OpenSSL command cheatsheet

January 10, 2018

by Alexey Samoshkin

When it comes to security-related tasks, like generating keys, CSRs, certificates, calculating digests, debugging TLS connections and other tasks related to PKI and HTTPS, you'd most likely end up using the OpenSSL tool.

OpenSSL includes tonnes of features covering a broad range of use cases, and it's difficult to remember its syntax for all of them and quite easy to get lost. `man` pages are not so helpful here, so often we just Google "openssl how to [use case here]" or look for some kind of "openssl cheatsheet" to recall the usage of a command and see examples.

This post is my personal collection of `openssl` command snippets and examples, grouped by use case.

Use cases

Here is a list of use cases, that I'll be covering:

Surely, this is not a complete list, but it covers the most common use cases and includes those I've been working with. For example, I skip encryption and decryption, or using openssl for CA management. `openssl` is like a universe. You never know where it ends. ?

Working with RSA and ECDSA keys

In the commands below, replace `[bits]` with the key size (For example, 2048, 4096, 8192).

Generate an RSA key:

```
openssl genrsa -out example.key [bits]
```

Print public key or modulus only:

```
openssl rsa -in example.key -pubout
openssl rsa -in example.key -noout -modulus
```

Print textual representation of RSA key:
`openssl rsa -in example.key -text -noout`

Generate new RSA key and encrypt with a pass phrase based on AES CBC 256 encryption:
`openssl genrsa -aes256 -out example.key [bits]`

Check your private key. If the key has a pass phrase, you'll be prompted for it:
`openssl rsa -check -in example.key`

Remove passphrase from the key:
`openssl rsa -in example.key -out example.key`

Encrypt existing private key with a pass phrase:
`openssl rsa -des3 -in example.key -out example_with_pass.key`

Generate ECDSA key. `curve` is to be replaced with: `prime256v1`, `secp384r1`, `secp521r1`, or any other supported elliptic curve:
`openssl ecparam -genkey -name [curve] | openssl ec -out example.ec.key`

Print ECDSA key textual representation:
`openssl ec -in example.ec.key -text -noout`

List available EC curves, that OpenSSL library supports:
`openssl ecparam -list_curves`

Generate DH params with a given length:
`openssl dhparam -out dhparams.pem [bits]`

Create certificate signing requests (CSR)

In the commands below, replace `[digest]` with the name of the supported hash function: `md5`, `sha1`, `sha224`, `sha256`, `sha384` or `sha512`, etc. It's better to avoid weak functions like `md5` and `sha1`, and stick to `sha256` and above.

Create a CSR from existing private key.
`openssl req -new -key example.key -out example.csr -[digest]`

Create a CSR and a private key without a pass phrase in a single command:
`openssl req -nodes -newkey rsa:[bits] -keyout example.key -out example.csr`

Provide CSR subject info on a command line, rather than through interactive prompt.
`openssl req -nodes -newkey rsa:[bits] -keyout example.key -out example.csr -subj "/C=UA/ST=Kharkov/L=Kharkov/O=Super Secure Company/OU=IT Department/CN=example.com"`

Create a CSR from existing certificate and private key:
`openssl x509 -x509toreq -in cert.pem -out example.csr -signkey`

`example.key`

Generate a CSR for multi-domain SAN certificate by supplying an openssl config file:
`openssl req -new -key example.key -out example.csr -config req.conf`

where `req.conf`:

```
[req]prompt=nodefault_md = sha256distinguished_name = dnreq_extensions = req_ext
```

```
[dn]CN=example.com
```

```
[req_ext]subjectAltName=@alt_names
```

```
[alt_names]DNS.1=example.comDNS.2=www.example.comDNS.3=ftp.example.com
```

Create X.509 certificates

Create self-signed certificate and new private key from scratch:
`openssl req -nodes -newkey rsa:2048 -keyout example.key -out example.crt -x509 -days 365`

Create a self signed certificate using existing CSR and private key:
`openssl x509 -req -in example.csr -signkey example.key -out example.crt -days 365`

Sign child certificate using your own “CA” certificate and it’s private key. If you were a CA company, this shows a very naive example of how you could issue new certificates.
`openssl x509 -req -in child.csr -days 365 -CA ca.crt -CAkey ca.key -set_serial 01 -out child.crt`

Print textual representation of the certificate
`openssl x509 -in example.crt -text -noout`

Print certificate’s fingerprint as md5, sha1, sha256 digest:
`openssl x509 -in cert.pem -fingerprint -sha256 -noout`

Verify CSRs or certificates

Verify a CSR signature:
`openssl req -in example.csr -verify`

Verify that private key matches a certificate and CSR:
`openssl rsa -noout -modulus -in example.key | openssl sha256`
`openssl x509 -noout -modulus -in example.crt | openssl sha256`
`openssl req -noout -modulus -in example.csr | openssl sha256`

Verify certificate, provided that you have root and any intemediate certificates configured as trusted on your machine:

```
openssl verify example.crt
```

Verify certificate, when you have intermediate certificate chain. Root certificate is not a part of bundle, and should be configured as a trusted on your machine.

```
openssl verify -untrusted intermediate-ca-chain.pem
example.crt
```

Verify certificate, when you have intermediate certificate chain and root certificate, that is not configured as a trusted one.

```
openssl verify -CAFile root.crt -untrusted intermediate-ca-
chain.pem child.crt
```

Verify that certificate served by a remote server covers given host name. Useful to check your mutlidomain certificate properly covers all the host names.

```
openssl s_client -verify_hostname www.example.com -connect
example.com:443
```

Calculate message digests and base64 encoding

Calculate `md5`, `sha1`, `sha256`, `sha384`, `sha512` digests:

```
openssl dgst -[hash_function] <input.file
cat input.file | openssl [hash_function]
```

Base64 encoding and decoding:

```
cat /dev/urandom | head -c 50 | openssl base64 | openssl
base64 -d
```

TLS client to connect to a remote server

Connect to a server supporting TLS:

```
openssl s_client -connect example.com:443
openssl s_client -host example.com -port 443
```

Connect to a server and show full certificate chain:

```
openssl s_client -showcerts -host example.com -port 443
</dev/null
```

Extract the certificate:

```
openssl s_client -connect example.com:443 2>&1 < /dev/null |
sed -n '/-----BEGIN/,/-----END/p' > certificate.pem
```

Override SNI (Server Name Indication) extension with another server name. Useful for testing when multiple secure sites are hosted on same IP address:

```
openssl s_client -servername www.example.com -host example.com
-port 443
```

Test TLS connection by forcibly using specific cipher suite, e.g. `ECDHE-RSA-AES128-GCM-SHA256`. Useful to check if a server can properly talk via different configured cipher suites, not one it prefers.

```
openssl s_client -host example.com -port 443 -cipher ECDHE-RSA-AES128-GCM-SHA256 2>&1 </dev/null
```

Measure TLS connection and handshake time

Measure SSL connection time without/with session reuse:

```
openssl s_time -connect example.com:443 -new
openssl s_time -connect example.com:443 -reuse
```

Roughly examine TCP and SSL handshake times using `curl`:

```
curl -kso /dev/null -w "tcp:%{time_connect}, ssldone:%{time_appconnect}\n" https://example.com
```

Measure speed of various security algorithms:

```
openssl speed rsa2048
openssl speed ecdsap256
```

Convert between encoding and container formats

Convert certificate between DER and PEM formats:

```
openssl x509 -in example.pem -outform der -out example.der
openssl x509 -in example.der -inform der -out example.pem
```

Combine several certificates in PKCS7 (P7B) file:

```
openssl crl2pkcs7 -nocrl -certfile child.crt -certfile ca.crt -out example.p7b
```

Convert from PKCS7 back to PEM. If PKCS7 file has multiple certificates, the PEM file will contain all of the items in it.

```
openssl pkcs7 -in example.p7b -print_certs -out example.crt
```

Combine a PEM certificate file and a private key to PKCS#12 (.pfx .p12). Also, you can add a chain of certificates to PKCS12 file.

```
openssl pkcs12 -export -out certificate.pfx -inkey privkey.pem -in certificate.pem -certfile ca-chain.pem
```

Convert a PKCS#12 file (.pfx .p12) containing a private key and certificates back to PEM:

```
openssl pkcs12 -in keystore.pfx -out keystore.pem -nodes
```

List cipher suites

List available TLS cipher suites, openssl client is capable of:

```
openssl ciphers -v
```

Enumerate all individual cipher suites, which are described by a short-hand OpenSSL cipher list string. This is useful when you’re configuring server (like Nginx), and you need to test your `ssl_ciphers` string.

```
openssl ciphers -v 'EECDH+ECDSA+AESGCM:EECDH+aRSA+SHA256:EECDH:DHE+AESGCM:DHE:!RSA!aNULL:!eNULL:!LOW:!RC4'
```

Manually check certificate revocation status from OCSP responder

This is a multi-step process:

1. Retrieve the certificate from a remote server
2. Obtain the intermediate CA certificate chain
3. Read OCSP endpoint URI from the certificate
4. Request a remote OCSP responder for certificate revocation status

First, retrieve the certificate from a remote server:
`openssl s_client -connect example.com:443 2>&1 < /dev/null | sed -n '/-----BEGIN/,/-----END/p' >` cert.pem

You'd also need to obtain intermediate CA certificate chain. Use `-showcerts` flag to show full certificate chain, and manually save all intermediate certificates to `chain.pem` file:
`openssl s_client -showcerts -host example.com -port 443 </dev/n`ull

Read OCSP endpoint URI from the certificate:
`openssl x509 -in cert.pem -noout -ocsp_uri`

Request a remote OCSP responder for certificate revocation status using the URI from the above step (e.g. http://ocsp.stg-int-x1.letsencrypt.org).
`openssl ocsp -header "Host" "ocsp.stg-int-x1.letsencrypt.org" -issuer chain.pem -VAfile chain.pem -cert cert.pem -text -url http://ocsp.stg-int-x1.letsencrypt.org`

Resources

I've put together a few resources about OpenSSL that you may find useful.

OpenSSL Essentials: Working with SSL Certificates, Private Keys and CSRs | DigitalOcean — https://www.digitalocean.com/community/tutorials/openssl-essentials-working-with-ssl-certificates-private-keys-and-csrs

The Most Common OpenSSL Commands — https://www.sslshopper.com/article-most-common-openssl-commands.html

OpenSSL: Working with SSL Certificates, Private Keys and CSRs — https://www.dynacont.net/documentation/linux/openssl/