The Most Common OpenSSL Commands

General OpenSSL Commands

These commands allow you to generate CSRs, Certificates, Private Keys and do other miscellaneous tasks.

Generate a new private key and Certificate Signing Request

```
openssl req -out CSR.csr -new -newkey rsa:2048 -nodes -keyout privateKey.key
```

Generate a self-signed certificate (see How to Create and Install an Apache Self Signed Certificate for more info)

```
openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout privateKey.key -out certificate.crt
```

Generate a certificate signing request (CSR) for an existing private key

```
openssl req -out CSR.csr -key privateKey.key -new
```

Generate a certificate signing request based on an existing certificate

```
openssl x509 -x509toreq -in certificate.crt -out CSR.csr -signkey privateKey.key
```

Remove a passphrase from a private key

```
openssl rsa -in privateKey.pem -out newPrivateKey.pem
```

Checking Using OpenSSL

If you need to check the information within a Certificate, CSR or Private Key, use these commands. You can also check CSRs and check certificates using our online tools.

Check a Certificate Signing Request (CSR)

```
openssl req -text -noout -verify -in CSR.csr
```

Check a private key

```
openssl rsa -in privateKey.key -check
```

Check a certificate

```
openssl x509 -in certificate.crt -text -noout
```

Check a PKCS#12 file (.pfx or .p12)

```
openssl pkcs12 -info -in keyStore.p12
```

Debugging Using OpenSSL

If you are receiving an error that the private doesn't match the certificate or that a certificate that you installed to a site is not trusted, try one of these commands. If you are trying to verify that an SSL certificate is installed correctly, be sure to check out the SSL Checker.

Check an MD5 hash of the public key to ensure that it matches with what is in a CSR or private key

```
openssl x509 -noout -modulus -in certificate.crt | openssl md5
openssl rsa -noout -modulus -in privateKey.key | openssl md5
openssl req -noout -modulus -in CSR.csr | openssl md5
```

Check an SSL connection. All the certificates (including Intermediates) should be displayed

```
openssl s_client -connect www.paypal.com:443
```

Converting Using OpenSSL

These commands allow you to convert certificates and keys to different formats to make them compatible with specific types of servers or software. For example, you can convert a normal PEM file that would work with Apache to a PFX (PKCS#12) file and use it with Tomcat or IIS. Use our SSL Converter to convert certificates without messing with OpenSSL.

Convert a DER file (.crt .cer .der) to PEM

```
openssl x509 -inform der -in certificate.cer -out certificate.pem
```

Convert a PEM file to DER

```
openssl x509 -outform der -in certificate.pem -out certificate.der
```

Convert a PKCS#12 file (.pfx .p12) containing a private key and certificates to PEM

```
openssl pkcs12 -in keyStore.pfx -out keyStore.pem -nodes
```

You can add -nocerts to only output the private key or add -nokeys to only output the certificates.

Convert a PEM certificate file and a private key to PKCS#12 (.pfx .p12)

```
openssl pkcs12 -export -out certificate.pfx -inkey privateKey.key -in certificate.crt -
certfile CACert.crt
```

Source

View CRL

```
openssl crl -inform DER -text -noout -in crl.crl
```

```
openssl crl -inform PEM -text -noout -in crl.crl
```

Generate a request for Windows CA Services

The process below shows an "easy" way to add the certificateTemplateName to the certificate request that will be submitted to Windows CA Services.

1. Create an openssl configuration file for your request based on the template shown below.
    1. Make sure you update all the subject alternative names entries.
2. Use the script below to create the certificate request.

```
# FILE: server.mydomain.com.openssl.cnf
#
# run the following to generate the request:
# openssl req -newkey rsa:2048 -keyout server.mydomain.com -out server.mydomain.com.req -nodes
-config server.mydomain.com.openssl.cnf

oid_section = OIDs

[OIDs]
certificateTemplateName = 1.3.6.1.4.1.311.20.2

[req]
default_bits = 2048
distinguished_name = req_distinguished_name
req_extensions = req_ext

[req_distinguished_name]
countryName = Country Name (2 letter code)
countryName_default = US
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = My State
localityName = Locality Name (eg, city)
localityName_default = My City
organizationalUnitName    = Organizational Unit Name (eg, section)
organizationalUnitName_default    = My Org
commonName = Common Name
commonName_default = server.mydomain.com
commonName_max    = 64

[req_ext]
subjectAltName = @alt_names

# Replace PKI-WebServer with your certificate template name, NOT the display name
certificateTemplateName = ASN1:PRINTABLESTRING:PKI-WebServer

[alt_names]
DNS.1 = server.mydomain.com
DNS.2 = server1.mydomain.com
DNS.3 = server2.mydomain.com
IP.1 = 10.1.2.3
```

```
#!/bin/bash

# change this to match your certificate primary common name
CERTBASE="server.mydomain.com"

# generate timestamp
printf -v TIMESTAMP '%(%Y%m%d-%H%M%S)T' -1

# generate new private key and csr
openssl req -newkey rsa:2048 -keyout "$CERTBASE.$TIMESTAMP.key" -out
"$CERTBASE.$TIMESTAMP.req" -nodes -config "$CERTBASE.openssl.cnf"
```

Revision #6
Created 15 December 2020 06:42:17 by bluecrow76
Updated 17 April 2025 10:08:43 by bluecrow76