

## Sommaire

I.Qu'est ce que OpenSSL ?.....	1
II.Certificats, certifications et autres concepts.....	2
a)CA : Certificate Authorities : autorité de certification.....	2
b)Contenu d'un certificat.....	2
III.Se connecter en SSL.....	2
IV.L'outil OpenSSL.....	5
a)Le fichier de configuration.....	6
a)Créer son autorité de certification.....	6
b)Génération de certificats.....	7
b)Fichier PEM.....	7
c)Question de droits.....	8
V.Public Key Infrastructure : une méthode plus simple.....	8
VI.Configuration avec Apache.....	9
a)Passphrase et démarrage.....	10
VII.Configuration avec Dovecot .....	10
a)Génération de certificats avec dovecot.....	11
VIII.Test de SSL ou TLS.....	11
Bibliographie.....	13

## I. Qu'est ce que OpenSSL ?

SSL a été à l'origine développé par Netscape.

OpenSSL est un version libre du protocole SSL (Secure Sockets Layer version 1 et 2) et TLS (Transport Layer Security = SSL version 3). Il permet de crypter toutes les données échangées entre le client et le serveur de façon à ce que seul le serveur puisse décrypter ce qui vient du client et inversement. Un éventuel pirate ne peut pas, dans un temps raisonnable, décrypter les informations.

SSL sert de support :

- à SSH pour donner un telnet sécurisé et faire des tunnels cryptés simplement
- à HTTP pour sécurisé les sites de Web marchands
- à POP ou IMAP pour sécurisé la récupération de mail
- à tout autre protocole en clair afin de le sécuriser

SSL a trois buts :

- confidentialité des échanges grâce au cryptage symétrique
- intégrité des données grâce au fonction de hachage
- authentification des entités communicantes grâce aux certificats

## II. Certificats, certifications et autres concepts

### a) CA : Certificate Authorities : autorité de certification

Les autorités de certifications sont des organisations qui émettent les certificats moyennant paiement. Elles garantissent les certificats par leur signature ce qui permet d'être sûr du serveur auquel on se connecte (vente en ligne, banque...).

### b) Contenu d'un certificat

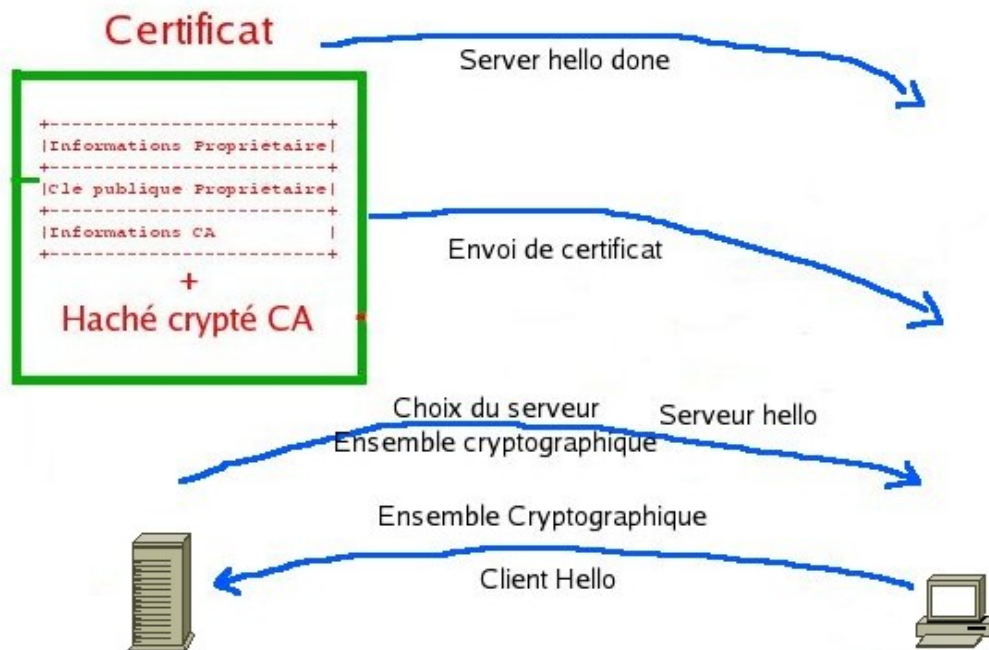
Un certificat contient (suivant la normalisation X.509):

- **des informations sur le certificat :**
  - la version de la norme X.509
  - le numéro de série du certificat (information purement administrative)
  - l'algorithme de cryptage utilisé pour la signature de celui-ci
  - la date de début de validité
  - la date de fin de validité
- **des informations sur le demandeur/propriétaire :**
  - le nom (DN : Distinguished Name) du propriétaire qui comprend :
    - Nom du serveur (Common Name = CN) : nom du serveur sécurisé
    - Organisation ou Entreprise (Organization or Company = O) : nom de l'entreprise de la personne qui demande le certificat
    - Service (Organizational Unit = OU) : nom du service ou du département de l'organisation qui demande le certificat.
    - Ville (City/Locality = L) : nom de la ville dans lequel se trouve l'organisation
    - Département/Région/Etat (State/Province = ST) : nom de la province dans laquelle se trouve l'organisation
    - Pays (Country = C) : code ISO du pays dans lequel se trouve l'organisation (FR, BE, ...)
    - adresse email de l'administrateur
  - **la clé publique du propriétaire du certificat**
- **des informations sur l'autorité de certification CA :**
  - le nom (DN : Distinguished Name) de la CA qui comprend :
    - Nom courant (Common Name = CN) : nom du CA
    - Organisation ou Entreprise (Organization or Company = O) : nom d'organisation du CA
    - Service (Organizational Unit = OU) : nom du service du CA qui signe le certificat
    - Ville (City/Locality = L) : nom de la ville dans lequel se trouve le CA
    - Département/Région/Etat (State/Province = ST) : nom de la province dans laquelle se trouve le CA
    - Pays (Country = C) : code ISO du pays dans lequel se trouve le CA (FR, BE, ...)
  - **la signature/sceau de la CA**

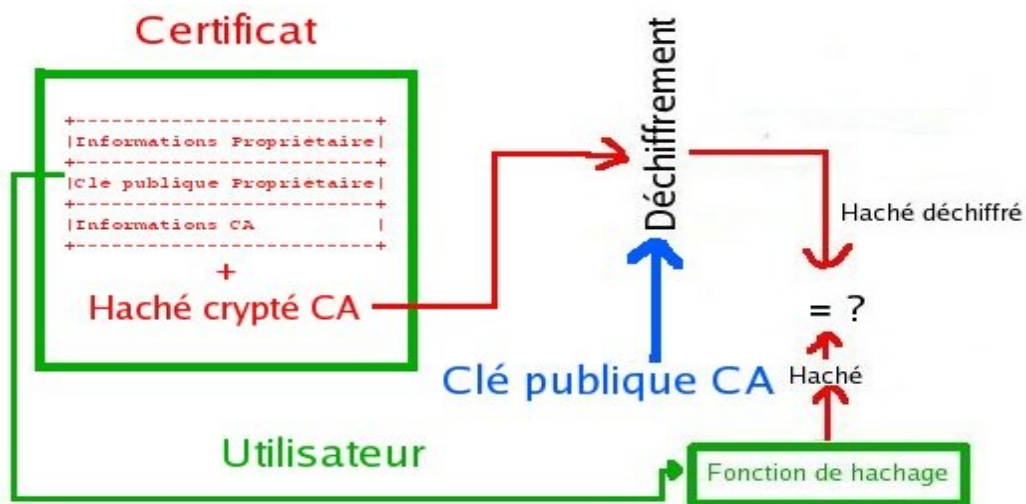
## III. Se connecter en SSL

Les étapes d'une connexion SSL sont les suivantes (du moins) :

1. établissement de la connexion : choix de la méthode de cryptage

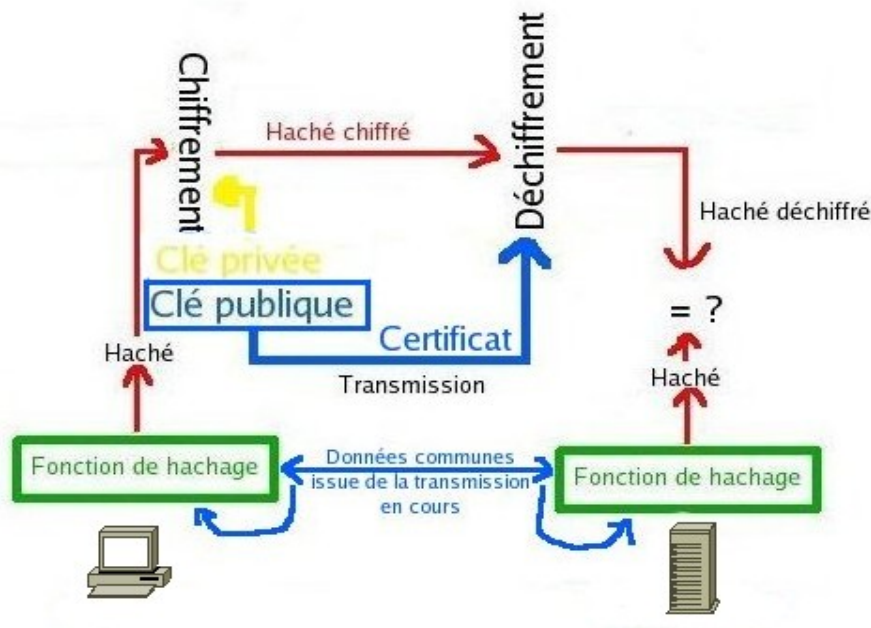


- le client envoie un « client-hello » au serveur qui comprend :
    - la version de SSL utilisée par le client
    - un id de session aléatoire
    - un ensemble cryptographique que le serveur devrait pouvoir supporter :
      - une méthode de cryptage à clé publique (asymétrique) pour le transfert des clés de session
      - une méthode de cryptage à clé privée (symétrique) (utilisera la clé de session) pour le transfert des données après l'établissement de la connexion (aucune, RC4(40bits), RC4(128bits), RC2(40bits), DES(40bits), DES(56bits), 3DES(168bits), IDEA (128bits) ou FORTEZZA(96bits))
      - une méthode de hachage pour calculer un digest des paquets transmis pendant la connexion, évite les attaques par relai (aucune, MD5(128bits) ou SHA-1(160bits)).
    - une méthode de compression
  - le serveur répond un « serveur-hello » au client (ou un code d'erreur = abandon de la connexion) contenant l'ensemble cryptographique qu'il a choisi sous la forme `SSL_X_WITH_Y_Z` :
    - la méthode de cryptage symétrique *X*
    - la méthode de cryptage asymétrique *Y*
    - la méthode de hachage *Z*
    - (la méthode de compression)
  - le serveur envoie son certificat et éventuellement une demande du certificat client (qui contient une liste de type de certificat et de CA autorisés par le serveur)
  - le serveur envoie un « server hello done »
1. authentification du serveur



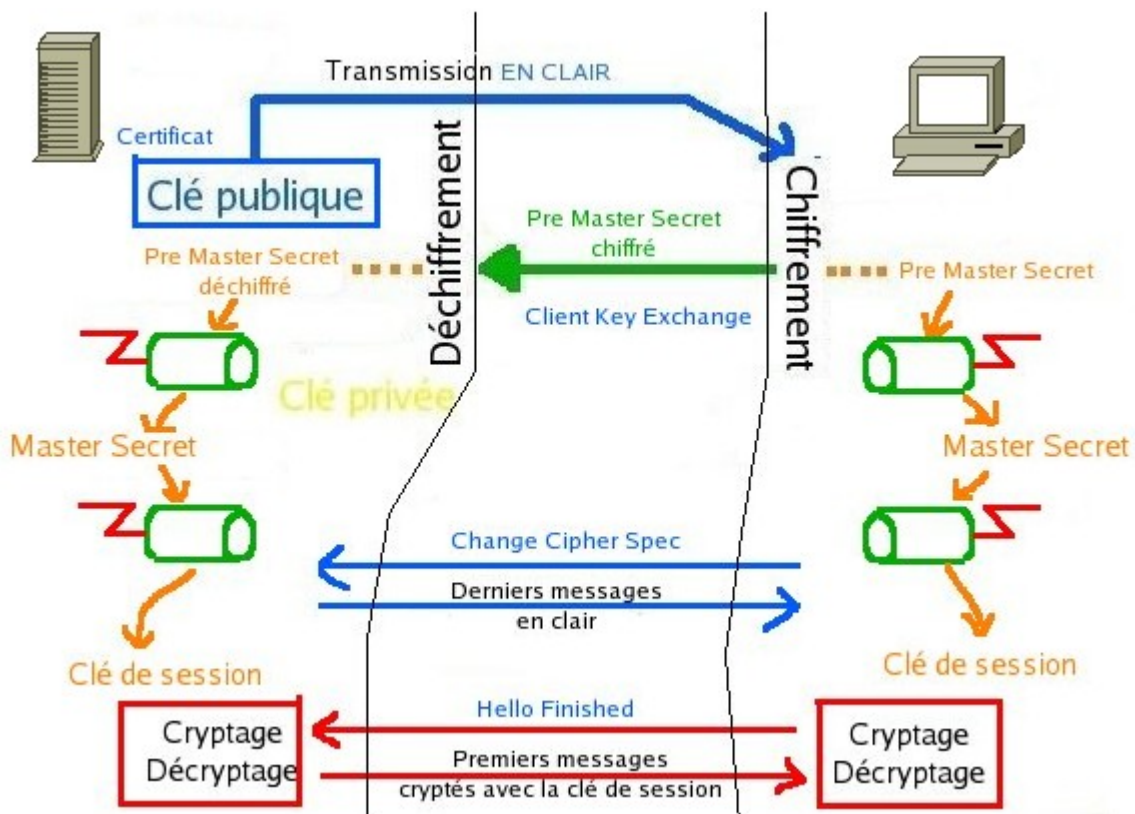
- une fois que le client reçoit le message « server hello done », il vérifie le certificat de la façon suivante :
  - trouver le nom de l’algorithme de chiffrement utilisé pour la signature du certificat dans celui-ci
  - calculer le digest du certificat
  - décrypter la signature présente dans le certificat avec l’algorithme trouvé et la clé publique de la CA **qu’il possède au préalable**
  - comparer le digest calculé et le digest décrypté : s’ils sont égaux, la connexion est établie sinon erreur

1. authentification facultative du client (si le serveur est configuré pour)



- le client scelle un morceau de donnée connu du client et du serveur avec la clé privée de son certificat puis envoie ce sceau et son certificat
- le serveur essaie de décrypter le sceau avec la clé publique du client trouvée dans son certificat, pour voir si les données correspondent à celle qu’il connaît. Si oui, la connexion continue, sinon annulation.

1. échange de la clé de session (dans le cas de la méthode RSA)



- le client envoie le « client key exchange » contenant le Pre Master Secret, un nombre aléatoire servant à générer le Master Secret, crypté avec la clé publique du serveur (se trouvant dans le certificat)
  - le serveur décrypte le Pre Master Secret avec sa clé privée
  - le client et le serveur calculent le Master Secret avec une série cryptographique qui va servir à calculer la clé de session
  - le client et le serveur calculent la clé de session (symétrique) à l'aide du Master Secret.
  - le client envoie le message « change cipher spec » afin de signaler la mise en place de la clé de session
  - le client envoie un message de fin crypté avec la clé de session
  - le serveur envoie le message « change cipher spec » afin de signaler la mise en place de la clé de session
  - le serveur envoie un message de fin crypté avec la clé de session
1. échanges cryptés
- maintenant, les échanges sont cryptés dans les deux sens
  - il peut arriver que le processus de génération de clé de session recommence au hasard.

## IV. L'outil OpenSSL

L'outil openssl permet de manipuler les objets nécessaires aux connexions sécurisées, entre autre :

- Création de certificats X.509, CSRs
  - génération de demande de certification et de certificats signés
- Tests SSL/TLS client et server
  - test du fonctionnement d'une connexion sécurisée SSL/TLS sur un serveur
  - permet de vérifier que le certificat est bien envoyé et que la connexion peut être établie

Mais aussi, comme gpg/pgp :

- Calcul de signature de messages (calcul d'un digest MD5)

- Chiffrement et Déchiffrement (d'un message, d'un fichier...)

## a) Le fichier de configuration

Le fichier `/etc/pki/tls/openssl.cnf` permet de configurer les informations par défaut afin de simplifier la génération. Voici par exemple, ce que vous devriez mettre dans la section `[req_distinguished_name]`:

```
[ req_distinguished_name ]
countryName                = Nom de pays (2 lettres)
countryName_default        = FR
countryName_min            = 2
countryName_max            = 2

stateOrProvinceName        = Etat ou province (Nom entier)
stateOrProvinceName_default = département

localityName                = Nom de localité
localityName_default        = ville

0.organizationName         = Nom d'organisation
0.organizationName_default = société

# we can do this but it is not needed normally :-)
#1.organizationName        = Nom d'organisation secondaire
#1.organizationName_default = société

organizationalUnitName     = Section de l'ogranisation
#organizationalUnitName_default =

commonName                  = Nom du serveur ou le votre
commonName_max              = 64

emailAddress                = Votre adresse Email
emailAddress_max            = 64
```

## a) Créer son autorité de certification

Il faut se placer dans le répertoire `/etc/pki/tls/misc`.

Pour créer son autorité de certification (par exemple pour du VPN) :

```
[root]# ./CA -newca
```

Il vous demandera une passphrase pour crypter la clé privée de l'autorité.

Il faut ensuite se placer dans le répertoire `/etc/pki/CA/` pour créer le certificat autosigné de l'autorité :

```
[root]# openssl x509 -in cacert.pem -days nb_jours_validité
-out cacert.pem -signkey ./private/cakey.pem
```

Le fichier contenant le certificat se trouve dans `/etc/pki/CA/cacert.pem` et la clé privée dans `/etc/pki/CA/private/cakey.pem`.

## b) Génération de certificats

Pour générer des certificats dans de fichiers séparés (**KEY, CSR, CRT**), on réalise les étapes suivantes :

- Générer une clef privée pour Apache qui servira pour le chiffrement : **fichier.key**. Une passphrase vous sera demandée afin de protéger la clé privée.

```
[root]# /usr/bin/openssl genrsa -des3 nb_bits_clé > fichier.key
```

*nb\_bits\_clé* : au moins 1024

- Il faut ensuite créer la requête signée de certification (Certificate Signing Request) qu'il faut normalement envoyer à une autorité de certification (Certifying Authority ou CA), cette étape demande un certain nombre de renseignements comme le nom du site, le pays, ... **fichier.csr**

```
[root]# /usr/bin/openssl req -new -key fichier.key -out fichier.csr
```

- Envoyer ce fichier à l'autorité de certification qui vous renvoie un **fichier.crt** (CeRtificaTe) après un avoir payé une certaine somme évidemment. Vous pouvez aussi auto signer votre certificat, dans ce cas les navigateurs web afficheront un message d'avertissement.

```
[root]# /usr/bin/openssl req -new -key fichier.key -x509 -days nb_jour_validité -out fichier.crt -set_serial numéro_série
```

ou

```
[root]# /usr/bin/openssl req -new -key fichier.key -x509 -startdate AAMMJJHHMMSSZ -enddate AAMMJJHHMMSSZ -out fichier.crt -set_serial numéro_série
```

Si vous voulez le signer avec une clé de votre CA :

```
[root]# openssl ca -cert /etc/pki/CA/cacert.pem -keyfile /etc/pki/CA/private/cakey.pem -in fichier.csr -out fichier.crt
```

Le *numéro\_série* est un nombre servant à identifier le certificat à l'intérieur de l'organisation demandeur.

## b) Fichier PEM

On peut générer toutes les clés dans le même fichier **PEM** (Privacy Enhanced Mail) qui a le format contient (dans l'ordre) :

- le fichier key (la paire de clé privée/publique)
- une ligne vide
- le fichier CRT

Pour générer un certificat (autosigné) en un seul fichier, on réalise les étapes suivantes :

```
[root]# /usr/bin/openssl req -newkey rsa:nb_bits_clé -keyout fichier.key -nodes -x509 -days nb_jours_validité -out fichier.crt -set_serial numéro_série
```

ou

```
[root]# /usr/bin/openssl genrsa -des3 nb_bits_clé > fichier.key
[root]# /usr/bin/openssl req -new -key fichier.key -out
fichier.csr
[root]# /usr/bin/openssl req -new -key fichier.key -x509 -days
nb_jour_validité -out fichier.crt -set_serial numéro_série
```

Une passphrase vous sera demandée afin de protéger la clé privée.

```
[root]# cat fichier.key > fichier.pem
[root]# echo "" >> fichier.pem
[root]# cat fichier.crt >> fichier.pem
[root]# rm -f fichier.key fichier.crt #éventuellement fichier.csr
```

Le *numéro\_série* est un nombre servant à identifier le certificat à l'intérieur de l'organisation demandeur.

Sinon, on peut générer un CSR et le signer avec son autorité de certification :

```
[root]# /usr/bin/openssl genrsa -des3 nb_bits_clé > fichier.key
[root]# /usr/bin/openssl req -new -key fichier.key -out
fichier.csr
[root]# openssl ca -cert /etc/pki/CA/cacert.pem -keyfile
/etc/pki/CA/private/cakey.pem -in fichier.csr -out fichier.crt
[root]# cat fichier.key > fichier.pem
[root]# echo "" >> fichier.pem
[root]# cat fichier.crt >> fichier.pem
[root]# rm -f fichier.key fichier.crt #éventuellement fichier.csr
```

On dispose alors d'un certificat *fichier.pem* signé par son propre CA. Cela est utile pour les VPN.

## c) Question de droits

**ATTENTION** : les fichiers **KEY**, **CSR**, **CRT** et **PEM** doivent impérativement être lisible uniquement par **ROOT**. Ils doivent avoir les droits **rw----- (600)**. Si vous ne prenez pas cette précaution, n'importe qui peut lire/copier les fichiers et se faire passer pour vous.

# V. Public Key Infrastructure : une méthode plus simple

Pour créer l'ensemble de ces clefs, on utilise l'outil *openssl*, avec des paramètres différents suivant le fichier à créer. Toutefois, des scripts sont livrés avec votre distribution afin de vous faciliter cette tâche. Vous allez donc consulter de près le fichier *Makefile* contenu dans le répertoire

/etc/pki/tls/certs et après avoir compris les différentes cibles, générer les clefs nécessaires à Apache grace à ce *Makefile*. Le fichier de configuration de https se trouve dans /etc/httpd/conf.d

Dans le dossier /etc/pki/tls/certs :

```
[root]# make fichier.key
```

```
[root]# make fichier.csr
```

```
[root]# make fichier.crt
```

ou

```
[root]# make fichier.pem
```

## VI. Configuration avec Apache

Pour obtenir un protocole HTTP sécurisé, appelé HTTPS, on utilise la couche SSL Secured Sockets Layer, qui est également utilisée avec ssh par exemple. Le protocole HTTPS utilise le port TCP 443. Assurez vous tout d'abord que vous avez le package mod\_ssl installé sur votre machine.

Dans le fichier /etc/httpd/conf.d/ssl.conf :

```
<VirtualHost tp9-pc01.prive.iut-amiens.fr:443>
# dossier des pages web du site HTTPS
DocumentRoot "/var/www/html"
# nom du serveur HTTPS
ServerName tp9-pc01.prive.iut-amiens.fr:443

# fichier de log des erreurs
ErrorLog logs/ssl_error_log
# fichier de log des accès
TransferLog logs/ssl_access_log
# niveau de log : warning
LogLevel warn

#activation de SSL pour ce virtual host
SSLEngine on

# algo de cryptage
SSLCipherSuite
ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP

# chemin du fichier certificats signé (ou autosigné). Peut être
dans /etc/pki/tls/certs
SSLCertificateFile /chemin/vers/fichier.crt #ou fichier.pem

#chemin du fichier de la paire de clé privée/publique .KEY. Peut
être dans /etc/pki/tls/private
SSLCertificateKeyFile /chemin/vers/fichier.key #ou fichier.pem

# fichier de log des requête pour le virtual host
CustomLog logs/ssl_request_log \
    "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
```

</VirtualHost>

## a) Passphrase et démarrage

Lors de la génération de la paire de clé publique/privée (pour un fichier .KEY ou .PEM), une passphrase vous sera demandée afin de protéger la clé privée.

A chaque démarrage de Apache la passphrase du fichier .KEY vous sera demandée. Ceci peut être embêtant si le serveur doit démarrer tout seul.

Pour éviter cela, on peut se passer de passphrase. (Il faudra ABSOLUMENT vérifier les droits des fichiers .KEY ou .PEM afin que la clé privée ne puisse pas être lue par n'importe qui, car elle est en clair dans le fichier).

Si l'on n'a pas encore créé le certificat, il suffit de retirer l'option `-des3` pour générer une paire de clé sans passphrase puis de continuer les étapes suivantes :

```
[root]# /usr/bin/openssl genrsa nb_bits_clé > fichier.key
```

Si on a des fichiers KEY et CRT, il faut :

- renommer le fichier *fichier.key* en *fichier.key.org* (`mv fichier.key fichier.key.org`)
- exécuter (décrypter le fichier key, s'assurer qu'il n'est lisible que par root)

```
[root]# openssl rsa -in fichier.key.org -out fichier.key
[root]# chmod 600 fichier.key
[root]# chown root.root fichier.key
```

Si on a un seul fichier PEM, il faut :

- extraire le début du fichier PEM jusqu'à la ligne vide EXCLUE dans un fichier *fichier.key.org*
- exécuter (décrypter le fichier key, s'assurer qu'il n'est lisible que par root)

```
[root]# openssl rsa -in fichier.key.org -out fichier.key
[root]# chmod 600 fichier.key
[root]# chown root.root fichier.key
```

- remplacer le début du fichier PEM jusqu'à la ligne vide EXCLUE par le contenu du fichier *fichier.key*

**Dans tous les cas, il faut vérifier que les fichiers KEY, CSR, CRT et PEM ne sont lisible que par root (600 rw-----).**

## VII. Configuration avec Dovecot

Vous allez également générer des clés pour pouvoir utiliser imap et pop en ssl (imaps et pops), et donc configurer votre serveur de mail pour qu'il utilise ssl.

Dans le fichier `/etc/dovecot/conf` :

```
protocols = imap imaps pop3 pop3s
ssl_disable=no
```

```
#chemin du fichier de la paire de clé privée/publique .KEY. Peut être dans /etc/pki/dovecot/private
ssl_key_file = /chemin/vers/fichier.key #ou fichier.pem
# chemin du fichier certificats signé (ou autosigné). Peut être dans /etc/pki/dovecot/certs
ssl_cert_file = /chemin/vers/fichier.crt #ou fichier.pem
```

## a) Génération de certificats avec dovecot

On se place dans le dossier /etc/pki/dovecot : (on prépare le terrain, peu générer des erreurs)

```
[root]# rm dovecot.pem
[root]# rm private/dovecot.pem
[root]# mkdir private certs
```

On édite le fichier dovecot-openssl.cnf :

```
[ req ]
default_bits = taille_clé #1024
encrypt_key = yes
distinguished_name = nom du demandeur du certificat
x509_extensions = cert_type
prompt = no
```

```
[ nom du demandeur ]
# country (2 letter code)
C=pays
# State or Province Name (full name)
ST=province
# Locality Name (eg. city)
L=ville
# Organization (eg. company)
O=entreprise
# Organizational Unit Name (eg. section)
OU=service ou section de l'entreprise
# Common Name (*.example.com is also possible)
CN=nom du serveur
# E-mail contact
emailAddress=email de l'administrateur
```

```
[ cert_type ]
nsCertType = server
```

On peut utiliser le script : mkcert.sh

```
[root]# cd /usr/share/doc/dovecot-x.xx.xx/examples/mkcert.sh
```

## VIII. Test de SSL ou TLS

On peut tester une connexion ssl en ligne de commande avec l'outil openssl, par exemple :

```
[root@server ~]# openssl s_client -connect localhost:443
```

```
CONNECTED(00000003)
depth=0 /C=--/ST=SomeState/L=SomeCity/O=SomeOrganization/OU=SomeOrganizationalUnit/CN=localhost.localdomain/emailAddress=root@localhost.localdomain
verify error:num=18:self signed certificate
verify return:1
depth=0 /C=--/ST=SomeState/L=SomeCity/O=SomeOrganization/OU=SomeOrganizationalUnit/CN=localhost.localdomain/emailAddress=root@localhost.localdomain
verify return:1
---
Certificate chain
 0 s:/C=--/ST=SomeState/L=SomeCity/O=SomeOrganization/OU=SomeOrganizationalUnit/CN=localhost.localdomain/emailAddress=root@localhost.localdomain
  i:/C=--/ST=SomeState/L=SomeCity/O=SomeOrganization/OU=SomeOrganizationalUnit/CN=localhost.localdomain/emailAddress=root@localhost.localdomain
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEEDjCCA3egAwIBAgICOTEwDQYJKoZIhvcNAQEEBQAwwgbsxCzAJBgNVBAYTAi0t
MRIwEAYDVQQIEw1Tb211U3RhZGUxETAPBgNVBACtCFNvbWVkaXR5MRkwFwYDVQK
ExBTb211T3JnYW5pemF0aW9uMR8wHQYDVQQLExZTb211T3JnYW5pemF0aW9uYWxV
bml0MR4wHAYDVQQDExVsb2NhbGhvc3QubG9jYWxkb21haW4xKTAnBgkqhkiG9w0B
CQEWGnJvb3RAbG9jYWxob3N0LmxvY2FsZG9tYWluMB4XDTA1MTAxMTE5Mzk1NFoX
DTA2MTAxMTE5Mzk1NFowgbsxCzAJBgNVBAYTAi0tMRIwEAYDVQQIEw1Tb211U3Rh
dGUxETAPBgNVBACtCFNvbWVkaXR5MRkwFwYDVQKExBTb211T3JnYW5pemF0aW9u
MR8wHQYDVQQLExZTb211T3JnYW5pemF0aW9uYWxVbml0MR4wHAYDVQQDExVsb2Nh
bGhvc3QubG9jYWxkb21haW4xKTAnBgkqhkiG9w0BCQEWGnJvb3RAbG9jYWxob3N0
LmxvY2FsZG9tYWluMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDMYPQLNqb/
Xcfbydinria2PijHQqnX4lFAziJXSSLswEq5q7e2tWhDbXlZgxXfc5tgjbWOk0G
i98mOG6yKa+DNsliXCPGQVX19uOt8T3Npf9Qjw6B0aAEVKJBmREV56Q+ouLSnS4m
xN+1tVfKsAR/nTXKivjdvk+uA/Q9fQ0YswIDAQABo4IBHTCCARkwhQYDVR0OBBYE
FNYypEadie35H54YFqGMTjrpcP8MIHpbGnVHSMEEgeEwg6AFNYypEadie35H54
YFqGMTjrpcP8oYHBPiG+MIG7MQswCQYDVQQGEwItLTESMBAGA1UECBMJU29tZVN0
YXR1MREwDwYDVQQHEw1Tb211Q210eTEZMBCGA1UEChMQU29tZU9yZ2FuaXphdGlv
bjEfmB0GA1UECXMWU29tZU9yZ2FuaXphdGlvbmFsVW5pdDEeMBWGA1UEAxMVbG9j
YWxob3N0LmxvY2FsZG9tYWluMSkwJwYJKoZIhvcNAQkBFhpyb290QGxvY2Fsag9z
dc5sb2NhbGRvbWVpboICOTEwDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQQFAAOB
gQBZgX6uYEHZ4PBszi7GiFGTQwd4Rg091uVo84mvFAfKMfK0TSp7bouwejDPNeA
VL4GU9DZ0oo7xZHvCFwKPPSiO4dt8WxwmsisKXrFAOxRBXjKShaFCHzuOMi8c5w7
Hq95KhwpE3Rp11BzD8ZSRkq+L9s2yviP5keOMKrvW+6KRA==
-----END CERTIFICATE-----
subject=/C=--/ST=SomeState/L=SomeCity/O=SomeOrganization/OU=SomeOrganizationalUnit/CN=localhost.localdomain/emailAddress=root@localhost.localdomain
issuer=/C=--/ST=SomeState/L=SomeCity/O=SomeOrganization/OU=SomeOrganizationalUnit/CN=localhost.localdomain/emailAddress=root@localhost.localdomain
---
No client certificate CA names sent
---
SSL handshake has read 1606 bytes and written 340 bytes
---
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 1024 bit
SSL-Session:
    Protocol    : TLSv1
    Cipher      : DHE-RSA-AES256-SHA
    Session-ID: 657FED46D7EEDF444509E52919435B9CF4BA2446F48FDE3368B58586C25FC756
    Session-ID-ctx:
    Master-Key: EECEAA37A75C168C3022927E0CF60EE31E705EEE7523A043CEF05EB9EA7E90AA
E2F9E6F9869BFAA4185E5867F2D10EFB
    Key-Arg     : None
    Krb5 Principal: None
    Start Time: 1144959797
    Timeout    : 300 (sec)
    Verify return code: 18 (self signed certificate)
```

---

# Bibliographie

[OpenSSL: The Open Source toolkit for SSL/TLS](#)

[mod\\_ssl: The Apache Interface to OpenSSL](#)

[SSL 3.0 Specification](#)

[Cryptographie - Secure Sockets Layers \(SSL\)](#)

[Transport Layer Security - Wikipédia](#)