# Command Line Utilities

[Jump to navigation](https://wiki.openssl.org/index.php/Command_Line_Utilities#column-one)[Jump to search](https://wiki.openssl.org/index.php/Command_Line_Utilities#searchInput)

The **openssl** program provides a rich variety of commands, each of which often has a wealth of options and arguments. Many commands use an external configuration file for some or all of their arguments and have a -config option to specify that file. The environment variable [OPENSSL\_CONF](https://wiki.openssl.org/index.php?title=OPENSSL_CONF&action=edit&redlink=1) can be used to specify the location of the configuration file. If the environment variable is not specified, a default file is created in the default certificate storage area called **openssl.cnf**. The settings in this default configuration file depend on the flags set when the version of OpenSSL being used was built.

This article is an overview of the available tools provided by OpenSSL. For all of the details on usage and implementation, you can find the [manpages](https://www.openssl.org/docs/manmaster/man1/), which are automatically generated from the source code at the [official OpenSSL project home](https://www.openssl.org/). Likewise, the source code itself may be found on the [OpenSSL project home page](https://www.openssl.org/source/), as well as on the [OpenSSL Github](https://github.com/openssl/openssl). The main OpenSSL site also includes an [overview of the command-line utilities](https://www.openssl.org/docs/manmaster/man1/openssl.html), as well as links to all of their respective documentation.

## **Contents**

* [1Getting Started](https://wiki.openssl.org/index.php/Command_Line_Utilities#Getting_Started)
* [2Basic Tasks](https://wiki.openssl.org/index.php/Command_Line_Utilities#Basic_Tasks)
* [2.1Getting Help](https://wiki.openssl.org/index.php/Command_Line_Utilities#Getting_Help)
* [2.2Getting Library Version Information](https://wiki.openssl.org/index.php/Command_Line_Utilities#Getting_Library_Version_Information)
* [3Commands](https://wiki.openssl.org/index.php/Command_Line_Utilities#Commands)
* [3.1Standard Commands](https://wiki.openssl.org/index.php/Command_Line_Utilities#Standard_Commands)
* [3.2Generating an RSA Private Key](https://wiki.openssl.org/index.php/Command_Line_Utilities#Generating_an_RSA_Private_Key)
* [3.3Generating a Public Key](https://wiki.openssl.org/index.php/Command_Line_Utilities#Generating_a_Public_Key)
* [3.4Generating Keys Based on Elliptic Curves](https://wiki.openssl.org/index.php/Command_Line_Utilities#Generating_Keys_Based_on_Elliptic_Curves)
* [3.4.1Generating the Curve Parameters](https://wiki.openssl.org/index.php/Command_Line_Utilities#Generating_the_Curve_Parameters)
* [3.4.1.1Printing Parameters to Standard Out](https://wiki.openssl.org/index.php/Command_Line_Utilities#Printing_Parameters_to_Standard_Out)
* [3.4.1.2Printing Parameters as C Code](https://wiki.openssl.org/index.php/Command_Line_Utilities#Printing_Parameters_as_C_Code)
* [3.4.2Generating the Key](https://wiki.openssl.org/index.php/Command_Line_Utilities#Generating_the_Key)
* [3.4.3Putting it All Together](https://wiki.openssl.org/index.php/Command_Line_Utilities#Putting_it_All_Together)
* [3.5Base64 Encoding Strings](https://wiki.openssl.org/index.php/Command_Line_Utilities#Base64_Encoding_Strings)
* [3.6Generating a File Hash](https://wiki.openssl.org/index.php/Command_Line_Utilities#Generating_a_File_Hash)
* [3.7File Encryption and Decryption](https://wiki.openssl.org/index.php/Command_Line_Utilities#File_Encryption_and_Decryption)
* [4Further reading](https://wiki.openssl.org/index.php/Command_Line_Utilities#Further_reading)

# Getting Started[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=1)]

The entry point for the OpenSSL library is the **openssl** binary, usually /usr/bin/openssl on Linux. The general syntax for calling **openssl** is as follows:

$ openssl command [ command\_options ] [ command\_arguments ]

Before OpenSSL 3.0, you could call **openssl** without arguments to enter the interactive mode prompt and then enter commands directly, exiting with either a quit command or by issuing a termination signal with either Ctrl+C or Ctrl+D. The following is a sample interactive session in which the user invokes the [prime](https://wiki.openssl.org/index.php?title=Prime&action=edit&redlink=1) command twice before using the quit command to terminate the session.

OpenSSL> prime -generate -bits 24  
13467269  
OpenSSL> prime -generate -bits 24  
16651079  
OpenSSL> quit

# Basic Tasks[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=2)]

This section is a brief tutorial on performing the most basic tasks using OpenSSL. For a detailed explanation of the rationale behind the syntax and semantics of the commands shown here, see the section on [Commands](https://wiki.openssl.org/index.php/Command_Line_Utilities#Commands).

## Getting Help[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=3)]

As mentioned previously, the general syntax of a command is openssl command [ command\_options ] [ command\_arguments ]. The **help** command is no different, but it does have its idiosyncrasies. To view the top-level help menu, you can call **openssl** as follows.

$ openssl help

Since OpenSSL 3.0, there are equivalent invocations such as:

$ openssl --help  
$ openssl -h

This query will print all of the available commands, like so:

Standard commands  
asn1parse ca ciphers cmp   
cms crl crl2pkcs7 dgst   
dhparam dsa dsaparam ec   
...

Note the above output was truncated, so only the first four lines of output are shown.

The same output is obtained also with

$ openssl list -standard-commands

A help menu for each command may be requested in two different ways. First, the same command used above may be repeated, followed by the name of the command to print help for.

$ openssl help genpkey

The program will then display the valid options for the given command.

$ openssl help genpkey  
General options:  
 -help Display this summary  
 -engine val Use engine, possibly a hardware device  
 -paramfile infile Parameters file  
 -algorithm val The public key algorithm  
 -quiet Do not output status while generating keys  
 -pkeyopt val Set the public key algorithm option as opt:value  
 -config infile Load a configuration file (this may load modules)  
Output options:  
 -out outfile Output file  
 -outform PEM|DER output format (DER or PEM)  
 -pass val Output file pass phrase source  
 -genparam Generate parameters, not key  
 -text Print the in text  
 -\* Cipher to use to encrypt the key  
Provider options:  
 -provider-path val Provider load path (must be before 'provider' argument if required)  
 -provider val Provider to load (can be specified multiple times)  
 -propquery val Property query used when fetching algorithms  
Order of options may be important! See the documentation.

The second way of requesting the help menu for a particular command is by using the first option in the output shown above, namely openssl command -help. Both commands will yield the same output; the help menu displayed will be exactly the same.

For additional information on the usage of a particular command, the project [manpages](https://www.openssl.org/docs/manmaster/man1/) are the definite source of information. The manpages may be views in a shell as usual, e.g.

$ man openssl  
$ man openssl-genpkey  
$ man genpkey

Another way of accessing the manpages is via the project perldocs. [perldoc](https://perldoc.perl.org/perldoc) is a utility included with most if not all [Perl](https://www.perl.org/) distributions, and it's capable of displaying documentation information in a variety of formats, one of which is as manpages. Not surprisingly, the project documentation is generated from the **pod** files located in the doc directory of the source code.

## Getting Library Version Information[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=4)]

$ openssl version  
OpenSSL 3.0.4 21 Jun 2022 (Library: OpenSSL 3.0.4 21 Jun 2022)

As mentioned above, the version command's help menu may be queried for additional options like so:

$ openssl version -help  
Usage: version [options]  
General options:  
 -help Display this summary  
Output options:  
 -a Show all data  
 -b Show build date  
 -d Show configuration directory  
 -e Show engines directory  
 -m Show modules directory  
 -f Show compiler flags used  
 -o Show some internal datatype options  
 -p Show target build platform  
 -r Show random seeding options  
 -v Show library version  
 -c Show CPU settings info

Using the -a option to show all version information yields the following output on my current machine:

$ openssl version -a  
OpenSSL 3.0.4 21 Jun 2022 (Library: OpenSSL 3.0.4 21 Jun 2022)  
built on: Fri Jun 24 08:58:53 2022 UTC  
platform: linux-x86\_64  
options: bn(64,64)  
compiler: gcc -fPIC -pthread -m64 -Wa,--noexecstack -Wall -O3 -DOPENSSL\_USE\_NODELETE -DL\_ENDIAN -DOPENSSL\_PIC -DOPENSSL\_BUILDING\_OPENSSL -DNDEBUG  
OPENSSLDIR: "/usr/local/ssl"  
ENGINESDIR: "/usr/local/lib64/engines-3"  
MODULESDIR: "/usr/local/lib64/ossl-modules"  
Seeding source: os-specific  
CPUINFO: OPENSSL\_ia32cap=0x7ffaf3ffffebffff:0x29c67af

# Commands[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=5)]

There are three different kinds of commands. These are standard commands, cipher commands, and message digest commands. Calling the OpenSSL top-level help command with no arguments will result in openssl printing all available commands by group, sorted alphabetically.

## Standard Commands[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=6)]

Overview of OpenSSL's command line utilities

|  |  |
| --- | --- |
| **Command** | **Description** |
| [asn1parse](https://www.openssl.org/docs/manmaster/man1/openssl-asn1parse.html) | Parse an ASN.1 sequence. |
| [ca](https://www.openssl.org/docs/manmaster/man1/openssl-ca.html) | Certificate Authority (CA) Management. |
| [ciphers](https://www.openssl.org/docs/manmaster/man1/openssl-ciphers.html) | Cipher Suite Description Determination. |
| [cmp](https://www.openssl.org/docs/manmaster/man1/openssl-cmp.html) | Certificate Management Protocol (CMP, RFC 4210) application. |
| [cms](https://www.openssl.org/docs/manmaster/man1/openssl-cms.html) | CMS (Cryptographic Message Syntax) utility. |
| [crl](https://www.openssl.org/docs/manmaster/man1/openssl-crl.html) | Certificate Revocation List (CRL) Management. |
| [crl2pkcs7](https://www.openssl.org/docs/manmaster/man1/openssl-crl2pkcs7.html) | CRL to PKCS#7 Conversion. |
| [dgst](https://www.openssl.org/docs/manmaster/man1/openssl-dgst.html) | Message Digest calculation. MAC calculations are superseded by mac(1). |
| [dhparam](https://www.openssl.org/docs/manmaster/man1/openssl-dhparam.html) | Generation and Management of Diffie-Hellman Parameters. Superseded by genpkey(1) and pkeyparam(1). |
| [dsa](https://www.openssl.org/docs/manmaster/man1/openssl-dsa.html) | DSA Data Management. |
| [dsaparam](https://www.openssl.org/docs/manmaster/man1/openssl-dsaparam.html) | DSA Parameter Generation and Management. Superseded by genpkey(1) and pkeyparam(1). |
| [ec](https://www.openssl.org/docs/manmaster/man1/openssl-ec.html) | EC (Elliptic curve) key processing. |
| [ecparam](https://www.openssl.org/docs/manmaster/man1/openssl-ecparam.html) | EC parameter manipulation and generation. |
| [enc](https://www.openssl.org/docs/manmaster/man1/openssl-enc.html) | Symmetric cipher routines. |
| [engine](https://www.openssl.org/docs/manmaster/man1/openssl-engine.html) | Engine (loadable module) information and manipulation. |
| [errstr](https://www.openssl.org/docs/manmaster/man1/openssl-errstr.html) | Error Number to Error String Conversion. |
| [fipsinstall](https://www.openssl.org/docs/manmaster/man1/openssl-fipsinstall.html) | IPS configuration installation. |
| [gendsa](https://www.openssl.org/docs/manmaster/man1/openssl-gendsa.html) | Generation of DSA Private Key from Parameters. Superseded by genpkey(1) and pkey(1). |
| [genpkey](https://www.openssl.org/docs/manmaster/man1/openssl-genpkey.html) | Generation of Private Key or Parameters. |
| [genrsa](https://www.openssl.org/docs/manmaster/man1/openssl-genrsa.html) | Generation of RSA Private Key. Superseded by genpkey(1). |
| [help](https://www.openssl.org/docs/manmaster/man1/openssl-help.html) | Display information about a command's options. |
| [info](https://www.openssl.org/docs/manmaster/man1/openssl-info.html) | Display diverse information built into the OpenSSL libraries. |
| [kdf](https://www.openssl.org/docs/manmaster/man1/openssl-kdf.html) | Key Derivation Functions. |
| [list](https://www.openssl.org/docs/manmaster/man1/openssl-list.html) | List algorithms and features. |
| [mac](https://www.openssl.org/docs/manmaster/man1/openssl-mac.html) | Message Authentication Code Calculation. |
| [nseq](https://www.openssl.org/docs/manmaster/man1/openssl-nseq.html) | Create or examine a Netscape certificate sequence. |
| [ocsp](https://www.openssl.org/docs/manmaster/man1/openssl-ocsp.html) | Online Certificate Status Protocol utility. |
| [passwd](https://www.openssl.org/docs/manmaster/man1/openssl-passwd.html) | Generation of hashed passwords. |
| [pkcs12](https://www.openssl.org/docs/manmaster/man1/openssl-pkcs12.html) | PKCS#12 Data Management. |
| [pkcs7](https://www.openssl.org/docs/manmaster/man1/openssl-pkcs7.html) | PKCS#7 Data Management. |
| [pkcs8](https://www.openssl.org/docs/manmaster/man1/openssl-pkcs8.html) | PKCS#8 format private key conversion tool. |
| [pkey](https://www.openssl.org/docs/manmaster/man1/openssl-pkey.html) | Public and private key management. |
| [pkeyparam](https://www.openssl.org/docs/manmaster/man1/openssl-pkeyparam.html) | Public key algorithm parameter management. |
| [pkeyutl](https://www.openssl.org/docs/manmaster/man1/openssl-pkeyutl.html) | Public key algorithm cryptographic operation utility. |
| [prime](https://www.openssl.org/docs/manmaster/man1/openssl-prime.html) | Compute prime numbers. |
| [rand](https://www.openssl.org/docs/manmaster/man1/openssl-rand.html) | Generate pseudo-random bytes -- see [Random Numbers](https://wiki.openssl.org/index.php/Random_Numbers) |
| [rehash](https://www.openssl.org/docs/manmaster/man1/openssl-rehash.html) | Create symbolic links to certificate and CRL files named by the hash values. |
| [req](https://www.openssl.org/docs/manmaster/man1/openssl-req.html) | PKCS#10 X.509 Certificate Signing Request (CSR) Management. |
| [rsa](https://www.openssl.org/docs/manmaster/man1/openssl-rsa.html) | RSA key management. |
| [rsautl](https://www.openssl.org/docs/manmaster/man1/openssl-rsautl.html) | RSA utility for signing, verification, encryption, and decryption. Superseded by pkeyutl(1). |
| [s\_client](https://www.openssl.org/docs/manmaster/man1/openssl-s_client.html) | This implements a generic SSL/TLS client which can establish a transparent connection to a remote server speaking SSL/TLS. |
| [s\_server](https://www.openssl.org/docs/manmaster/man1/openssl-s_server.html) | This implements a generic SSL/TLS server which accepts connections from remote clients speaking SSL/TLS. |
| [s\_time](https://www.openssl.org/docs/manmaster/man1/openssl-s_time.html) | SSL Connection Timer. |
| [sess\_id](https://www.openssl.org/docs/manmaster/man1/openssl-sess_id.html) | SSL Session Data Management. |
| [smime](https://www.openssl.org/docs/manmaster/man1/openssl-smime.html) | S/MIME mail processing. |
| [speed](https://www.openssl.org/docs/manmaster/man1/openssl-openssl-speed.html) | Algorithm Speed Measurement. |
| [spkac](https://www.openssl.org/docs/manmaster/man1/openssl-spkac.html) | SPKAC printing and generating utility. |
| [srp](https://www.openssl.org/docs/manmaster/man1/openssl-srp.html) | Maintain SRP password file. |
| [storeutl](https://www.openssl.org/docs/manmaster/man1/openssl-storeutl.html) | Utility to list and display certificates, keys, CRLs, etc. |
| [ts](https://www.openssl.org/docs/manmaster/man1/openssl-ts.html) | Time Stamping Authority tool (client/server). |
| [verify](https://www.openssl.org/docs/manmaster/man1/openssl-verify.html) | X.509 Certificate Verification. |
| [version](https://www.openssl.org/docs/manmaster/man1/openssl-version.html) | OpenSSL Version Information. |
| [x509](https://www.openssl.org/docs/manmaster/man1/openssl-x509.html) | X.509 Certificate Data Management. |

## Generating an RSA Private Key[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=7)]

Generating a private key can be done in a variety of different ways depending on the type of key, algorithm, bits, and other options your specific use case may require. In this example, we are generating a private key using RSA and a key size of 2048 bits.

$ openssl genpkey -algorithm RSA -pkeyopt rsa\_keygen\_bits:2048 -out private-key.pem

To generate a password protected private key, the previous command may be slightly amended as follows:

$ openssl genpkey -aes256 -algorithm RSA -pkeyopt rsa\_keygen\_bits:2048 -out private-key.pem

The addition of the -aes256 option specifies the cipher to use to encrypt the private key file. For a list of available ciphers in the library, you can run the following command:

$ openssl list -cipher-algorithms

With your private key in hand, you can use the following command to see the key's details, such as its modulus and its constituent primes. Remember to change the name of the input file to the file name of your private key.

$ openssl pkey -in private-key.pem -text

The above command yields the following output in my specific case. Your output will differ but should be structurally similar.

-----BEGIN PRIVATE KEY-----  
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQDZD6IMLRFk4CaF  
w0rhRienwuE5EZ6xFE8e3C5TVi1+d9Enhi38RgkwD7UlWxPE6AWhp5T3kfrFWdak  
1lZFVPp7/btOKLjKUru15nLoA4AKYtz9W9PhsM0dyzLc6FQ6K4ReQam5pHCqI2zF  
82MwE+eIAduvuqyoQLKiI608EArWZqDtMUpBJzv0UVEYvRdnMWpCwfzpI+hPJywV  
CcTlNCT/ctGgBKyIx+dDuZ7bR9MNmSW7GreJEbTH+R13xT3dd/JCka1+LYCl4h0q  
oWhFPhOkvQzmmSzUmZlAlTDQLv2eAdJIrQcsnKZ3SsIOCC/3IpqwSzpid38Ill4O  
xH6XIrVFAgMBAAECggEBAJ2MC0JrM8TULSHJrf/0u7O4b2DMuTIuW386sSUr17mD  
nfviGF6TNvf7bq++e4rgHbZHvIg1HJ9Bpdne+J86HtUARYNlazru8fAFZEGiyLzB  
JUV/8TpO6ZJGepR8zSWrkFgZsOddw6i6LalADy5GRDcjoiDajZdR3lZxLrv5qOQU  
I1vKTf4Zs2Tl3gnaJ/Il1gBHIQ9W9xUH8jPBIwj51iXwCh8H0BiDPvFkU7cHIFCP  
sJhGsGp6OS3uSwwQuSE+NqbuPfVilysCcwgZduknyio0QO1YfMBL6+XoKE/bFHsn  
N+FzzczQg9sWyiwVR+3EeI9kp4JSElNh2nqG96i4QAECgYEA76OLUGrShHb4saoP  
aYnBAKLEdWj5K483JdY6BSbdd5RkDbJG8ExmcbfTas/BGdKc4iVCkxV3ysxKnX18  
PfxATHDLL8NMa+gGgZY5oTKUsrXEpS132HhCJ9T9LoesQjRb4kOZH8POVqm6O4Xf  
lCt0y1+M1eQHI1NPO9CmPBgouEUCgYEA5+F4SS8RMyYRkU/kx195fwh0hhaOElzr  
E8mZou3NFL/XT6/9t+2+7sMTuiQCP9zIa6s+/rrXdjWtrTcDp4WlDITas0UUgZhv  
YVBQBF4vhHxIVwJxnT9Gwi4XM1JlFmVHofWD71P6DRe7jSWRS3CujP3AE9vmpWMx  
tE1D9qLiWQECgYB445LzFYBvrKjWz4iI4CJKFNJwvGz+iXfzkXehg7KzkVtMAYSB  
0rjXYzm3J2ktgq778nn8Qxc0agy2GEil6GvzY+9MgAQ8Z0do9gTKif6zjLjP7vkH  
bdtJxsuWPoEqwMkdgqZrfNbJp0O4pVddovJ/agtdF3R2YJ+W+DH0HOfl1QKBgFnM  
c2zEEYEhaQRBUHP1gXO0rouPCI4L9e2/0QPL2/QBJzzxBuzH4X1NhsI7V7OrqOIp  
e0fiy7Y3q369I2ko1HY4rQln4z0c72VcWOCYKQbBqrInfCBNdPWWK93wNr2pk0gh  
cGqqtteDLVrIBbCVfsOTMWN/cZ7y/zi4A23sPoQBAoGAEPzcIjOyoB97Pzd7iNim  
Gin8RkwXIiFGSHo8vAh74CKBNokThM50OUNm5T2eJ4huzPpowQ+ID1mB5EjEai9n  
JY9ll3cUpawiIIW/6uGTHyXfvZWNtqEYXrVJ6fcDaKcW4y3cplNj/SJaBW8HXsW7  
YGHW3zHsgy7EOAOzPwlm9oE=  
-----END PRIVATE KEY-----  
RSA Private-Key: (2048 bit, 2 primes)  
modulus:  
 00:d9:0f:a2:0c:2d:11:64:e0:26:85:c3:4a:e1:46:  
 27:a7:c2:e1:39:11:9e:b1:14:4f:1e:dc:2e:53:56:  
 2d:7e:77:d1:27:86:2d:fc:46:09:30:0f:b5:25:5b:  
 13:c4:e8:05:a1:a7:94:f7:91:fa:c5:59:d6:a4:d6:  
 56:45:54:fa:7b:fd:bb:4e:28:b8:ca:52:bb:b5:e6:  
 72:e8:03:80:0a:62:dc:fd:5b:d3:e1:b0:cd:1d:cb:  
 32:dc:e8:54:3a:2b:84:5e:41:a9:b9:a4:70:aa:23:  
 6c:c5:f3:63:30:13:e7:88:01:db:af:ba:ac:a8:40:  
 b2:a2:23:ad:3c:10:0a:d6:66:a0:ed:31:4a:41:27:  
 3b:f4:51:51:18:bd:17:67:31:6a:42:c1:fc:e9:23:  
 e8:4f:27:2c:15:09:c4:e5:34:24:ff:72:d1:a0:04:  
 ac:88:c7:e7:43:b9:9e:db:47:d3:0d:99:25:bb:1a:  
 b7:89:11:b4:c7:f9:1d:77:c5:3d:dd:77:f2:42:91:  
 ad:7e:2d:80:a5:e2:1d:2a:a1:68:45:3e:13:a4:bd:  
 0c:e6:99:2c:d4:99:99:40:95:30:d0:2e:fd:9e:01:  
 d2:48:ad:07:2c:9c:a6:77:4a:c2:0e:08:2f:f7:22:  
 9a:b0:4b:3a:62:77:7f:08:96:5e:0e:c4:7e:97:22:  
 b5:45  
publicExponent: 65537 (0x10001)  
privateExponent:  
 00:9d:8c:0b:42:6b:33:c4:d4:2d:21:c9:ad:ff:f4:  
 bb:b3:b8:6f:60:cc:b9:32:2e:5b:7f:3a:b1:25:2b:  
 d7:b9:83:9d:fb:e2:18:5e:93:36:f7:fb:6e:af:be:  
 7b:8a:e0:1d:b6:47:bc:88:35:1c:9f:41:a5:d9:de:  
 f8:9f:3a:1e:d5:00:45:83:65:6b:3a:ee:f1:f0:05:  
 64:41:a2:c8:bc:c1:25:45:7f:f1:3a:4e:e9:92:46:  
 7a:94:7c:cd:25:ab:90:58:19:b0:e7:5d:c3:a8:ba:  
 2d:a9:40:0f:2e:46:44:37:23:a2:20:da:8d:97:51:  
 de:56:71:2e:bb:f9:a8:e4:14:23:5b:ca:4d:fe:19:  
 b3:64:e5:de:09:da:27:f2:25:d6:00:47:21:0f:56:  
 f7:15:07:f2:33:c1:23:08:f9:d6:25:f0:0a:1f:07:  
 d0:18:83:3e:f1:64:53:b7:07:20:50:8f:b0:98:46:  
 b0:6a:7a:39:2d:ee:4b:0c:10:b9:21:3e:36:a6:ee:  
 3d:f5:62:97:2b:02:73:08:19:76:e9:27:ca:2a:34:  
 40:ed:58:7c:c0:4b:eb:e5:e8:28:4f:db:14:7b:27:  
 37:e1:73:cd:cc:d0:83:db:16:ca:2c:15:47:ed:c4:  
 78:8f:64:a7:82:52:12:53:61:da:7a:86:f7:a8:b8:  
 40:01  
prime1:  
 00:ef:a3:8b:50:6a:d2:84:76:f8:b1:aa:0f:69:89:  
 c1:00:a2:c4:75:68:f9:2b:8f:37:25:d6:3a:05:26:  
 dd:77:94:64:0d:b2:46:f0:4c:66:71:b7:d3:6a:cf:  
 c1:19:d2:9c:e2:25:42:93:15:77:ca:cc:4a:9d:7d:  
 7c:3d:fc:40:4c:70:cb:2f:c3:4c:6b:e8:06:81:96:  
 39:a1:32:94:b2:b5:c4:a5:2d:77:d8:78:42:27:d4:  
 fd:2e:87:ac:42:34:5b:e2:43:99:1f:c3:ce:56:a9:  
 ba:3b:85:df:94:2b:74:cb:5f:8c:d5:e4:07:23:53:  
 4f:3b:d0:a6:3c:18:28:b8:45  
prime2:  
 00:e7:e1:78:49:2f:11:33:26:11:91:4f:e4:c7:5f:  
 79:7f:08:74:86:16:8e:12:5c:eb:13:c9:99:a2:ed:  
 cd:14:bf:d7:4f:af:fd:b7:ed:be:ee:c3:13:ba:24:  
 02:3f:dc:c8:6b:ab:3e:fe:ba:d7:76:35:ad:ad:37:  
 03:a7:85:a5:0c:84:da:b3:45:14:81:98:6f:61:50:  
 50:04:5e:2f:84:7c:48:57:02:71:9d:3f:46:c2:2e:  
 17:33:52:65:16:65:47:a1:f5:83:ef:53:fa:0d:17:  
 bb:8d:25:91:4b:70:ae:8c:fd:c0:13:db:e6:a5:63:  
 31:b4:4d:43:f6:a2:e2:59:01  
exponent1:  
 78:e3:92:f3:15:80:6f:ac:a8:d6:cf:88:88:e0:22:  
 4a:14:d2:70:bc:6c:fe:89:77:f3:91:77:a1:83:b2:  
 b3:91:5b:4c:01:84:81:d2:b8:d7:63:39:b7:27:69:  
 2d:82:ae:fb:f2:79:fc:43:17:34:6a:0c:b6:18:48:  
 a5:e8:6b:f3:63:ef:4c:80:04:3c:67:47:68:f6:04:  
 ca:89:fe:b3:8c:b8:cf:ee:f9:07:6d:db:49:c6:cb:  
 96:3e:81:2a:c0:c9:1d:82:a6:6b:7c:d6:c9:a7:43:  
 b8:a5:57:5d:a2:f2:7f:6a:0b:5d:17:74:76:60:9f:  
 96:f8:31:f4:1c:e7:e5:d5  
exponent2:  
 59:cc:73:6c:c4:11:81:21:69:04:41:50:73:f5:81:  
 73:b4:ae:8b:8f:08:8e:0b:f5:ed:bf:d1:03:cb:db:  
 f4:01:27:3c:f1:06:ec:c7:e1:7d:4d:86:c2:3b:57:  
 b3:ab:a8:e2:29:7b:47:e2:cb:b6:37:ab:7e:bd:23:  
 69:28:d4:76:38:ad:09:67:e3:3d:1c:ef:65:5c:58:  
 e0:98:29:06:c1:aa:b2:27:7c:20:4d:74:f5:96:2b:  
 dd:f0:36:bd:a9:93:48:21:70:6a:aa:b6:d7:83:2d:  
 5a:c8:05:b0:95:7e:c3:93:31:63:7f:71:9e:f2:ff:  
 38:b8:03:6d:ec:3e:84:01  
coefficient:  
 10:fc:dc:22:33:b2:a0:1f:7b:3f:37:7b:88:d8:a6:  
 1a:29:fc:46:4c:17:22:21:46:48:7a:3c:bc:08:7b:  
 e0:22:81:36:89:13:84:ce:74:39:43:66:e5:3d:9e:  
 27:88:6e:cc:fa:68:c1:0f:88:0f:59:81:e4:48:c4:  
 6a:2f:67:25:8f:65:97:77:14:a5:ac:22:20:85:bf:  
 ea:e1:93:1f:25:df:bd:95:8d:b6:a1:18:5e:b5:49:  
 e9:f7:03:68:a7:16:e3:2d:dc:a6:53:63:fd:22:5a:  
 05:6f:07:5e:c5:bb:60:61:d6:df:31:ec:83:2e:c4:  
 38:03:b3:3f:09:66:f6:81

Keep in mind the above key was generated solely for pedagogical purposes; never give anyone access to your private keys.

## Generating a Public Key[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=8)]

Having previously generated your private key, you may generate the corresponding public key using the following command.

$ openssl pkey -in private-key.pem -out public-key.pem -pubout

You may once again view the key details, using a slightly different command this time.

$ openssl pkey -in public-key.pem -pubin -text

The output for the public key will be shorter, as it carries much less information, and it will look something like this.

-----BEGIN PUBLIC KEY-----  
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA2Q+iDC0RZOAmhcNK4UYn  
p8LhORGesRRPHtwuU1YtfnfRJ4Yt/EYJMA+1JVsTxOgFoaeU95H6xVnWpNZWRVT6  
e/27Tii4ylK7teZy6AOACmLc/VvT4bDNHcsy3OhUOiuEXkGpuaRwqiNsxfNjMBPn  
iAHbr7qsqECyoiOtPBAK1mag7TFKQSc79FFRGL0XZzFqQsH86SPoTycsFQnE5TQk  
/3LRoASsiMfnQ7me20fTDZkluxq3iRG0x/kdd8U93XfyQpGtfi2ApeIdKqFoRT4T  
pL0M5pks1JmZQJUw0C79ngHSSK0HLJymd0rCDggv9yKasEs6Ynd/CJZeDsR+lyK1  
RQIDAQAB  
-----END PUBLIC KEY-----  
RSA Public-Key: (2048 bit)  
Modulus:  
 00:d9:0f:a2:0c:2d:11:64:e0:26:85:c3:4a:e1:46:  
 27:a7:c2:e1:39:11:9e:b1:14:4f:1e:dc:2e:53:56:  
 2d:7e:77:d1:27:86:2d:fc:46:09:30:0f:b5:25:5b:  
 13:c4:e8:05:a1:a7:94:f7:91:fa:c5:59:d6:a4:d6:  
 56:45:54:fa:7b:fd:bb:4e:28:b8:ca:52:bb:b5:e6:  
 72:e8:03:80:0a:62:dc:fd:5b:d3:e1:b0:cd:1d:cb:  
 32:dc:e8:54:3a:2b:84:5e:41:a9:b9:a4:70:aa:23:  
 6c:c5:f3:63:30:13:e7:88:01:db:af:ba:ac:a8:40:  
 b2:a2:23:ad:3c:10:0a:d6:66:a0:ed:31:4a:41:27:  
 3b:f4:51:51:18:bd:17:67:31:6a:42:c1:fc:e9:23:  
 e8:4f:27:2c:15:09:c4:e5:34:24:ff:72:d1:a0:04:  
 ac:88:c7:e7:43:b9:9e:db:47:d3:0d:99:25:bb:1a:  
 b7:89:11:b4:c7:f9:1d:77:c5:3d:dd:77:f2:42:91:  
 ad:7e:2d:80:a5:e2:1d:2a:a1:68:45:3e:13:a4:bd:  
 0c:e6:99:2c:d4:99:99:40:95:30:d0:2e:fd:9e:01:  
 d2:48:ad:07:2c:9c:a6:77:4a:c2:0e:08:2f:f7:22:  
 9a:b0:4b:3a:62:77:7f:08:96:5e:0e:c4:7e:97:22:  
 b5:45  
Exponent: 65537 (0x10001)

For more information on generating keys, see the source code documentation, located in the doc/HOWTO/keys.txt file.

## Generating Keys Based on Elliptic Curves[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=9)]

There are essentially two steps to generating a key:

1. Generate the parameters for the specific curve you are using
2. Use those parameters to generate the key

To see the list of curves instrinsically supported by openssl, you can use the -list\_curves</t> option when calling the ecparam command.

$ openssl ecparam -list\_curves  
 secp112r1 : SECG/WTLS curve over a 112 bit prime field  
 secp112r2 : SECG curve over a 112 bit prime field  
 secp128r1 : SECG curve over a 128 bit prime field  
 secp128r2 : SECG curve over a 128 bit prime field  
 secp160k1 : SECG curve over a 160 bit prime field  
 ...

For this example I will use the prime256v1 curve, which is an X9.62/SECG curve over a 256 bit prime field.

### Generating the Curve Parameters[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=10)]

Having selected our curve, we now call ecparam to generate our parameters file.

$ openssl ecparam -name prime256v1 -out prime256v1.pem

#### Printing Parameters to Standard Out[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=11)]

You can print the generated curve parameters to the terminal output with the following command:

$ openssl ecparam -in prime256v1.pem -noout -text  
ASN1 OID: prime256v1  
NIST CURVE: P-256

#### Printing Parameters as C Code[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=12)]

Analogously, you may also output the generated curve parameters as C code. The parameters can then be loaded by calling the get\_ec\_group\_XXX() function. To print the C code to the current terminal's output, the following command may be used:

$ openssl ecparam -in prime256v1.pem -noout -C

And here are the first few lines of the corresponding output:

EC\_GROUP \*get\_ec\_group\_256(void)  
{  
 static unsigned char ec\_p\_256[] = {  
 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00,  
 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,  
 ...

### Generating the Key[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=13)]

With the curve parameters in hand, we are now free to generate the key. Just as with the [#Generating an RSA Private Key|RSA] example above, we may optionally specify a cipher algorithm with which to encrypt the private key. The call to generate the key using the elliptic curve parameters generated in the example above looks like this:

$ openssl genpkey -aes256 -paramfile prime256v1.pem -out private-key.pem  
Enter PEM pass phrase:  
Verifying - Enter PEM pass phrase:

### Putting it All Together[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=14)]

The process of generation a curve based on elliptic-curves can be streamlined by calling the genpkey command directly and specifying both the algorithm and the name of the curve to use for parameter generation. In it's simplest form, the command to generate a key based on the same curve as in the example above looks like this:

$ openssl genpkey -algorithm EC -pkeyopt ec\_paramgen\_curve:P-256

This command will result in the generated key being printed to the terminal's output.

$ openssl genpkey -algorithm EC -pkeyopt ec\_paramgen\_curve:P-256  
  
-----BEGIN PRIVATE KEY-----  
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgqqYoJGowXJ5/GTkB  
SRLnBMNWLoQ2RM/QxrY+bfDDGRahRANCAASPY4eTANkwIIAWhh32eoFl2YFLJSWy  
bdITdZ82O5JDpDijmGmJ2hepe5afek9WVqxMPYjmbTwMPO3xMGbqUiJD  
-----END PRIVATE KEY-----

Remember that you can specify a cipher algorithm to encrypt the key with, which something you may or may not want to do, depending on your specific use case. Here is a slightly more complete example showing a key generated with a password and written to a specific output file.

$ openssl genpkey -aes256 -algorithm EC -pkeyopt ec\_paramgen\_curve:P-256 -out private-key.pem  
Enter PEM pass phrase:  
Verifying - Enter PEM pass phrase:

Just as with the previous example, you can use the pkey command to inspect your newly-generated key.

$ openssl pkey -in private-key.pem -text  
Enter pass phrase for private-key.pem:  
-----BEGIN PRIVATE KEY-----  
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgEO7CxgTwi0hsjdbp  
sXWuU2x2flLthxqXabYDOqOZCvuhRANCAAQVTLkeCBJdvMnqwZKYJxrPvTTuanrD  
NkyAPQCARKsQ7bVrP6ky/5uAcAvjuZB0xKCcSp7roXLWRzD/y/ik8P5R  
-----END PRIVATE KEY-----  
Private-Key: (256 bit)  
priv:  
 10:ee:c2:c6:04:f0:8b:48:6c:8d:d6:e9:b1:75:ae:  
 53:6c:76:7e:52:ed:87:1a:97:69:b6:03:3a:a3:99:  
 0a:fb  
pub:  
 04:15:4c:b9:1e:08:12:5d:bc:c9:ea:c1:92:98:27:  
 1a:cf:bd:34:ee:6a:7a:c3:36:4c:80:3d:00:80:44:  
 ab:10:ed:b5:6b:3f:a9:32:ff:9b:80:70:0b:e3:b9:  
 90:74:c4:a0:9c:4a:9e:eb:a1:72:d6:47:30:ff:cb:  
 f8:a4:f0:fe:51  
ASN1 OID: prime256v1  
NIST CURVE: P-256

For more details on elliptic curve cryptography or key generation, check out the [manpages](https://www.openssl.org/docs/manpages.html).

## Base64 Encoding Strings[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=15)]

For simple string encoding, you can use "here string" syntax with the [base64](https://wiki.openssl.org/index.php?title=Base64_Encoding&action=edit&redlink=1) command as below. Intuitively, the -e flag specifies the action to be encoding.

$ openssl base64 -e <<< 'Welcome to openssl wiki'  
V2VsY29tZSB0byBvcGVuc3NsIHdpa2kK

Similarly, the base64 command's -d flag may be used to indicate decoding mode.

$ openssl base64 -d <<< 'V2VsY29tZSB0byBvcGVuc3NsIHdpa2kK'  
Welcome to openssl wiki

***Note:*** base64 line length is limited to 76 characters by default in openssl (and generated with 64 characters per line).

openssl base64 -e <<< 'Welcome to openssl wiki with a very long line that splits...'  
V2VsY29tZSB0byBvcGVuc3NsIHdpa2kgd2l0aCBhIHZlcnkgbG9uZyBsaW5lIHRo  
YXQgc3BsaXRzLi4uCg==  
openssl base64 -d <<< 'V2VsY29tZSB0byBvcGVuc3NsIHdpa2kgd2l0aCBhIHZlcnkgbG9uZyBsaW5lIHRoYXQgc3BsaXRzLi4uCg=='

=> NOTHING!

To be able to decode a base64 line without line feeds that exceeds the default 76 character length restriction use the -A option.

openssl base64 -d -A <<< 'V2VsY29tZSB0byBvcGVuc3NsIHdpa2kgd2l0aCBhIHZlcnkgbG9uZyBsaW5lIHRoYXQgc3BsaXRzLi4uCg=='  
Welcome to openssl wiki with a very long line that splits...

It is recommended to actually split base64 strings into multiple lines of 64 characters, however, since the -A option is buggy, particularly with its handling of long files.

## Generating a File Hash[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=16)]

One of the most basic uses of the [dgst](https://wiki.openssl.org/index.php?title=Dgst&action=edit&redlink=1) command (short for digest) is viewing the hash of a given file. To do this, simply invoke the command with the specified digest algorithm to use. For this example, I will be hashing an arbitrary file on my system using the [MD5](https://wiki.openssl.org/index.php?title=MD5&action=edit&redlink=1), [SHA1](https://wiki.openssl.org/index.php?title=SHA1&action=edit&redlink=1), and [SHA384](https://wiki.openssl.org/index.php?title=SHA384&action=edit&redlink=1) algorithms.

$ openssl dgst -md5 primes.dat  
MD5(primes.dat)= 7710839bb87d2c4c15a86c2b2c805664  
  
$ openssl dgst -sha1 primes.dat  
SHA1(primes.dat)= 5dfab70ce825591689f4a3f65910870a9022cd32  
  
$ openssl dgst -sha384 primes.dat  
SHA384(primes.dat)= 41399bdffe6850f5a44852d967f3db415654f20dc2eb6cd231772f6ea411876d85d44091ebbc6b1f4ce8673e64617271

For a list of the available digest algorithms, you can use the following command.

$ openssl list -digest-algorithms  
RSA-MD4 => MD4  
RSA-MD5 => MD5  
RSA-MDC2 => MDC2  
RSA-RIPEMD160 => RIPEMD160  
RSA-SHA1 => SHA1  
RSA-SHA1-2 => RSA-SHA1  
...

You can also use a similar command to see the available [digest commands](https://wiki.openssl.org/index.php?title=Digest_Commands&action=edit&redlink=1):

$ openssl list -digest-commands  
blake2b512 blake2s256 md5 sha1   
sha224 sha256 sha3-224 sha3-256   
sha3-384 sha3-512 sha384 sha512   
sha512-224 sha512-256 shake128 shake256   
sm3

Below are three sample invocations of the [md5](https://wiki.openssl.org/index.php?title=Md5&action=edit&redlink=1), [sha1](https://wiki.openssl.org/index.php?title=Sha1&action=edit&redlink=1), and [sha384](https://wiki.openssl.org/index.php?title=Sha384&action=edit&redlink=1) digest commands using the same file as the [dgst](https://wiki.openssl.org/index.php?title=Dgst&action=edit&redlink=1) command invocation above.

$ openssl md5 primes.dat  
MD5(primes.dat)= 7710839bb87d2c4c15a86c2b2c805664  
  
$ openssl sha1 primes.dat  
SHA1(primes.dat)= 5dfab70ce825591689f4a3f65910870a9022cd32  
  
$ openssl sha384 primes.dat  
SHA384(primes.dat)= 41399bdffe6850f5a44852d967f3db415654f20dc2eb6cd231772f6ea411876d85d44091ebbc6b1f4ce8673e64617271

## File Encryption and Decryption[[edit](https://wiki.openssl.org/index.php?title=Command_Line_Utilities&action=edit&section=17)]

The following example demonstrates a simple file encryption and decryption using the [enc](https://wiki.openssl.org/index.php/Enc) command. The first argument is the cipher algorithm to use for encrypting the file. For this example I carefully selected the [AES-256](https://wiki.openssl.org/index.php?title=AES-256&action=edit&redlink=1) algorithm in [CBC Mode](https://wiki.openssl.org/index.php?title=CBC_Mode&action=edit&redlink=1) by looking up the available ciphers and picking out the first one I saw. To see the list of available ciphers, you can use the following command.

$ openssl enc -ciphers  
Supported ciphers:  
-aes-128-cbc -aes-128-cfb -aes-128-cfb1   
-aes-128-cfb8 -aes-128-ctr -aes-128-ecb   
-aes-128-ofb -aes-192-cbc -aes-192-cfb   
-aes-192-cfb1 -aes-192-cfb8 -aes-192-ctr  
...

You can also use the following command:

$ openssl list -cipher-algorithms  
AES-128-CBC  
AES-128-CBC-HMAC-SHA1  
AES-128-CBC-HMAC-SHA256  
id-aes128-CCM  
AES-128-CFB  
AES-128-CFB1  
AES-128-CFB8  
AES-128-CTR  
...

Having selected an encryption algorithm, you must then specify whether the action you are taking is either encryption or decryption via the -e or -d flags, respectively. The -iter flag specifies the number of iterations on the password used for deriving the encryption key. A higher iteration count increases the time required to brute-force the resulting file. Using this option implies enabling use of the [Password-Based Key Derivation Function 2](https://wiki.openssl.org/index.php?title=Password-Based_Key_Derivation_Function_2&action=edit&redlink=1), usually set using the -pbkdf2 flag. We then use the -salt flag to enable the use of a randomly generated salt in the key-derivation function.

Putting it all together, you can see the command to encrypt a file and the corresponding output below. Note that the passwords entered by the user are blank, just as they would usually be in a terminal session.

$ openssl enc -aes-256-cbc -e -iter 1000 -salt -in primes.dat -out primes.enc  
enter aes-256-cbc encryption password:  
Verifying - enter aes-256-cbc encryption password:

The analogous decryption command is as follows:

$ openssl enc -aes-256-cbc -d -iter 1000 -in primes.enc -out primes.dec  
enter aes-256-cbc decryption password: