OpenSSL Keys, certs, TLS

START HERE

`openssl genpkey -algorithm ED25519 -out key.pem`
New private key

`openssl req -new -key key.pem -out req.csr`
Certificate request

`openssl req -x509 -newkey rsa:4096 -noenc -days 365 -keyout key.pem -out cert.pem`
Self-signed cert

`openssl x509 -in cert.pem -noout -text`
Inspect a certificate

`openssl s_client -connect <host>:443 -servername <host>`
Test a TLS server

`openssl dgst -sha256 <file>`
Hash a file

`openssl verify -CAfile ca.pem cert.pem`
Check a chain

`openssl rand -hex 32`
Random bytes

Private key genpkey / req -newkey → `req -new` → **CSR** request with public key → `signed by a CA` → **Certificate** x509 -text to inspect → `pkcs12 -export` → **Bundle** .p12 / fullchain.pem

CONCEPTS

Usage is openssl <subcommand> <options>. Most subcommands read stdin and write stdout, so they pipe into each other. Files are PEM (base64 text) unless you ask for DER.

Passwords on the command line show up in ps and shell history. Use -passin/-passout with env:VAR, file:path or stdin instead of pass:.

OpenSSL 3 is built from providers. Old algorithms (RC2, 3DES in PKCS#12 and similar) may need -legacy. LibreSSL on macOS has different defaults and lacks some options such as -noenc.

KEYS

`openssl genpkey -algorithm ED25519 -out <key.pem>`
Generate a private key
Other algorithms: RSA, EC, X25519

`openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:4096 -out <key.pem>`
RSA key with chosen size

`openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -out <key.pem>`
Elliptic curve key

`openssl genpkey -aes-256-cbc -algorithm ED25519 -out <key.pem>`
Key protected by a passphrase

`openssl pkey -in <key.pem> -pubout -out <pub.pem>`
Extract the public key

`openssl pkey -in <key.pem> -text -noout`
Show key components

`openssl pkey -in <key.pem> -aes-256-cbc -out <enc.pem>`
Add a passphrase to a key

`openssl pkey -in <enc.pem> -out <plain.pem>`
Remove a passphrase

`openssl rand -base64 <num>`
Random bytes as base64

`openssl rand -hex <num>`
Random bytes as hex

REQUESTS AND CERTIFICATES

`openssl req -new -key <key.pem> -out <req.csr>`
CSR from an existing key

`openssl req -new -newkey rsa:2048 -noenc -keyout <key.pem> -out <req.csr>`
New key and CSR together
-noenc skips encrypting the key (older versions: -nodes)

`openssl req -new -key <key.pem> -subj "/CN=<name>" -addext "subjectAltName=DNS:<name>" -out <req.csr>`
CSR with name and SAN, no prompts
Browsers ignore CN and require a SAN

`openssl req -in <req.csr> -noout -text`
Read a CSR

`openssl req -in <req.csr> -verify -noout`
Verify CSR signature

`openssl req -x509 -newkey rsa:4096 -noenc -days <num> -keyout <key.pem> -out <cert.pem>`
Self-signed certificate

`openssl req -x509 -key <key.pem> -days <num> -subj "/CN=<name>" -addext "subjectAltName=DNS:<name>" -out <cert.pem>`
Self-signed cert from a key

`openssl x509 -req -in <req.csr> -CA <ca.pem> -CAkey <ca.key> -days <num> -copy_extensions copy -out <cert.pem>`
Sign a CSR with a CA
Add -CAcreateserial on the first run

INSPECT CERTIFICATES

`openssl x509 -in <cert.pem> -noout -text`
Full certificate details

`openssl x509 -in <cert.pem> -noout -subject -issuer`
Subject and issuer

`openssl x509 -in <cert.pem> -noout -dates`
Validity period

`openssl x509 -in <cert.pem> -noout -enddate`
Expiry date only

INSPECT CERTIFICATES (cont.)

`openssl x509 -in <cert.pem> -noout -checkend <secs>`
Exit 1 if expiring within N seconds
86400 tests for expiry within a day

`openssl x509 -in <cert.pem> -noout -ext subjectAltName`
Show the SAN list

`openssl x509 -in <cert.pem> -noout -fingerprint -sha256`
SHA-256 fingerprint

`openssl verify -CAfile <ca.pem> <cert.pem>`
Verify against a CA

`openssl verify -CAfile <ca.pem> -untrusted <inter.pem> <cert.pem>`
Verify with intermediates

`openssl pkey -in <key.pem> -pubout | sha256sum`
Hash of key public part
Compare with the cert's -pubkey output to check a key matches

`openssl x509 -in <cert.pem> -pubkey -noout | sha256sum`
Hash of cert public key

TEST A TLS SERVER

`openssl s_client -connect <host>:443 -servername <host>`
Open a TLS connection
-servername sets SNI; needed for virtual hosts

`openssl s_client -connect <host>:443 -brief </dev/null`
Short connection summary

`openssl s_client -connect <host>:443 -showcerts </dev/null`
Show the full chain sent

`openssl s_client -connect <host>:443 -tls1_3 </dev/null`
Force TLS 1.3

`openssl s_client -connect <host>:587 -starttls smtp`
Upgrade SMTP to TLS
Also imap, pop3, ftp and others

`openssl s_client -connect <host>:443 </dev/null | openssl x509 -noout -dates`
Check remote cert expiry

TEST A TLS SERVER (cont.)

`openssl s_client -connect <host>:443 </dev/null | openssl x509 -out <cert.pem>`
Save the server cert

DIGESTS AND ENCRYPTION

`openssl dgst -sha256 <file>`
SHA-256 of a file

`openssl dgst -sha256 -sign <key.pem> -out <sig> <file>`
Sign a file
RSA/EC keys only; Ed25519 rejects a digest: use pkeyutl -rawin

`openssl dgst -sha256 -verify <pub.pem> -signature <sig> <file>`
Verify a signature

`openssl enc -aes-256-cbc -pbkdf2 -salt -in <file> -out <file.enc>`
Encrypt a file with a password
Always add -pbkdf2; the default key derivation is weak

`openssl enc -d -aes-256-cbc -pbkdf2 -in <file.enc> -out <file>`
Decrypt it

CONVERT FORMATS

`openssl x509 -in <cert.pem> -outform DER -out <cert.der>`
PEM cert to DER

`openssl x509 -inform DER -in <cert.der> -out <cert.pem>`
DER cert to PEM

`openssl pkcs12 -export -inkey <key.pem> -in <cert.pem> -certfile <chain.pem> -out <bundle.p12>`
Bundle key and cert as PKCS#12

`openssl pkcs12 -in <bundle.p12> -nokeys -out <cert.pem>`
Extract certificates

`openssl pkcs12 -in <bundle.p12> -nodes -out <all.pem>`
Extract everything unencrypted

`openssl version -a`
Version and build info