

Public Key Infrastructure Creation HOWTO

Matthieu Hautreux
CEA/DAM Ile-de-France

November 2008

1 Introduction

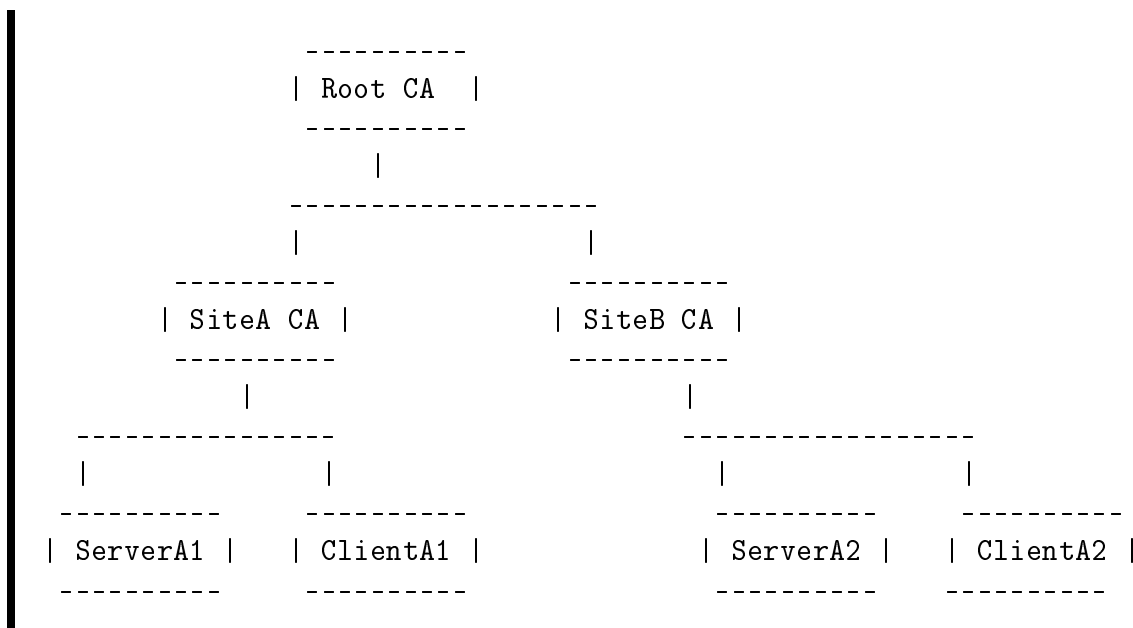
This document describes how to build a hierarchical **Public Key Infrastructure (PKI)** using **OpenSSL**.

It addresses the following topics :

- x509 Certification Authorities
 - how to build a self-signed root certificate authority
 - how to build trusted certificate authorities (signed by an upper level CA)
- x509 Certificates
 - how to build server certificates (signed by a CA)
 - how to build client certificates (signed by a CA)

The **PKI** that illustrates this **HOWTO** represents a basic double production sites company where each site manages its own *certification authority*. The *root CA* is managed by the head quarter of the company in a third site. For clarity purpose, each production site will have one server and one client only. A more complex schema should be easy to extend from this simple example.

We will set up the following hierarchical tree :



The example company will be called *CNY*.

Client and server certificates are often partially built using third party applications (e.g. **UNICORE Eclipse Client** for client certificates). We will here describe how to build them with **openssl**.

2 Certification Authorities

We here consider the creation of each required **CA**, that is to say, the root one as well as the production sites ones.

Depending on the roles of the **CA**, they will have different configuration files. For each **CA**, we will first describe how to set up **OpenSSL** and then how to create its certificate.

We will use the following naming schema :

- **Root CA** machine will be called **nodeR**
- **siteA** machine will be called **nodeA**
- **siteB** machine will be called **nodeB**

2.1 Root CA (Self-signed)

2.1.1 OpenSSL configuration

We assume that we are connected to node **nodeR**. This node will principally serves CA signature purposes. We will create 2 configuration files, one for **root CA** creation, the other for child CA certificates validation and signature.

We first create **OpenSSL** working directories and files :

```
nodeR# mkdir -p -m0755 /var/openssl/etc
nodeR# mkdir -p -m0700 /var/openssl/CA/{csr,certs,crl,private}
nodeR# touch /var/openssl/CA/index.txt
nodeR# echo 01 > /var/openssl/CA/serial
nodeR# dd if=/dev/urandom of=/var/openssl/CA/private/.rand bs=1k count=16
```

We then create the configuration file `/var/openssl/etc/ca-cert.cnf`. This configuration file is dedicated to certificate request creation :

```
[ ca ]
default_ca = CA_CNY

[ CA_CNY ]
dir = /var/openssl/CA
crl_dir = $dir/crl
new_certs_dir = $dir/certs
database = $dir/index.txt
serial = $dir/serial
certificate = $dir/ca.crt
crl = $dir/ca.crl
private_key = $dir/private/akey.pem
RANDFILE = $dir/private/.rand
default_days = 1825
default_crl_days = 30
default_md = sha1

[ req ]
```

```

default_bits = 4096
default_md = sha1
distinguished_name = req_distinguished_name
x509_extensions = rootca

[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = FR
countryName_min = 2
countryName_max = 2
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = IDF
localityName = Locality Name (eg, city)
localityName_default = Paris
0.organizationName = Organization Name (eg, company)
0.organizationName_default = CNY
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = IT
commonName = Common Name (eg, your name or your server\'s hostname)
commonName_max = 64
emailAddress = Email Address
emailAddress_max = 64

[ rootca ]
# warning, some broken softwares chokes on critical extensions...
basicConstraints = critical, CA:true
subjectKeyIdentifier = hash
keyUsage = critical, keyCertSign, cRLSign

```

We then create a configuration file, `/var/openssl/etc/ca-sign.cnf`, for signing purposes :

```

[ ca ]
default_ca = CA_CNY

[ CA_CNY ]
dir = /var/openssl/CA
crl_dir = $dir/crl
new_certs_dir = $dir/certs
database = $dir/index.txt
serial = $dir/serial
certificate = $dir/ca.crt
crl = $dir/ca.crl
private_key = $dir/private/cakey.pem
RANDFILE = $dir/private/.rand
default_days = 1825
default_crl_days = 30
default_md = sha1

```

```
# only former attributes are kept in signed cert
preserve = no
policy = Policy_CNY

[ Policy_CNY ]
countryName = match
stateOrProvinceName = optional
organizationName = match
organizationalUnitName = optional
commonName = supplied
emailAddress = optional

[ req ]
default_bits = 4096
default_md = sha1
distinguished_name = req_distinguished_name
x509_extensions = subca

[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = FR
countryName_min = 2
countryName_max = 2
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = IDF
localityName = Locality Name (eg, city)
localityName_default = Paris
0.organizationName = Organization Name (eg, company)
0.organizationName_default = CNY
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = IT
commonName = Common Name (eg, your name or your server\'s hostname)
commonName_max = 64
emailAddress = Email Address
emailAddress_max = 64

[ subca ]
# warning, some broken softwares chokes on critical extensions...
basicConstraints = critical, CA:true
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always, issuer:always
keyUsage = critical, keyCertSign, cRLSign
```

2.1.2 Root CA self-signed certificate creation

Here are the commands to enter on **nodeR** to build the **Root CA** :

```
nodeR# cd /var/openssl/CA
```

We create the required key material :

```
nodeR# openssl genrsa -aes256 -out private/cakey.pem -rand private/.rand 4096
16384 semi-random bytes loaded
Generating RSA private key, 4096 bit long modulus
.....++
.....++
.....++
.....++
.....++
e is 65537 (0x10001)
```

We set certificate request configuration file in the environment :

```
nodeR# export OPENSSL_CONF=/var/openssl/etc/ca-cert.cnf
```

We then launch the certificate request creation :

```
nodeR# openssl req -new -x509 -sha1 -days 1825 -key \
private/cakey.pem -out ca.crt
Enter pass phrase for private/cakey.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:
State or Province Name (full name) [IDF]:
Locality Name (eg, city) [Paris]:
Organization Name (eg, company) [CNY]:
Organizational Unit Name (eg, section) [IT]:Security Lab
Common Name (eg, your name or your server's hostname) []:CA
Email Address []:
```

We can now view the content of our **root CA** certificate :

```
nodeR# openssl x509 -noout -text -in ca.crt
Certificate:
```

```

Data:
  Version: 3 (0x2)
  Serial Number:
    cd:4f:46:40:18:a6:aa:74
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: C=FR, ST=IDF, L=Paris, O=CNY, OU=Security Lab, CN=CA
  Validity
    Not Before: Oct 24 12:31:19 2008 GMT
    Not After : Oct 24 12:31:19 2009 GMT
  Subject: C=FR, ST=IDF, L=Paris, O=CNY, OU=Security Lab, CN=CA
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (4096 bit)
      Modulus (4096 bit):
        4e:ac:c6:9a:5e:72:0b:12:4a:2f:7a:b5:1c:35:f7:
        ...
        d2:ca:85
      Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Basic Constraints: critical
      CA:TRUE
    X509v3 Subject Key Identifier:
      E5:C5:2B:E9:56:C4:B1:14:F2:C2:F9:C2:B1:E9:C5:DA:F5:A7:56:D5
    X509v3 Key Usage: critical
      Certificate Sign, CRL Sign
  Signature Algorithm: sha1WithRSAEncryption
    03:68:55:9e:9f:10:1a:af:cc:74:94:de:cc:d4:1b:be:20:1a:
    ...
    85:de:25:1b:cc:65:85:5f

```

2.2 Production Site CA

2.2.1 OpenSSL configuration

We assume that we are connected to host **nodeA** or **nodeB**. Those nodes will serve server and client certificate validation and signature. We will create 3 configuration files, one for CA creation, one for server certificate validation and signature and the last for client certificate validation and signature.

We first create **OpenSSL** working directories and files like before :

```

nodeA# mkdir -p -m0755 /var/openssl/etc
nodeA# mkdir -p -m0700 /var/openssl/CA/{csr,certs,crl,private}
nodeA# touch /var/openssl/CA/index.txt
nodeA# echo 01 > /var/openssl/CA/serial
nodeA# dd if=/dev/urandom of=/var/openssl/CA/private/.rand bs=1k count=16

```

We then create a configuration file, `/var/openssl/etc/ca-cert.cnf`, dedicated to CA cer-

tificate request creation :

```
[ ca ]
default_ca = CA_Prod_CNY

[ CA_Prod_CNY ]
dir = /var/openssl/CA
crl_dir = $dir/crl
new_certs_dir = $dir/certs
database = $dir/index.txt
serial = $dir/serial
certificate = $dir/siteA_ca.crt
crl = $dir/siteA_ca.crl
private_key = $dir/private/siteA_cakey.pem
RANDFILE = $dir/private/.rand
default_days = 1825
default_crl_days = 30
default_md = sha1

[ req ]
default_bits = 4096
default_md = sha1
distinguished_name = req_distinguished_name
x509_extensions = prodca

[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = FR
countryName_min = 2
countryName_max = 2
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = IDF
localityName = Locality Name (eg, city)
localityName_default = Paris
0.organizationName = Organization Name (eg, company)
0.organizationName_default = CNY
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = IT
commonName = Common Name (eg, your name or your server\'s hostname)
commonName_max = 64
emailAddress = Email Address
emailAddress_max = 64

[ prodca ]
# warning, some broken softwares chokes on critical extensions...
basicConstraints = critical, CA:true
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer:always
```

```
keyUsage = critical, keyCertSign, cRLSign
```

After that, we create a configuration file, `/var/openssl/etc/ca-server-sign.cnf`, for server certificate signing purposes :

```
[ ca ]
default_ca = CA_Prod_CNY

[ CA_Prod_CNY ]
dir = /var/openssl/CA
crl_dir = $dir/crl
new_certs_dir = $dir/certs
database = $dir/index.txt
serial = $dir/serial
certificate = $dir/siteA_ca.crt
crl = $dir/siteA_ca.crl
private_key = $dir/private/siteA_cakey.pem
RANDFILE = $dir/private/.rand
default_days = 1825
default_crl_days = 30
default_md = sha1
# only former attributes are kept in signed cert
preserve = no
policy = Policy_Prod_CNY

[ Policy_Prod_CNY ]
countryName = match
stateOrProvinceName = optional
organizationName = match
organizationalUnitName = optional
commonName = supplied
emailAddress = optional

[ req ]
default_bits = 4096
default_md = sha1
distinguished_name = req_distinguished_name
attributes = req_attributes
x509_extensions = servers

[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = FR
countryName_min 2
countryName_max = 2
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = IDF
```

```

localityName = Locality Name (eg, city)
localityName_default = Paris
0.organizationName = Organization Name (eg, company)
0.organizationName_default = CNY
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = IT
commonName = Common Name (eg, your name or your server\'s hostname)
commonName_max = 64
emailAddress = Email Address
emailAddress_max = 64

[ servers ]
# warning, some broken softwares chokes on critical extensions...
basicConstraints = critical, CA:false
authorityKeyIdentifier = keyid:always
subjectKeyIdentifier = hash
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth, clientAuth
nsCertType = server
nsComment = "Certificate issued by CNY/SiteA"
# or whatever you want
crlDistributionPoints = URI:http://siteA.cny.org/crl/server.crl
# or whatever you want
authorityInfoAccess = OCSP;URI:http://siteA.cny.org:8090/

```

The next step is to create a configuration file, `/var/openssl/etc/ca-user-sign.cnf`, for user certificate signing purposes :

```

[ ca ]
default_ca = CA_Prod_CNY

[ CA_Prod_CNY ]
dir = /var/openssl/CA
crl_dir = $dir/crl
new_certs_dir = $dir/certs
database = $dir/index.txt
serial = $dir/serial
certificate = $dir/siteA_ca.crt
crl = $dir/siteA_ca.crl
private_key = $dir/private/siteA_cakey.pem
RANDFILE = $dir/private/.rand
default_days = 1825
default_crl_days = 30
default_md = sha1
# only former attributes are kept in signed cert
preserve = no
policy = Policy_Prod_CNY

```

```
[ Policy_Prod_CNY ]
countryName = match
stateOrProvinceName = optional
organizationName = match
organizationalUnitName = optional
commonName = supplied
emailAddress = supplied

[ req ]
default_bits = 4096
default_md = sha1
distinguished_name = req_distinguished_name
attributes = req_attributes
x509_extensions = users

[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = FR
countryName_min = 2
countryName_max = 2
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = IDF
localityName = Locality Name (eg, city)
localityName_default = Paris
0.organizationName = Organization Name (eg, company)
0.organizationName_default = CNY
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = IT
commonName = Common Name (eg, your name or your server\'s hostname)
commonName_max = 64
emailAddress = Email Address
emailAddress_max = 64

[ users ]
# warning, some broken softwares chokes on critical extensions...
basicConstraints = critical, CA:false
authorityKeyIdentifier = keyid:always
subjectKeyIdentifier = hash
keyUsage = digitalSignature, keyEncipherment, nonRepudiation
extendedKeyUsage = clientAuth, emailProtection
nsCertType = client, email
nsComment = "Certificate issued by CNY/SiteA"
# or whatever you want
crlDistributionPoints = URI:http://siteA.cny.org/crl/user.crl
# or whatever you want
authorityInfoAccess = OCSP;URI:http://siteA.cny.org:8090/
```

2.2.2 Certificate signing request (CSR) creation

Here are the commands to enter on node **nodeA** to build **SiteA** CA certificate signing request (the same should be applied on site **siteB**) :

```
nodeA# cd /var/openssl/CA
nodeA# openssl genrsa -aes256 -out private/siteA_cakey.pem \
      -rand private/.rand 4096
16384 semi-random bytes loaded
Generating RSA private key, 4096 bit long modulus
.....++
.....++
.....++
.....++
.....++
e is 65537 (0x10001)
nodeA# export OPENSSL_CONF=/var/openssl/etc/ca-cert.cnf
nodeA# openssl req -new -key private/siteA_cakey.pem \
      -out csr/siteA_ca.csr
Enter pass phrase for private/siteA_cakey.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:
State or Province Name (full name) [IDF]:
Locality Name (eg, city) [Paris]:
Organization Name (eg, company) [CNY]:
Organizational Unit Name (eg, section) [IT]:Security Lab
Common Name (eg, your name or your server's hostname) []:SiteA
Email Address []:
nodeA# openssl req -noout -text -in csr/siteA_ca.csr
Certificate Request:
  Data:
    Version: 0 (0x0)
    Subject: C=FR, ST=IDF, L=Paris, O=CNY, OU=Security Lab, CN=SiteA
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (4096 bit)
        Modulus (4096 bit):
          00:f4:ea:ac:d9:dd:e2:b5:4f:a3:b8:7b:90:7c:55:
          ...
          2a:5d:6d
        Exponent: 65537 (0x10001)
    Attributes:
      a0:00
```

```

Signature Algorithm: sha1WithRSAEncryption
25:f0:e1:8f:5f:1b:1d:97:84:15:7c:9d:ef:98:74:19:8d:f0:
...
08:d2:d9:45:79:72:7c:e6

```

The **CSR** can now be send to **Root CA** for approval and signature.

2.2.3 CSR validation and signature (CA certificate creation)

Here are the commands to enter on **nodeR** for *certificate signing requests* validation and signature. We assume that `/tmp/siteA_ca.csr` and `/tmp/siteB_ca.csr` are the two received request files.

```

nodeR# cd /var/openssl/CA
nodeR# export OPENSSL_CONF=/var/openssl/etc/ca-sign.cnf
nodeR# cp /tmp/siteA_ca.csr csr/
nodeR# openssl ca -in csr/siteA_ca.csr -out certs/siteA_ca.crt
Using configuration from /var/openssl/etc/ca-sign.cnf
Enter pass phrase for /var/openssl/CA/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:'IDF'
localityName         :PRINTABLE:'Paris'
organizationName     :PRINTABLE:'CNY'
organizationalUnitName:PRINTABLE:'Security Lab'
commonName           :PRINTABLE:'SiteA'
Certificate is to be certified until Oct 24 13:07:00 2009 GMT (1825 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
nodeR# openssl x509 -noout -text -in certs/siteA_ca.crt
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 1 (0x1)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=FR, ST=IDF, L=Paris, O=CNY, OU=Security Lab, CN=CA
        Validity
            Not Before: Oct 24 13:07:00 2008 GMT
            Not After : Oct 24 13:07:00 2009 GMT
        Subject: C=FR, ST=IDF, O=CNY, OU=Security Lab, CN=SiteA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption

```

```

    RSA Public Key: (4096 bit)
      Modulus (4096 bit):
        00:f4:ea:ac:d9:dd:e2:b5:4f:a3:b8:7b:90:7c:55:
        ...
        2a:5d:6d
      Exponent: 65537 (0x10001)
Signature Algorithm: sha1WithRSAEncryption
  50:8a:7e:07:f2:b6:99:1a:ee:93:b4:56:d4:4c:dd:85:22:a2:
  ...
  97:b4:fe:b1:f7:1b:83:31

```

```

nodeR# cp /tmp/siteB_ca.csr csr/
nodeR# openssl ca -in csr/siteB_ca.csr -out certs/siteB_ca.crt
Using configuration from /var/openssl/etc/ca-sign.cnf
Enter pass phrase for /var/openssl/CA/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:'IDF'
localityName          :PRINTABLE:'Paris'
organizationName      :PRINTABLE:'CNY'
organizationalUnitName:PRINTABLE:'Security Lab'
commonName            :PRINTABLE:'SiteB'
Certificate is to be certified until Oct 24 13:11:46 2009 GMT (1825 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
nodeR# openssl x509 -noout -text -in certs/siteB_ca.crt
Certificate:
  Data:
    Version: 1 (0x0)
    Serial Number: 2 (0x2)
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: C=FR, ST=IDF, L=Paris, O=CNY, OU=Security Lab, CN=CA
    Validity
      Not Before: Oct 24 13:11:46 2008 GMT
      Not After : Oct 24 13:11:46 2009 GMT
    Subject: C=FR, ST=IDF, O=CNY, OU=Security Lab, CN=SiteB
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (4096 bit)
        Modulus (4096 bit):
          00:ce:03:f9:37:68:52:c3:f9:58:93:f4:ad:9c:71:
          ...

```

```

12:26:27
Exponent: 65537 (0x10001)
Signature Algorithm: sha1WithRSAEncryption
50:8d:e8:58:01:0b:b9:30:cf:be:f5:6c:74:66:4d:0d:a5:ae:
...
13:b8:f5:51:d6:a9:3d:39

```

`/var/openssl/CA/certs/siteA_ca.crt` and `/var/openssl/CA/certs/siteB_ca.crt` can now be send to corresponding production site for further use (we directly put those files into the directory mentioned in their respective **openssl** configuration file) :

```

nodeR# scp /var/openssl/CA/certs/siteA_ca.crt nodeA:/var/openssl/CA/
nodeR# scp /var/openssl/CA/certs/siteB_ca.crt nodeB:/var/openssl/CA/

```

Certification authorities are now created and ready to serve.

3 Server certificates creation

The method commonly used to get a valid certificate for a server is the same as for a non root CA :

- key material generation (server side)
- **CSR** (Certificate Signing Request) generation using this material (server side)
- **CSR** validation and signature (corresponding CA side)

CA can also manage all the process but it's not the recommended way because of security concerns. **CSR** enables server owners not to expose their private keys. When **CA** manages server certificate creation, both of the **CA** and the servers will know the private part of the keys and that can be a problem. We will treat here the basic approach, that is to say, the approach that let the server create and keep safe its key material.

We will consider that the machine hosting **serverA** is called **nodeSA** and the one hosting **serverB** is **nodeSB**.

3.1 Server's OpenSSL configuration

We have to configure **OpenSSL** to build a **CSR** for our two servers, **serverA** on **siteA** and **serverB** on site **siteB**. We will only describe the process on **siteA**, **siteB** will be the roughly same.

We suppose that we are on the server for which we want to get a certificate on **siteA**.

We first create **OpenSSL** directories as usual :

```

nodeSA# mkdir -p -m0755 /var/openssl/etc
nodeSA# mkdir -p -m0700 /var/openssl/{csr,certs,crl,private}
nodeSA# dd if=/dev/urandom of=/var/openssl/private/.rand bs=1k count=16

```

We then create a configuration file `/var/openssl/etc/server-cert.cnf`, for certificate request creation :

```

[ req ]
default_bits = 1024
default_md = sha1
distinguished_name = req_distinguished_name
x509_extensions = servers

[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = FR
countryName_min = 2
countryName_max = 2
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = IDF
localityName = Locality Name (eg, city)
localityName_default = Paris
0.organizationName = Organization Name (eg, company)
0.organizationName_default = CNY
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = IT
commonName = Common Name (eg, your name or your server\'s hostname)
commonName_max = 64
emailAddress = Email Address
emailAddress_max = 64

[ servers ]
# warning, some broken softwares chokes on critical extensions...
basicConstraints = critical, CA:false
authorityKeyIdentifier = keyid:always
subjectKeyIdentifier = hash
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth, clientAuth
nsCertType = server

```

3.2 Server's CSR creation

We first create a couple of private and public keys.

```

nodeSA# cd /var/openssl
nodeSA# openssl genrsa -aes256 -out private/serverA_key.pem \
        -rand private/.rand 1024
16384 semi-random bytes loaded
Generating RSA private key, 1024 bit long modulus
.....++++++
.....++++++
e is 65537 (0x10001)
Enter pass phrase for private/serverA_key.pem:

```

```
Verifying - Enter pass phrase for private/serverA_key.pem:
nodeSA#
```

We then use this key material to create a **CSR** :

```
nodeSA# cd /var/openssl
nodeSA# export OPENSSL_CONF=/var/openssl/etc/server-cert.cnf
nodeSA# openssl req -new -key private/serverA_key.pem -out csr/serverA.csr
Enter pass phrase for private/serverA_key.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:
State or Province Name (full name) [IDF]:
Locality Name (eg, city) [Paris]:
Organization Name (eg, company) [CNY]:
Organizational Unit Name (eg, section) [IT]:
Common Name (eg, your name or your server's hostname) []:serverA.siteA.cny.org
Email Address []:
nodeSA# openssl req -noout -text -in csr/serverA.csr
Certificate Request:
  Data:
    Version: 0 (0x0)
    Subject: C=FR, ST=IDF, L=Paris, O=CNY, OU=IT, CN=serverA.siteA.cny.org
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (1024 bit)
        Modulus (1024 bit):
          00:ee:cd:f8:d3:70:d7:8c:cd:81:b8:fb:58:8f:dd:
          ...
          bc:9d:b3:ab:67:23:ab:0e:f5
        Exponent: 65537 (0x10001)
    Attributes:
      a0:00
  Signature Algorithm: sha1WithRSAEncryption
    01:ec:79:5c:4b:91:34:ca:12:46:6c:56:0f:01:5b:81:3c:1e:
    ...
    c6:3d
nodeSA#
```

It's now possible to send the **CSR** to the *SiteA CA* for validation and signature.

3.3 Server's CSR validation and signature

Here we suppose that we are back on host *nodeA* and that serverA CSR is in */tmp*. Here are the commands to create the certificate :

```
nodeA# cd /var/openssl/CA
nodeA# export OPENSSL_CONF=/var/openssl/etc/ca-server-sign.cnf
nodeA# cp /tmp/serverA.csr csr/
nodeA# openssl ca -in csr/serverA.csr -out certs/serverA.crt
Using configuration from /var/openssl/etc/ca-server-sign.cnf
Enter pass phrase for /var/openssl/CA/private/siteA_cakey.pem:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'FR'
stateOrProvinceName  :PRINTABLE:'IDF'
localityName          :PRINTABLE:'Paris'
organizationName      :PRINTABLE:'CNY'
organizationalUnitName:PRINTABLE:'IT'
commonName            :PRINTABLE:'serverA.siteA.cny.org'
Certificate is to be certified until Oct 24 14:37:46 2009 GMT (1825 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
nodeA# openssl x509 -noout -text -in certs/serverA.crt
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 1 (0x1)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=FR, ST=IDF, O=CNY, OU=Security Lab, CN=SiteA
        Validity
            Not Before: Oct 24 14:37:46 2008 GMT
            Not After : Oct 24 14:37:46 2009 GMT
        Subject: C=FR, ST=IDF, O=CNY, OU=IT, CN=serverA.siteA.cny.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:ee:cd:f8:d3:70:d7:8c:cd:81:b8:fb:58:8f:dd:
                    ...
                    bc:9d:b3:ab:67:23:ab:0e:f5
                Exponent: 65537 (0x10001)
        Signature Algorithm: sha1WithRSAEncryption
        7b:cd:72:5d:95:c8:36:06:1c:f5:f9:b6:df:28:ff:91:60:f7:
        ...
        b8:7e:65:e9:89:b6:1a:9e
```

```
nodeA#
```

Certificate file, `certs/serverA.crt`, is now ready to send back to the server `serverA` on node `nodeSA`. The same should be applied on node `nodeSB` for `serverB`.

3.4 Server's certificate and keys combinations

It is often recommended to combine the key material and the corresponding certificate into a unique file. One format that can be used to do this wrapping is `pkcs12`. We here suppose that we are on node `nodeSA` and that we are completing this combination (the same procedure should be done on `nodeSB`) :

```
nodeSA# cd /var/openssl
nodeSA# openssl pkcs12 -export -in serverA.crt \
    -inkey private/serverA_key.pem -out private/serverA.p12
```

The `p12` file can now be used by the server to secure its services.

4 Users certificates creation

The method commonly used to get a valid certificate for a user is the same as for a server :

- key material generation (user side)
- **CSR** (*certificate signing request*) generation using this material (user side)
- **CSR** validation and signature (corresponding CA side)

CA can also manage all the process but it's not the recommended way because of security concerns. **CSR** enables users not to expose their private keys. When **CA** manages user certificates creation, both of the **CA** and the users will know the private part of the keys and that can be a problem. We will treat here the basic approach, that is to say, the approach that let the user create and keep safe its key material.

We will consider that the machine used by user `userA` is called `nodeUA` and the one used by user `userB` is `nodeUB`.

4.1 User's OpenSSL configuration

We have to configure **OpenSSL** to build a **CSR** for our two users, `userA` on `siteA` and `userB` on site `siteB`. We will only describe the process on `siteA`, the one on `siteB` will be roughly the same.

We suppose that we are on a linux box with **OpenSSL** installed, connected as `userA`.

We first create **OpenSSL** directories as usual :

```
nodeUA# mkdir -p -m0700 ~userA/openssl/{etc,csr,certs,private}
nodeUA# dd if=/dev/urandom of=~userA/openssl/private/.rand bs=1k count=16
```

We then create a configuration file, `userA/openssl/etc/user-cert.cnf`, for certificate signing request creation :

```
[ req ]
default_bits = 1024
default_md = sha1
distinguished_name = req_distinguished_name
x509_extensions = users

[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = FR
countryName_min = 2
countryName_max = 2

stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = IDF

localityName = Locality Name (eg, city)
localityName_default = Paris

0.organizationName = Organization Name (eg, company)
0.organizationName_default = CNY

organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = IT

commonName = Common Name (eg, your name or your server\'s hostname)
commonName_max = 64

emailAddress = Email Address
emailAddress_max = 64

[ users ]
basicConstraints = critical, CA:false
authorityKeyIdentifier = keyid:always
subjectKeyIdentifier = hash
keyUsage = digitalSignature, keyEncipherment, nonRepudiation
extendedKeyUsage = clientAuth, emailProtection
nsCertType = client, email
```

4.2 User's CSR creation

We first create a couple of private and public keys.

```
nodeUA# cd ~userA/openssl
nodeUA# openssl genrsa -aes256 -out private/userA_key.pem -rand private/.rand 1024
16384 semi-random bytes loaded
Generating RSA private key, 1024 bit long modulus
```

```

.....+++++
.....+++++
e is 65537 (0x10001)
Enter pass phrase for private/serverA_key.pem:
Verifying - Enter pass phrase for private/userA_key.pem:
nodeUA#

```

We then use this key material to create a **CSR** :

```

nodeUA# cd ~userA/openssl
nodeUA# export OPENSSL_CONF=~userA/openssl/etc/user-cert.cnf
nodeUA# openssl req -new -key private/userA_key.pem -out csr/userA.csr
Enter pass phrase for private/userA_key.pem:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:
State or Province Name (full name) [IDF]:
Locality Name (eg, city) [Paris]:
Organization Name (eg, company) [CNY]:
Organizational Unit Name (eg, section) [IT]:
Common Name (eg, your name or your server's hostname) []:userA
Email Address []:userA@cny.org
nodeUA# openssl req -noout -text -in csr/userA.csr
Certificate Request:
  Data:
    Version: 0 (0x0)
    Subject: C=FR, ST=IDF, L=Paris, O=CNY, OU=IT, CN=userA/emailAddress=userA@cny.o
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (1024 bit)
        Modulus (1024 bit):
          00:c8:7c:cd:bc:32:c9:b8:ae:86:82:ac:07:86:e0:
          ...
          74:77:44:21:d4:9c:8d:19:81
        Exponent: 65537 (0x10001)
    Attributes:
      a0:00
    Signature Algorithm: sha1WithRSAEncryption
      8a:a1:90:ae:1b:27:9b:5e:54:68:86:1c:98:e2:68:5e:1f:2d:
      ...
      b2:93
nodeUA#

```

It's now possible to send the **CSR** to the *SiteA CA* for validation and signature.

4.3 User's CSR validation and signature

Here we suppose that we are back on host **nodeA** and that *userA CSR* is in **/tmp**. The command lines to create the certificate are :

```
nodeA# cd /var/openssl/CA
nodeA# export OPENSSL_CONF=/var/openssl/etc/ca-user-sign.cnf
nodeA# cp /tmp/userA.csr csr/userA.csr
nodeA# openssl ca -in csr/userA.csr -out certs/userA.crt
Using configuration from /var/pki/siteA/etc/ca-user-sign.cnf
Enter pass phrase for /var/pki/siteA/CA/private/siteA_cakey.pem:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :PRINTABLE:'IDF'
localityName          :PRINTABLE:'Paris'
organizationName      :PRINTABLE:'CNY'
organizationalUnitName:PRINTABLE:'IT'
commonName            :PRINTABLE:'userA'
emailAddress          :IA5STRING:'userA@cny.org'
Certificate is to be certified until Oct 27 10:46:53 2009 GMT (1825 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
nodeUA# openssl x509 -noout -text -in certs/userA.crt
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 2 (0x2)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=FR, ST=IDF, O=CNY, OU=Security Lab, CN=SiteA
        Validity
            Not Before: Oct 27 10:46:53 2008 GMT
            Not After : Oct 27 10:46:53 2009 GMT
        Subject: C=FR, ST=IDF, O=CNY, OU=IT, CN=userA/emailAddress=userA@cny.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c8:7c:cd:bc:32:c9:b8:ae:86:82:ac:07:86:e0:
                    ...
                    74:77:44:21:d4:9c:8d:19:81
```

```

                Exponent: 65537 (0x10001)
Signature Algorithm: sha1WithRSAEncryption
                bf:a8:55:49:1a:d1:46:dc:fd:1a:98:50:a0:81:07:cb:73:6b:
                ...
                05:7c:55:d2:93:fc:73:3a
nodeUA#

```

Certificate file, **certs/userA.crt**, is now ready to send back to the user on node **nodeUA**. The same should be applied on node **nodeUB** for **userB**.

4.4 User's certificate and keys combinations

As for servers, it is often recommended to combine users' key material and their corresponding certificate into unique files using the **pkcs12** format. We here suppose that we are on node **nodeUA** and that we are completing this combination for **userA** (the same procedure should be done on **nodeUB** for **userB**) :

```

nodeUA# cd ~userA/openssl
nodeUA# openssl pkcs12 -export -in userA.crt \
                -inkey private/userA_key.pem -out private/userA.p12

```

The **p12** file can now be used to securely access **PKI** resources (like **serverA**, **serverB**,...).

5 Important files summary

Here is a brief retrospective of created files and their roles.

5.1 Certification Authorities related

- on node **nodeR**, the **self-signed CA** :
 - **/var/openssl/etc/ca-sign.cnf** : configuration files used for validation and signature of **CSR**
 - **/var/openssl/CA/private/cakey.pem** : private key of the **root CA**
 - **/var/openssl/CA/ca.crt** : x509 certificate of the **root CA**
 - **/var/openssl/CA/certs/siteA_ca.crt** : x509 certificate of **siteA CA**
 - **/var/openssl/CA/certs/siteB_ca.crt** : x509 certificate of **siteB CA**
- on node **nodeA**, production site **siteA CA** :
 - **/var/openssl/etc/ca-server-sign.cnf** : configuration files used for validation and signature of servers **CSR**
 - **/var/openssl/etc/ca-user-sign.cnf** : configuration files used for validation and signature of users **CSR**
 - **/var/openssl/CA/private/siteA_cakey.pem** : private key of **siteA CA**
 - **/var/openssl/CA/siteA_ca.crt** : x509 certificate of **siteA CA** received from **root CA**
 - **/var/openssl/CA/certs/serverA.crt** : x509 certificate of **serverA**
 - **/var/openssl/CA/certs/userA.crt** : x509 certificate of user **userA**

- on node **nodeB**, production site **siteB CA** :
 - **/var/openssl/etc/ca-server-sign.cnf** : configuration files used for validation and signature of servers **CSR**
 - **/var/openssl/etc/ca-user-sign.cnf** : configuration files used for validation and signature of users **CSR**
 - **/var/openssl/CA/private/siteB_cakey.pem** : private key of **siteB CA**
 - **/var/openssl/CA/siteB_ca.crt** : x509 certificate of **siteB CA** received from **root CA**
 - **/var/openssl/CA/certs/serverB.crt** : x509 certificate of server **serverA**
 - **/var/openssl/CA/certs/userB.crt** : x509 certificate of user **userA**

5.2 Production Server related

- on node **nodeSA**, server **serverA** of production site **siteA** :
 - **/var/openssl/private/serverA_key.pem** : private key of **serverA**
 - **/var/openssl/certs/serverA.crt** : x509 certificate of **serverA** received from **siteA CA**
 - **/var/openssl/private/serverA.p12** : cyphered combination of the two previous items
- on node **nodeSB**, server **serverB** of production site **siteB** :
 - **/var/openssl/private/serverB_key.pem** : private key of **serverB**
 - **/var/openssl/certs/serverB.crt** : x509 certificate of **serverB** received from **siteB CA**
 - **/var/openssl/private/serverB.p12** : cyphered combination of the two previous items

5.3 User related

- on node **nodeUA**, user **userA** 's machine of production site **siteA** :
 - **userA/openssl/private/userA_key.pem** : private key of **userA**
 - **userA/openssl/certs/userA.crt** : x509 certificate of **userA** received from **siteA CA**
 - **userA/openssl/private/userA.p12** : cyphered combination of the two previous items
- on node **nodeUB**, user **userB** 's machine of production site **siteB** :
 - **userB/openssl/private/userB_key.pem** : private key of **userB**
 - **userB/openssl/certs/userB.crt** : x509 certificate of **userB** received from **siteB CA**
 - **userB/openssl/private/userB.p12** : cyphered combination of the two previous items