OpenSSL Cheat Sheet

list

`openssl list -commands`	Display a list of standard commands.
`openssl list -digest-commands`	Display a list of message digest commands, typically used as input to the *dgst* or *speed* commands. For example, *echo test | openssl dgst -sha256*
`openssl list -digest-algorithms`	Display a list of message digest algorithms.
`openssl list -cipher-commands`	Display a list of message cipher commands, typically used as input to the enc or *speed* commands.
`openssl list -cipher-algorithms`	Display a list of cipher algorithms.

Read more about list utility and options → https://www.openssl.org/docs/man1.1.1/man1/openssl-list.html

genrsa

`openssl genrsa -out yourprivatekey.pem 2048`	Create a new rsa private key with PEM encoding with a length of 2048 bits.
`openssl genrsa -des3 -out yourprivatekey.pem 2048`	Create a new rsa private key that is passphrase protected using specified cipher.

Read more about genrsa utility and options → https://www.openssl.org/docs/man1.1.1/man1/openssl-genrsa.html

rsa

`openssl rsa -in yourprivatekey.pem -text`	Print various public and private key components (plain text format) in addition to encoded version of private key.
`openssl rsa -in yourprivatekey.pem -text -noout`	Print various public and private key components without encoded version of private key.

`openssl rsa -in yourprivatekey.pem -des3 -out yourencryptedprivatekey.pem`	Encrypt a private key using specified cipher and passphrase.
`openssl rsa -in yourencryptedprivatekey.pem -des3 -out yourprivatekey.pem`	Remove passphrase from a private key.
`openssl rsa -in yourprivatekey.pem -pubout`	Print a public part of a private key.
`openssl rsa -in yourprivatekey.pem -pubout -out yourpublickey.pem`	Export a public part to output file.
`openssl rsa -in yourprivatekey.pem -RSAPublicKey_out`	Print a public part of a private key in RSAPublicKey format.
`openssl rsa -in yourprivatekey.pem -modulus`	Print modulus of a private key.
`openssl rsa -in yourprivatekey.pem -inform pem -out yourprivatekey.der -outform der`	Convert a rsa private key from PEM to DER encoding without encryption.
`openssl rsa -in yourprivatekey.der -inform der -out yourprivatekey.pem -outform pem`	Convert a rsa private key from DER to PEM encoding.
`openssl rsa -in yourprivatekey.pem -inform pem -out yourprivatekey.der -outform der -des3`	Convert a rsa private key from PEM to DER encoding with triple DES encryption.
`openssl rsa -in yourprivatekey.der -inform der -out yourprivatekey.pem -outform pem -des3`	Convert a rsa private key from DER to PEM encoding with triple DES encryption.

Read more about rsa utility and options → https://www.openssl.org/docs/man1.1.1/man1/openssl-rsa.html

rsautl

`openssl rsautl -sign -in data.txt -inkey yourprivatekey.pem`	Sign a file using rsa private key with default PKCS#1 v1.5 padding.
`openssl rsautl -sign -in data.txt -inkey yourprivatekey.pem -keyform pem -out signature.bin`	Sign a file using rsa private key and store result in output file.
`echo test | openssl rsautl -sign -inkey yourprivatekey.pem -keyform pem -out signature.bin`	Sign an input data using rsa private key and store result in output file.
`openssl rsautl -verify -in signature.bin -inkey yourpublickey.pem -pubin`	Recover a signed data. Note, the "rsautl -verify" does not perform integrity check (or validation) but only decrypts the signature and returns the hash.

`openssl rsautl -verify -in signature.bin -inkey yourpublickey.pem -pubin -out data.txt`	Recover a signed data using public key and store result in output file. The decrypted hash is not validated with original file hash to check integrity.
`openssl rsautl -verify -in signature.bin -inkey yourpublickey.pem -pubin -raw -hexdump`	Examine a raw signed data in hexadecimal view.

Read more about rsautil utility and options → https://www.openssl.org/docs/man1.1.1/man1/rsautl.html

asn1parse

`openssl asn1parse -in yourcert.pem`	Parse a certificate file with PEM encoding.
`openssl asn1parse -inform der -in yourcert.cer`	Parse a certificate file with DER encoding. Normally, using cer or crt extension.
`openssl asn1parse -in yourcert.pem -out certrawsignature.bin -noout -strparse <pos>`	Extract a raw signature (ASN.1 structure) from certificate. The final BIT STRING contains the actual signature. Enter <pos> as offset value.
`openssl rsautl -in certrawsignature.bin -verify -asn1parse -inkey yourpublickey.pem -pubin`	Analyze a raw signature using public key. Print a parsed version of an ASN1 DigestInfo structure.
`openssl asn1parse -in yourcert.pem -out signedattr.tbs -noout -strparse 4`	Extract an actual part of a certificate that was signed and hash function used.
`openssl sha256 -c signedattr.tbs`	Print a digest computed.
`openssl asn1parse -genstr "UTF8:test"`	Perform an ASN.1 encoding of an input string in ASN1_TYPE structure.
`openssl asn1parse -genstr "UTF8:test" -noout -out utf8.asn1.der`	Perform an ASN.1 encoding of an input string, and store result in output file.
`openssl asn1parse -genconf asn1.cnf -noout -out asn1.der`	Perform an ASN.1 encoding of data from a config file, and store result in output file.
`openssl asn1parse -inform pem -in signed.p7b`	Parse a signed and/or enveloped PKCS#7 file with PEM encoding.
`openssl asn1parse -inform der -in signed.p7b`	Parse a signed and/or enveloped PKCS#7 file with DER encoding.
`openssl asn1parse -inform der -in signed.p7b -out rawsignature.bin -noout -strparse <pos>`	Extract a raw signature from PKCS#7 file. The final BIT STRING contains the actual signature. Enter <pos> as offset value.

Read more about req utility and options → https://www.openssl.org/docs/man1.1.1/man1/openssl-asn1parse.html

req

`openssl req -newkey rsa:2048 -nodes -keyout yourprivatekey.pem -new -out yourcert.csr`	Create a new rsa private key and certificate request (csr).
`openssl req -key yourprivatekey.pem -new -out yourcert.csr`	Create a csr from a private key.
`openssl req -in yourcert.csr -noout -text`	Print a csr file content.
`openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout yourprivatekey.pem -out yourcert.pem`	Create a new rsa private key and self-signed certificate.

Read more about req utility and options → https://www.openssl.org/docs/man1.1.1/man1/openssl-req.html

x509

`openssl x509 -req -days 365 -sha256 -in yourcert.csr -signkey yourprivatekey.pem -out yourcert.pem`	Create a self-signed certificate from a csr using private key.
`openssl x509 -in yourcert.pem -text`	Print a certificate file content.
`openssl x509 -in yourcert.pem -serial -noout`	Print a serial number of a certificate in hex format.
`openssl x509 -inform der -in yourcert.crt -outform pem -out yourcert.pem`	Convert a DER certificate to a PEM format.
`openssl x509 -inform pem -in yourcert.pem -outform der -out yourcert.crt`	Convert a PEM certificate to a DER format.
`openssl x509 -x509toreq -in yourcert.pem -signkey yourprivatekey.pem -out newcert.csr`	Create a certificate request (csr) from an existing certificate using private key.

Read more about x509 utility and options → https://www.openssl.org/docs/man1.1.1/man1/x509.html

dgst

`openssl dgst -sha256 data.txt`	Compute a hash of the file using specified digest algorithm.

`openssl dgst -sha256 -hex data.txt`	Compute a hash of the file using specified digest algorithm and encode in hex format.
`openssl dgst -sha256 -binary -sign yourprivatekey.pem -out data.rsasha256.bin data.txt`	Compute a signature (RSA PKCS#1 v1.5 style) using private key.
`openssl dgst -sha256 -verify yourpublickey.pem -signature data.rsasha256.bin data.txt`	Validate a signature and file integrity using public key. Note, openssl dgst can't read the public key from a certificate, you must pass the key as input.

Read more about dgst utility and options → https://www.openssl.org/docs/man1.1.1/man1/dgst.html

cms

`openssl cms -sign -in data.txt -text -out signed.msg -signer yourcert.pem`	Sign a mail (input) file using supplied certificate (embed private key), and store result in output file. If necessary, more than one signer can be added.
`openssl cms -sign -in data.txt -text -out signed.msg -signer yourcert.pem -inkey yourprivatekey.pem -nodetach`	Create an opaque (or bundled) signed mail using supplied certificate and private key, and store result in output file. The *-nodetach* option embeds data in output file.
`openssl cms -sign -in data.txt -text -out signed.msg -signer yourcert.pem -inkey yourprivatekey.pem -certfile yourcerts.pem`	Create a signed mail, bundled with additional certificates and then store result in output file. For example, include ca certs.
`openssl cms -sign -in data.txt -text -out signed.msg -signer yourcert.pem -inkey yourprivatekey.pem -keyid`	Sign a mail file using subject key identifier x509 extension instead of issuer name and serial number.
`openssl cms -sign -in message.txt -text -out signed.msg -signer mycert.pem -keyopt rsa_padding_mode:pss`	Sign a mail using RSA-PSS.
`openssl cms -sign -in data.txt -text -signer yourcert.pem -inkey yourprivatekey.pem -from sender@domain.com -to receiver@domain.com -subject "subject line" -out signed.msg`	Create a signed mail and then store result in output file. *On Unix like system, you can deliver signed mail directly like sendmail -t mail.msg*
`openssl cms -sign -in data.txt -text -signer yourcert.pem -inkey yourprivatekey.pem -from sender@domain.com -to receiver@domain.com -subject "subject line" | sendmail receiver@domain.com`	Delivery a signed mail directly to recipient. The *sendmail* utility is available on the Unix-like operating system.
`openssl cms -sign -in data.txt -binary -out signed.msg -inkey yourprivatekey.pem -signer yourcert.pem`	Sign a mail file. Unlike *-text*, the *-binary* option prevents adding a content-type (text/plain) MIME header to the supplied message before signing.
`openssl cms -cmsout -in signed.msg -noout -print`	Read a signed mail (MIME format) and writes out CMS structure (PEM encoded).
`openssl cms -cmsout -in signed.p7 -inform der -noout -print`	Print a CMS structure of a DER encoded PKCS#7 document.

`openssl cms -cmsout -in signed.p7 -inform pem -noout -print`	Print a CMS structure of a PEM encoded PKCS#7 document.
`openssl cms -verify -in signed.msg -signer signercert.pem`	Verify a signed mail and certificate chains. On success, print a signed data and export signer's certificate to cert file.
`openssl cms -verify -in signed.msg -signer signercert.pem -out data.out.txt`	Verify a signed mail and certificate chains. On success, extract the signed data part to an output file and signer's certificate to cert file, respectively.
`openssl cms -verify -in signed.msg -signer signercert.pem -out data.out.txt -content data.in.txt`	Verify a signed mail using a file containing a detached content. The *-content* option is used only with the *-verify* command in a detached signature form.
`openssl cms -verify -in signed.p7 -inform pem -content data.in.txt`	Verify a signature in PEM encoded PKCS#7 structure, supply or override content for detached signature.
`openssl cms -verify -in signed.p7 -inform der -content data.in.txt`	Verify a DER encoded signature in PKCS#7 structure, supply or override content for detached signature.
`openssl cms -resign -in mail.msg -signer newsigner.pem -out signed.msg`	Add a signer to an existing message.
`openssl cms -encrypt -in data.txt -out encrypted.msg -from sender@domain.com -to receiver@domain.com -subject "subject line" -des3 yourcert.pem`	Encrypt a mail.
`openssl cms -encrypt -in plain.txt -out encrypted.msg -recip cert.pem -keyopt rsa_padding_mode:oaep`	Encrypt a mail using RSA-OAEP.
`openssl cms -encrypt -in plain.txt -out encrypted.msg -recip cert.pem -keyopt ecdh_kdf_md:sha256`	Encrypt a mail using SHA256 KDF with an ECDH certificate.
`openssl cms -encrypt -in data.txt -binary -out encrypted.msg -signer yourcert.pem`	Encrypt a mail with no translation on input data.
`openssl cms -decrypt -in encrypted.msg -binary -recip yourcert.pem -inkey yourprivatekey.pem -out decrypted.msg`	Decrypt a mail.
`openssl cms -decrypt -in encrypted.msg -recip yourcert.pem -inkey yourprivatekey.pem`	Decrypt a mail.
`openssl cms -sign -in data.txt -text -signer yourcert.pem -inkey yourprivatekey.pem | openssl cms -encrypt -out encrypted.msg -from sender@domain.com -to receiver@domain.com -subject "subject line" -des3 usercert.pem`	Sign and encrypt a mail.

Read more about cms utility and options → https://www.openssl.org/docs/man1.1.1/man1/cms.html

smime

`openssl smime -sign -binary -in data.bin -signer yourcert.pem -inkey yourprivatekey.pem -outform der -out signed.p7s`	Create a signed mail using supplied certificate and private key, and later store result in output file.
`openssl smime -sign -in data.txt -signer yourcert.pem -inkey yourprivatekey.pem -certfile mycerts.pem -noattr -nodetach -outform pem -out signed.p7s`	Create a signed mail, bundled with additional certificates and then store result in output file.
`openssl smime -verify -in signed.p7s -inform PEM -signer yourcert.pem -CAfile cachain.pem -noverify`	Verify a signed mail.
`openssl smime -pk7out -in data.mime.msg`	Read a message (mime format) and print a PEM encoded PKCS#7 structure.
`openssl smime -pk7out -in data.mime.msg -out signature.pem`	Read a message and write a PEM encoded PKCS#7 structure in output file.
`openssl smime -pk7out -in data.mime.msg | openssl pkcs7 -print_certs`	Display a list of signer certificates from a PEM encoded PKCS#7 structure.

Read more about smime utility and options → https://www.openssl.org/docs/man1.1.1/man1/smime.html

enc

`openssl enc -aes-256-cbc -in data.txt -out data.encrypted`	Encrypt a file using specified cipher (password protected) and store result in output file.
`openssl enc -d -aes-256-cbc -in data.encrypted -out data.decrypted`	Decrypt a file using specified cipher.
`openssl enc -aes-256-cbc -pbkdf2 -in data.txt -out data.encrypted`	Encrypt a file using specified cipher and PBKDF2 key derivation, and store result in output (binary) file.
`openssl enc -aes-256-cbc -pbkdf2 -d -in data.encrypted -out data.decrypted -pass pass:<password>`	Decrypt a file using specified cipher.
`openssl enc -aes-256-cbc -pbkdf2 -a -in data.txt -out data.encrypted`	Encrypt file using specified cipher, PBKDF2 key derivation and convert data into base64 encoded before storing result in output (base64 format) file.
`openssl enc -aes-256-cbc -pbkdf2 -d -a -in data.encrypted -out data.decrypted -pass pass:<password>`	Decode a base64-encoded file and then decrypt it using specified cipher.
`echo test | openssl enc -e -a`	Encode an input test to base64. The *-e* option is optional.

`echo dGVzdCANCg== | openssl enc -d -a`	Decode a base64-encoded input.
`openssl enc -e -a -in data.txt -out data.base64`	Encode a text (or binary) file to base64 and store result in output file.
`openssl enc -d -a -in data.base64 -out data.txt`	Decode a base64 encoded file and store result in output file.

Read more about enc utility and options → https://www.openssl.org/docs/man1.1.1/man1/enc.html

base64

`openssl base64 -in data.bin -out data.b64`	Encode a binary (or any) file to base64.
`openssl base64 -d -in data.bin -out data.b64`	Decode a base64 encoded file.

Read more about enc utility and options → https://www.openssl.org/docs/man1.1.1/man1/enc.html

pkcs7

`openssl pkcs7 -inform der -in file.p7b -outform pem -print_certs`	Print all certificates in pkcs7 file.
`openssl pkcs7 -inform der -in file.p7b -print_certs -out certs.pem`	Export all certificates in pkcs7 to output file.
`openssl pkcs7 -in file.pem -outform DER -out file.der`	Convert a pkcs7 file format from PEM to DER.

Read more about enc utility and options → https://www.openssl.org/docs/man1.1.1/man1/pkcs7.html

pkcs12

`openssl pkcs12 -export -inkey yourprivatekey.pem -in yourcert.pem -out keycert.pfx`	Bundle a certificate and private key, and store result in output file (p12 or pfx).
`openssl pkcs12 -export -inkey yourprivatekey.pem -in yourcert.pem -certfile othercerts.pem -out keycert.pfx`	Bundle a certificate and private key including some other certs, and store result in output file (p12 or pfx).

`openssl pkcs12 -in keycert.pfx -info`	Print pkcs12 file content. Prompt to PEM passphrase to encrypt private key before printing.
`openssl pkcs12 -in keycert.pfx -info -nodes`	Print pkcs12 file content. The *-nodes* option allows to print private key without protection (or encryption).
`openssl pkcs12 -in keycert.pfx -info -nocerts -nodes`	Print only private key from pkcs12 file.
`openssl pkcs12 -in keycert.pfx -nocerts -nodes -out yourprivatekey.pem`	Export a private key from pkcs12 to output file.
`openssl pkcs12 -in keycert.pfx -info -nokeys`	Print only cert.
`openssl pkcs12 -in keycert.pfx -nokeys -out yourcert.pem`	Export a certificate from pkcs12 file.

Read more about pkcs12 utility and options → https://www.openssl.org/docs/man1.1.1/man1/pkcs12.html

crl2pkcs7

`openssl crl2pkcs7 -in crl.pem -certfile yourcert.pem -out bundle.pem.p7b`	Wrap or bundle a CRL and Certificate inside a PLCS#7 archive in PEM format.
`openssl crl2pkcs7 -nocrl -certfile yourcert.pem -certfile cacert.pem -outform der -out p7.der`	Create a pkcs#7 file (DER encoded) wrapping inside certificates but no CRL.

Read more about pkcs12 utility and options → https://www.openssl.org/docs/man1.1.1/man1/crl2pkcs7.html

crl

`openssl crl -inform der -text -noout -in your.crl`	Decode a PEM-encoded CRL file to print revoked certificates info.
`openssl verify -crl_check -CAfile cachain.pem -CRLfile cacrl.pem yourcert.pem`	Check a certificate for revocation using CRL and CA certs.
`openssl crl -inform der -in cacrl.crl -outform pem -out cacrl.pem`	Convert a CRL file from DER to PEM encoding.
`openssl crl -inform pem -in cacrl.pem -outform der -out cacrl.crl`	Convert a CRL file from PEM to DER encoding.

Read more about pkcs12 utility and options → https://www.openssl.org/docs/man1.1.1/man1/crl.html

s_server

`openssl s_server -cert yourcert.pem -key yourprivatekey.pem`	Implement a generic SSL/TLS server, listening for connections on default port (4433) using SSL/TLS.
`openssl s_server -cert yourcert.pem -key yourprivatekey.pem -accept 9443`	Implement a generic SSL/TLS server, listening for connections on a given port using SSL/TLS.
`openssl s_server -cert yourcert.pem -key yourprivatekey.pem -cipher ECDHE`	Use any ECDHE based ciphersuites for TLSv1.2 and below. For TLSv1.3, default ciphers will be used.
`openssl s_server -cert yourcert.pem -key yourprivatekey.pem -cipher ECDHE -ciphersuites "TLS_CHACHA20_POLY1305_SHA256"`	Use any ECDHE based ciphersuites for TLSv1.2 and below. For TLSv1.3, set the *-ciphersuites* option to configure the cipher suites that can be used.

Read more about pkcs12 utility and options → https://www.openssl.org/docs/man1.1.1/man1/s_server.html

s_client

`openssl s_client -connect <hostname>`	Connect to an SSL server.
`openssl s_client -tls1_3 -connect <hostname>:9443`	Connect to an SSL server on a given SSL port. Force to use TLS version 1.3 to establish connectivity between two ends.
`openssl s_client -tls1_3 -ciphersuites "TLS_CHACHA20_POLY1305_SHA256" -connect <hostname>:9443`	Connect to an SSL server on a given SSL port. Force to use TLS version 1.3 and specified cipher suite(s) to establish connectivity between two ends.
`openssl s_client -tls1_2 -cipher "ECDHE-RSA-AES256-GCM-SHA384" -connect <hostname>:9443`	Connect to an SSL server on a given SSL port. Force to use TLS version 1.2 and specified cipher suite(s) to establish connectivity between two ends.

Read more about pkcs12 utility and options → https://www.openssl.org/docs/man1.1.1/man1/s_client.html