
Eclectica

Software
Documentation
SSL Certificates
Self-Check Digits
Bare Metal Reload
Linux
popbsmtpd
Reference
Installation
Changelog
FAQ
Mailing Lists
Download
Postfix
EnGarde
iSeries
CPYTOIFSF
FTP Backup
Windows

Creating and Using SSL Certificates

This document describes how to establish yourself as a root certificate authority (root CA) using the OpenSSL toolset. As a root CA, you are able to sign and install certificates for use in your Internet server applications, such as Apache and Stunnel.

Table of Contents

Scope

This document covers a very specific, limited purpose, but one that meets a common need: preventing browser, mail, and other clients from complaining about the certificates installed on your server.

Not covered is dealing with a commercial root certificate authority (CA). Instead, we will become our own root CA, and sign our own certificates.

These procedures were developed using OpenSSL 0.9.6, 24 Sep 2000, on Linux.

Back to top

Quick Start

Those who want to start creating certificates right away without reading this whole document should skip to the summary at the end.

Back to top

Background

Why be our own root CA? So that we can take advantage of SSL encryption without spending unnecessary money on having our certificates signed.

A drawback is that browsers will still complain about our site not being trusted until our root certificate is imported. However, once this is done, we are no different from the commercial root CAs.

Clients will only import our root certificate if they trust us. This is where the commercial CAs come in: they purport to do extensive research into the people and organizations for whom they sign certificates. By importing (actually, by the browser vendors incorporating) their trusted root certificates, we are saying that we trust them when they guarantee that someone else is who they say they are. We can trust additional root CAs (like ourselves) by importing their CA certificates.

Note: If you are in the business of running a commercial secure site, obtaining a commercially signed certificate is the only realistic choice.

Back to top

Prerequisites

You will need an installed copy of OpenSSL for this, which is available from http://www.openssl.org. Chances are it is already installed on your machine. This document will not cover the installation procedure.

Back to top

Initial Setup

First, we will create a directory where we can work. It does not matter where this is; I am arbitrarily going to create it in my home directory.

```
# mkdir CA
# cd CA
# mkdir newcerts private
```

The CA directory will contain:

- Our Certificate Authority (CA) certificate
- The database of the certificates that we have signed
- The keys, requests, and certificates we generate

It will also be our working directory when creating or signing certificates.

The CA/newcerts directory will contain:

- A copy of each certificate we sign

The CA/private directory will contain:

- Our CA private key

This key is important:

- Do not lose this key. Without it, you will not be able to sign or renew any certificates.
- Do not disclose this key to anyone. If it is compromised, others will be able to impersonate you.

Our next step is to create a database for the certificates we will sign:

```
# echo '01' >serial
# touch index.txt
```

Rather than use the configuration file that comes with OpenSSL, we are going to create a minimal configuration of our own in this directory. Start your editor (vi, pico, ...) and create a basic openssl.cnf:

```
---Begin---
#
# OpenSSL configuration file.
#

# Establish working directory.

dir                     = .

----End----
```

Back to top

Creating a Root Certificate

With OpenSSL, a large part of what goes into a certificate depends on the contents of the configuration file, rather than the command line. This is a good thing, because there is a lot to specify.

The configuration file is divided into sections, which are selectively read and processed according to openssl command line arguments. Sections can include one or more other sections by referring to them, which helps to make the configuration file more modular. A name in square brackets (e.g. "[req]") starts each section.

We now need to add the section that controls how certificates are created, and a section to define the type of certificate to create.

The first thing we need to specify is the Distinguished Name. This is the text that identifies the owner of the certificate when it is viewed. It is not directly referenced in the configuration file, but is included into the section processed when certificate requests are created. The command is "openssl req <args>", so the section is titled [req].

Add the following to openssl.cnf:

```
---Begin---

[ req ]
default_bits            = 1024                  # Size of keys
default_keyfile         = key.pem               # name of generated keys
default_md              = md5                   # message digest algorithm
string_mask             = nombstr               # permitted characters
distinguished_name      = req_distinguished_name

[ req_distinguished_name ]
# Variable name           Prompt string
#----------------------   ----------------------------------
0.organizationName      = Organization Name (company)
organizationalUnitName  = Organizational Unit Name (department, division)
emailAddress            = Email Address
emailAddress_max        = 40
localityName            = Locality Name (city, district)
stateOrProvinceName     = State or Province Name (full name)
countryName             = Country Name (2 letter code)
countryName_min         = 2
countryName_max         = 2
commonName              = Common Name (hostname, IP, or your name)
commonName_max          = 64

# Default values for the above, for consistency and less typing.
# Variable name                   Value
#------------------------------   ------------------------------
0.organizationName_default      = The Sample Company
localityName_default            = Metropolis
stateOrProvinceName_default     = New York
countryName_default             = US

[ v3_ca ]
basicConstraints        = CA:TRUE
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid:always,issuer:always

----End----
```

In order to protect ourselves from unauthorized use of our CA certificate, it is passphrase protected. Each time you use the CA certificate to sign a request, you will be prompted for the passphrase. Now would be a good time to pick a secure passphrase and put it in a safe place.

All the preparation is now in place for creating our self-signed root certificate. For this, we want to override some of the defaults we just put into the configuration, so we will specify our overrides on the command line.

Our overrides to the "openssl req" command are:

- Create a new self-signed certificate: -new -x509
- Create a CA certificate: -extensions v3_ca
- Make it valid for more than 30 days: -days 3650
- Write output to specific locations: -keyout, -out
- Use our configuration file: -config ./openssl.cnf

(A note on the term of validity of root certificates: When a root certificate expires, all of the certificates signed with it are no longer valid. To correct this situation, a new root certificate must be created and distributed. Also, all certificates signed with the expired one must be revoked, and re-signed with the new one. As this can be a lot of work, you want to make your root certificate valid for as long as you think you will need it. In this example, we are making it valid for ten years.)

Run the command as shown. In this case, the PEM pass phrase it asks for is a new one, which you must enter twice:

```
# openssl req -new -x509 -extensions v3_ca -keyout private/cakey.pem \
-out cacert.pem -days 3650 -config ./openssl.cnf
Using configuration from ./openssl.cnf
Generating a 1024 bit RSA private key
.......++++++
.........................++++++
writing new private key to 'private/cakey.pem'
Enter PEM pass phrase:demo
Verifying password - Enter PEM pass phrase:demo
-----
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.
-----
Organization Name (company) [The Sample Company]:<enter>
Organizational Unit Name (department, division) []:CA Division
Email Address []:ca@sample.com
Locality Name (city, district) [Metropolis]:<enter>
State or Province Name (full name) [New York]:<enter>
Country Name (2 letter code) [US]:<enter>
Common Name (hostname, IP, or your name) []:TSC Root CA
```

This process produces two files as output:

- A private key in **private/cakey.pem**
- A root CA certificate in **cacert.pem**

cacert.pem is the file you want to distribute to your clients.

The private key (cakey.pem) looks like this:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,0947F49BB28FE5F4

jlQvt9WdR9Vpg3WQT5+C3HU17bUOwvhp/r0+viMcBUCRW85UqI2BJJKTi1IwQQ4c
tyTrhYJYOP+A6JXt5BzDzZy/B7tjEMDBosPiwH2m4MaP+6wTbi1qR1pFDL3fXYDr
ZsuN08dkbw9ML6LOX5Rl6bIBL3i5hnGiqm338Fl52gNstThv0C/OZhXT3B4qsJn8
qZb3mC6U2nRaP/NpZPcEx4lv2vH7OzHTu1TZ7t0asSpgpuH58dfHPw775kZDep2F
LXA3Oeavg0TLFHkaFBUx2xaeEG6Txpt9I74aAsw1T6UbTSjqgtsK0PHdjPNfPGlY
5U3Do1pnU9hfoem/4RAOe0cCovP/xf6YPBraSFPs4XFfnWwgEtL09ReFqO9T0aSp
5ajLyBOYOBKQ3PCSu1HQDw/OzphInhKxdYg81WBBEfELzSdMFQZgmfGrt5DyyWmq
TADwWtGVvO3pEhO1STmCaNqZQSpSwEGPGo5RFkyFvyvyozWX2SZg4g1o1X40qSg9
0FMHTEB5HQebEkKBoRQMCJN/uyKXTLjNB7ibtVbZmfjsi9oNd3NJNVQQH+o9I/rP
wtFsjs+t7SKrsFB2cxZQdDlFzD6EBA+5ytebGEI1lJHcOUEa6P+LTphlwh/o1QuN
IKX2YKHA4ePrBzdgZ+xZuSLn/Qtjg/eZv6i73VXoHk8EdxfOk5xkJ+DnsNmyx0vq
W53+O05j5xsxzDJfWrllqBlFF/OkIYCPcyK1iLs4GOwe/V0udDNwr2Uw90tefr3q
X1OZ9Dix+U0u6xXTZTETJ5dF3hV6GF7hP3Tmj9/UQdBwBzr+D8YWzQ==
-----END RSA PRIVATE KEY-----
```

Of course, you don't want to show this to anyone! Needless to say, the one shown here is now useless as a private key.

The certificate (cacert.pem) looks like this:

```
-----BEGIN CERTIFICATE-----
MIIDrTCCAxagAwIBAgIBADANBgkqhkiG9w0BAQQFADCBnDEbMBkGA1UEChMSVGhl
IFNhbXBsZSBDb21wYW55MRQwEgYDVQQLEwtDQSBEaXZpc2lvbjEcMBoGCSqGSIb3
DQEJARYNY2FAc2FtcGxlLmNvbTETMBEGA1UEBxMKTWV0cm9wb2xpczERMA8GA1UE
CBMITmV3IFlvcmsxCzAJBgNVBAYTAlVTMRQwEgYDVQQDEwtUU0MgUm9vdCBDQTAe
Fw0wMTEyMDgwNDI3MDVaFw0wMjEyMDgwNDI3MDVaMIGcMRswGQYDVQQKExJUaGUg
U2FtcGxlIENvbXBhbnkxFDASBgNVBAsTC0NBIERpdmlzaW9uMRwwGgYJKoZIhvcN
AQkBFg1jYUBzYW1wbGUuY29tMRMwEQYDVQQHEwpNZXRyb3BvbGlzMREwDwYDVQQI
EwhOZXcgWW9yazELMAkGA1UEBhMCVVMxFDASBgNVBAMTC1RTQyBSb290IENBMIGf
MA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDaiAwfKB6ZBtnTRTIo6ddomt0S9ec0
NcuvtJogt0s9dXpHowh98FCDjnLtCi8du6LDTZluhlOtTFARPlV/LVnpsbyMCXMs
G2qpdjJop+XIBdvoCz2HpGXjUmym8WLqt+coWwJqUSwiEba74JG93v7TU+Xcvc00
5MWnxmKZzD/R3QIDAQABo4H8MIH5MAwGA1UdEwQFMAMBAf8wHQYDVR0OBBYEFG/v
yytrBtEquMX2dreysix/MlPMMIHJBgNVHSMEgcEwgb6AFG/vyytrBtEquMX2drey
six/MlPMoYGipIGfMIGcMRswGQYDVQQKExJUaGUgU2FtcGxlIENvbXBhbnkxFDAS
BgNVBAsTC0NBIERpdmlzaW9uMRwwGgYJKoZIhvcNAQkBFg1jYUBzYW1wbGUuY29t
MRMwEQYDVQQHEwpNZXRyb3BvbGlzMREwDwYDVQQIEwhOZXcgWW9yazELMAkGA1UE
BhMCVVMxFDASBgNVBAMTC1RTQyBSb290IENBggEAMA0GCSqGSIb3DQEBBAUAA4GB
ABclymJfsPOUazNQO8aIaxwVbXWS+8AFEkMMRx6O68ICAMubQBvs8Buz3ALXhqYe
FS5G13pW2ZnAlSdTkSTKkE5wGZ1RYSfyiEKXb+uOKhDN9LnajDzaMPkNDU2NDXDz
SqHk9ZiE1boQaMzjNLu+KabTLpmL9uXvFA/i+gdenFHv
-----END CERTIFICATE-----
```

We can query the contents of this certificate with openssl to learn to whom belongs, what it is valid for, etc.:

```
# openssl x509 -in cacert.pem -noout -text
# openssl x509 -in cacert.pem -noout -dates
# openssl x509 -in cacert.pem -noout -purpose
```

Back to top

Creating a Certificate Signing Request (CSR)

Now that we have a root certificate, we can create any number of certificates for installation into our SSL applications such as https, spop, or simap. The procedure involves creating a private key and certificate request, and then signing the request to generate the certificate.

Our configuration file needs some more definitions for creating non-CA certificates. Add the following at the end of the file:

```
---Begin---
[ v3_req ]
basicConstraints        = CA:FALSE
subjectKeyIdentifier    = hash

----End----
```

To avoid having to repeatedly put this on the command line, insert the following line to the [req] section after the distinguished_name line as shown:

```
---Begin---
distinguished_name      = req_distinguished_name
req_extensions          = v3_req

----End----
```

Now we are ready to create our first certificate request. In this example, we are going to create a certificate for a secure POP server at mail.sample.com. Everything looks the same as when we created the CA certificate, but three of the ensuing prompts get different responses.

- Organizational Unit: a reminder of what the certificate is for
- Email Address: the postmaster
- Common Name: the server hostname

The Common Name must be (or the IP address must resolve to) the server name your clients use to contact your host. If this does not match, every time they connect your clients will get a message asking them if they want to use this server. In effect, the client software is saying, "Warning! You asked for **mail.sample.com**; the responding machine's certificate is for **smtp.sample.com**. Are you sure you want to continue?"

```
# openssl req -new -nodes -out req.pem -config ./openssl.cnf
...
Organizational Unit Name (department, division) []:Mail Server
Email Address []:postmaster@sample.com
Common Name (hostname, IP, or your name) []:mail.sample.com
...
```

This process produces two files as output:

- A private key in **key.pem**
- A certificate signing request in **req.pem**

These files should be kept. When the certificate you are about to create expires, the request can be used again to create a new certificate with a new expiry date. The private key is of course necessary for SSL encryption. When you save these files, meaningful names will help; for example, mailserver.key.pem and mailserver.req.pem.

The certificate signing request looks like this:

```
-----BEGIN CERTIFICATE REQUEST-----
MIICJDCCAY0CAQAwgagxGzAZBgNVBAoTElRoZSBTYW1wbGUgQ29tcGFueTEUMBIG
A1UECxMLTWFpbCBTZXJ2ZXIxJDAiBgkqhkiG9w0BCQEWFXBvc3RtYXN0ZXJAc2Ft
cGxlLmNvbTETMBEGA1UEBxMKTWV0cm9wb2xpczERMA8GA1UECBMITmV3IFlvcmsx
CzAJBgNVBAYTAlVTMRgwFgYDVQQDEw9tYWlsLnNhbXBsZS5jb20wgZ8wDQYJKoZI
hvcNAQEBBQADgY0AMIGJAoGBAPJhc++WxcBaoDbJpzFbDg42NcOz/ELVFMU4FlPa
yUzUO+xXkdFRMPKo54d4Pf1w575Jhlu9lE+kJ8QN2st6JFySbc9QjPwVwl9D2+I3
SSf2kVTu+2Ur5izCPbVAfU0rPZxxK8ELoOkA1uwwjFz6EFuVvnHwlguonWKDtmYW
u7KTAgMBAAGgOzA5BgkqhkiG9w0BCQ4xLDAqMAkGA1UdEwQCMAAwHQYDVR0OBBYE
FLWaQsUVIQzWr58HtDinH1JfeCheMA0GCSqGSIb3DQEBBAUAA4GBAAbe0jrGEQ3i
tyVfy5Lg4/f69rKvDGs+uhZJ9ZRx7Dl92Qq2osE7XrLB1bANmcoEv/ORLZOjWZEY
NjMvuz60O7R8GKBrvb/YhAwWhIIt2LJqPkpAEWS0kY0AkoQcfZ7h6oC35+eJ7okg
Uu3WuE57RgcNt7/ftr0sG1jUyRwMLvhv
-----END CERTIFICATE REQUEST-----
```

We can view the contents to make sure our request is correct:

```
# openssl req -in req.pem -text -verify -noout
```

Back to top

Signing a Certificate

Now we need to add the configuration file section that deals with being a Certificate Authority. This section will identify the paths to the various pieces, such as the database, the CA certificate, and the private key. It also provides some basic default values. Insert the following into openssl.cnf just before the [req] section:

```
---Begin---
[ ca ]
default_ca              = CA_default

[ CA_default ]
serial                  = $dir/serial
database                = $dir/index.txt
new_certs_dir           = $dir/newcerts
certificate             = $dir/cacert.pem
private_key             = $dir/private/cakey.pem
default_days            = 365
default_md              = md5
preserve                = no
email_in_dn             = no
nameopt                 = default_ca
certopt                 = default_ca
policy                  = policy_match

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

----End----
```

To sign the request we made in the previous step, execute the following and respond to the prompts. Note that you are asked for the PEM passphrase selected earlier:

```
# openssl ca -out cert.pem -config ./openssl.cnf -infiles req.pem
Using configuration from ./openssl.cnf
Enter PEM pass phrase:demo
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
organizationName      :PRINTABLE:'The Sample Company'
organizationalUnitName:PRINTABLE:'Mail Server'
emailAddress          :IA5STRING:'postmaster@sample.com'
localityName          :PRINTABLE:'Metropolis'
stateOrProvinceName   :PRINTABLE:'New York'
countryName           :PRINTABLE:'US'
commonName            :PRINTABLE:'mail.sample.com'
Certificate is to be certified until Dec  8 04:37:38 2002 GMT (365 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
```

This process updates the CA database, and produces two files as output:

- A certificate in **cert.pem**
- A copy of the certificate in **newcerts/<serial>.pem**

Again, you can inspect the certificate:

```
# openssl x509 -in cert.pem -noout -text -purpose | more
```

The certificate has both the encoded version and a human-readable version in the same file. You can strip off the human-readable portion as follows:

```
# mv cert.pem tmp.pem
# openssl x509 -in tmp.pem -out cert.pem
```

Back to top

Installing the Certificate and Key

This depends on the application. Some want the key and the certificate in the same file, and others want them separately. Combining them is easily done with:

```
# cat key.pem cert.pem >key-cert.pem
```

After this step, you have three installable components to choose from:

- A private key in **key.pem**
- A certificate in **cert.pem**
- A combined private key and certificate in **key-cert.pem**

Copy the appropriate files into the locations specified by the instructions for your application and system. Restart the applications, and you are in operation with your new certificate.

Apache

Apache has separate configuration directives for the key and the certificate, so we keep each in its own file. These files should be kept outside of the DocumentRoot subtree, so a reasonable directory structure might be:

File	Comment
/home/httpd/html	Apache DocumentRoot
/home/httpd/ssl	SSL-related files
/home/httpd/ssl/cert.pem	Site certificate
/home/httpd/ssl/key.pem	Site private key

Within the <VirtualHost> directive for the site (which of course should be on port 443), include the directives that point to these files:

```
<VirtualHost 192.168.1.1:443>
   ServerName mail.sample.com
   DocumentRoot /home/httpd/html
   ... other directives for this site ...
   SSLEngine on
   SSLLog /var/log/ssl_engine_log
   SSLCertificateFile /home/httpd/ssl/cert.pem
   SSLCertificateKeyFile /home/httpd/ssl/key.pem
</VirtualHost>
```

Stunnel

stunnel is used as an SSL wrapper for normal non-secure services such as IMAP and POP. It accepts as arguments (among other things) the service to execute, and the location of the certificate and private key.

The key and the certificate are provided in the same file. These can go anywhere, but a good location might be **/etc/ssl/certs**. Specify it on the stunnel command line as follows:

```
stunnel -p /etc/ssl/certs/key-cert.pem <other stunnel args...>
```

More to come...

Back to top

Distributing the CA Certificate

This, finally, is the step that stops the clients from complaining about untrusted certificates. Send **cacert.pem** to anyone who is going to use your secure servers, so they can install it in their browsers, mail clients, et cetera as a root certificate.

Back to top

Renewing Certificates

Your certificate chain can break due to certificate expiry in two ways:

- The certificates you signed with your root certificate have expired.
- Your root certificate itself has expired.

In the second case, you have some work to do. A new root CA certificate must be created and distributed, and then your existing certificates must be recreated or re-signed.

In the first case, you have two options. You can either generate new certificate signing requests and sign them as described above, or (if you kept them) you can re-sign the original requests. In either case, the old certificates must be revoked, and then the new certificates signed and installed into your secure applications as described earlier.

You cannot issue two certificates with the same Common Name, which is why the expired certificates must be revoked. The certificate is in the newcerts directory; you can determine its filename by browsing index.txt and searching for the Common Name (CN) on it. The filename is the index plus the extension ".pem", for example "02.pem". To revoke a certificate:

```
# openssl ca -revoke newcerts/02.pem -config ./openssl.cnf
Using configuration from ./openssl.cnf
Enter PEM pass phrase: demo
Revoking Certificate 02.
Data Base Updated
```

Now that the certificate has been revoked, you can re-sign the original request, or create and sign a new one as described above.

Back to top

Getting a Commercially Signed Certificate

The process is basically the same as the one just demonstrated, but the CA does most of it. You need to generate a Certificate Signing Request as shown above, and then submit it for signing. You will receive a signed certificate for installation.

This certificate will automatically be trusted by your client's browser, as the browser has the commercial CA's certificate built in. There is no need to distribute anything.

The configuration described here may be inadequate for this purpose, as there is much more that can go into a request. Different certificate authorities require different features in the certificate signing request, none of which we have gone into here. This additional material is beyond the current scope of this document.

Back to top

Publishing Your CA Certificate

You can post the certificate on your web site for download. If you do this, you should also post a Certificate Revocation List (CRL), and a means of displaying a certificate given its serial number. This is outside the current scope of this document.

Apache will serve your certificate in a form recognizable to browsers if you specify its MIME type. For example, you can use the filename extension ".crt" for downloadable certificates, and put the following into the general section of your Apache configuration:

```
AddType application/x-x509-ca-cert .crt
```

Now you can post the certificate for download with a link like
Our Root Certificate, and when the link is followed the visitor's browser would offer to install the certificate.

The CRL can be created as follows:

```
# openssl ca -gencrl -crldays 31 -config ./openssl.cnf -out rootca.crl
```

More to come...

Back to top

Summary

You now have enough information to create and sign certificates on your own behalf. While this is a fairly long document, the procedure can be summarized easily.

One-Time Setup

Set up, and create a root CA certificate.

Commands

```
# mkdir CA
# cd CA
# mkdir newcerts private
# echo '01' >serial
# touch index.txt
# (IMPORTANT: Install and edit the configuration file shown below.)
# openssl req -new -x509 -extensions v3_ca -keyout private/cakey.pem \
-out cacert.pem -days 365 -config ./openssl.cnf
```

Output

File	Purpose
cacert.pem	CA certificate
private/cakey.pem	CA private key

Distribute **cacert.pem** to your clients.

Per Certificate

Create certificate signing requests and sign them, supplying appropriate values for the Common Name and the Organizational Unit.

Commands

```
# openssl req -new -nodes -out req.pem -config ./openssl.cnf
# openssl ca -out cert.pem -config ./openssl.cnf -infiles req.pem
# cat key.pem cert.pem >key-cert.pem
```

Output

File	Purpose
key.pem	Private key
req.pem	Certificate signing request
cert.pem	Certificate
key-cert.pem	Combined private key and certificate

Install **key.pem** and **cert.pem**, or just **key-cert.pem** as appropriate for your server application.

Per Certificate - Renewal

Revoke the expired certificate, and re-sign the original request.

Commands

```
# openssl ca -revoke newcerts/<serial>.pem -config ./openssl.cnf
# openssl ca -out cert.pem -config ./openssl.cnf -infiles req.pem
```

Install the renewed certificates in the same manner as the original ones.

Back to top

Configuration File

(This file is available for download.)

```
---Begin---
#
# OpenSSL configuration file.
#

# Establish working directory.

dir                     = .

[ ca ]
default_ca              = CA_default

[ CA_default ]
serial                  = $dir/serial
database                = $dir/index.txt
new_certs_dir           = $dir/newcerts
certificate             = $dir/cacert.pem
private_key             = $dir/private/cakey.pem
default_days            = 365
default_md              = md5
preserve                = no
email_in_dn             = no
nameopt                 = default_ca
certopt                 = default_ca
policy                  = policy_match

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

[ req ]
default_bits            = 1024                  # Size of keys
default_keyfile         = key.pem               # name of generated keys
default_md              = md5                   # message digest algorithm
string_mask             = nombstr               # permitted characters
distinguished_name      = req_distinguished_name
req_extensions          = v3_req

[ req_distinguished_name ]
# Variable name           Prompt string
#----------------------   ----------------------------------
0.organizationName      = Organization Name (company)
organizationalUnitName  = Organizational Unit Name (department, division)
emailAddress            = Email Address
emailAddress_max        = 40
localityName            = Locality Name (city, district)
stateOrProvinceName     = State or Province Name (full name)
countryName             = Country Name (2 letter code)
countryName_min         = 2
countryName_max         = 2
commonName              = Common Name (hostname, IP, or your name)
commonName_max          = 64

# Default values for the above, for consistency and less typing.
# Variable name                   Value
#------------------------------   ------------------------------
0.organizationName_default      = The Sample Company
localityName_default            = Metropolis
stateOrProvinceName_default     = New York
countryName_default             = US

[ v3_ca ]
basicConstraints        = CA:TRUE
subjectKeyIdentifier    = hash
authorityKeyIdentifier  = keyid:always,issuer:always

[ v3_req ]
basicConstraints        = CA:FALSE
subjectKeyIdentifier    = hash

----End----
```

Back to top

References

More information is available at the following sites (opens in new window):

- OpenSSL Home Page
- OpenSSL Documentation
- OpenSSL FAQ
- Nick Burch's Certificate Management and Installation with OpenSSL
- Franck Martin's SSL Certificates HOWTO

Back to top

Researched and written by Marcus Redivo.
Permission to use this document for any purpose is hereby granted, providing that the copyright information and this disclaimer is retained. Author accepts no responsibility for any consequences arising from the use of this information.
