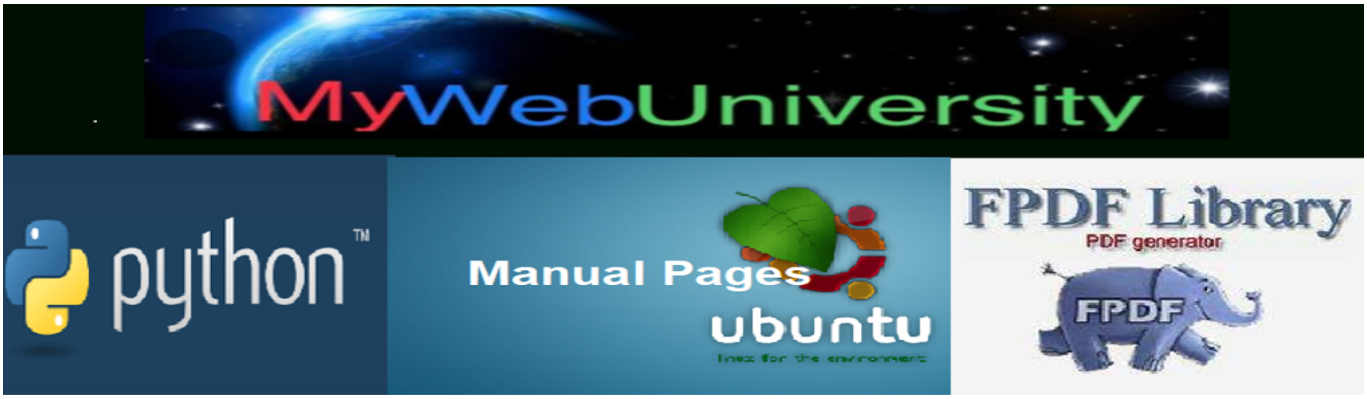




## *Rocky Enterprise Linux 9.2 Manual Pages on command 'openssl-enc.1ssl'*

***$ man openssl-enc.1ssl***

OPENSSL-ENC(1SSL) OpenSSL OPENSSL-ENC(1SSL)

NAME

openssl-enc - symmetric cipher routines

SYNOPSIS

openssl enc|cipher [-cipher] [-help] [-list] [-ciphers] [-in filename] [-out filename] [-pass arg] [-e] [-d] [-a] [-base64] [-A] [-k password] [-kfile filename] [-K key] [-iv IV] [-S salt] [-salt] [-nosalt] [-z] [-md digest] [-iter count] [-pbkdf2] [-p] [-P] [-bufsize number] [-nopad] [-v] [-debug] [-none] [-engine id] [-rand files] [-writerand file] [-provider name] [-provider-path path] [-propquery propq]

openssl cipher [...]

DESCRIPTION

The symmetric cipher commands allow data to be encrypted or decrypted using various block and stream ciphers using keys based on passwords or explicitly provided. Base64 encoding or decoding can also be performed either by itself or in addition to the encryption or decryption.

OPTIONS

-cipher

The cipher to use.

-help

Print out a usage message.

-list

List all supported ciphers.

-ciphers

Alias of -list to display all supported ciphers.

-in filename

The input filename, standard input by default.

-out filename

The output filename, standard output by default.

-pass arg

The password source. For more information about the format of arg see openssl-passphrase-options(1).

-e Encrypt the input data: this is the default.

-d Decrypt the input data.

-a Base64 process the data. This means that if encryption is taking place the data is base64 encoded after encryption. If decryption is set then the input data is base64 decoded before being decrypted.

-base64

Same as -a

-A If the -a option is set then base64 process the data on one line.

-k password

The password to derive the key from. This is for compatibility with previous versions of OpenSSL. Superseded by the -pass argument.

-kfile filename

Read the password to derive the key from the first line of filename. This is for compatibility with previous versions of OpenSSL. Superseded by the -pass argument.

-md digest

Use the specified digest to create the key from the passphrase. The default algorithm is sha-256.

-iter count

Use a given number of iterations on the password in deriving the encryption key. High values increase the time required to brute-force the resulting file. This option enables the use of PBKDF2 algorithm to derive the key.

-pbkdf2

Use PBKDF2 algorithm with default iteration count unless otherwise specified.

-nosalt

Don't use a salt in the key derivation routines. This option SHOULD NOT be used except

for test purposes or compatibility with ancient versions of OpenSSL.

-salt

Use salt (randomly generated or provide with -S option) when encrypting, this is the default.

-S salt

The actual salt to use: this must be represented as a string of hex digits. If this option is used while encrypting, the same exact value will be needed again during decryption.

-K key

The actual key to use: this must be represented as a string comprised only of hex digits. If only the key is specified, the IV must additionally specified using the -iv option. When both a key and a password are specified, the key given with the -K option will be used and the IV generated from the password will be taken. It does not make much sense to specify both key and password.

-iv IV

The actual IV to use: this must be represented as a string comprised only of hex digits. When only the key is specified using the -K option, the IV must explicitly be defined. When a password is being specified using one of the other options, the IV is generated from this password.

-p Print out the key and IV used.

-P Print out the key and IV used then immediately exit: don't do any encryption or decryption.

-bufsize number

Set the buffer size for I/O.

-nopad

Disable standard block padding.

-v Verbose print; display some statistics about I/O and buffer sizes.

-debug

Debug the BIOs used for I/O.

-z Compress or decompress encrypted data using zlib after encryption or before decryption. This option exists only if OpenSSL was compiled with the zlib or zlib-dynamic option.

-none

Use NULL cipher (no encryption or decryption of input).

-rand files, -writerand file

See "Random State Options" in openssl(1) for details.

-provider name

-provider-path path

-propquery propq

See "Provider Options" in openssl(1), provider(7), and property(7).

-engine id

See "Engine Options" in openssl(1). This option is deprecated.

NOTES

The program can be called either as "openssl cipher" or "openssl enc -cipher". The first form doesn't work with engine-provided ciphers, because this form is processed before the configuration file is read and any ENGINEs loaded. Use the openssl-list(1) command to get a list of supported ciphers.

Engines which provide entirely new encryption algorithms (such as the ccgost engine which provides gost89 algorithm) should be configured in the configuration file. Engines specified on the command line using -engine option can only be used for hardware-assisted implementations of ciphers which are supported by the OpenSSL core or another engine specified in the configuration file.

When the enc command lists supported ciphers, ciphers provided by engines, specified in the configuration files are listed too.

A password will be prompted for to derive the key and IV if necessary.

The -salt option should ALWAYS be used if the key is being derived from a password unless you want compatibility with previous versions of OpenSSL.

Without the -salt option it is possible to perform efficient dictionary attacks on the password and to attack stream cipher encrypted data. The reason for this is that without the salt the same password always generates the same encryption key.

When the salt is generated at random (that means when encrypting using a passphrase without explicit salt given using -S option), the first bytes of the encrypted data are reserved to store the salt for later decrypting.

Some of the ciphers do not have large keys and others have security implications if not used correctly. A beginner is advised to just use a strong block cipher, such as AES, in CBC mode.

All the block ciphers normally use PKCS#5 padding, also known as standard block padding.
This allows a rudimentary integrity or password check to be performed. However, since the chance of random data passing the test is better than 1 in 256 it isn't a very good test.
If padding is disabled then the input data must be a multiple of the cipher block length.
All RC2 ciphers have the same key and effective key length.
Blowfish and RC5 algorithms use a 128 bit key.

SUPPORTED CIPHERS

Note that some of these ciphers can be disabled at compile time and some are available only if an appropriate engine is configured in the configuration file. The output when invoking this command with the -list option (that is "openssl enc -list") is a list of ciphers, supported by your version of OpenSSL, including ones provided by configured engines.
This command does not support authenticated encryption modes like CCM and GCM, and will not support such modes in the future. This is due to having to begin streaming output (e.g., to standard output when -out is not used) before the authentication tag could be validated. When this command is used in a pipeline, the receiving end will not be able to roll back upon authentication failure. The AEAD modes currently in common use also suffer from catastrophic failure of confidentiality and/or integrity upon reuse of key/iv/nonce, and since openssl enc places the entire burden of key/iv/nonce management upon the user, the risk of exposing AEAD modes is too great to allow. These key/iv/nonce management issues also affect other modes currently exposed in this command, but the failure modes are less extreme in these cases, and the functionality cannot be removed with a stable release branch. For bulk encryption of data, whether using authenticated encryption modes or other modes, openssl-cms(1) is recommended, as it provides a standard data format and performs the needed key/iv/nonce management.

| | |
|---|---|
| base64 | Base 64 |
| bf-cbc | Blowfish in CBC mode |
| bf | Alias for bf-cbc |
| blowfish | Alias for bf-cbc |
| bf-cfb | Blowfish in CFB mode |
| bf-ecb | Blowfish in ECB mode |
| bf-ofb | Blowfish in OFB mode |
| cast-cbc | CAST in CBC mode |

| | |
|---|---|
| cast | Alias for cast-cbc |
| cast5-cbc | CAST5 in CBC mode |
| cast5-cfb | CAST5 in CFB mode |
| cast5-ecb | CAST5 in ECB mode |
| cast5-ofb | CAST5 in OFB mode |
| chacha20 | ChaCha20 algorithm |
| des-cbc | DES in CBC mode |
| des | Alias for des-cbc |
| des-cfb | DES in CFB mode |
| des-ofb | DES in OFB mode |
| des-ecb | DES in ECB mode |
| des-ede-cbc | Two key triple DES EDE in CBC mode |
| des-ede | Two key triple DES EDE in ECB mode |
| des-ede-cfb | Two key triple DES EDE in CFB mode |
| des-ede-ofb | Two key triple DES EDE in OFB mode |
| des-ede3-cbc | Three key triple DES EDE in CBC mode |
| des-ede3 | Three key triple DES EDE in ECB mode |
| des3 | Alias for des-ede3-cbc |
| des-ede3-cfb | Three key triple DES EDE CFB mode |
| des-ede3-ofb | Three key triple DES EDE in OFB mode |
| desx | DESX algorithm. |
| gost89 | GOST 28147-89 in CFB mode (provided by ccgost engine) |
| gost89-cnt | GOST 28147-89 in CNT mode (provided by ccgost engine) |
| idea-cbc | IDEA algorithm in CBC mode |
| idea | same as idea-cbc |
| idea-cfb | IDEA in CFB mode |
| idea-ecb | IDEA in ECB mode |
| idea-ofb | IDEA in OFB mode |
| rc2-cbc | 128 bit RC2 in CBC mode |
| rc2 | Alias for rc2-cbc |
| rc2-cfb | 128 bit RC2 in CFB mode |
| rc2-ecb | 128 bit RC2 in ECB mode |
| rc2-ofb | 128 bit RC2 in OFB mode |

rc2-64-cbc 64 bit RC2 in CBC mode

rc2-40-cbc 40 bit RC2 in CBC mode

rc4 128 bit RC4

rc4-64 64 bit RC4

rc4-40 40 bit RC4

rc5-cbc RC5 cipher in CBC mode

rc5 Alias for rc5-cbc

rc5-cfb RC5 cipher in CFB mode

rc5-ecb RC5 cipher in ECB mode

rc5-ofb RC5 cipher in OFB mode

seed-cbc SEED cipher in CBC mode

seed Alias for seed-cbc

seed-cfb SEED cipher in CFB mode

seed-ecb SEED cipher in ECB mode

seed-ofb SEED cipher in OFB mode

sm4-cbc SM4 cipher in CBC mode

sm4 Alias for sm4-cbc

sm4-cfb SM4 cipher in CFB mode

sm4-ctr SM4 cipher in CTR mode

sm4-ecb SM4 cipher in ECB mode

sm4-ofb SM4 cipher in OFB mode

aes-[128|192|256]-cbc 128/192/256 bit AES in CBC mode

aes[128|192|256] Alias for aes-[128|192|256]-cbc

aes-[128|192|256]-cfb 128/192/256 bit AES in 128 bit CFB mode

aes-[128|192|256]-cfb1 128/192/256 bit AES in 1 bit CFB mode

aes-[128|192|256]-cfb8 128/192/256 bit AES in 8 bit CFB mode

aes-[128|192|256]-ctr 128/192/256 bit AES in CTR mode

aes-[128|192|256]-ecb 128/192/256 bit AES in ECB mode

aes-[128|192|256]-ofb 128/192/256 bit AES in OFB mode

aria-[128|192|256]-cbc 128/192/256 bit ARIA in CBC mode

aria[128|192|256] Alias for aria-[128|192|256]-cbc

aria-[128|192|256]-cfb 128/192/256 bit ARIA in 128 bit CFB mode

aria-[128|192|256]-cfb1 128/192/256 bit ARIA in 1 bit CFB mode

aria-[128|192|256]-cfb8 128/192/256 bit ARIA in 8 bit CFB mode

aria-[128|192|256]-ctr 128/192/256 bit ARIA in CTR mode

aria-[128|192|256]-ecb 128/192/256 bit ARIA in ECB mode

aria-[128|192|256]-ofb 128/192/256 bit ARIA in OFB mode

camellia-[128|192|256]-cbc 128/192/256 bit Camellia in CBC mode

camellia[128|192|256] Alias for camellia-[128|192|256]-cbc

camellia-[128|192|256]-cfb 128/192/256 bit Camellia in 128 bit CFB mode

camellia-[128|192|256]-cfb1 128/192/256 bit Camellia in 1 bit CFB mode

camellia-[128|192|256]-cfb8 128/192/256 bit Camellia in 8 bit CFB mode

camellia-[128|192|256]-ctr 128/192/256 bit Camellia in CTR mode

camellia-[128|192|256]-ecb 128/192/256 bit Camellia in ECB mode

camellia-[128|192|256]-ofb 128/192/256 bit Camellia in OFB mode

EXAMPLES

Just base64 encode a binary file:

```
openssl base64 -in file.bin -out file.b64
```

Decode the same file

```
openssl base64 -d -in file.b64 -out file.bin
```

Encrypt a file using AES-128 using a prompted password and PBKDF2 key derivation:

```
openssl enc -aes128 -pbkdf2 -in file.txt -out file.aes128
```

Decrypt a file using a supplied password:

```
openssl enc -aes128 -pbkdf2 -d -in file.aes128 -out file.txt \
   -pass pass:<password>
```

Encrypt a file then base64 encode it (so it can be sent via mail for example) using AES-256 in CTR mode and PBKDF2 key derivation:

```
openssl enc -aes-256-ctr -pbkdf2 -a -in file.txt -out file.aes256
```

Base64 decode a file then decrypt it using a password supplied in a file:

```
openssl enc -aes-256-ctr -pbkdf2 -d -a -in file.aes256 -out file.txt \
   -pass file:<passfile>
```

BUGS

The -A option when used with large files doesn't work properly.

The openssl enc command only supports a fixed number of algorithms with certain parameters. So if, for example, you want to use RC2 with a 76 bit key or RC4 with an 84 bit key you can't use this program.

## HISTORY

The default digest was changed from MD5 to SHA256 in OpenSSL 1.1.0.

The -list option was added in OpenSSL 1.1.1e.

The -ciphers and -engine options were deprecated in OpenSSL 3.0.

## COPYRIGHT