Commonly Used OpenSSL Commands

Posted on 20 June 2014

Introduction

Over the years I have had to do a lot of repetitive tasks in OpenSSL, and I've always had to hunt down what command I needed to use. So, I finally made a list of the most common use cases and commands, and now it's time to share.

A Word About Certificate Formats and Encoding

There are two main types of encoding of certificates; `DER` and `PEM`.

`DER` is a binary encoding of a certificate. Typically these use the file extension of `.crt` or `.cer`.

`PEM` is a Base64 encoding of a certificate represented in ASCII therefore it is readable as a block of text. This is very useful as you can open it in a text editor work with the data more easily. The data itself is contained between a prefix of:

```
-----BEGIN CERTIFICATE-----
```

and a postfix of:

```
-----END CERTIFICATE-----
```

Similarly, RSA keys have a prefix and postfix as well. They are denoted with:

`-----BEGIN PRIVATE KEY-----`

and

`-----END PRIVATE KEY-----`

Certificate Signing Requests use:

`-----BEGIN CERTIFICATE REQUEST-----`

and

`-----END CERTIFICATE REQUEST-----`

Typically these use the file extension of `.pem`. RSA private and public keys use the file extension of `.key`. Certificate Signing Requests (CSRs) use the file extension of `.csr`.

In the event that you are getting errors when running any OpenSSL commands, you may need to explicitly declare the input format and/or the output format. This can be done by adding the following flags to almost any command:

`-inform <pem|der>` and `-outform <pem|der>`

Creating an RSA Private Key

Create a 2048 bit RSA private key that is unencrypted:

```
openssl genrsa -out name.unencrypted.priv.key 2048
```

Create a 2048 bit RSA private key that is encrypted with 3DES:

```
openssl genrsa -des3 -out name.encrypted.priv.key 2048
```

Encrypting/Decrypting an RSA Private Key

Encrypt an RSA private key with 3DES:

```
openssl rsa -des3 -in name.unencrypted.priv.key -out name.encrypted.priv.key
```

Decrypt an RSA private key:

```
openssl rsa -in name.encrypted.priv.key -out name.unencrypted.priv.key
```

Creating a Certificate Signing Request

Create a CSR for an existing private key:

```
openssl req -new -key name.<en|unen>crypted.priv.key -out name.csr
```

Create a CSR based on a previously issued certificate:

```
openssl x509 -x509toreq -in name.cer -signkey name.<en|unen>crypted.priv.key -out name.csr
```

Create an unencrypted private key and CSR in one command:

```
openssl req -new -newkey rsa:2048 -nodes -keyout name.unencrypted.priv.key -out name.csr
```

Create an encrypted private key and CSR in one command:

```
openssl req -new -newkey rsa:2048 -keyout name.encrypted.priv.key -out name.csr
```

Creating Certificate Signing Requests with

Subject Alternate Names

Creating a CSR with Subject Alternate Names (SANs) requires creating a configuration file with the specifics. Then you call it with OpenSSL.

Create a file, `name.req.config`:

```
[ req ]
default_bits        = 2048
distinguished_name  = req_distinguished_name
req_extensions     = req_ext

[ req_distinguished_name ]
countryName            = Country Name (2 letter code)
countryName_default    = US
stateOrProvinceName    = State or Province Name (full name)
stateOrProvinceName_default = Florida
localityName           = Locality Name (eg, city)
localityName_default   = Tampa
organizationName          = Organization Name (eg, company)
organizationName_default  = Acme Corporation
commonName             = Common Name (eg, YOUR name)
commonName_max         = 64

[ req_ext ]
subjectAltName          = @alt_names

[alt_names]
DNS.1   = host1.domain.com
DNS.2   = host2.domain.com
DNS.3   = host3.domain.com
DNS.4   = host.differentdomain.com
```

Create the CSR by referencing the above configuration file:

`openssl req -new -key name.encrypted.priv.key -config name.req.config -out name.csr`

Showing Contents of Certificate Signing Requests

Print out the contents of the CSR in human-readable format:

```
openssl req -in name.csr -noout -text
```

Showing Contents of Certificates

Print out the contents of the certificate in human-readable format:

```
openssl x509 -in name.pem -noout -text
```

Verifying Association of Private Key to Certificate

To compare whether a private key and certificate match you need to compare the modulus of both. Considering these are very long strings of text and numbers, it's easier to perform an MD5 checksum and compare the hashes.

Output the modulus MD5 hash of the certificate:

```
openssl x509 -noout -modulus -in name.pem | openssl md5
```

Output the modulus MD5 hash of the private key:

```
openssl rsa -noout -modulus -in name.<en|unen>crypted.priv.key | ope
nssl md5
```

Compare the outputs to make sure the MD5 hashes match.

Bash one-liner:

For unencrypted private key:

```
diff -q -s <(openssl x509 -noout -modulus -in name.pem | openssl md5
) <(openssl rsa -noout -modulus -in name.unencrypted.priv.key | opens
sl md5)
```

For encrypted private key:

```
diff -q -s <(openssl x509 -noout -modulus -in name.pem | openssl md5
) <(openssl rsa -noout -modulus -in name.encrypted.priv.key -passin p
ass:SuperSecretPassword | openssl md5)
```

The above command will show `Files /dev/fd/63 and /dev/fd/62 are iden tical` if the MD5 hashes match, and will show `Files /dev/fd/63 and /dev/f d/62 differ` if the MD5 hashes are different.

Combining Root CA and Intermediate CA Certificates into One File

In order to work with certificates that have more than one CA certificate in the issuance path, you have to combine all of the certificates into one single file. Most certificates will be issued by an intermediate authority, and then that intermediate will have been issued by a root authority.

To combine multiple `PEM` certificates, you just need to put the ASCII data from all of the certificates into one file. Below is an example of this.

```
-----BEGIN CERTIFICATE-----
MIIEKjCCAxKgAwIBAgIEOGPe+DANBgkqhkiG9w0BAQUFADCBtDEUMBIGA1UE
ChML
RW50cnVzdC5uZXQxQDA+BgNVBAsUN3d3dy5lbnRydXN0Lm5ldC9DUFNfMjA0
OCBp
bmNvcnAuIGJ5IHJlZi4gKGxpbWl0cyBsaWFiLikxJTAjBgNVBAsTHChjKSAx
OTk5
IEVudHJ1c3QubmV0IExpbWl0ZWQxMzAxBgNVBAMTKkVudHJ1c3QubmV0IENl
```

```
cnRp
ZmljYXRpb24gQXV0aG9yaXR5ICgyMDQ4KTAeFw05OTEyMjQxNzUwNTFaFw0y
OTA3
MjQxNDE1MTJaMIG0MRQwEgYDVQQKEwtFbnRydXN0Lm5ldDFAMD4GA1UECxQ3
d3d3
LmVudHJ1c3QubmV0L0NQU18yMDQ4IGluY28ycC4gYnkgcmVmLiAobGltaXRz
IGxp
YWIuKTElMCMGA1UECxMcKGMpIDE5OTkgRW50cnVzdC5uZXQgTGltaXRlZDEz
MDEG
A1UEAxMqRW50cnVzdC5uZXQgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkgKDIw
NDgp
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEArU1LqRKGsuqjIAcV
FmQq
K0vRvwtKTY7tgHalZ7d4QMBzQshowNtTK91euHaYNZOLGp18EzoOH1u3Hs/l
JBQe
sYGpjX24zGtLA/ECDNyrpUAkAH90lKGdCCmziAv1h3edVc3kw37XamSrhRSG
lVuX
MlBvPci6Zgzj/L24ScF2iUkZ/cCovYmjZy/Gn7xxGWC4LeksyZB2ZnuU4q94
1mVT
XTzWnLLPKQP5L6RQstRIzgUyVYr9smRMDuSYB3Xbf9+5CFVghTAp+XtIpGmG
4zU/
HoZdenoVve8AjhUiVBcAkCaTvA5JaJG/+EfTnZVCwQ5N328mz8MYIWJmQ3DW
1cAH
4QIDAQABo0IwQDAOBgNVHQ8BAf8EBAMCAQYwDwYDVR0TAQH/BAUwAwEB/zAd
BgNV
HQ4EFgQUVeSB0RGAvtiJuQijMfmhJAkWuXAwDQYJKoZIhvcNAQEFBQADggEB
ADub
j1abMOdTmXx6eadNl9cZlZD7Bh/KM3xGY4+WZiT6QBshJ8rmcnPyT/4xmf3I
DExo
U8aAghOY+rat2l098c5u9hURlIIM7j+VrxGrD9cv3h8Dj1csHsm7mhpElesY
T6Yf
zX1XEC+bBAlahLVu2B064dae0Wx5XnkcFMXj0EyTO2U87d89vqbllRrDtRnD
vV5b
u/8j72gZyxKTJ1wDLW8w0B62GqzeWvfRqqgnpv55gcR5mTNXuhKwqeBCbJPK
Vt7+
bYQLCIt+jerXmCHG8+c8eS9enNFMFY3h7CI3zJpDC5fcgJCNs2ebb0gIFVbP
v/Er
fF6adulZkMV8gzURZVE=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
```

MIIE9TCCA92gAwIBAgIETA6MOTANBgkqhkiG9w0BAQUFADCBtDEUMBIGA1UE
ChML
RW50cnVzdC5uZXQxQDA+BgNVBAsUN3d3dy5lbnRydXN0Lm5ldC9DUFNfMjA0
OCBp
bmNvcnAuIGJ5IHJlZi4gKGxpbWl0cyBsaWFiLikxJTAjBgNVBAsTHChjKSAx
OTk5
IEVudHJ1c3QubmV0IExpbWl0ZWQxMzAxBgNVBAMTKkVudHJ1c3QubmV0IENl
cnRp
ZmljYXRpb24gQXV0aG9yaXR5ICgyMDQ4KTAeFw1xMTExMTExNTQwNDBaFw0y
MTEx
MTIwMjUxMTdaMIGxMQswCQYDVQQGEwJVUzEWMBQGA1UEChMNRW50cnVzdCwg
SW5j
LjE5MDcGA1UECxMwd3d3LmVudHJ1c3QubmV0L3JwYSBpcyBpbmNvcnBvcmF0
ZWQg
YnkgcmVmZXJlbmNlMR8wHQYDVQQLExYoYykgMjAwOSBFbnRydXN0LCBJbmMu
MS4w
LAYDVQQDEyVFbnRydXN0IENlcnRpZmljYXRpb24gQXV0aG9yaXR5IC0gTDFD
MIIB
IjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAl6MtPJ7eBdoTwhGNnY7j
f8dL
flqfs/9iq3PIKGu6EGSChxPNVxj/KM7A5g4GkVApg9Hywyrb2NtOBMwA64u2
lty8
qvpSdwTB2xnkrpz9PIsD7028GgNl+cGxP3KG8jiqGa4QiHgo2nXDPQKCApy5
wWV3
diRMmPdtMTj72/7bNwJ2oRiXpszeIAlJNiRpQvbkN2LxWW2pPO00nKOO29w6
1/cK
b+8u2NWTWnrtCElo4kHjWpDBhlX8UUOd4LLEZ7TLMjEl8FSfS9Fv29Td/K9e
bHiQ
ld7KOki5eTybGdZ1BaD5iNfB6KUJ5BoV3IcjqrJ1jGMlh9j4PabCzGb/pWZo
VQID
AQABo4IBDjCCAQowDgYDVR0PAQH/BAQDAgEGMBIGA1UdEwEB/wQIMAYBAf8C
AQAw
MwYIKwYBBQUHAQEEJzAlMCMGCCsGAQUFBzABhhdodHRwOi8vb2NzcC5lbnRy
dXN0
Lm5ldDAyBgNVHR8EKzApMCegJaAjhiFodHRwOi8vY3JsLmVudHJ1c3QubmV0
LzIw
NDhjYS5jcmwwOwYDVR0gBDQwMjAwBgRVHSAAMCgwJgYIKwYBBQUHAgEWGmh0
dHA6
Ly93d3cuZW50cnVzdC5uZXQvcnBhMB0GA1UdDgQWBBQe8auJBvhJDwEzd+4U
eu4Z

```
fJMoTTAfBgNVHSMEGDAWgBRV5IHREYC+2Im5CKMx+aEkCRa5cDANBgkqhkiG
9w0B
AQUFAAOCAQEAQJqHfojUzCanS/p4SiDV+aI2IbvuW6BPRI3PqvmXF5aEqchn
m7vm
EN551lZqpHgUSdl87TBeaeptJEZaiDQ9JifPaUGEHATaGTgu24lBOX5lH51a
Oszh
DEw3oc5gk6i1jMo/uitdTBuBiXrKNjCc/4Tj/jrx93lxybXTMwPKd86wuinS
NF1z
/6T98iW4NUV5eh+Xrsm+CmiEmXQ5qE56JvXN3iXiN4VlB6fKxQW3EzgNLfBt
Gc7e
mWEn7kVuxzn/9sWL4Mt8ih7VegcxKlJcOlAZOKlE+jyoz+95nWrZ5S6hjyko
1+yq
wfsm5p9GJKaxB825DOgNghYAHZaS/KYIoA==
-----END CERTIFICATE-----
```

Verifying Validity of Certificate Chain

Verify validity of certificate for `sslserver` usage:

```
openssl verify -verbose -purpose sslserver -CAfile CAchain.pem name.pem
```

Combining Private Key, Certificate, and CA Chain into a PFX

Combine into PFX:

```
openssl pkcs12 -export -out name.pfx -inkey name.<en|unen>crypted.priv.key -in name.pem -certfile CAchain.pem
```

Breaking Apart a PFX into Private Key, Certificate, and CA Chain

Extract Private Key

Extract encrypted private key:

```
openssl pkcs12 -in name.pfx -nocerts -out name.encrypted.priv.key
```

Extract unencrypted private key:

```
openssl pkcs12 -in name.pfx -nocerts -nodes -out name.unencrypted.priv.key
```

Extract Certificate

Extract only the certificate:

```
openssl pkcs12 -in name.pfx -nokeys -clcerts -out name.pem
```

Extract Certificate Authority Chain

Extract CA chain. If there are multiple certificates in the chain, they will all be in the same output file.

```
openssl pkcs12 -in name.pfx -nokeys -cacerts -out CAchain.pem
```

Converting To/From PEM & DER

Convert from PEM to DER:

```
openssl x509 -in name.pem -inform pem -out name.cer -outform der
```

Convert from DER to PEM:

```
openssl x509 -in name.cer -inform der -out name.pem -outform pem
```