb:c7:bc:64:75:23:4c:2f:
                    a5:06:47:a6:39:46:74:f1:8a:91:ed:ad:8f:7d:d3:
                    34:c0:87:79:61:a7:b9:8c:1d:8c:45:0d:c4:df:a8:
                    ea:22:0c:5b:f2:0c:6b:11:8e:2b:f6:6a:9e:b0:1d:
                    ef:53:e7:ba:32:28:ee:98:d4:83:10:63:64:c2:4f:
                    44:12:51:f2:a6:31:68:30:d2:f9:8d:7a:bf:23:09:
                    18:a4:ab:0a:21:fc:25:88:ba:09:5a:70:82:33:41:
                    93:f8:d8:0a:b4:38:ae:47:8c:4a:d1:88:d1:af:b6:
                    18:77:b8:84:4e:7a:ad:c0:a6:23:e7:94:6c:76:cb:
                    0d:da:16:16:cb:a5:0f:75:a6:e0:78:ff:33:e1:d1:
                    5b:3c:8d:dd:06:7a:72:e2:be:48:c3:17:4a:4f:bb:
                    e0:bb:fb:e6:bd:7e:d4:f8:c1:2a:5a:4f:6a:3a:e0:
                    20:e6:63:e5:d5:65:ee:d3:8c:72:22:54:ca:f2:40:
                    ef:c1
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha1WithRSAEncryption
        54:1b:b8:52:28:f1:29:e9:5c:28:bc:e3:f6:58:cd:0f:e8:2c:
        a8:83:1d:d6:f3:6b:46:f8:d5:c6:37:bb:15:f5:34:58:56:3d:
        7b:f1:ce:1d:bf:14:09:fe:fe:f3:f1:07:54:27:60:45:6a:ef:
        88:8b:b7:89:74:77:5e:4d:a5:5c:ea:5c:b8:1c:7a:57:b3:83:
        0d:2e:1e:62:86:f4:01:d4:4d:39:51:25:04:cb:00:33:2a:84:
        5c:ae:0a:a0:6f:65:c7:70:0a:cf:56:95:d1:70:22:05:e1:fc:
        2a:d8:9b:21:3e:05:bb:5c:97:b4:35:67:85:e3:1f:8b:82:31:
        ef:76:0b:17:53:a2:06:43:79:1e:cb:a5:45:1d:6a:c9:d1:04:
        0c:44:d7:87:44:2c:4b:ef:38:5b:72:38:de:ff:74:ae:59:17:
        12:f5:87:8c:00:56:00:3f:f0:d4:08:d2:c1:4b:ae:84:0f:f3:
        d5:dd:ea:48:0b:86:31:82:c5:f5:ae:1d:52:b2:c6:74:62:d1:
        f7:94:43:b2:5d:9a:e5:52:c1:48:10:ff:27:bb:d5:ec:00:b8:
        aa:88:77:c3:36:f9:87:41:d9:da:58:9a:c3:26:8c:ba:e3:19:
        fe:25:42:25:12:b4:d4:79:e1:d9:b9:94:d8:83:90:36:ac:ca:
        c6:a7:b2:9c
```

Exercice 10. Expliquez les différents éléments contenus dans cette requête. La clé privée du sujet y figure–t–elle ?

3.3 Demande de signature de certificat

Une fois que vous avez établi une requete de certificat, il vous reste à contacter une autorité de certification qui vous délivrera un certificat signé, après avoir procédé (normalement) à quelques vérifications vous concernant.

L'autorité de certification *Père Ubu* Vous jouerez dans ce TP le rôle de l'autorité de certification. Pour cela il vous faut un certificat d'autorité de certification, ainsi qu'une paire de clés. Afin de ne pas multiplier les autorités, vous utiliserezvous la très notable autorité *Père Ubu* dont vous trouverez ici le certificat et la paire de clés RSA.

Un certificat produit par `openssl` est un fichier au format PEM.

```
$ cat unCertif.pem
-----BEGIN CERTIFICATE-----

    ....

-----END CERTIFICATE-----
```

Pour visualiser le contenu d'un certificat

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

Exercice 11. Après avoir récupéré le certificat de l'autorité, ainsi que sa paire de clés RSA, cherchez quelle est la date d'expiration du certificat et la taille de la clé.

Création d'un certificat Pour créer et signer un certificat à partir d'une requête `maRequete.pem`, l'autorité invoque la commande `x509`

```
$ openssl x509 -days 10\
          -CAserial PereUbu.srl \
          -CA PereUbuCertif.pem -CAkey PereUbuCle.pem\
          -extfile usr.ext -extensions x509_ext\
          -in maRequete.pem -req -out monCertif.pem
```

dans laquelle

- l'option `-days` détermine la durée de validité du certificat (ici 10 jours) ;
- `PereUbu.srl` est un fichier contenant le numéro de série du prochain certificat à signer (ce fichier est un fichier texte contenant une ligne donnant un nombre codé avec un nombre pair de chiffres hexadécimaux) ;
- et `usr.ext` est un fichier contenant une description des extensions au certificat décrivant entre autres les usages auxquels sont destinés les certificats, l'option `-extension` précisant le nom de la section dans laquelle se trouvent les différentes extensions (voici un exemple de tel fichier).

Exercice 12. Créez un certificat pour votre clé publique. (Lors de la signature du certificat, la commande `x509` invite l'autorité certifiante à donner son mot de passe. Le mot de passe de Père Ubu ne devrait pas vous être inconnu, puisqu'il s'agit DU mot du Père Ubu (celui prononcé au tout début d'Ubu Roi). Néanmoins, le voici codé en base 64 `bWVyZHJlCg==` S'il le faut, utilisez la commande `openssl base64 -d` pour le décoder.)

Puis contrôlez le contenu du certificat obtenu avec les options appropriées de la commande `x509`.

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

```
$ openssl verify -CAfile PereUbuCertif.pem  monCertif.pem
```

3.4 Signature et chiffrement de courriers électroniques

Lorsque vous disposez d'une paire de clés publique/privée, et d'un certificat qui les accompagne, et qui atteste de votre adresse électronique, il vous est possible, entre autres choses, de signer vos mails, et de recevoir des mails chiffrés de la part de vos correspondants qui disposent de votre certificat.

3.4.1 Avec openssl

Signature de courriers Vous disposez d'un certificat `monCertif.pem`, d'une clé privée `maCle.pem` et vous voulez envoyer un courrier *signé* dont le contenu est dans le fichier `blabla.txt` à l'adresse `haha@palotin.pl`. En supposant que tous les fichiers se trouvent dans le répertoire courant, il suffit d'invoquer la commande

```
$ openssl smime -sign -in blabla.txt -text\
        -signer monCertif.pem -inkey maCle.pem\
        -from moi@palotin.pl \
        -to haha@palotin.pl \
        -subject "courrier signe"
```

Il est bien videmment possible de rediriger la sortie de la comande précédente vers un fichier (avec l'option `-out` par exemple), ou vers la commande `mail`.

Vérification de courriers signés Pour vérifier un courrier signé, il faut disposer du certificat que l'émetteur a utilisé pour signer, ainsi que celui de l'autorité ayant émis ce certificat. On invoque la commande

```
$ openssl smime -verify -in courrier.signe \
              -CAfile PereUbuCertif.pem
```

Chiffrement de courriers Pour envoyer un courrier chiffré, par exemple avec du Triple-DES, à un destinataire dont on dispose d'un certificat `sonCertif.pem`, il suffit d'invoquer la commande

```
$ openssl smime -encrypt -in blabla.txt -text \
        -from moi@palotin.pl \
        -to haha@palotin.pl \
        -subject "courrier chiffre" \
        -des3 sonCertif.pem
```

Il est bien videmment possible de rediriger la sortie de la comande précédente vers un fichier (avec l'option `-out` par exemple), ou vers la commande `mail`.

Déchiffrement de courriers Pour déchiffrer un courrier chiffré

```
$ openssl smime -decrypt -in courrier.chiffre \
              -recip sonCertif.pem \
              -inkey maCle.pem
```

où `maCle.pem` est ma clé privée RSA, et `sonCertif.pem` est le certificat de l'émetteur.

Remarque Lorsqu'on chiffre un courrier électronique, il faut s'assurer que le destinataire est en mesure de le déchiffrer, autrement dit que le logiciel du destinataire est en mesure de le faire.

3.4.2 Avec Mozilla Thunderbird

La première chose à faire pour pouvoir échanger des courriers signés et/ou chiffrés avec Mozilla Thunderbird est d'importer votre certificat et votre clé privée que vous venez d'obtenir dans la base de données de certificats gérées par votre navigateur.

La base de données de certificats Vous pouvez consulter cette base de données par le menu `Edition/Preferences`.

Vous pouvez alors consulter
- vos certificats (a priori aucun),
- ceux des personnes dont vous en avez obtenu un,
- ceux de serveurs WEB sécurisés que vous avez eu l'occasion de rencontrer,
- et enfin, ceux des autorités de certification que vous connaissez (il y en a plusieurs dizaines normalement, mais cette liste ne contient pas le Père Ubu)

Importation d'un certificat Pour importer votre certificat dans votre base de données, il faut créer d'abord une *enveloppe PKCS#12*. Il s'agit d'une norme de fichier adoptée pour "ficeler" ensemble un ou plusieurs certificats et clés.

Pour créer une enveloppe PKCS#12 contenant votre certificat, votre clé privée, et le certificat de l'autorité ayant signé le votre, il suffit d'invoquer la commande

```
$ openssl pkcs12 -export -in monCertif.pem \
        -inkey maCle.pem -certfile PereUbuCertif.pem \
        -name "EW" -caname "Pere Ubu" -out monEnveloppe.p12
```

Un mot de passe protégeant cette enveloppe vous est demandé.

Une fois cette enveloppe réalisée, il reste à importer tout cela dans la base de données de certificats de votre navigateur. Pour cela passez par le menu sécurité. Un mot de passe vous sera demandé pour la base de données, puis celui que vous avez utilisé pour protéger l'enveloppe PKCS#12.

Cela réalisé, vous devez voir maintenant votre certificat dans la liste des votres. Demandez à le visualiser, puis à le vérifier. Votre certificat n'est pas valide car ...

Acceptation de l'autorité Père Ubu ... l'autorité Père Ubu n'est pas acceptée par votre navigateur.

Pour que Père Ubu soit accepté vous devez éditer son certificat (depuis la liste des certificats d'autorités signataires) et cocher la case concernant l'acceptation de cette autorité pour les emails.

Vérifiez maintenant votre certificat.

Vérifiez aussi la signature du mail de mercredi dernier.

Envoyer un courrier signé Pour envoyer un courrier signé, il suffit de cocher la cas adéquate qui apparaît lorsqu'on coche la case sécurité (en bas à gauche) de la fenêtre de composition du mail.

Envoyer un mail chiffré Pour envoyer un courrier chiffré, il suffit de faire la même chose. ATTENTION, vous devez posséder un certificat valide de votre destinataire.