Symmetric Encryption with OpenSSL

Giuseppe Persiano

Università di Salerno

November, 2020

Encrypting with OpenSSL

`openssl enc`

Parameters:

- `-in`: input file
- `-out`: output file
- specify the algorithm
- specify the key

Encrypting with AES

openssl enc: specifying the algorithm

- AES, Blowfish, Camellia, Chacha20, SEED, CAST-128, DES, IDEA, RC5, Triple DES
- AES:
  - 128 bit keys:
    AES-128-CBC, AES-128-CBC-HMAC-SHA1,
    AES-128-CBC-HMAC-SHA256, id-aes128-CCM, id-aes128-GCM,
    AES-128-CFB, AES-128-CFB1, AES-128-CFB8, AES-128-CTR,
    AES-128-ECB, AES-128-OCB, AES-128-OFB, AES-128-XTS
  - 256-bit keys:
    AES-256-CBC, AES-256-CBC-HMAC-SHA1,
    AES-256-CBC-HMAC-SHA256, id-aes256-CCM, AES-256-CFB,
    AES-256-CFB1, AES-256-CFB8, AES-256-CTR, AES-256-ECB,
    id-aes256-GCM, AES-256-OCB, AES-256-OFB, AES-256-XTS
  - shortcuts:
    aes128 $\Rightarrow$ AES-128-CBC, aes128-wrap $\Rightarrow$ id-aes128-wrap,
    aes192 $\Rightarrow$ AES-192-CBC, aes192-wrap $\Rightarrow$ id-aes192-wrap,
    aes256 $\Rightarrow$ AES-256-CBC, aes256-wrap $\Rightarrow$ id-aes256-wrap

Encrypting with AES

openssl enc: using a password

- openssl derives a key from a *password* + *salt* using a key derivation function KDF
- OpenSSL 1.1.1f, PBKDF1 (with $c = 1$) and SHA256
  - `KEY=SHA256(Password||Salt)`
  - `IV=SHA256(KEY||Password||Salt)`

```
Encrypting a file using AES cbc with 256-bit key
        Key generated from password and salt
        Using PBKDF1 (deprecated)
Encryption command: openssl enc -aes-256-cbc -salt -in example.txt -out example1.cpt

enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=8247A8168AA852FA
key=C846F0E378C120516288FAFEB097C28D02CE5D08C1AF6C27DA37972FF20BF6CA
iv =45D27ADDCA8926BE4A1F4806A67BBCD9

Decryption command: openssl enc -aes-256-cbc -salt -p -d -in example1.cpt -out example.txt.new

enter aes-256-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
salt=8247A8168AA852FA
key=C846F0E378C120516288FAFEB097C28D02CE5D08C1AF6C27DA37972FF20BF6CA
iv =45D27ADDCA8926BE4A1F4806A67BBCD9
```

Encrypting with AES

openssl enc: using a password

- We can specify `-pbkdf2` or `-iter` followed by the number of iterations that will force the use of PBKDF2

```
Encrypting a file using AES cbc with 256-bit key
        Key generated from password and salt
        Using PBKDF2
Encryption command: openssl enc -aes-256-cbc -salt -iter 10000 -in example.txt -out example2.cpt

enter aes-256-cbc encryption password:
Verifying - enter aes-256-cbc encryption password:
salt=939E3E229CD9763D
key=B0393BE15BECA5E86D1E42525B25CCA938CD14A94F833ACDA6469D4BD44D5898
iv =4159F394563EDC8540820E8E79E007D4

Decryption command: openssl enc -aes-256-cbc -salt -p -d -iter 10000 -in example2.cpt -out example.t
xt.new

enter aes-256-cbc decryption password:
salt=939E3E229CD9763D
key=B0393BE15BECA5E86D1E42525B25CCA938CD14A94F833ACDA6469D4BD44D5898
iv =4159F394563EDC8540820E8E79E007D4
```

Salted File Format

```
0000000   S   a   l   t   e   d   _   _ stx   G   ( syn  nl   (   R   z
           6153    746c    6465    5f5f    4782    16a8    a88a    fa52
0000020   C   m   , etx   `   O  rs   H   !   G dc1   P   <   i   R   4
           6dc3    832c    cf60    c81e    c7a1    5091    e93c    b452
```

PBKDF1

- uses a Hash function (SHA-256, by default)
- inputs:
  - P: password
  - S: salt, 8-byte string
  - c: iteration count ≥ 1
  - `dkLen`: length of the derived key
- algorithm:
  - $T_1 = \text{Hash}(P||S)$
  - $T_2 = \text{Hash}(T_1)$
  - .....
  - $T_c = \text{Hash}(T_{c-1})$
  - Output first `dkLen` byte of T_c

`openssl` uses $c = 1$ with SHA-256 as a default

PBKDF2

- $\mathtt{DK} = \mathtt{PBKDF2}(\mathtt{PRF}, \mathtt{Password}, \mathtt{Salt}, \mathtt{c}, \mathtt{dkLen})$
- $\mathtt{DK} = \mathtt{T1}||\mathtt{T2}||\ldots||\mathtt{Tdklen/hlen}$
  - $\mathtt{Ti} = \mathtt{F}(\mathtt{Password}, \mathtt{Salt}, \mathtt{c}, \mathtt{i})$
  - $\mathtt{F}(\mathtt{Password}, \mathtt{Salt}, \mathtt{c}, \mathtt{i}) = \mathtt{U}_1 \oplus \mathtt{U}_2 \oplus \cdots \oplus \mathtt{U}_\mathtt{c}$
  - $\mathtt{U}_1 = \mathtt{PRF}(\mathtt{Password}, \mathtt{Salt}||\mathtt{i})$
  - $\mathtt{U}_2 = \mathtt{PRF}(\mathtt{Password}, \mathtt{U}_1)$
  - ……
  - $\mathtt{U}_\mathtt{c} = \mathtt{PRF}(\mathtt{Password}, \mathtt{U}_{\mathtt{c}-1})$

The PRF used is HMAC-SHA-256

Encrypting with AES

openssl enc: specifying a password and the IV

- use `-K` followed by the key in hexadecimal
- use `-iv` followed by the IV in hexadecimal

```
Encrypting a file using AES cbc with 256-bit key
        Key specified in the command line
Encryption command: openssl enc -aes-256-cbc -K 000102030405060708090A0B0C0D0E0F10111213141516171819
1A1B1C1D1E1F -iv AABBCCDDEEFF00112233445566778899 -in example.txt -out example3.cpt


Decryption command: openssl enc -d -aes-256-cbc -K 000102030405060708090A0B0C0D0E0F10111213141516171
8191A1B1C1D1E1F -iv AABBCCDDEEFF00112233445566778899 -in example3.cpt -out example.txt.new
```

Encrypting with AES

openssl enc: randomly generating a password and the IV

- use command `openssl rand -hex 32` to generate the 32 bytes needed for an AES-256 key
- use command `openssl rand -hex 16` to generate the 16 bytes needed for an AES-256 IV