

Introduction to OpenSSL

Author: Sanjay Ahuja

Table of Contents

Introduction:.....	3
OpenSSL Capabilities:.....	3
OpenSSL Commands:.....	3
OpenSSL version:	3
List of the available commands:	3
How to generate a self-signed RSA type certificate?	5
How to generate a Certificate Request?.....	6
How to generate a self-signed DSA type certificate?	7
How to extract information from a certificate?	7
How to convert certificate from PEM to DER?.....	9
How to convert certificate from DER to PEM?.....	10
How to Encrypt a text file?.....	10
How to Decrypt a text file?.....	10
How to sign a digest?.....	10
How to verify a signed digest?.....	11
How do I generate an RSA key?	11
How do I generate a public RSA key?	11
How to generate OCSP response from openvalidation.org site?.....	11
How to verify a OCSP response from openvalidation.org site?.....	12
How to decode OCSP response?.....	12
How to Install OpenSSL based local OCSP Server?.....	12
How to generate OCSP response from local server?.....	12
How to verify OCSP responses from local server?.....	13
How to generate Abnormal OCSP response?.....	14

Introduction:

- OpenSSL is a library written in the C programming language that provides routines for cryptographic primitives utilized in implementing the Secure Sockets Layer (SSL) protocol.
- OpenSSL also includes routines for implementing the SSL protocol itself
- It includes an application called openssl that provides a command line interface.

OpenSSL Capabilities:

OpenSSL supports following algorithms:

- Symmetric (single key) ciphers:
 - aes-128-cbc, aes-128-cfb, aes-128-cfb1, aes-128-cfb8, aes-128-ecb, aes-128-ofb, aes-192-cbc, aes-192-cfb, aes-192-cfb1, aes-192-cfb8, aes-192-ecb, aes-192-ofb, aes-256-cbc, aes-256-cfb, aes-256-cfb1, aes-256-cfb8, aes-256-ecb, aes-256-ofb
 - bf, bf-cbc, bf-cfb, bf-ecb, bf-ofb, blowfish
 - cast, cast-cbc, cast5-cbc, cast5-cfb, cast5-ecb, cast5-ofb
 - des, des-cbc, des-cfb, des-cfb1, des-cfb8, des-ecb, des-ede, des-ede-cbc, des-ede-cfb, des-ede-ofb, des-ede3, des-ede3-cbc, des-ede3-cfb, des-ede3-ofb, des-ofb, des3, desx, desx-cbc
 - rc2, rc2-40-cbc, rc2-64-cbc, rc2-cbc, rc2-cfb, rc2-ecb, rc2-ofb, rc4, rc4-40
- Asymmetric (Dual key) ciphers:
 - rsa, dsa
- Hash functions:
 - md2, md4, md5, rmd160, sha, sha1.
- Message Authentication Code (MAC)

OpenSSL Commands:

OpenSSL version:

The version of OpenSSL can be found by **version** command.

```
$ openssl version
OpenSSL 0.9.8b 04 May 2006
```

You can get much more information with the **version -a** option.

```
$ openssl version -a
OpenSSL 0.9.8b 04 May 2006
built on: Thu Jun  1 18:04:23 WEDT 2006
platform: Cygwin
options:  bn(64,32) md2(int) rc4(idx,int) des(ptr,risc1,16,long) blowfish(idx)
compiler: gcc -D_WINDLL -DOPENSSL_PIC -DOPENSSL_THREADS -D_DSO_DLFCN -DHAVE_DLFCN
N_H -DTERMIOS -DL_ENDIAN -fomit-frame-pointer -O3 -march=i486 -Wall -DOPENSSL_BN
_ASM_PART_WORDS -DOPENSSL_IA32_SSE2 -DSHA1_ASM -DMD5_ASM -DRMD160_ASM -DAES_ASM
```

```
OPENSSLDIR: "/usr/ssl"
```

List of the available commands:

The best thing to do is provide an invalid command (help or -h will do nicely) to get a readable answer.

```
$ openssl --help
openssl:Error: '--help' is an invalid command.

Standard commands
asn1parse      ca          ciphers      crl          crl2pkcs7
dgst           dh          dhparam      dsa          dsaparam
ec             ecparam    enc          engine       errstr
gendh          gendsa     genrsa       nseq        ocsf
passwd        pkcs12     pkcs7        pkcs8       prime
rand           req        rsa          rsautl      s_client
s_server       s_time     sess_id      smime       speed
spkac          verify     version      x509

Message Digest commands (see the `dgst' command for more details)
md2            md4         md5          rmd160      sha
sha1

Cipher commands (see the `enc' command for more details)
aes-128-cbc    aes-128-ecb aes-192-cbc  aes-192-ecb aes-256-cbc
aes-256-ecb    base64      bf          bf-cbc      bf-cfb
bf-ecb        bf-ofb     cast       cast-cbc    cast5-cbc
cast5-cfb     cast5-ecb  cast5-ofb  des         des-cbc
des-cfb       des-ecb    des-ede    des-ede-cbc des-ede-cfb
des-ede-ofb   des-ede3   des-ede3-cbc des-ede3-cfb des-ede3-ofb
des-ofb       des3       desx       rc2         rc2-40-cbc
rc2-64-cbc    rc2-cbc    rc2-cfb    rc2-ecb    rc2-ofb
rc4           rc4-40
```

You can use the same trick with any of the subcommands.

```
$ openssl ca --help
unknown option --help
usage: ca args

-verbose          - Talk alot while doing things
-config file      - A config file
-name arg         - The particular CA definition to use
-gencrl           - Generate a new CRL
-crl days days    - Days is when the next CRL is due
-crl hours hours  - Hours is when the next CRL is due
-startdate YYMMDDHHMMSSZ - certificate validity notBefore
-enddate YYMMDDHHMMSSZ   - certificate validity notAfter (overrides -days)
-days arg         - number of days to certify the certificate for
-md arg          - md to use, one of md2, md5, sha or sha1
-policy arg       - The CA 'policy' to support
-keyfile arg     - private key file
-keyform arg     - private key file format (PEM or ENGINE)
-key arg         - key to decode the private key if it is encrypted
```

```

-cert file      - The CA certificate
-selfsign      - sign a certificate with the key associated with it
-in file       - The input PEM encoded certificate request(s)
-out file      - Where to put the output file(s)
-outdir dir    - Where to put output certificates
-infiles ....  - The last argument, requests to process
-spkac file    - File contains DN and signed public key and challenge
-ss_cert file  - File contains a self signed cert to sign
-preservedDN   - Don't re-order the DN
-noemailDN    - Don't add the EMAIL field into certificate' subject
-batch        - Don't ask questions
-msie_hack     - msie modifications to handle all those universal strings
-revoke file   - Revoke a certificate (given in file)
-subj arg      - Use arg instead of request's subject
-utf8         - input characters are UTF8 (default ASCII)
-multivalue-rdn - enable support for multivalued RDNs
-extensions .. - Extension section (override value in config file)
-extfile file  - Configuration file with X509v3 extensions to add
-crlxets ..   - CRL extension section (override value in config file)
-engine e     - use engine e, possibly a hardware device.
-status serial - Shows certificate status given the serial number
-updatedb     - Updates db for expired certificates

```

How to generate a self-signed RSA type certificate?

To generate a new RSA certificate with 1024 bit key the command is as below:

```

$ openssl req \
> -x509 -nodes -days 365 \
> -newkey rsa:1024 -keyout mycert.pem -out mycert.pem

```

The options are self explanatory. The above command will generate a x509 type RSA certificate mycert.pem with key size 1024, validity of 365 days.

Using this command-line invocation, you'll have to answer a lot of questions: Country Name, State, City, and so on. The tricky question is "Common Name." You'll want to answer with the *hostname or CNAME by which people will address the server*. This is very important. If your web server's real hostname is mybox.mydomain.com but people will be using www.mydomain.com to address the box, then use the latter name to answer the "Common Name" question.

```

$ openssl req \
> -x509 -nodes -days 365 \
> -newkey rsa:1024 -keyout mycert.pem -out mycert.pem
Generating a 1024 bit RSA private key
.....++++++
.....++++++
writing new private key to 'mycert.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank

```

```

For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [IN]:
State or Province Name (full name) [New Delhi]:
Locality Name (eg, city) []:
Organization Name (eg, company) [SafeNet Infotech Pvt. Ltd.]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) [Sanjay]:
Email Address [skahuja@safenet-inc.com]:

```

The parameters can also be provided by command line option also. For *e.g.* You can use *-subj* option to provide the information on command line as shown below.

```

$ openssl req \
> -x509 -nodes -days 365 \
> -subj '/C=IN/ST=Delhi/L=Okhla/CN=www.safenet-inc.com' \
> -newkey rsa:1024 -keyout mycert.pem -out mycert.pem
Generating a 1024 bit RSA private key
.....++++++
.....++++++
writing new private key to 'mycert.pem'
-----

```

The default value of these questions is defined in the file `\usr\ssl\openssl.cnf`, which can be changed if required.

How to generate a Certificate Request?

For generating a certificate request the first step is to generate private key. The command for generating private key is as below:

```

$ openssl genrsa -out my.key 504
Generating RSA private key, 504 bit long modulus
.....++++++
.....++++++
e is 65537 (0x10001)

```

By default the exponent value is 65537 (0x10001). If you wish to generate key with different exponent for *e.g.* Exponent 3 then the same can be done by defining exponent value at command line as shown below:

```

$ openssl genrsa -des3 -3 -out my.key 3072
Generating RSA private key, 3072 bit long modulus
.....++
.....++
e is 3 (0x3)
Enter pass phrase for my.key:
Verifying - Enter pass phrase for my.key:

```

Once you have generated your key then the command for generating a certificate request is as below:

You are about to be asked to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

```
-----  
Country Name (2 letter code) [IN]:  
State or Province Name (full name) [New Delhi]:  
Locality Name (eg, city) []:  
Organization Name (eg, company) [SafeNet Infotech Pvt. Ltd.]:  
Organizational Unit Name (eg, section) []:  
Common Name (eg, YOUR name) [Sanjay]:  
Email Address [skahuja@safenet-inc.com]:
```

How to extract information from a certificate?

An x509 certificate contains a wide range of information: issuer, valid dates, subject, and some hardcore crypto stuff. The command for extracting the information from certificate is as below:

```
$ openssl x509 -in mycert.pem -text
```

where mycert.pem is the x509 type certificate.

It will show the contents of certificate in readable format as shown below:

```
Certificate:  
  Data:  
    Version: 3 (0x2)  
    Serial Number:  
      80:90:8c:6d:55:71:cd:72  
    Signature Algorithm: sha1WithRSAEncryption  
    Issuer: C=IN, ST=New Delhi, O=SafeNet Infotech Pvt. Ltd., CN=Sanjay/emailAddress=skahuja@safenet-inc.com  
    Validity  
      Not Before: Oct 17 05:13:22 2006 GMT  
      Not After : Oct 17 05:13:22 2007 GMT  
    Subject: C=IN, ST=New Delhi, O=SafeNet Infotech Pvt. Ltd., CN=Sanjay/emailAddress=skahuja@safenet-inc.com  
    Subject Public Key Info:  
      Public Key Algorithm: rsaEncryption  
      RSA Public Key: (1024 bit)  
        Modulus (1024 bit):  
          00:df:f9:07:1d:5b:94:ec:e9:e8:e0:9e:65:b1:2e:  
          75:1f:a2:31:ef:01:cd:e5:a1:2e:f5:a1:13:53:5e:  
          08:03:d3:f3:01:3e:78:f8:d0:e3:02:42:20:40:4b:  
          f7:7f:3b:58:cb:15:20:e0:db:51:16:fc:c5:e0:91:  
          b8:31:b7:c2:40:63:2b:c1:3e:25:dc:87:8c:ff:4c:  
          7a:f1:89:1e:dc:6e:5c:22:b5:1c:f5:a7:50:51:c2:  
          84:bf:86:c8:ef:4c:84:c1:bc:c6:5e:a3:00:65:93:  
          9a:1f:e1:3c:74:d2:20:c8:0f:df:5c:cf:f0:d5:f8:  
          d2:4a:f7:72:3c:da:eb:9b:25  
        Exponent: 65537 (0x10001)
```

```

X509v3 extensions:
  X509v3 Basic Constraints: critical
    CA:TRUE
  X509v3 Key Usage:
    Digital Signature, Non Repudiation, Key Encipherment, Certificate Sign, CRL Sign
  X509v3 Subject Key Identifier:
    79:A9:64:D9:99:51:CD:1E:4B:59:0A:A9:96:73:F8:47:19:7B:38:E7
  X509v3 Authority Key Identifier:
    keyid:79:A9:64:D9:99:51:CD:1E:4B:59:0A:A9:96:73:F8:47:19:7B:38:E7

    DirName:/C=IN/ST=New Delhi/O=SafeNet Infotech Pvt. Ltd./CN=Sanjay/emailAddress=skahuja@safenet-inc.com
    serial:80:90:8C:6D:55:71:CD:72

  X509v3 Subject Alternative Name:
    email:skahuja@safenet-inc.com
  X509v3 Issuer Alternative Name:
    email:skahuja@safenet-inc.com
  Netscape Cert Type:
    SSL CA, S/MIME CA, Object Signing CA
  Netscape Comment:
    This certificate was issued for testing only!
Signature Algorithm: sha1WithRSAEncryption
d8:af:a3:72:4e:73:05:ec:04:44:7c:a3:b3:9a:19:78:1d:ba:
89:f5:31:5b:39:08:e0:b4:69:f7:35:5b:c6:1a:d5:a1:fb:c9:
09:44:55:54:ff:a1:59:76:fb:e6:f3:f0:02:f7:ce:b0:ec:4d:
3c:d0:7b:99:76:94:7f:34:de:b2:a0:54:54:94:d4:60:5b:2d:
a8:95:f3:43:54:69:87:f8:8d:1e:69:51:ae:c6:ca:b4:2a:f4:
79:12:55:e5:ac:df:86:86:ac:5b:08:b2:78:64:bd:8c:f5:fe:
6f:d0:7a:31:33:61:17:8f:f1:a2:2e:4b:a2:dc:34:ac:35:72:
5f:c6
-----BEGIN CERTIFICATE-----
MIIEDjCCA3egAwIBAgIJAICQjG1Vcc1yMA0GCSqGSIb3DQEBBQUAMH8xCzAJBgNV
BAYTAKlOMRIwEAYDVQQIEw10ZXcgRGVsaGkxIzAhBgNVBAoTGlNhZmVOZXQgSW5m
b3RlY2ggUHZ0LiBMDGQuMQ8wDQYDVQQDEwZTYW5qYXkxJjAkBgkqhkiG9w0BCQEW
F3NrYWh1amFAC2FmZW5ldC1pbmMuY29tMB4XDTA2MTAxNzA1MTMyMl0XDTA3MTAx
NzA1MTMyMl0wZjzELMAkGA1UEBhMCSU4xEjAQBgNVBAgTCU5ldyBEZWxoatejMCEG
A1UEChMaU2FmZU5ldCBJbMzvdGVjaCBQdnQuIEx0ZC4xDzANBgNVBAMTB1Nhbmph
eTEEmMCQGCSqGSIb3DQEJARYXc2thaHVqYUBzYWZlbnV0LWluYy5jb2wgZ8wDQYJ
KoZlHvcNAQEBBQADgY0AMIGJAoGBAN/5Bx1bl0zp60CeZbEudR+iMe8BzeWhLvWh
E1NeCAPT8we+ePjQ4wJCIEBL9387WMSVIODbURb8xeCRuDg3wkBjK8E+JdyHjP9M
evGJhtxuXCK1HPWnUFHChL+GyO9MhMG8xl6jAGWTmh/hPHTSIMGP31zP8NX40kr3
cjza65slAgMBAAGjggQMIIBjDAPBgNVHRMBAf8EBTADAQH/MASGA1UdDwQEAwIB
5jAdBgNVHQ4EFgQealk2ZlRzR5LWQqplnP4Rxl700cwgbMGA1UdIwSBqzCBqIAU
ealk2ZlRzR5LWQqplnP4Rxl700ehgYSkgYEwfzELMAkGA1UEBhMCSU4xEjAQBgNV
BAGTCU5ldyBEZWxoatejMCEGA1UEChMaU2FmZU5ldCBJbMzvdGVjaCBQdnQuIEx0
ZC4xDzANBgNVBAMTB1NhbmphTeTEEmMCQGCSqGSIb3DQEJARYXc2thaHVqYUBzYWZl
bnV0LWluYy5jb22CCQCAkIxtVXHNcjAiBgNVHREEGzAZgRdza2FodWphQHNhZmVu
ZXQtaW5jLmNvbTAiBgNVHRIEGzAZgRdza2FodWphQHNhZmVuZXQtaW5jLmNvbTAR
BglghkgBhvhCAQEEBAMCAAcwPAYJYIZIAyb4QgENBC8WLVRoaxMgY2VydGlmawNh
dGUgd2FzIGlzc3VlZCBmb3IgdGVzdGluZyBvbmx5ITANBgkqhkiG9w0BAQUFAAOB
gQDYr6NyTnMF7AREfKozmhl4HbqJ9TFbOQjgtGn3NVvGGtWh+8kJRFVU/6FZdvvm
8/AC986w7E080HuZdpr/NN6yofRU1NRgWy2o1fNDVGmH+I0eaVGuxsq0KvR5E1X1
rN+GhqxbCLJ4ZL2M9f5v0HoxM2EXj/GiLkui3DSsNXJfxg==
-----END CERTIFICATE-----

```

Other options will provide more targeted sets of data.

```
# who issued the cert?
openssl x509 -noout -in mycert.pem -issuer

# to whom was it issued?
openssl x509 -noout -in mycert.pem -subject

# for what dates is it valid?
openssl x509 -noout -in mycert.pem -dates

# the above, all at once
openssl x509 -noout -in mycert.pem -issuer -subject -dates

# what is its hash value?
openssl x509 -noout -in mycert.pem -hash

# what is its MD5 fingerprint?
openssl x509 -noout -in mycert.pem -fingerprint
```

How to convert certificate from PEM to DER?

The certificate can be converted from PEM format to DER format by following command:

```
$ openssl x509 -inform PEM -outform DER -in mycert.pem -out mycert.der
```

where mycert.pem is certificate in PEM format and mycer.der is the name of the target certificate.

How to convert certificate from DER to PEM?

The certificate can be converted from PEM format to DER format by following command:

```
$ openssl x509 -inform DER -outform PEM -in mycert.der -out mycert.pem
```

where mycert.der is certificate in DER format and mycer.pem is the name of the target certificate.

How to Encrypt a text file?

In OpenSSL a text file can be encrypted by using rsautl utility. The command to encrypt a text file is as below:

```
$ openssl rsautl -pubin -inkey pubkey.pem -encrypt -in plaintext.txt -out ciphertext.txt
```

Where pubkey.pem is public key.
plaintext.txt is file for plain text to be encrypted.
ciphertext.txt is name of output cipher text file.

How to Decrypt a text file?

In OpenSSL a ciphertext file can be decrypted by using rsautl utility. The command to decrypt a ciphertext file is as below:

```
$ openssl rsautl -inkey mykey.pem -decrypt -in ciphertext.txt -out plaintext.txt
```

Where:

mykey.pem is private key,
ciphertext.txt is file for cipher text to be decrypted,
plaintext.txt is name of output plain text file.

How to sign a digest?

In OpenSSL a message can be signed using command:

```
$ openssl rsautl -inkey mykey.pem -sign -in digest.txt -out sign.txt
```

Where:

mykey.pem is private key,
digest.txt is digest to be signed,
sign.txt is output signed file.

Another way to sign a digest is as below:

```
$ openssl dgst -hex -sha1 -sign mykey.pem digest.txt
```

How to verify a signed digest?

In OpenSSL a message can be verified using command:

```
$ openssl rsautl -pubin -inkey pubkey.pem -verify -in sign.txt -out verify.txt
```

Where:

pubkey.pem is public key,
sign.txt is signed file,
verify.txt is output file.

How do I generate an RSA key?

RSA key can be generated using **genrsa** command as below:

```
# default 512-bit key, sent to standard output
openssl genrsa

# 1024-bit key, saved to file named mykey.pem
openssl genrsa -out mykey.pem 1024

# same as above, but encrypted with a passphrase
openssl genrsa -des3 -out mykey.pem 1024
```

How do I generate a public RSA key?

RSA public key can be generated from private key using rsa option as below:

```
$ openssl rsa -in mykey.pem -pubout
```

How to generate OCSP response from openvalidation.org site?

The OCSP responses can be generated from openvalidation.org site and by sending request from OpenSSL. We need to provide issuer certificate, user certificate for the same. The command is as below:

```
$ openssl ocsp -issuer RootCAcert.pem -cert User.pem -url
http://ocsp.openvalidation.org:80 -resp_text -respout resp.crt
```

Where 80 is port for normal valid response.

There are few other ports available for generating other types of response as mentioned below:

- Port: 80 Standard configuration. OCSP Responder will accept all proper requests and send a signed response.
- Port: 8080 Response does not contain any attached certificates. Client must accept this response
- Port: 8081 Never replies nonce. Insecure but standard conform mode. Client application should warn in case of replay-attacks
- Port: 8082 The OCSP Responder will sign the response with randomized bytecode. Client should NOT accept this response.
- Port: 8083 OCSP response will always be revoked.
- Port: 8084 OCSP response will always be unknown.
- Port: 8085 OCSP response will always be malformed.
- Port: 8086 OCSP response will always be internal error.
- Port: 8087 OCSP response will always be try later.
- Port: 8088 OCSP response will always be signature required.
- Port: 8089 OCSP response will always be unauth.
- Port: 8090 Standard configuration with full Debuglogs. Access the logs at » <http://www.openvalidation.org/en/test/logs.html>
- Port: 8091 Internal test responder. Configuration will change on demand.

How to verify a OCSP response from openvalidation.org site?

The OCSP responses can be verified from openvalidation.org site and by sending request from OpenSSL. We need to provide RootCA certificate, Server certificate and User certificate for the same. The command is as below:

```
$ openssl ocsf -url http://ocsp.openvalidation.org -issuer RootCAcert.pem -VAfile OCSPServer.pem -cert User.pem
```

How to decode OCSP response?

The command to decode OCSP response is as below:

```
$ openssl ocsf -respin ocsf.der -text
```

How to Install OpenSSL based local OCSP Server?

The OpenSSL-based OCSP server is started with the following command:

```
# openssl ocsf -index index.txt -CA cacert.pem -port 8880 \  
-rkey ocsfkey.pem -rsigner ocsfcert.pem \  
-resp_no_certs -nmin 60 -text
```

How to generate OCSP response from local server?

The command to generate OCSP response from local server is as below:

```
$ openssl ocsf -issuer RootCA/RootCAcert.pem -cert RootCA/newcerts/shalusercert.pem -VAfile RootCA/newcerts/ocspservercert.pem -url http://127.0.0.1:8080 -resp_text -respout shalresp.crt
```

The other OCSP options are as below:

-out file	output filename
-issuer file	issuer certificate
-cert file	certificate to check
-serial n	serial number to check
-signer file	certificate to sign OCSP request with
-signkey file	private key to sign OCSP request with
-sign_other file	additional certificates to include in signed request
-no_certs	don't include any certificates in signed request
-req_text	print text form of request
-resp_text	print text form of response
-text	print text form of request and response
-reqout file	write DER encoded OCSP request to "file"
-respout file	write DER encoded OCSP response to "file"
-reqin file	read DER encoded OCSP request from "file"
-respin file	read DER encoded OCSP response from "file"
-nonce	add OCSP nonce to request

```

-no_nonce          don't add OCSP nonce to request
-url URL          OCSP responder URL
-host host:n      send OCSP request to host on port n
-path            path to use in OCSP request
-CApath dir      trusted certificates directory
-CAfile file     trusted certificates file
-VAfile file     validator certificates file
-validity_period n maximum validity discrepancy in seconds
-status_age n    maximum status age in seconds
-noverify        don't verify response at all
-verify_other file additional certificates to search for signer
-trust_other     don't verify additional certificates
-no_intern      don't search certificates contained in response for signer
-no_signature_verify don't check signature on response
-no_cert_verify don't check signing certificate
-no_chain       don't chain verify response
-no_cert_checks don't do additional checks on signing certificate
-port num       port to run responder on
-index file     certificate status index file
-CA file       CA certificate
-rsigner file   responder certificate to sign responses with
-rkey file     responder key to sign responses with
-rother file   other certificates to include in response
-resp_no_certs don't include any certificates in response
-nmin n       number of minutes before next update
-ndays n      number of days before next update
-resp_key_id   identify response by signing certificate key ID
-nrequest n    number of requests to accept (default unlimited)

```

How to verify OCSP responses from local server?

The command to verify OCSP response from local server is as below:

```
$ openssl ocsp -url http://127.0.0.1:8080 -issuer RootCA/RootCAcert.pem -VAfile
RootCA/newcerts/ocspservercert.pem -cert RootCA/newcerts/shalusercert.pem
```

How to generate Abnormal OCSP response?

To generate abnormal response for e.g. Revoked etc we need to generate **crl** by using following command:

```
$ openssl ca -gencrl -out crl.pem
```

The certificate can be revoked by using **crl** command. The options for **crl** command are as below:

```
$ openssl ca -revoke host.pem
```

where **host.pem** is the host certificate to be revoked.

Next the CRL file must be updated by following command:

```
$ openssl crl -updatedb
```

The content of the CRL file can be listed with the command:

```
$ openssl crl -in crl.pem -noout -text
```

Now you can follow the above mentioned steps to generate corresponding OCSP responses.

How to extract public key from a key pair?