stackoverflow.com

How do you sign a Certificate Signing Request with your Certification Authority?

Bernard Rosset

15-19 minutes

```
1. Using the x509 module
openssl x509 ...
...

2 Using the ca module
openssl ca ...
...
```

You are missing the prelude to those commands.

This is a two-step process. First you set up your CA, and then you sign an end entity certificate (a.k.a server or user). Both of the two commands elide the two steps into one. And both assume you have a an OpenSSL configuration file already setup for both CAs and Server (end entity) certificates.

First, create a basic configuration file:

```
$ touch openssl-ca.cnf
```

Then, add the following to it:

```
HOME            = .
RANDFILE        = $ENV::HOME/.rnd

####################################################################
[ ca ]
default_ca    = CA_default      # The default ca section

[ CA_default ]
```

```
default_days     = 1000         # How long to certify for
default_crl_days = 30           # How long before next CRL
default_md       = sha256       # Use public key default MD
preserve         = no           # Keep passed DN ordering

x509_extensions = ca_extensions # The extensions to add to the
cert

email_in_dn     = no            # Don't concat the email in
the DN
copy_extensions = copy          # Required to copy SANs from
CSR to cert

####################################################################
[ req ]
default_bits       = 4096
default_keyfile    = cakey.pem
distinguished_name = ca_distinguished_name
x509_extensions    = ca_extensions
string_mask        = utf8only

####################################################################
[ ca_distinguished_name ]
countryName         = Country Name (2 letter code)
countryName_default = US

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

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

organizationName            = Organization Name (eg, company)
organizationName_default    = Test CA, Limited
```

```
organizationalUnitName         = Organizational Unit (eg,
division)
organizationalUnitName_default = Server Research Department

commonName         = Common Name (e.g. server FQDN or YOUR
name)
commonName_default = Test CA

emailAddress         = Email Address
emailAddress_default = test@example.com

#####################################################################
[ ca_extensions ]

subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always, issuer
basicConstraints       = critical, CA:true
keyUsage               = keyCertSign, cRLSign
```

The fields above are taken from a more complex `openssl.cnf` (you can find it in `/usr/lib/openssl.cnf`), but I think they are the essentials for creating the CA certificate and private key.

Tweak the fields above to suit your taste. The defaults save you the time from entering the same information while experimenting with configuration file and command options.

I omitted the CRL-relevant stuff, but your CA operations should have them. See `openssl.cnf` and the related `crl_ext` section.

Then, execute the following. The `-nodes` omits the password or passphrase so you can examine the certificate. It's a *really* **bad** idea to omit the password or passphrase.

```
$ openssl req -x509 -config openssl-ca.cnf -newkey rsa:4096
-sha256 -nodes -out cacert.pem -outform PEM
```

After the command executes, `cacert.pem` will be your certificate for CA operations, and `cakey.pem` will be the private key. Recall the private key *does not* have a password or passphrase.

You can dump the certificate with the following.

```
$ openssl x509 -in cacert.pem -text -noout
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 11485830970703032316
(0x9f65de69ceef2ffc)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=MD, L=Baltimore, CN=Test
CA/emailAddress=test@example.com
        Validity
            Not Before: Jan 24 14:24:11 2014 GMT
            Not After : Feb 23 14:24:11 2014 GMT
        Subject: C=US, ST=MD, L=Baltimore, CN=Test
CA/emailAddress=test@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:

00:b1:7f:29:be:78:02:b8:56:54:2d:2c:ec:ff:6d:
                    ...

39:f9:1e:52:cb:8e:bf:8b:9e:a6:93:e1:22:09:8b:
                    59:05:9f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:

4A:9A:F3:10:9E:D7:CF:54:79:DE:46:75:7A:B0:D0:C1:0F:CF:C1:8A
            X509v3 Authority Key Identifier:

keyid:4A:9A:F3:10:9E:D7:CF:54:79:DE:46:75:7A:B0:D0:C1:0F:CF:C1:8A

            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage:
                Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
```

```
4a:6f:1f:ac:fd:fb:1e:a4:6d:08:eb:f5:af:f6:1e:48:a5:c7:
         ...

cd:c6:ac:30:f9:15:83:41:c1:d1:20:fa:85:e7:4f:35:8f:b5:
         38:ff:fd:55:68:2c:3e:37
```

And test its purpose with the following (don't worry about the `Any Purpose: Yes`; see "critical,CA:FALSE" but "Any Purpose CA : Yes").

```
$ openssl x509 -purpose -in cacert.pem -inform PEM
Certificate purposes:
SSL client : No
SSL client CA : Yes
SSL server : No
SSL server CA : Yes
Netscape SSL server : No
Netscape SSL server CA : Yes
S/MIME signing : No
S/MIME signing CA : Yes
S/MIME encryption : No
S/MIME encryption CA : Yes
CRL signing : Yes
CRL signing CA : Yes
Any Purpose : Yes
Any Purpose CA : Yes
OCSP helper : Yes
OCSP helper CA : Yes
Time Stamp signing : No
Time Stamp signing CA : Yes
-----BEGIN CERTIFICATE-----
MIIFpTCCA42gAwIBAgIJAJ9l3mnO7y/8MA0GCSqGSIb3DQEBCwUAMGExCzAJBgNV
...
aQUtFrV4hpmJUaQZ7ySr
/RjCb4KYkQpTkOtKJOU1Ic3GrDD5FYNBwdEg+oXnTzWP
tTj//VVoLD43
-----END CERTIFICATE-----
```

For part two, I'm going to create another configuration file that's easily digestible. First, `touch` the `openssl-server.cnf` (you can make one of these for user certificates also).

```
$ touch openssl-server.cnf
```

Then open it, and add the following.

```
HOME            = .
RANDFILE        = $ENV::HOME/.rnd

####################################################################
[ req ]
default_bits       = 2048
default_keyfile    = serverkey.pem
distinguished_name = server_distinguished_name
req_extensions     = server_req_extensions
string_mask        = utf8only

####################################################################
[ server_distinguished_name ]
countryName         = Country Name (2 letter code)
countryName_default = US

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

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

organizationName            = Organization Name (eg, company)
organizationName_default    = Test Server, Limited

commonName           = Common Name (e.g. server FQDN or YOUR
name)
commonName_default   = Test Server

emailAddress         = Email Address
emailAddress_default = test@example.com
```

```
####################################################################
[ server_req_extensions ]

subjectKeyIdentifier = hash
basicConstraints     = CA:FALSE
keyUsage             = digitalSignature, keyEncipherment
subjectAltName       = @alternate_names
nsComment            = "OpenSSL Generated Certificate"

####################################################################
[ alternate_names ]

DNS.1  = example.com
DNS.2  = www.example.com
DNS.3  = mail.example.com
DNS.4  = ftp.example.com
```

If you are developing and need to use your workstation as a server, then you may need to do the following for Chrome. Otherwise Chrome may complain a *Common Name* is invalid (ERR_CERT_COMMON_NAME_INVALID). I'm not sure what the relationship is between an IP address in the SAN and a CN in this instance.

```
# IPv4 localhost
IP.1     = 127.0.0.1

# IPv6 localhost
IP.2     = ::1
```

Then, create the server certificate request. Be sure to *omit* `-x509*`. Adding `-x509` will create a certificate, and not a request.

```
$ openssl req -config openssl-server.cnf -newkey rsa:2048
-sha256 -nodes -out servercert.csr -outform PEM
```

After this command executes, you will have a request in `servercert.csr` and a private key in `serverkey.pem`.

And you can inspect it again.

```
$ openssl req -text -noout -verify -in servercert.csr
```

```
Certificate:
    verify OK
    Certificate Request:
        Version: 0 (0x0)
        Subject: C=US, ST=MD, L=Baltimore, CN=Test
Server/emailAddress=test@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:

00:ce:3d:58:7f:a0:59:92:aa:7c:a0:82:dc:c9:6d:
                    ...

f9:5e:0c:ba:84:eb:27:0d:d9:e7:22:5d:fe:e5:51:
                    86:e1
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
            X509v3 Subject Key Identifier:

1F:09:EF:79:9A:73:36:C1:80:52:60:2D:03:53:C7:B6:BD:63:3B:61
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Key Encipherment
            X509v3 Subject Alternative Name:
                DNS:example.com, DNS:www.example.com,
DNS:mail.example.com, DNS:ftp.example.com
            Netscape Comment:
                OpenSSL Generated Certificate
    Signature Algorithm: sha256WithRSAEncryption

6d:e8:d3:85:b3:88:d4:1a:80:9e:67:0d:37:46:db:4d:9a:81:
         ...

76:6a:22:0a:41:45:1f:e2:d6:e4:8f:a1:ca:de:e5:69:98:88:
```

```
        a9:63:d0:a7
```

Next, you have to sign it with your CA.

You are almost ready to sign the server's certificate by your CA. The CA's `openssl-ca.cnf` needs two more sections before issuing the command.

First, open `openssl-ca.cnf` and add the following two sections.

```
####################################################################
[ signing_policy ]
countryName            = optional
stateOrProvinceName    = optional
localityName           = optional
organizationName       = optional
organizationalUnitName = optional
commonName             = supplied
emailAddress           = optional

####################################################################
[ signing_req ]
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid,issuer
basicConstraints       = CA:FALSE
keyUsage               = digitalSignature, keyEncipherment
```

Second, add the following to the `[ CA_default ]` section of `openssl-ca.cnf`. I left them out earlier, because they can complicate things (they were unused at the time). Now you'll see how they are used, so hopefully they will make sense.

```
base_dir      = .
certificate   = $base_dir/cacert.pem   # The CA certifcate
private_key   = $base_dir/cakey.pem    # The CA private key
new_certs_dir = $base_dir              # Location for new
certs after signing
database      = $base_dir/index.txt    # Database index file
serial        = $base_dir/serial.txt   # The current serial
number

unique_subject = no  # Set to 'no' to allow creation of
```

```
                    # several certificates with same subject.
```

Third, touch `index.txt` and `serial.txt`:

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

Then, perform the following:

```
$ openssl ca -config openssl-ca.cnf -policy signing_policy
-extensions signing_req -out servercert.pem -infiles
servercert.csr
```

You should see similar to the following:

```
Using configuration from openssl-ca.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'US'
stateOrProvinceName   :ASN.1 12:'MD'
localityName          :ASN.1 12:'Baltimore'
commonName            :ASN.1 12:'Test CA'
emailAddress          :IA5STRING:'test@example.com'
Certificate is to be certified until Oct 20 16:12:39 2016 GMT
(1000 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
```

After the command executes, you will have a freshly minted server certificate in `servercert.pem`. The private key was created earlier and is available in `serverkey.pem`.

Finally, you can inspect your freshly minted certificate with the following:

```
$ openssl x509 -in servercert.pem -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 9 (0x9)
```

```
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=MD, L=Baltimore, CN=Test
CA/emailAddress=test@example.com
        Validity
            Not Before: Jan 24 19:07:36 2014 GMT
            Not After : Oct 20 19:07:36 2016 GMT
        Subject: C=US, ST=MD, L=Baltimore, CN=Test Server
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:

00:ce:3d:58:7f:a0:59:92:aa:7c:a0:82:dc:c9:6d:
                    ...

f9:5e:0c:ba:84:eb:27:0d:d9:e7:22:5d:fe:e5:51:
                    86:e1
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:

1F:09:EF:79:9A:73:36:C1:80:52:60:2D:03:53:C7:B6:BD:63:3B:61
            X509v3 Authority Key Identifier:

keyid:42:15:F2:CA:9C:B1:BB:F5:4C:2C:66:27:DA:6D:2E:5F:BA:0F:C5:9E

            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Key Encipherment
            X509v3 Subject Alternative Name:
                DNS:example.com, DNS:www.example.com,
DNS:mail.example.com, DNS:ftp.example.com
            Netscape Comment:
                OpenSSL Generated Certificate
    Signature Algorithm: sha256WithRSAEncryption
```

```
b1:40:f6:34:f4:38:c8:57:d4:b6:08:f7:e2:71:12:6b:0e:4a:
        ...

45:71:06:a9:86:b6:0f:6d:8d:e1:c5:97:8d:fd:59:43:e9:3c:
        56:a5:eb:c8:7e:9f:6b:7a
```

Earlier, you added the following to `CA_default`: `copy_extensions = copy`. This copies extension provided by the person making the request.

If you omit `copy_extensions = copy`, then your server certificate will lack the Subject Alternate Names (SANs) like `www.example.com` and `mail.example.com`.

If you use `copy_extensions = copy`, but don't look over the request, then the requester might be able to trick you into signing something like a subordinate root (rather than a server or user certificate). Which means he/she will be able to mint certificates that chain back to your trusted root. Be sure to verify the request with `openssl req -verify` before signing.

If you *omit* `unique_subject` or set it to `yes`, then you will only be allowed to create **one** certificate under the subject's distinguished name.

```
unique_subject = yes            # Set to 'no' to allow
creation of
                                # several ctificates with same
subject.
```

Trying to create a second certificate while experimenting will result in the following when signing your server's certificate with the CA's private key:

```
Sign the certificate? [y/n]:Y
failed to update database
TXT_DB error number 2
```

So `unique_subject = no` is perfect for testing.

If you want to ensure the *Organizational Name* is consistent between self-signed CAs, *Subordinate CA* and *End-Entity* certificates, then add the following to your CA configuration files:

```
[ policy_match ]
organizationName = match
```

If you want to allow the *Organizational Name* to change, then use:

```
[ policy_match ]
organizationName = supplied
```

There are other rules concerning the handling of DNS names in X.509/PKIX certificates. Refer to these documents for the rules:

- RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
- RFC 6125, Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)
- RFC 6797, Appendix A, HTTP Strict Transport Security (HSTS)
- RFC 7469, Public Key Pinning Extension for HTTP
- CA/Browser Forum Baseline Requirements
- CA/Browser Forum Extended Validation Guidelines

RFC 6797 and RFC 7469 are listed, because they are more restrictive than the other RFCs and CA/B documents. RFC's 6797 and 7469 *do not* allow an IP address, either.