Becoming a X.509 Certificate Authority

Normally when you configure a server to use TLS or SSL you have two choices; Either you pay someone like Verisign or Thawte to sign a certificate or you generate a self-signed certificate. However there is an alternative, which is to generate your own certificate authority or CA. This article will explain how to generate your CA and sign and revoke certificates, as well as how to get common applications to trust your CA.

Why Become Your Own Certificate Authority

Which route you choose depends on your circumstances and why you need a certificate. For a large public service like an e-commerce website, you'll want a certificate signed by an established trusted root CA, who, like Verisign, have their root keys bundled with web browsers and operating systems. This allows anyone to trust your server is the server it claims to be and traffic is encrypted, without having to install any additional certificate. The downside to this is the cost of getting a certificate. At the time of writing, Verisign were charging 2,480USD for a 3 year 128bit certificate. Other companies can provide you with certificates for around 100USD for a year.

If on the other hand you have a personal website with a webmail application on, you probably want to use https to prevent anyone sniffing your login password. For a personal site it isn't worth paying 100USD a year for a trusted certificate. In this case you'd generate a self-signed certificate. That is a certificate signed by itself. This allows you to use https for encryption, but your browser can not trust the server as there is no chain from a trusted root to the server certificate. The client will probably also give you several warnings that it isn't trusted and give you the option of whether you want to trust it. The advantage of self-signed certificates is that they are free, but the disadvantages are that you can't use them for trust and your client will complan every time you connect.

If you have a small (or large) organisation, you may have several different services that need encrypting. You'll need one for every website (e.g. www, intranet, wiki, webmail) and you'd need one for mail services. This soon mounts up to quite a few certificates. Because you have a limited number of users, it becomes possible to generate your own root CA and distribute the public key to your users. This has the advantages of being free and can be used for trusting servers, but at the expense of requiring your users to import your root certificate. This article will explain how to go about generating your certificate authority and using it to sign and revoke

certificates. It will also explain how to import the root certificate into common applications.

Creating A Root Certificate Authority

We will be using the openssl program which is included in the OpenSSL SSL suite. It should be possible to use other tools like GnuTLS or SSLeay, but they are not described here.

The first thing to do is to create a location for storing your files. You need to keep your private key secure from unauthorised access. Create a new directory for your CA directory layout. All the commands will be run from inside this new directory.

```
mkdir -p root-ca/{conf,private,public}
chmod 400 root-ca/private
cd root-ca
```

I like to use a separate configuration file for the CA, rather than use the central configuration file. The central file is fine for generating certificates and signing requests, but the CA needs some special options. During each section I'll build up the config file until we have a complete config file for the CA. Create `conf/openssl.cnf` with the following contents:

```
[ req ]
default_bits            = 2048
default_keyfile         = ./private/root.pem
default_md              = sha1
prompt                  = no
distinguished_name      = root_ca_distinguished_name
x509_extensions = v3_ca

[ root_ca_distinguished_name ]
countryName             = UK
stateOrProvinceName     = Sussex
localityName            = Brighton
0.organizationName      = Example Inc
commonName              = Example Inc Root CA
emailAddress            = david@example.com

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

The only section you need to change is the `root_ca_distinguished_name` options to define your location and organisation that will appear in your trusted root. Once you've done that, you can generate your key.

```
root$ openssl req -nodes -config conf/openssl.cnf -days 1825 \
   -x509 -newkey rsa:1024 -out public/root.pem -outform PEM
Generating a 1024 bit RSA private key
.........++++++
...++++++
writing new private key to './private/root.pem'
-----
```

You should now have a `public/root.pem` file and a `private/root.pem` which contains the public and private key respectively. You may want to configure the number of bits in the key by changing the `rsa:1024` to `rsa:2048` for 2048bit keys. You may also want to change the expiry of the key by changing the days option. I've generated a key that will last 5 years (1825 days).

Lets look at these files. They are base64 encoded, so they are plain ASCII files. We can also decode the files using openssl. The private key using the rsa subcommand and the public key using the x509 subcommand.

```
root$ cat private/root.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQD0Yl4ySPCWCOuA+T2JnbGBStlgmaIsnayy5TPArq0jFGheq0Ch
hRiK/rkCs6YB9smsCu/cSXr0IVCZWlBa82GdZ3v3vBULm9E/Xlc7pU3LNijBeQbk
bZajXWzRrrEEwM0TawjIUbdh3tM/hBUO0MyuP29IvsLv6R/jdCiyu+r16wIDAQAB
AoGAUGODRNtFeLUacuSJ35jl6PfsJaCj/jUnt/k5WuqavW1bH7kJm5giiKZdyYu6
0e6STGuUl0v8jA3ha3CP+jsRIodvX+270f/0XsLLR9Jki6sGAposhX9G5MKlBIJb
cSRpITqlXlbAzp2l3zv/xkh7O0GQD95A2MVSw13J7EZExSECQQD/FlhjaMIt9iLI
9Y7EMC9g9i3d+lugk5Warc42xs0Qecyvao7kSq8U4OnFO4o93h0BnAYCgDUJamwT
jvPBDuFjAkEA9UI4C4M5+HrGDIKuKm1cMC/ryWtM4dwUaoqtyUhYIgNEaHZue7Vb
VhzWNc0Fpsrl7JXBV5+47KJjEIxTmrtD2QJBAKh2CF92Yq7Z23Ud/nAAZnnBet8b
JQHyFZ32ZLZ2Kh1UYuICHGYWEHRYaOpRfOWI110lvi5OnzgdqUxtXvtDqYMCQQDx
q3iST5KV+FSjCzoXcH+AtBfA98xgWn57pX8tce32RgXocZrYp5ICvpONnCLPgesi
P0au0yJjenfnXH8dXx2xAkEAnGynAkEb9JwQKoaWLlA2yiuRGT0Q+qz248Ni6cyA
soEu1m8bH6SYh7p8UOmkDKksoGLZIXbUmumVHQjzzzb/Xg==
-----END RSA PRIVATE KEY-----
root$ openssl rsa -in private/root.pem -noout -text
Private-Key: (1024 bit)
modulus:
    00:f4:62:5e:32:48:f0:96:08:eb:80:f9:3d:89:9d:
    b1:81:4a:d9:60:99:a2:2c:9d:ac:b2:e5:33:c0:ae:
    ad:23:14:68:5e:ab:40:a1:85:18:8a:fe:b9:02:b3:
    a6:01:f6:c9:ac:0a:ef:dc:49:7a:f4:21:50:99:5a:
    50:5a:f3:61:9d:67:7b:f7:bc:15:0b:9b:d1:3f:5e:
    57:3b:a5:4d:cb:36:28:c1:79:06:e4:6d:96:a3:5d:
    6c:d1:ae:b1:04:c0:cd:13:6b:08:c8:51:b7:61:de:
    d3:3f:84:15:0e:d0:cc:ae:3f:6f:48:be:c2:ef:e9:
    1f:e3:74:28:b2:bb:ea:f5:eb
publicExponent: 65537 (0x10001)
privateExponent:
```

```
    50:63:83:44:db:45:78:b5:1a:72:e4:89:df:98:e5:
    e8:f7:ec:25:a0:a3:fe:35:27:b7:f9:39:5a:ea:9a:
    bd:6d:5b:1f:b9:09:9b:98:22:88:a6:5d:c9:8b:ba:
    d1:ee:92:4c:6b:94:97:4b:fc:8c:0d:e1:6b:70:8f:
    fa:3b:11:22:87:6f:5f:ed:bb:d1:ff:f4:5e:c2:cb:
    47:d2:64:8b:ab:06:02:9a:2c:85:7f:46:e4:c2:a5:
    04:82:5b:71:24:69:21:3a:a5:5e:56:c0:ce:9d:a5:
    df:3b:ff:c6:48:7b:3b:41:90:0f:de:40:d8:c5:52:
    c3:5d:c9:ec:46:44:c5:21
prime1:
    00:ff:16:58:63:68:c2:2d:f6:22:c8:f5:8e:c4:30:
    2f:60:f6:2d:dd:fa:5b:a0:93:95:9a:ad:ce:36:c6:
    cd:10:79:cc:af:6a:8e:e4:4a:af:14:e0:e9:c5:3b:
    8a:3d:de:1d:01:9c:06:02:80:35:09:6a:6c:13:8e:
    f3:c1:0e:e1:63
prime2:
    00:f5:42:38:0b:83:39:f8:7a:c6:0c:82:ae:2a:6d:
    5c:30:2f:eb:c9:6b:4c:e1:dc:14:6a:8a:ad:c9:48:
    58:22:03:44:68:76:6e:7b:b5:5b:56:1c:d6:35:cd:
    05:a6:ca:e5:ec:95:c1:57:9f:b8:ec:a2:63:10:8c:
    53:9a:bb:43:d9
exponent1:
    00:a8:76:08:5f:76:62:ae:d9:db:75:1d:fe:70:00:
    66:79:c1:7a:df:1b:25:01:f2:15:9d:f6:64:b6:76:
    2a:1d:54:62:e2:02:1c:66:16:10:74:58:68:ea:51:
    7c:e5:88:d7:5d:25:be:2e:4e:9f:38:1d:a9:4c:6d:
    5e:fb:43:a9:83
exponent2:
    00:f1:ab:78:92:4f:92:95:f8:54:a3:0b:3a:17:70:
    7f:80:b4:17:c0:f7:cc:60:5a:7e:7b:a5:7f:2d:71:
    ed:f6:46:05:e8:71:9a:d8:a7:92:02:be:93:8d:9c:
    22:cf:81:eb:22:3f:46:ae:d3:22:63:7a:77:e7:5c:
    7f:1d:5f:1d:b1
coefficient:
    00:9c:6c:a7:02:41:1b:f4:9c:10:2a:86:96:2e:50:
    36:ca:2b:91:19:3d:10:fa:ac:f6:e3:c3:62:e9:cc:
    80:b2:81:2e:d6:6f:1b:1f:a4:98:87:ba:7c:50:e9:
    a4:0c:a9:2c:a0:62:d9:21:76:d4:9a:e9:95:1d:08:
    f3:cf:36:ff:5e

root$ cat public/root.pem
-----BEGIN CERTIFICATE-----
MIIDYzCCAsygAwIBAgIJAJjXuFpakUIPMA0GCSqGSIb3DQEBBQUAMH8xCzAJBgNV
BAYTAlVLMQ8wDQYDVQQIEwZTdXNzZXgxETAPBgNVBAcTCEJyaWdodG9uMQ8wDQYD
VQQKEwZDYXRuaXAxFzAVBgNVBAMTDkNhdG5pcCBSb290IENBMSIwIAYJKoZIhvcN
AQkBFhNkYXZpZEBjYXRuaXAub3JnLnVrMB4XDTA2MDQwODE3MDUzNVoXDTExMDQw
NzE3MDUzNVowfzELMAkGA1UEBhMCVUsxDzANBgNVBAgTBlN1c3NleDERMA8GA1UE
BxMIQnJpZ2h0b24xDzANBgNVBAoTBkNhdG5pcDEXMBUGA1UEAxMOQ2F0bmlwIFJv
b3QgQ0ExIjAgBgkqhkiG9w0BCQEWE2RhdmlkQGNhdG5pcC5vcmcudWswgZ8wDQYJ
```

```
KoZIhvcNAQEBBQADgY0AMIGJAoGBAPRiXjJI8JYI64D5PYmdsYFK2WCZoiydrLLl
M8CurSMUaF6rQKGFGIr+uQKzpgH2yawK79xJevQhUJlaUFrzYZ1ne/e8FQub0T9e
VzulTcs2KMF5BuRtlqNdbNGusQTAzRNrCMhRt2He0z+EFQ7QzK4/b0i+wu/pH+N0
KLK76vXrAgMBAAGjgeYwgeMwHQYDVR0OBBYEFNX+V7njwm+VfCmNdxLXlAlD1ixS
MIGzBgNVHSMEgaswgaiAFNX+V7njwm+VfCmNdxLXlAlD1ixSoYGEpIGBMH8xCzAJ
BgNVBAYTAlVLMQ8wDQYDVQQIEwZTdXNzZXgxETAPBgNVBAcTCEJyaWdodG9uMQ8w
DQYDVQQKEwZDYXRuaXAxFzAVBgNVBAMTDkNhdG5pcCBSb290IENBMSIwIAYJKoZI
hvcNAQkBFhNkYXZpZEBjYXRuaXAub3JnLnVrggkAmNe4WlqRQg8wDAYDVR0TBAUw
AwEB/zANBgkqhkiG9w0BAQUFAAOBgQA23D/zpCC9lOxjzkAd7ZXTwrPbfgH9MmeB
gOnzwpvlequTnTHB2g9h5lSOLb3y0iT5Pd1sIsIPohMqLy8uN0abOoVeUxAjudUM
kfz2ZyiD0Fz/V1KOXrqTl5Xr/M+yuqslYJUC/E4Lpjl6JDaJ+DsdoiNNuJCLqBGK
0JjKsPTB2w==
-----END CERTIFICATE-----
root$ openssl x509 -in public/root.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            98:d7:b8:5a:5a:91:42:0f
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=UK, ST=Sussex, L=Brighton, O=Example Inc,
CN=Example Inc Root CA/emailAddress=david@example.com
        Validity
            Not Before: Apr  8 17:05:35 2006 GMT
            Not After : Apr  7 17:05:35 2011 GMT
        Subject: C=UK, ST=Sussex, L=Brighton, O=Example Inc,
CN=Example Inc Root CA/emailAddress=david@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:f4:62:5e:32:48:f0:96:08:eb:80:f9:3d:89:9d:
                    b1:81:4a:d9:60:99:a2:2c:9d:ac:b2:e5:33:c0:ae:
                    ad:23:14:68:5e:ab:40:a1:85:18:8a:fe:b9:02:b3:
                    a6:01:f6:c9:ac:0a:ef:dc:49:7a:f4:21:50:99:5a:
                    50:5a:f3:61:9d:67:7b:f7:bc:15:0b:9b:d1:3f:5e:
                    57:3b:a5:4d:cb:36:28:c1:79:06:e4:6d:96:a3:5d:
                    6c:d1:ae:b1:04:c0:cd:13:6b:08:c8:51:b7:61:de:
                    d3:3f:84:15:0e:d0:cc:ae:3f:6f:48:be:c2:ef:e9:
                    1f:e3:74:28:b2:bb:ea:f5:eb
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:

D5:FE:57:B9:E3:C2:6F:95:7C:29:8D:77:12:D7:94:09:43:D6:2C:52
            X509v3 Authority Key Identifier:

keyid:D5:FE:57:B9:E3:C2:6F:95:7C:29:8D:77:12:D7:94:09:43:D6:2C:52
                DirName:/C=UK/ST=Sussex/L=Brighton/O=Example
```

```
Inc/CN=Example Inc Root CA/emailAddress=david@example.com
                serial:98:D7:B8:5A:5A:91:42:0F

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha1WithRSAEncryption
        36:dc:3f:f3:a4:20:bd:94:ec:63:ce:40:1d:ed:95:d3:c2:b3:
        db:7e:01:fd:32:67:81:80:e9:f3:c2:9b:e5:7a:ab:93:9d:31:
        c1:da:0f:61:e6:54:8e:2d:bd:f2:d2:24:f9:3d:dd:6c:22:c2:
        0f:a2:13:2a:2f:2f:2e:37:46:9b:3a:85:5e:53:10:23:b9:d5:
        0c:91:fc:f6:67:28:83:d0:5c:ff:57:52:8e:5e:ba:93:97:95:
        eb:fc:cf:b2:ba:ab:25:60:95:02:fc:4e:0b:a6:39:7a:24:36:
        89:f8:3b:1d:a2:23:4d:b8:90:8b:a8:11:8a:d0:98:ca:b0:f4:
        c1:db
```

Distributing Your Root CA Public Certificate

The easiest way to distribute your key is to put up a web page and link to your public certificate. You will need to serve it using the right mime type. Under apache you can do this using:

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

We can now copy our public key to our web site as something like `Example-Inc-Root-CA.crt`, and direct users to http://www.example.com/Example-Inc-Root-CA.crt to download it.

Some clients require different formats to import, so we can make life easier for our users by offering our public key in diffferent formats. Some clients require the key in DER format.

```
root$ openssl x509 -in public/root.pem -outform DER -out
public/root.der
```

Some software, like cfengine, require the filename of the certificate to be a hash of the certificate. You can generate this filename using:

```
root$ cp public/root.pem public/$(openssl x509 -noout -hash -in
public/root.pem).0
```

Signing Certificate Signing Requests

we now need to add support for signing certificate signing requests (CSR). A someone will generate a CSR and give it to you to sign. You then need to check they should have the certificate they've asked for. If the information in the CSR is valid, then you sign it, generating a new public certificate for the CSR, which you return to the person making the request.

Add the following to your `conf/openssl.cnf`:

```
[ ca ]
default_ca              = CA_default

[ CA_default ]
dir                     = .
new_certs_dir           = ./signed-keys/
database                = ./conf/index
certificate             = ./public/root.pem
serial                  = ./conf/serial
private_key             = ./private/root.pem
x509_extensions         = usr_cert
name_opt                = ca_default
cert_opt                = ca_default
default_crl_days        = 30
default_days            = 365
default_md              = sha1
preserve                = no
policy                  = policy_match

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

[ usr_cert ]
basicConstraints=CA:FALSE
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always
nsCaRevocationUrl     = https://www.example.com/example-ca-crl.pem
```

The only options you might want to change are `nsCaRevocationUrl`, which clients can use to check if the certificate has been revoked and `default_days` which defines the length of the certificate before it expires.

We also need to set up a couple more files and directories before we can sign CSRs. We need a directory to store copies of our signed keys, so we can easily revoke them. We also need a file to store a serial number in and a file to keep an index of our signed keys.

```
root$ mkdir signed-keys
root$ echo "01" > conf/serial
root$ touch conf/index
```

Now we have the infrastructure available, we can start signing requests. The first thing to do is to view the request and make sure we are happy to sign it. The important bit is the `CN=`section as that is the bit that the client software will check.

```
root$ openssl req -in request.csr -noout -text
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=UK, ST=Sussex, L=Brighton, O=Example Inc,
CN=www.example.com/emailAddress=david@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (512 bit)
                Modulus (512 bit):
                    00:d9:58:b4:ca:b5:0e:8b:86:f7:8c:16:7f:c6:a4:
                    74:90:cb:66:09:b6:a7:4d:e5:a1:d4:2e:cb:98:dc:
                    10:72:a0:9c:42:78:24:17:82:2c:0b:ff:d6:ea:67:
                    76:c7:60:01:ea:c7:cd:31:12:24:b4:c5:9d:02:09:
                    0a:d9:2b:f2:bd
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha1WithRSAEncryption
        a6:e6:00:9c:f7:9e:20:08:b7:5c:4d:d4:32:e4:cb:0c:69:d2:
        ad:19:f9:de:7c:9f:e1:76:05:a9:59:3e:05:6d:8b:3c:69:a2:
        e3:8e:fe:e6:8b:a1:3f:a9:36:6a:80:da:c1:bb:5d:71:b3:63:
        df:d4:17:6c:a3:9d:2a:62:3f:ff
```

We need to be certain that the person requesting this certificate controls www.example.com. When we are happy with the details in the request, we can sign it using:

```
root$ openssl ca -batch -config root.cnf -in request.csr -out
request.cert
Using configuration from conf/openssl.cnf
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Apr  8 18:29:33 2006 GMT
            Not After : Apr  8 18:29:33 2007 GMT
        Subject:
            countryName               = UK
            stateOrProvinceName       = Sussex
            organizationName          = Example Inc
            commonName                = www.example.com
            emailAddress              = david@example.com
```

```
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:

4B:26:25:75:EE:74:63:2D:B5:07:E7:92:9F:B1:85:F6:B7:7A:78:24
            X509v3 Authority Key Identifier:

keyid:D5:FE:57:B9:E3:C2:6F:95:7C:29:8D:77:12:D7:94:09:43:D6:2C:52
                DirName:/C=UK/ST=Sussex/L=Brighton/O=Example
Inc/CN=Example Inc Root CA/emailAddress=david@example.com
                serial:98:D7:B8:5A:5A:91:42:0F

            Netscape CA Revocation Url:
                https://www.example.com/example-ca-crl.pem
Certificate is to be certified until Apr  8 18:29:33 2007 GMT (365
days)

Write out database with 1 new entries
Data Base Updated
```

You should find that you now have a `request.cert`, which you can return to the requestor. You should also have a `signed-keys/01.pem`, which you can use if you ever need to revoke the signature. You should also find that `conf/serial` has been incremented, and that`conf/index` has been updated to include the new key's details.

Revoking keys

The final step is the ability to revoke certificates that you have signed. You would probably want to revoke certificates if you discover that a certificate was incorrectly signed or someone has failed to keep their private key secure and needs to replace it. The process for revoking keys is to revoke the key itself and then generate and publish a new certificate revokation list (CRL).

The first thing to do is to add a couple of lines to the config file:

```
[ crl_ext ]
authorityKeyIdentifier=keyid:always,issuer:always
```

First we need to revoke the key using:

```
root$ openssl ca -config conf/openssl.cnf -revoke request.cert
Using configuration from conf/openssl.cnf
Revoking Certificate 01.
Data Base Updated
```

You should find that the only change has been to the `conf/index` file. Now we need to generate our CRL so users can tell it's been revoked.

```
root$ openssl ca -config conf/openssl.cnf -gencrl -out example-ca-crl.pem
Using configuration from conf/openssl.cnf
```

Again we can decode this file using **openssl crl**:

```
root$ openssl crl -in example-ca-crl.pem -noout -text
Certificate Revocation List (CRL):
        Version 1 (0x0)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: /C=UK/ST=Sussex/L=Brighton/O=Example Inc/CN=Example
Inc Root CA/emailAddress=david@example.com
        Last Update: Apr  9 08:51:40 2006 GMT
        Next Update: May  9 08:51:40 2006 GMT
Revoked Certificates:
    Serial Number: 01
        Revocation Date: Apr  9 08:46:05 2006 GMT
    Signature Algorithm: sha1WithRSAEncryption
        a0:94:8e:ed:b6:87:af:59:cc:cb:4d:f7:99:b7:ba:d8:14:d0:
        53:96:4b:a7:2b:e3:2b:b2:cd:5b:2a:57:46:21:e4:c9:5c:71:
        b9:b5:03:a2:2b:a5:5f:42:2b:1d:ee:dc:10:63:8a:40:17:e7:
        0a:ea:5d:e6:56:d6:38:cc:68:7b:f3:14:ec:1d:93:9b:13:6a:
        fc:89:96:ed:39:c7:e4:32:6d:9d:b7:92:3d:2f:95:20:e6:06:
        e8:13:f4:cc:77:36:9a:83:1b:2c:f4:02:3e:1c:37:59:08:f2:
        ff:46:eb:bc:3a:fb:ca:32:0c:57:e7:26:32:77:d6:f5:d0:2f:
        4b:d0
```

This file now needs to be uploaded to `https://www.example.com/example-ca-crl.pem` as this is the location we specified when we signed the certificates. Clients will look at the information in the certificate and attempt to check it against that url to see if it has been revoked. If you don't publish it at the location in the certificates the client will never know it's been revoked.

Importing the Trusted Root Certificate

Firefox

There are two methods for installing a certificate in Firefox; from a url or from a file. To load from a url, just browse to the url and, providing the server serves the file with the right MIME type, you will be prompted for which uses you trust the certificate. You probably want to tick "Trust this CA to identify websites". Finally click okay.

Firefox prompt

If you have the root certificate on disk, you can install it by going to Tools -> Options. Select the Advanced section and the Security tab.


Click on the “View Certificates” button and then the “Authorities” tab.


Click on “Import” and then browse to the certificate file.


Once you click open, you’ll be prompted for which uses you trust the certificate. You probably want to tick “Trust this CA to identify websites”. Finally click okay.


You can check that it’s installed correctly by scrolling down the list until you find your root certificate.

Internet Explorer

You can import the certificate into Internet Explorer by browsing the the certificate. You will be prompted if you want to open or save the file. Select Open.


You’ll be shown information about the certificate. Select Install Certificate.


Choose Place all certificates in the following store and select Browse.


Select Third-Party Root Certification Authorities and press OK.


Select Next


Select Finish and you’ll be told the certificate imported sucessfully. You can close all the dialogs.


You can check the certificate installed correctly by opening Internet Options and choosing the Certificate button from the Content tab.


Select the Trusted Root Certification Authories tab and scroll down to your certificate.


If you select View, you can view details about your certificate.

Debian

You can install your certificate as part of the ca-certificate infrastructure. You need to copy your .pem file to `/usr/share/ca-certificates/` directory, add the filename to `/etc/ca-certificates.conf` file and then run **update-ca-certificates** command.

```
root# cp public/root.pem /usr/share/ca-certificates/Example-Inc.crt
root# echo Example-Inc.crt >> /etc/ca-certificates.conf
root# update-ca-certificates
```

Once you've done this, you can point programs at `/etc/ssl/certs/ca-certificates.crt`.

Mutt

You can add the following line to your `~/.muttrc` file.

```
set certificate_file=/etc/ssl/certs/ca-certificates.crt
```