

Working with Certificate and Key Files in MatrixSSL

Generating Certificates for use with MatrixSSL

The most common way to obtain a certificate is to buy one from a commercial certificate authority. This will result in a public key that has been digitally signed by a trusted third-party so that clients using the certificate can be very sure they are communicating with the entity they think they are. However, in a trusted environment it is possible to use an unsigned certificate or to create an in-house CA to self-sign the certificate.

MatrixSSL is designed as an embedded library to enable SSL for devices and does not contain code to generate keys or certificates. There are several free software packages available to do this. This document describes how to use the standard OpenSSL package to generate certificates. It contains complete instructions on how to obtain your own certificates suitable for use with MatrixSSL. For instructions on how to use these keys with MatrixSSL are provided in the MatrixSSL developers guide and the MatrixSSL API document.

Using the OpenSSL package to create certificates

OpenSSL is a widely used SSL toolkit available for free download at <http://www.openssl.org>. It comes with a command line utility for generating keys, creating CAs, and creating certificates. The following instructions assume the OpenSSL package has been installed and configured properly. These instructions will walk you through using OpenSSL to create an unsigned certificate, to create a Certificate Authority to sign your own certificates, and to generate the proper requests in order to receive a signed certificate from a commercial CA.

Creating an unsigned certificates

The certificate generated from these steps should never be used in any environment other than for internal testing purposes. The private key file is generated in an unencrypted format and the certificate is created from a generic configuration supplied in the OpenSSL package.

1. Create an RSA key

With the OpenSSL package installed, create an empty directory to work in. From the command prompt type the following to generate an unencrypted `privkey.pem` file with a 1024 bit key:

```
openssl genrsa -out privkey.pem 1024
```

2. Create a test certificate

OpenSSL uses a configuration text file to read information about the certificate being generated. This file is typically created or edited to suit the specific details of the subject, but there is a test configuration file provided in the OpenSSL distribution. The file is named `test.cnf` and can be found in the top level `test` directory of where the OpenSSL distribution was installed. In order for the file to

be found a system environment variable called `OPENSSL_CONF` must be set to reference that specific file.

On a Linux system type the following to set the environment variable (assuming an installation directory of `/OpenSSL`):

```
export OPENSSL_CONF=/OpenSSL/test/test.cnf
```

On a Windows system type the following to set the environment variable (assuming an installation directory of `C:\OpenSSL`):

```
set OPENSSL_CONF=C:\OpenSSL\test\test.cnf
```

Now the certificate can be generated with the following command (all text is a single command typed on one line):

```
openssl req -new -x509 -key privkey.pem -out cert.pem -days 1095
```

There should now be a *cert.pem* and *privkey.pem* file in the current directory for use in testing a MatrixSSL session.

Creating a Certificate Authority to sign your own certificates

The certificate generated from these steps could potentially be suitable for use outside a testing environment. In this case **you** are the trusted authority, so the users of these certificates would have to be confident of your identity. For example, off-site users of a corporate network could secure their communications with the company network using certificates signed by a company created Certificate Authority.

For this example we will create a test Certificate Authority. In order to store the files created, create an empty directory called *testCA* with the subfolders *certs* and *private*. The *testCA* directory should be the working directory of the command shell when following these instructions.

1. Create the test CA environment

The creation of a CA will produce several files that should be preserved throughout the life of the CA. You can sign an unlimited number of certificates with a single CA, these files will be written to each time you sign a certificate.

- a. Create a new file named *serial* and add *01* as the only text
- b. Create an empty file named *index.txt*

2. Create the test CA configuration file

While you could simply enter all the configuration information into the command line, creating a configuration file makes these steps much easier to reproduce and allows you to save the options used to create a CA.

Create a new file named *CAcnf.cnf* and add the following basic CA configuration information:

```
[ ca ]
default_ca = exampleca

[ exampleca ]
dir = /testCA
certificate = $dir/cacert.pem
database = $dir/index.txt
new_certs_dir = $dir/certs
private_key = $dir/private/caprivkey.pem
serial = $dir/serial

default_crl_days = 7
default_days = 365
default_md = md5

policy = exampleca_policy
x509_extensions = certificate_extensions

[ exampleca_policy ]
commonName = supplied
stateOrProvinceName = supplied
countryName = supplied
emailAddress = supplied
organizationName = supplied
organizationalUnitName = optional

[ certificate_extensions ]
basicConstraints = CA:false

[ req ]
dir = /testCA
default_bits = 1024
default_keyfile = $dir/private/caprivkey.pem
default_md = md5

prompt = no
distinguished_name = root_ca_dn
x509_extensions = root_ca_extensions

[ root_ca_dn ]
commonName = My CA
stateOrProvinceName = Washington
countryName = US
emailAddress = myemail@mydomain.com
organizationName = My Organization

[ root_ca_extensions ]
basicConstraints = CA:true
```

The two *dir* entries should be set to the path to the *testCA* directory created earlier. The *root_ca_dn* section could be changed to enter information specific to your organization, although for testing purposes this isn't necessary.

3. Create a self-signed root certificate

A certificate authority is essentially a self-signed root certificate. This root certificate is then used to respond to new certificate requests to create a signed certificate. In this case, we are the CA and the requestor so there are no identity verification issues, but in a more typical situation, a CA can only be trusted if they do sufficient background checks into the requestor of the certificate to verify their identity.

- a. Set the OPENSSL_CONF system environment variable to point to the newly created configuration file:

```
OPENSSL_CONF=/testCA/CAcnf.cnf
export OPENSSL_CONF
```

- on a Windows system type the following -

```
set OPENSSL_CONF=C:\testCA\CAcnf.cnf
```

- b. Enter the command to generate the self-signed root certificate (all text is a single command typed on one line):

```
openssl req -x509 -newkey rsa -out cacert.pem -outform PEM
```

- c. You will then be prompted for a 'PEM pass phrase', this will be your password to the CA private key. It is essential to the security of the system that this password and the CA private key are kept secret.

At the conclusion of this step, there should be an encrypted *caprivkey.pem* in the *private* subdirectory that is the private key file for the CA. There should also be a self-signed *cacert.pem* file in the top level of the *testCA* directory that will be used to sign new certificate requests in the next steps.

4. Create a Certificate Request

Now that the CA has been created, we can use it to sign new certificates. In this example, we're playing the role of the CA, the certificate subject, and the end-user of the certificate so we don't have to worry about any trust issues. But the typical process involves communication between the certificate subject(you) and a trusted CA. Usually someone wishing to issue certificates to an end user would generate a certificate request file and submit it to the administrators of a CA. Once the administrators of the CA have determined the request to be valid, a self-signed root certificate would be used to sign the certificate request and create a new certificate to be returned to requestor, and eventually the end user.

- a. Reset the OPENSSL_CONF environment variable to the default *openssl.cnf* file. Generating a request has nothing to do with a CA before it is actually submitted. It is safe to point OPENSSL_CONF to the default configuration file because it will force the request command to prompt the

user for all information regarding the certificate request. Set the environment variable to the default file by typing the following:

```
OPENSSL_CONF=/OpenSSL/apps/openssl.cnf
export OPENSSL_CONF
```

- or on Windows -

```
set OPENSSL_CONF=C:\OpenSSL\apps\openssl.cnf
```

b. Generate the request with the following command (the text is one command to be typed on a single line) and answer all questions at the prompt:

```
openssl req -newkey rsa:1024 -keyout myprivkey.pem -keyform PEM
-out myreq.pem -outform PEM
```

If you do not want an encrypted private key, add *-nodes* to the above command. At the conclusion of this step two new files will have been created. The *myprivkey.pem* file containing the encrypted private key. This file should never be shared, not even with the CA. The other file is the certificate request file, *myreq.pem*, that will be used by the CA to create the final signed certificate.

5. Use the test CA to issue the certificate

The final step of the process is to use the CA self-signed certificate to sign the certificate and return it to the requestor (subject).

a. Reset the OPENSSL_CONF system environment variable to reference the CA configuration file again.

```
OPENSSL_CONF=/testCA/CAcnf.cnf
export OPENSSL_CONF
```

- on a Windows system type the following -

```
set OPENSSL_CONF=C:\testCA\CAcnf.cnf
```

b. Assure that the request file is in the current directory and run the following command. The prompt for the PEM password is the password to the CA private key file:

```
openssl ca -in myreq.pem
```

Answer 'y' at the next two prompts, then at the conclusion of this step, several files will have been updated and a new certificate will have been created. The new certificate can be found in the *certs* subdirectory of the CA directory structure. It will be named as the serial number it is associated with by the CA. The file can be renamed to whatever the subject would like, but the .pem extension should be preserved for clarity. The *serial* file itself will have

incremented its count for the next certificate request and the *index.txt* file will show a record of the creation.

The new certificate file and the *myprivkey.pem* file are now suitable for use in MatrixSSL.

Obtaining a certificate from a commercial certificate authority

The certificate generated from these steps is suitable for use in a production environment (ie. an Internet Web Site conducting financial transactions).

1. Create a certificate request

A commercial certificate authority requires the subject to generate a certificate request to send to them. The CA will then take the request along with payment and perform some background checks to verify the subject is a valid entity and return a digitally signed certificate.

- a. Set the OPENSSL_CONF system environment variable to the default *openssl.cnf* file. It is safe to point OPENSSL_CONF to the default configuration file because it will force the request command to prompt the user for all information regarding the certificate request. Set the environment variable to the default file by typing the following:

```
OPENSSL_CONF=/OpenSSL/apps/openssl.cnf
export OPENSSL_CONF
```

- or on Windows -

```
set OPENSSL_CONF=C:\OpenSSL\apps\openssl.cnf
```

- b. Generate the request with the following command (the text is one command to be typed on a single line) and answer all questions at the prompt:

```
openssl req -newkey rsa:1024 -keyout myprivkey.pem -keyform PEM
-out myreq.pem -outform PEM
```

At the conclusion of this step two new files will have been created. The *myprivkey.pem* file containing the encrypted private key. Again, this file is never to be shared, not even with the CA. The other file is the certificate request file, *myreq.pem* that will be sent to the CA to create the final signed certificate.

2. Send the request to a CA

Select a reputable CA and work within their guidelines for submitting a certificate request. Choose to receive the returned certificate in an X.509 PEM format. The resulting certificate along with the private key is suitable for use in MatrixSSL.