OpenSSL Quick Reference

Certificates, keys, encryption, and debugging

Certificates

View Certificate Details

```
openssl x509 -in cert.pem -text -noout
openssl x509 -in cert.pem -subject -noout
openssl x509 -in cert.pem -dates -noout
openssl x509 -in cert.pem -issuer -noout
```

Convert Formats

```
# PEM to DER
openssl x509 -in cert.pem -outform DER \
    -out cert.der
# DER to PEM
openssl x509 -in cert.der -inform DER \
    -out cert.pem
```

Common Formats

PEM	Base64-encoded, **-----BEGIN CERTIFICATE-----**
DER	Binary format, compact
PFX / P12	PKCS#12 bundle (cert + key + chain)
CRT / CER	Certificate file (usually PEM or DER)

Key Generation

RSA Keys

```
openssl genrsa -out key.pem 4096
openssl rsa -in key.pem -pubout \
    -out pubkey.pem
openssl rsa -in key.pem -text -noout
```

EC Keys

```
openssl ecparam -genkey -name prime256v1 \
    -out ec_key.pem
openssl ec -in ec_key.pem -pubout \
    -out ec_pub.pem
```

Ed25519 Keys

```
openssl genpkey -algorithm Ed25519 \
    -out ed25519_key.pem
openssl pkey -in ed25519_key.pem -pubout \
    -out ed25519_pub.pem
```

Key Algorithms Comparison

RSA 2048/4096	Widely supported, larger keys
ECDSA (P-256)	Smaller keys, faster, modern TLS
Ed25519	Fastest, smallest, not in all systems

CSR

Generate CSR

```
openssl req -new -key key.pem \
    -out request.csr
# Non-interactive
openssl req -new -key key.pem -out req.csr \
    -subj "/CN=example.com/O=MyOrg/C=US"
```

Generate Key + CSR Together

```
openssl req -new -newkey rsa:4096 \
    -nodes -keyout key.pem -out req.csr \
    -subj "/CN=example.com"
```

Inspect CSR

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

Common CSR Fields

CN	Common Name (domain or hostname)
O	Organization name
OU	Organizational unit
C	Country (2-letter code)
ST	State or province
L	Locality / city

Self-Signed

Quick Self-Signed Certificate

```
openssl req -x509 -newkey rsa:4096 -nodes \
    -keyout key.pem -out cert.pem -days 365 \
    -subj "/CN=localhost"
```

With SAN (Subject Alternative Name)

```
openssl req -x509 -newkey rsa:4096 -nodes \
    -keyout key.pem -out cert.pem -days 365 \
    -subj "/CN=myapp.local" \
    -addext "subjectAltName=\
DNS:myapp.local,DNS:*.myapp.local,IP:127.0.0.1"
```

From Existing Key

```
openssl req -x509 -key key.pem \
    -out cert.pem -days 365 \
    -subj "/CN=example.com"
```

Verification

Verify Certificate

```
openssl verify -CAfile ca.pem cert.pem
openssl verify -CAfile ca.pem \
    -untrusted intermediate.pem cert.pem
```

Check Key / Cert Match

```
# Modulus must match for key and cert
openssl x509 -in cert.pem -modulus -noout
openssl rsa -in key.pem -modulus -noout
openssl req -in req.csr -modulus -noout
```

Check Expiration

```
openssl x509 -in cert.pem -checkend 86400
# Returns 0 if valid for 86400s (24h)
openssl x509 -in cert.pem -enddate -noout
```

Remote Server Certificate

```
openssl s_client -connect example.com:443 \
    < /dev/null 2>/dev/null \
    | openssl x509 -text -noout
```

Encryption

Symmetric Encryption

```
openssl enc -aes-256-cbc -salt -pbkdf2 \
    -in plain.txt -out encrypted.bin
openssl enc -aes-256-cbc -d -pbkdf2 \
    -in encrypted.bin -out plain.txt
```

Asymmetric Encryption

```
# Encrypt with public key
openssl pkeyutl -encrypt \
    -pubin -inkey pub.pem \
    -in secret.txt -out secret.enc
# Decrypt with private key
openssl pkeyutl -decrypt \
    -inkey key.pem \
    -in secret.enc -out secret.txt
```

Common Ciphers

aes-256-cbc	AES 256-bit, CBC mode (common default)
aes-256-gcm	AES 256-bit, GCM mode (authenticated)
chacha20-poly1305	Modern stream cipher (fast on ARM)

List all: `openssl enc -list`

Hashing

File Hashes

```
openssl dgst -sha256 file.txt
openssl dgst -sha512 file.txt
openssl dgst -md5 file.txt  # legacy only
```

HMAC

```
openssl dgst -sha256 -hmac "secret" file.txt
echo -n "message" | openssl dgst \
    -sha256 -hmac "mykey"
```

Hash Algorithms

SHA-256	Standard choice for integrity checks
SHA-384 / SHA-512	Stronger SHA-2 variants
SHA3-256	Latest standard (Keccak-based)
MD5	Broken, legacy only — do not use for security
BLAKE2	Fast, secure alternative (if supported)

S/MIME

Sign Email

```
openssl smime -sign -in msg.txt \
    -signer cert.pem -inkey key.pem \
    -out signed.msg
```

Verify Signed Email

```
openssl smime -verify -in signed.msg \
    -CAfile ca.pem -out original.txt
```

Encrypt / Decrypt Email

```
# Encrypt for recipient
openssl smime -encrypt -aes256 \
    -in msg.txt -out encrypted.msg \
    recipient_cert.pem
# Decrypt
openssl smime -decrypt -in encrypted.msg \
    -recip cert.pem -inkey key.pem
```

Debugging

Test TLS Connection

```
openssl s_client -connect host:443
openssl s_client -connect host:443 \
    -servername example.com  # SNI
openssl s_client -connect host:443 \
    -tls1_3  # force TLS 1.3
```

OpenSSL Quick Reference

Show Certificate Chain

```
openssl s_client -connect host:443 \
    -showcerts < /dev/null
```

Check TLS Ciphers

```
openssl ciphers -v 'HIGH:!aNULL'
openssl s_client -connect host:443 \
    -cipher 'ECDHE-RSA-AES256-GCM-SHA384'
```

PKCS#12 Operations

```
# Create PFX bundle
openssl pkcs12 -export -out bundle.pfx \
    -inkey key.pem -in cert.pem -certfile ca.pem
# Extract from PFX
openssl pkcs12 -in bundle.pfx -nodes \
    -out all.pem
```

Common Patterns

Generate Secure Random

```
openssl rand -hex 32    # 32 random bytes, hex
openssl rand -base64 24 # 24 random bytes, b64
```

Base64 Encode / Decode

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

Password Hashing

```
openssl passwd -6 -salt xyz "password"
# -6 = SHA-512, -5 = SHA-256, -1 = MD5
```

Quick Cheat: Key + Cert + Verify

```
openssl req -x509 -newkey rsa:4096 -nodes \
    -keyout k.pem -out c.pem -days 365 \
    -subj "/CN=test"
openssl x509 -in c.pem -text -noout
```