Simple Introduction to using OpenSSL on Command Line

By Steven Gordon on Wed, 31/07/2013 - 1:36pm

OpenSSL is a program and library that supports many different cryptographic operations, including:

- Symmetric key encryption
- Public/private key pair generation
- Public key encryption
- Hash functions
- Certificate creation
- Digital signatures
- Random number generation

Each of the operations supported by OpenSSL have a variety of options, such as input/output files, algorithms, algorithm parameters and formats. This article aims to give a demonstration of some simple and common operations.

To start learning the details of OpenSSL, read the man page, i.e. `man openssl`. You'll soon learn that each of the operations (or commands) have their own man pages. For example, the operation of symmetric key encryption is `enc`, which is described in `man enc`. Although it is good to read the man pages, in my (and others) experience, the man pages of OpenSSL can be very detailed, hard to follow, confusing and out of date. So hopefully this article will make life easier for those getting started.

There are other [3] websites [4] that give an overview of OpenSSL operations, as well as programming [5] with the API. I used some of them to write the following notes. Check them out for more details.

1. Initial Steps

Lets first determine the current versions of Ubuntu, Linux and OpenSSL I am using:

```
$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 12.04.2 LTS
Release:        12.04
Codename:       precise
$ uname -a
Linux lime 3.2.0-51-generic #77-Ubuntu SMP Wed Jul 24 20:18:19 UTC 2013 x86_64 x86_64
$ openssl version
OpenSSL 1.0.1 14 Mar 2012
```

If you are using different versions, then it is still a very good chance that all the following commands will work. In the past I have had problems [6] with different versions of OpenSSL but for only for very specific operations.

Update (2013-08-02): I just tested on a Apple iMac using OS X 10.8.4 and OpenSSL version 0.9.8x 10 May 2012. Most operations worked. See my additional comments at the end of this article if you are using a similar version of OpenSSL.

As input plaintext I will copy some files on Ubuntu Linux into my home directory. You don't need to do this if you already have some files to encrypt. It doesn't matter what files you use. I have chosen the following three, and will rename them simply to `plaintext1.in`, `plaintext2.in`, `plaintext3.in`:

1. `/usr/share/dict/words` - a large text file containing a list of words, i.e. a dictionary
2. `/usr/bin/openssl` - the binary for the program OpenSSL
3. `/etc/legal` - a short text file containing the Ubuntu legal notice

```
$ cp /usr/share/dict/words plaintext1.in
$ cp /usr/bin/openssl plaintext2.in
$ cp /etc/legal plaintext3.in
$ ls -l plaintext*
-rw-r--r-- 1 sgordon sgordon 938848 Jul 31 13:32 plaintext1.in
-rwxr-xr-x 1 sgordon sgordon 513208 Jul 31 13:32 plaintext2.in
-rw-r--r-- 1 sgordon sgordon    267 Jul 31 13:32 plaintext3.in
```

2. Symmetric Key Encryption

The most common cryptographic operation is encryption. Lets encrypt some files using selected symmetric key (conventional) ciphers such as DES, 3DES and AES.

Symmetric key encryption is performed using the `enc` operation of OpenSSL. To encrypt we need to choose a cipher. A list of supported ciphers can be found using:

```
$ openssl list-cipher-algorithms
AES-128-CBC
AES-128-CBC-HMAC-SHA1
AES-128-CFB
AES-128-CFB1
AES-128-CFB8
...
seed => SEED-CBC
SEED-CBC
SEED-CFB
SEED-ECB
SEED-OFB
```

The lowercase `seed` is an alias for the actual cipher `SEED-CBC`, i.e. SEED using CBC mode of operation. You can use the cipher names in either lowercase or uppercase.

Now lets encrypt using DES and ECB, creating an output file `ciphertext1.bin`. Enter a password when prompted - OpenSSL will automatically convert it to a key appropriate for DES:

```
$ openssl enc -des-ecb -in plaintext1.in -out ciphertext1.bin
enter des-ecb encryption password: password
Verifying - enter des-ecb encryption password: password
$ ls -l plaintext1.in ciphertext1.bin
-rw-rw-r-- 1 sgordon sgordon 938872 Jul 31 14:15 ciphertext1.bin
-rw-r--r-- 1 sgordon sgordon 938848 Jul 31 13:32 plaintext1.in
```

To decrypt, include the `-d` option:

```
$ openssl enc -d -des-ecb -in ciphertext1.bin -out plaintext1.out
enter des-ecb decryption password: password
$ ls -l plaintext1.in plaintext1.out
-rw-r--r-- 1 sgordon sgordon 938848 Jul 31 13:32 plaintext1.in
-rw-rw-r-- 1 sgordon sgordon 938848 Jul 31 14:18 plaintext1.out
$ diff plaintext1.in plaintext1.out
$ xxd -l 96 ciphertext1.bin
0000000: 5361 6c74 6564 5f5f f253 8361 b87d 1a3e  Salted__.S.a.}.>
0000010: 30ed be95 5b38 ebf9 a013 ca64 bbf4 03ea  0...[8.....d....
0000020: 3ebb cdf8 483d 5a12 acd8 bc75 140c 920b  >...H=Z....u....
0000030: da41 7376 edc3 b9bd 59c4 a5ce 0a67 408a  .Asv....Y....g@.
0000040: d23e 10ee 7ac3 f5b6 4f09 4aaf 88e4 1f96  .>..z...O.J.....
0000050: 3171 7277 91a7 100c ac04 7871 dd39 cf4c  1qrw......xq.9.L
```

The lack of output from the `diff` indicates the files `plaintext1.in` and `plaintext1.out` are identical. We've retrieved the original plaintext.

`xxd` was used to view the first 96 bytes, in hexadecimal, of the ciphertext. The first 8 bytes contain the special string `Salted__` meaning the DES key was generated using a password and a salt. The salt is stored in the next 8 bytes of ciphertext, i.e. the value `f2538361b87d1a3e` in hexadecimal. So when decrypting, the user supplies the password and OpenSSL combines with the salt to determine the DES 64 bit key.

Lets try an example where we select a key. I will use AES with a 128 bit key and Counter (CTR) mode of operation. In addition to the key, an initialisation vector (IV) is needed.

```
$ openssl enc -aes-128-ctr -in plaintext2.in -out ciphertext2.bin -K 0123456789abcdef
$ openssl enc -d -aes-128-ctr -in ciphertext2.bin -out plaintext2.out -K 0123456789ab
$ ls -l *2*
-rw-rw-r-- 1 sgordon sgordon 513208 Jul 31 14:29 ciphertext2.bin
-rwxr-xr-x 1 sgordon sgordon 513208 Jul 31 13:32 plaintext2.in
-rw-rw-r-- 1 sgordon sgordon 513208 Jul 31 14:30 plaintext2.out
$ diff plaintext2.in plaintext2.out
$ xxd -l 96 ciphertext2.bin
0000000: 06ee 8984 3a69 ac84 d388 ce61 110a 6274  ....:i.....a..bt
0000010: c1ed f9ed f193 f2d2 bf8d 29e2 1577 5d32  ..........)..w]2
0000020: 1e25 cc36 bb37 baa7 eb65 402b a8ef 421b  .%.6.7...e@+..B.
0000030: a6f7 073c a08a e698 747d 5153 8df1 ed88  ...<....t}QS....
0000040: 1131 f4e0 2014 1392 ee36 2b54 27eb ca72  .1.. ....6+T'..r
0000050: 4b88 e623 ed28 2da7 87cd 0c1a 5441 5d7c  K..#.(-.....TA]|
```

Both the Key (not uppercase `-K`) and IV were specified on the command line as a hexadecimal string. With AES-128, they must be 32 hex digits (128 bits). You may choose any value you wish.

3. Public Key Encryption, Certificates and Digital Signatures

I have written several guides that introduce topics related to public key cryptography, including:

- Setting up a Certificate Authority and generating RSA certificates [7]
- Generating RSA key pairs and using for encryption and digital signatures [8]
- Performing a secret key exchange using Diffie-Hellman [9]

4. Hash Functions

Hash functions (like MD5 and SHA) as well as MAC functions (e.g. using HMAC) are available via the message digest (`dgst`) operating of OpenSSL. To list the available algorithms:

```
$ openssl list-message-digest-algorithms
DSA
DSA-SHA
DSA-SHA1 => DSA
DSA-SHA1-old => DSA-SHA1
DSS1 => DSA-SHA1
MD4
MD5
...
ssl3-md5 => MD5
ssl3-sha1 => SHA1
whirlpool
```

Calculate the MD5 hash of a file:

```
$ openssl dgst -md5 plaintext3.in
MD5(plaintext3.in)= 0110925f6e068836ef2e09356e3651d9
```

Now create a new file, slightly different from the previous and see that the MD5 hash is significantly different:

```
$ cat plaintext3.in

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

$ sed 's/U/X/g' plaintext3.in > plaintext4.in
$ cat plaintext4.in

The programs included with the Xbuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Xbuntu comes with ABSOLXTELY NO WARRANTY, to the extent permitted by
applicable law.
```

```
$ openssl dgst -md5 plaintext4.in
MD5(plaintext4.in)= 0b4974e95714c429e40cfad510286827
```

Use SHA-256, first outputing to the terminal and then in binary to a file:

```
$ openssl dgst -sha256 plaintext3.in
SHA256(plaintext3.in)= 9fa4ad4d7c2a346540c64c4c3619e389db894116f99a0fbbcc75a58bf28512
$ openssl dgst -sha256 -binary -out dgst3.bin plaintext3.in
$ xxd dgst3.bin
0000000: 9fa4 ad4d 7c2a 3465 40c6 4c4c 3619 e389  ...M|*4e@.LL6...
0000010: db89 4116 f99a 0fbb cc75 a58b f285 1262  ..A......u.....b
```

Create a MAC using HMAC and MD5. First generate a random 128 bit key (see Random Number below for further explanation), then pass the key as an option when using HMAC:

```
$ openssl rand 32 -hex
36463a4eb02b5ab9776aa8ed51f4e8a34f4bd785597fd74d4277652fd9f743d5
$ openssl dgst -md5 -mac hmac -macopt hexkey:36463a4eb02b5ab9776aa8ed51f4e8a34f4bd785
HMAC-MD5(plaintext3.in)= 85e0bbf0a14559699c4b8e04bd1c1665
```

A much simpler alternative to calculate hash values is to use the Linux programs `md5sum` and `sha1sum` (and its variants `sha224sum`, `sha256sum` and so on). For example:

```
$ sha256sum plaintext3.in
9fa4ad4d7c2a346540c64c4c3619e389db894116f99a0fbbcc75a58bf2851262  plaintext3.in
```

5. Random Numbers

The `rand` operation of OpenSSL can be used to produce random numbers, either printed on the screen or stored in a file. Some quick examples:

Write 8 random bytes to a file (then view that file with `xxd` in both hexadecimal and binary):

```
$ openssl rand 8 -out rand1.bin
$ ls -l rand1.bin
-rw-rw-r-- 1 sgordon sgordon 8 Jul 31 15:14 rand1.bin
$ xxd rand1.bin
0000000: 7d12 162f 1a18 c331                      }../...1
$ xxd -b -g 8 -c 8 rand1.bin  | cut -d " " -f 2
0111110100010010000101100010111100011010000110001100001100110001
```

Generate a 128 bit (16 byte) random value, shown in hexadecimal:

```
$ openssl rand 16 -hex
04d6b077d60e323711b37813b3a68a71
```

Another way to generate random values on Linux (without using OpenSSL) is using `urandom`:

```
$ cat /dev/urandom | xxd -l 32
```

```
0000000: 4b69 76ca f6ee 4663 e467 f767 d560 07cd  Kiv...Fc.g.g.`..
0000010: dac6 5020 62ac 02db ea2e 6f0a 60ba 5031  ..P b.....o.`.P1
```

Read `man rand` and `man urandom` for further details.

6. OpenSSL 0.9.8x on Mac OS X

Running the above commands on Mac OS X 10.8.4 which uses OpenSSL 0.9.8x produces correct results, except for the following:

- The OpenSSL `list-` operations do not work, e.g. `list-cipher-algorithms` and `list-message-digest-algorithms`. But its not a problem because in fact if you give an invalid option with OpenSSL it prints an error followed by the algorithms that are supported.
- Counter (CTR) mode is not supported. So I replaced *`aes-128-ctr`* with *`aes-128-cfb`* (or you can choose from any of the supported modes of operation).
- The command line options for performing a HMAC are different. Instead of *`-mac hmac -macopt hexkey:KEY`* use *`-hmac KEY`*.
- Although not an issue with OpenSSL, the Linux programs `md5sum` and `sha256sum` are not supported on Mac OS X. Instead you can use `md5` and `shasum -a`.

If you have an older version of OpenSSL (pre 1.0) - no matter what operating system - then you may try the above commands instead.

Interest: Ubuntu Linux [10]
/dev/urandom [11]
cut [12]
lsb_release [13]
OpenSSL [14]
sed [15]
uname [16]
xxd [17]
Topic: Security [18]
Content: Howto [19]

Source URL: http://sandilands.info/sgordon/simple-introduction-to-using-openssl-on-command-line

Links:
[1] http://sandilands.info/sgordon/simple-introduction-to-using-openssl-on-command-line
[2] http://sandilands.info/sgordon/user/2
[3] http://www.madboa.com/geek/openssl/
[4] https://help.ubuntu.com/community/OpenSSL
[5] http://www.ibm.com/developerworks/linux/library/l-openssl/index.html
[6] http://sandilands.info/sgordon/upgrade-latest-version-openssl-on-ubuntu
[7] http://sandilands.info/sgordon/key-generation-and-encryption-examples-using-openssl
[8] http://sandilands.info/sgordon/public-key-encryption-and-digital-signatures-using-openssl
[9] http://sandilands.info/sgordon/diffie-hellman-secret-key-exchange-with-openssl
[10] http://sandilands.info/sgordon/taxonomy/term/302
[11] http://sandilands.info/sgordon/taxonomy/term/359
[12] http://sandilands.info/sgordon/taxonomy/term/355
[13] http://sandilands.info/sgordon/taxonomy/term/357
[14] http://sandilands.info/sgordon/taxonomy/term/338
[15] http://sandilands.info/sgordon/taxonomy/term/360

[16] http://sandilands.info/sgordon/taxonomy/term/358
[17] http://sandilands.info/sgordon/taxonomy/term/356
[18] http://sandilands.info/sgordon/taxonomy/term/116
[19] http://sandilands.info/sgordon/taxonomy/term/212