OpenSSL Quick Reference Guide

Solution ID : INFO141 **Last Modified : 06/28/2024**

Learn how to use the most common OpenSSL commands

OpenSSL is an open-source command line tool that is commonly used to generate private keys, create CSRs, install your SSL/TLS certificate, and identify certificate information. We designed this quick reference guide to help you understand the most common OpenSSL commands and how to use them.

This guide is not meant to be comprehensive. If you're looking for a more in-depth and comprehensive look at OpenSSL, we recommend you check out the *OpenSSL Cookbook* by Ivan Ristić.

Note: Ubuntu 16.04.3 LTS was the system used to write this guide. Some command examples use a '\' (backslash) to create a line break to make them easier to understand.

If you don't have the time to get into the nitty-gritty of OpenSSL commands and CSR generation, or you want to save some time, check out our OpenSSL CSR Wizard.

CERTIFICATE.

Checking Your OpenSSL Version

Identifying which version of OpenSSL you are using is an important first step when preparing to generate a private key or CSR. Your version of OpenSSL dictates which cryptographic algorithms can be used when generating keys as well as which protocols are supported. For example, OpenSSL version 1.0.1 was the first version to support TLS 1.1 and TLS 1.2. Knowing which version of OpenSSL you are using is also important when getting help troubleshooting problems you may run into.

Use the following command to identify which version of OpenSSL you are running:

```
openssl version -a
```

In this command, the -a switch displays complete version information, including:

- The version number and version release date (OpenSSL 1.0.2g 1 Mar 2016).
- The options that were built with the library (options)

(options).

- The directory where certificates and private keys are stored (OPENSSLDIR).

Using the openssl version -a command, the following output was generated:

```
OpenSSL 1.0.2g 1 Mar 2016
built on: reproducible build, date unspecified
platform: debian-amd64
options: bn(64,64) rc4(16x,int) des(idx,cisc,16,int) blowfish(idx)
compiler: cc -I. -I.. -I../include -fPIC -DOPENSSL_PIC -
DOPENSSL_THREADS -
D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -m64 -DL_ENDIAN -
g -O2 -fstack-protector-
strong -Wformat -Werror=format-security -Wdate-time -
D_FORTIFY_SOURCE=2 -Wl,-
Bsymbolic-functions -Wl,-z,relro -Wa,--noexecstack -Wall -
DMD32_REG_T=int -
DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -
DOPENSSL_BN_ASM_MONT5 -
DOPENSSL_BN_ASM_GF2m -DSHA1_ASM -DSHA256_ASM -
DSHA512_ASM -DMD5_ASM -DAES_ASM -
DVPAES_ASM -DBSAES_ASM -DWHIRLPOOL_ASM -DGHASH_ASM -
DECP_NISTZ256_ASM
OPENSSLDIR: "/usr/lib/ssl"
```

OpenSSL and CSR Creation

The first step to obtaining an SSL certificate is using OpenSSL to create a certificate signing request (CSR)

OpenSSL to create a certificate signing request (CSR) that can be sent to a Certificate Authority (CA) (e.g., DigiCert). The CSR contains the common name(s) you want your certificate to secure, information about your company, and your public key. In order for a CSR to be created, it needs to have a private key from which the public key is extracted. This can be done by using an existing private key or generating a new private key.

Security Note: Because of the security issues associated with using an existing private key, and because it's very easy and entirely free to create a private key, we recommend you generate a brand new private key whenever you create a CSR.

Deciding on Key Generation Options

When generating a key, you have to decide three things: the key algorithm, the key size, and whether to use a passphrase.

Key Algorithm

For the key algorithm, you need to take into account its compatibility. For this reason, we recommend you use RSA. However, if you have a specific need to use another algorithm (such as ECDSA), you can use that too, but be aware of the compatibility issues you might run into.

Note: This guide only covers generating keys using the RSA

Note: This guide only covers generating keys using the RSA algorithm.

Key Size

For the key size, you need to select a bit length of at least 2048 when using RSA and 256 when using ECDSA; these are the smallest key sizes allowed for SSL certificates. Unless you need to use a larger key size, we recommend sticking with 2048 with RSA and 256 with ECDSA.

Note: In older versions of OpenSSL, if no key size is specified, the default key size of 512 is used. Any key size lower than 2048 is considered unsecure and should never be used.

Passphrase

For the passphrase, you need to decide whether you want to use one. If used, the private key will be encrypted using the specified encryption method, and it will be impossible to use without the passphrase. Because there are pros and cons with both options, it's important you understand the implications of using or not using a passphrase. In this guide, we will not be using a passphrase in our examples.

Generating Your Private Key

After deciding on a key algorithm, key size, and whether to use a passphrase, you are ready to

Whether to use a passphrase, you are ready to generate your private key.

Use the following command to generate your private key using the RSA algorithm:

```
openssl genrsa -out yourdomain.key 2048
```

This command generates a private key in your current directory named *yourdomain.key* (-out yourdomain.key) using the RSA algorithm (genrsa) with a key length of 2048 bits (2048). The generated key is created using the OpenSSL format called PEM.

Use the following command to view the raw, encoded contents (PEM format) of the private key:

```
cat yourdomain.key
```

Even though the contents of the file might look like a random chunk of text, it actually contains important information about the key.

Use the following command to decode the private key and view its contents:

```
openssl rsa -text -in yourdomain.key -noout
```

The -noout switch omits the output of the encoded

version of the private key.

Extracting Your Public Key

The private key file contains both the private key and the public key. You can extract your public key from your private key file if needed.

Use the following command to extract your public key:

```
openssl rsa -in yourdomain.key -pubout -out yourdomain_public.key
```

Creating Your CSR

After generating your private key, you are ready to create your CSR. The CSR is created using the PEM format and contains the public key portion of the private key as well as information about you (or your company).

Use the following command to create a CSR using your newly generated private key:

```
```

```
openssl req -new -key yourdomain.key -out yourdomain.csr
```

After entering the command, you will be asked series of questions. Your answers to these questions will be embedded in the CSR. Answer the questions as described below:

Country Name (2 letter code)	The two-letter country code where your company is legally located.
State or Province Name (full name)	The state/province where your company is legally located.
Locality Name (e.g., city)	The city where your company is legally located.
Organization Name (e.g., city)	The city where your company is legally located.
Organizational Unit Name (e.g., section)	The name of your department within the organization. (You can leave this option blank; simply press Enter.)
Common Name (e.g., server FQDN)	The fully-qualified domain name (FQDN) (e.g., www.example.com).
Email Address	Your email address. (You can leave this option blank; simply press Enter.)

A challenge password	Your email address. (You can leave this option blank; simply press Enter.)
An optional company name	Leave this option blank (simply press Enter).

Some of the above CSR questions have default values that will be used if you leave the answer blank and press Enter. These default values are pulled from the OpenSSL configuration file located in the OPENSSLDIR (see Checking Your OpenSSL Version). If you want to leave a question blank without using the default value, type a "." (period) and press Enter.

Using the -subj Switch

Another option when creating a CSR is to provide all the necessary information within the command itself by using the -subj switch.

Use the following command to disable question prompts when generating a CSR:

```
openssl req -new -key yourdomain.key -out yourdomain.csr \
```

```
-subj "/C=US/ST=Utah/L=Lehi/O=Your Company, Inc./OU=IT/CN=yourdom
```

This command uses your private key file (-key yourdomain.key) to create a new CSR (-out yourdomain.csr) and disables question prompts by providing the CSR information (-subj).

Creating Your CSR with One Command

Instead of generating a private key and then creating a CSR in two separate steps, you can actually perform both tasks at once.

Use the following command to create both the private key and CSR:

```
openssl req -new \
-newkey rsa:2048 -nodes -keyout yourdomain.key \
-out yourdomain.csr \
-subj "/C=US/ST=Utah/L=Lehi/O=Your Company, Inc./OU=IT/CN=yourdom
```

This command generates a new private key (-newkey) using the RSA algorithm with a 2048-bit key length (rsa:2048) without using a passphrase (-nodes) and then creates the key file with a name of yourdomain.key (-keyout yourdomain.key).

The command then generates the CSR with a filename of yourdomain.csr (-out yourdomain.csr) and the information for the CSR is supplied (-subj).

Note: While it is possible to add a subject alternative name (SAN) to a CSR using OpenSSL, the process is a bit complicated and involved. If you do need to add a SAN to your certificate, this can easily be done by adding them to the order form when purchasing your DigiCert certificate.

Verifying CSR Information

After creating your CSR using your private key, we recommend verifying that the information contained in the CSR is correct and that the file hasn't been modified or corrupted.

Use the following command to view the information in your CSR before submitting it to a CA (e.g., DigiCert):

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

The -noout switch omits the output of the encoded version of the CSR. The -verify switch checks the signature of the file to make sure it hasn't been modified.

Running this command provides you with the following output:

```
verify OK
```

```
verify OK
Certificate Request:
Data:
Version: 0 (0x0)
Subject: C=US, ST=Utah, L=Lehi, O=Your Company, Inc., OU=IT, CN=yourdc
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
Public-Key: (2048 bit)
Modulus:
00:bb:31:71:40:81:2c:8e:fb:89:25:7c:0e:cb:76:
[...17 lines removed]
Exponent: 65537 (0x10001)
Attributes:
a0:00
Signature Algorithm: sha256WithRSAEncryption
0b:9b:23:b5:1f:8d:c9:cd:59:bf:b7:e5:11:ab:f0:e8:b9:f6:
[...14 lines removed]
```

On the first line of the above output, you can see that the CSR was verified (verify OK). On the fourth line, the Subject: field contains the information you provided when you created the CSR. Make sure this information is correct.

If any of the information is wrong, you will need to create an entirely new CSR to fix the errors. This is because CSR files are digitally signed, meaning if even a single character is changed in the file it will be rejected by the CA.

Sending the CSR to the CA

When you are ready to send the CSR to the CA (e.g., DigiCert), you need to do so using the PEM format—

the raw, encoded text of the CSR that you see when opening it in a text editor.

Use the following command to view the raw output of the CSR:

```
cat yourdomain.csr
```

You must copy the entire contents of the output (including the -----BEGIN CERTIFICATE REQUEST----- and -----END CERTIFICATE REQUEST----- lines) and paste it into your DigiCert order form.

Viewing Certificate Information

After receiving your certificate from the CA (e.g., DigiCert), we recommend making sure the information in the certificate is correct and matches your private key. You do this by using the x509 command

the x509 command.

Use the following command to view the contents of your certificate:

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

Verifying Your Keys Match

To verify the public and private keys match, extract the public key from each file and generate a hash output for it. All three files should share the same public key and the same hash value.

Use the following commands to generate a hash of each file's public key:

```
openssl pkey -pubout -in .\private.key | openssl sha256
```

```
openssl req -pubkey -in .\request.csr -noout | openssl sha256
```

```
openssl x509 -pubkey -in .\certificate.crt -noout | openssl sha256
```

Note: The above commands should be entered one by one to generate three separate outputs.

Each command will output (stdin)= followed by a string of characters. If the output of each command matches, then the keys for each file are the same. However, if there is any mismatch, then the keys are not the same and the certificate cannot be installed.

Key mismatch errors are typically caused by installing a certificate on a machine different from the one used to generate the CSR.

If you run into a key mismatch error, you need to do one of the following:

- Transfer the private key from the machine used to generate the CSR to the one you are trying to install the certificate on.
- Install the certificate on the machine with the private key.
- Generate an entirely new key and create a new CSR on the machine that will use the certificate.

Converting Certificate Formats

By default, OpenSSL generates keys and CSRs using

the PEM format. However, there might be occasions when you need to convert your key or certificate into a different format to export it to another system.

PEM to PKCS#12

The PKCS#12 format is an archival file that stores both the certificate and the private key. This format is useful for migrating certificates and keys from one system to another as it contains all the necessary files. PKCS#12 files use either the *.pfx* or *.p12* file extension.

Use the following command to convert your PEM key and certificate into the PKCS#12 format (i.e., a single .pfx file):

```
openssl pkcs12 -export -name "yourdomain-digicert-(expiration date)" \
-out yourdomain.pfx -inkey yourdomain.key -in yourdomain.crt
```

Note: After you enter the command, you will be asked to provide a password to encrypt the file. Because the PKCS#12 format is often used for system migration, we recommend encrypting the file using a very strong password.

This command combines your private key (-inkey yourdomain.key) and your certificate (-in yourdomain.crt) into a single *.pfx* file (-out yourdomain.pfx) with a friendly name (-name "yourdomain-digicert-(expiration date)"), where

the *expiration date* is the date that the certificate expires.

PKCS#12 to PEM

Because the PKCS#12 format contains both the certificate and private key, you need to use two separate commands to convert a .pfx file back into the PEM format.

Use the following command to extract the private key from a PKCS#12 (.pfx) file and convert it into a PEM encoded private key:

```
openssl pkcs12 -in yourdomain.pfx -nocerts -out yourdomain.key -nodes
```

Use the following command to extract the certificate from a PKCS#12 (.pfx) file and convert it into a PEM encoded certificate:

```
openssl pkcs12 -in yourdomain.pfx -nokeys -clcerts -out yourdomain.crt
```

Note: You will need to provide the password used to encrypt the .pfx file in order to convert the key and certificate into the PEM format.

PEM to DER

The DER format uses ASN.1 encoding to store certificate or key information. Similar to the PEM format, DER stores key and certificate information in two separate files and typically uses the same file extensions (i.e., *.key*, *.crt*, and *.csr*). The file extension *.der* was used in the below examples for clarity.

Use the following command to convert a PEM encoded certificate into a DER encoded certificate:

```
openssl x509 -inform PEM -in yourdomain.crt -outform DER -out yourdomain.der
```

Use the following command to convert a PEM encoded private key into a DER encoded private key:

```
openssl rsa -inform PEM -in yourdomain.key -outform DER -out yourdomain_key.der
```

DER to PEM

Use the following command to convert a DER encoded certificate into a PEM encoded certificate:

```
openssl x509 -inform DER -in yourdomain.der -outform PEM -out yourdomain.crt
```

Use the following command to convert a DER encoded private key into a PEM encoded private key:

```
openssl rsa -inform DER -in yourdomain_key.der -outform PEM -out yourdo
```

Cookie Settings