OpenSSL

> 本文对密码学中常用的底层库OpenSSL进行简单说明，包括基本介绍、简单使用等内容。

简单介绍

OpenSSL 是密码学中一个流行的底层密码库，也是SSL和TLS最常见的开源实现。

> ! SSL 全称为 Secure Sockets Layer ，**安全套接层协议**(对网络连接进行加密处理)。
> TLS 全称为 Transport Layer Security ，**安全传输层协议**(由记录协议和握手协议组成)。

SSL协议由 Netscape (网景)公司开发，因为应用广泛，到了1999年SSL已经成为互联网上的事实标准。同年， IETF把SSL标准化后改名为TLS，它建立在SSL 3.0协议规范上，是SSL的后续版本 。 TLS 和 SSL，两者差别极小，可以简单看做是同一个东西的两个不同阶段。

OpenSSL 是由一些志愿者合作开发的开源工具库，其目标是开发一个强壮的、具有完备功能的商业级工具集，以实现SSL 和 TLS协议以及一个全功能的通用加密库。OpenSSL主要以C语言实现，目前最新稳定版本为2018年9月11日发布的 1.1.1版本 。

OpenSSL 的主要组成

- ❏ **openssl** 多用途的命令行工具。
- ❏ **libcrypto** 常用加密算法库。
- ❏ **libssl** 实现了SSL及TLS的加密模块应用库。

OpenSSL 支持许多不同的加密算法：其中包括但不限于**单向散列函数**的 MD5、MD4、MD2、SHA-1、SHA-2、SHA-3、RIPEMD-160、MDC-2、GOST R 34.11-94、BLAKE2、Whirlpool、SM3等算法，**对称加密**有 AES、Blowfish、Camellia、Chacha20、Poly1305、SEED、CAST-128、DES、IDEA、RC2、RC4、RC5、SM4、TDES、GOST 28147-89等算法，以及**非对称加密**的 RSA、DSA、迪菲-赫尔曼密钥交换、椭圆曲线、SM2、X25519、Ed25519、X448、Ed448、GOST R 34.10-2001等算法。

参考资料

OpenSSL官网
OpenSSL 维基百科
OpenSSL GitHub组织
OpenSSL Command-Line HOWTO
OpenSSL 版本下载地址① OpenSSL 版本下载地址②

opsnssl 命令行简单说明

OpenSSL 的安装

通常，很多操作系统都会默认安装OpenSSL 工具，我们可以通过在终端中输入 `openssl version` 命令来查看和进行验证。

> ! 早些年，Mac OSX默认自带安装 OpenSSL库，但是因为OpenSSL“心脏病失血”事件，Mac OSX 自 10.11 El Capitan 起，原本默认安装的 OpenSSL 被替换为 LibreSSL 。除Mac外，其它使用LibreSSL 取代 OpenSSL的系统还有 **OpenBSD** (`5.6 +`)和 **Alpine Linux** (`3.5.0 +`) 等。

我当前使用的是Mac OSX 系统10.13.3 版本，在终端中输入 `openssl version` 命令 可以看到本地显示的是 **LibreSSL** 的版本，为 `LibreSSL 2.2.7` 。**LibreSSL** 是 OpenBSD 开发者在 OpenSSL 爆出心脏出血漏洞之后 fork 的一个分支，旨在提供一个比 OpenSSL 更安全的替代品，更详细的信息可以参考扔掉 OpenSSL，拥抱 LibreSSL。

```
wendingding$ openssl version
LibreSSL 2.2.7
wendingding$ which openssl
/usr/bin/openssl
```

说明 上面的命令用于检查和验证当前系统中使用的openssl的版本和路径(位置)。**LibreSSL**可以理解为更安全的**OpenSSL**，使用方式几乎一致，接下来我就以电脑中已经安装的 `LibreSSL 2.2.7` 版本为例进行说明。在终端查看(输入任意无效命令)该工具支持的所有命令，如下：

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

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

Message Digest commands (see the `dgst' command for more details)
gost-mac          md4               md5               md_gost94
ripemd160         sha               sha1              sha224
sha256            sha384            sha512            streebog256
streebog512       whirlpool

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
camellia-128-cbc  camellia-128-ecb  camellia-192-cbc  camellia-192-ecb
camellia-256-cbc  camellia-256-ecb  cast              cast-cbc
cast5-cbc         cast5-cfb         cast5-ecb         cast5-ofb
chacha            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
```

通过上面列出的庞大的可选命令项，不难看出OpenSSL工具的强大和庞大，如果需要获取指定算法的帮助信息，只需要在终端输入 `openssl 算法名称 --help` 形式的命令即可。

```
wendingding$ openssl enc -help
usage: enc -ciphername [-AadePp] [-base64] [-bufsize number] [-debug]
    [-engine id] [-in file] [-iv IV] [-K key] [-k password]
    [-kfile file] [-md digest] [-none] [-nopad] [-nosalt]
    [-out file] [-pass arg] [-S salt] [-salt]

 -A                Process base64 data on one line (requires -a)
 -a                Perform base64 encoding/decoding (alias -base64)
 -bufsize size     Specify the buffer size to use for I/O
 -d                Decrypt the input data
 -debug            Print debugging information
 -e                Encrypt the input data (default)
 -engine id        Use the engine specified by the given identifier
 -in file          Input file to read from (default stdin)
 -iv IV            IV to use, specified as a hexidecimal string
 -K key            Key to use, specified as a hexidecimal string
 -md digest        Digest to use to create a key from the passphrase
 -none             Use NULL cipher (no encryption or decryption)
 -nopad            Disable standard block padding
 -out file         Output file to write to (default stdout)
 -P                Print out the salt, key and IV used, then exit
                     (no encryption or decryption is performed)
 -p                Print out the salt, key and IV used
 -pass source      Password source
 -S salt           Salt to use, specified as a hexidecimal string
 -salt             Use a salt in the key derivation routines (default)
 -v                Verbose

Valid ciphername values:

 -aes-128-cbc              -aes-128-cfb              -aes-128-cfb1
 -aes-128-cfb8             -aes-128-ctr              -aes-128-ecb
 -aes-128-gcm              -aes-128-ofb              -aes-128-xts
 -aes-192-cbc              -aes-192-cfb              -aes-192-cfb1
 -aes-192-cfb8             -aes-192-ctr              -aes-192-ecb
 -aes-192-gcm              -aes-192-ofb              -aes-256-cbc
 -aes-256-cfb              -aes-256-cfb1             -aes-256-cfb8
 -aes-256-ctr              -aes-256-ecb              -aes-256-gcm
 -aes-256-ofb              -aes-256-xts              -aes128
 -aes192                   -aes256                   -bf
 -bf-cbc                   -bf-cfb                   -bf-ecb
```

```
-bf-ofb                 -blowfish               -camellia-128-cbc
-camellia-128-cfb       -camellia-128-cfb1      -camellia-128-cfb8
-camellia-128-ecb       -camellia-128-ofb       -camellia-192-cbc
-camellia-192-cfb       -camellia-192-cfb1      -camellia-192-cfb8
-camellia-192-ecb       -camellia-192-ofb       -camellia-256-cbc
-camellia-256-cfb       -camellia-256-cfb1      -camellia-256-cfb8
-camellia-256-ecb       -camellia-256-ofb       -camellia128
-camellia192            -camellia256            -cast
-cast-cbc               -cast5-cbc              -cast5-cfb
-cast5-ecb              -cast5-ofb              -chacha
-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-cfb1          -des-ede3-cfb8
-des-ede3-ofb           -des-ofb                -des3
-desx                   -desx-cbc               -gost89
-gost89-cnt             -gost89-ecb             -id-aes128-GCM
-id-aes192-GCM          -id-aes256-GCM          -rc2
-rc2-40-cbc             -rc2-64-cbc             -rc2-cbc
-rc2-cfb                -rc2-ecb                -rc2-ofb
-rc4
```

opsnssl + 单向散列函数应用

单向散列函数的特点

> **① 散列计算后的密文是定长的。**
> **② 明文相同，密文一定相同。**
> **③ 明文不同，密文一定不同。**
> **④ 计算过程不可逆，算法公开，效率高性能好。**

经典加密算法 MD5加密 、SHA1 和 SHA512等
散列函数进阶

HTML

```
1）先加盐，然后再进行MD5
2）先乱序，再进行MD5加密
3）乱序 + 加盐，多次MD5加密等
4）使用消息认证机制，即HMAC-MD5-先对密钥进行加密，加密之后进行两次MD5散列
```

001 对字符串进行简单的MD5加密

```
wendingding$ echo -n "wendingding" | md5
d661517da45e21c9d180ad50ffcdf18d
```

002 对任意的文件进行MD5加密

```
wendingding$ ls -a
.               .DS_Store       All             Node            Vue
```

```
..              123.png         Canvas          Other
wendingding$ md5 123.png
MD5 (123.png) = 2feeda7a43bc96b94fa0fbe64673a593
```

003 使用sha家族算法来对字符串进行加密

```
wendingding$ echo -n "wendingding" | openssl sha -sha1
(stdin)= 13fb6a32c9876c0b4aa05c5e930ff9332f84c62d

wendingding$ echo -n "wendingding" | openssl sha -sha256
(stdin)= a3c02021d3630ec36fd07c4f14236cac691f499bc47428a245e8618319d812df

wendingding$ echo -n "wendingding" | openssl sha -sha512
(stdin)= 9b4c77e0e1a99b99fc46bfa58dab5070d707b84384a8659c69b19442d1d19490cf63b145b645c7
a7fe42ffba9a3bff800911f5957bc6a0a219b708c6c2dc9c2c
```

004 对字符串应用hmacMD5加密

```
wendingding$ echo -n "wendingding" |openssl dgst -md5 -hmac "123"
(stdin)= 82fce83fe44d72969f69565f3b16d3e9
```

> ! 消息认证机制（HMAC）原理是：消息的发送者和接收者有一个共享密钥，发送者使用共享密钥对消息加密计算得到MAC值，消息接收者使用共享密钥对消息加密计算得到MAC值，比较两个MAC值是否一致。在具体使用的时候，客户端需要在发送的时候把 （消息）+（消息·HMAC） 打包发送给服务器，服务器接收到数据后，对拿到的消息用共享的KEY进行HMAC，比较是否一致，如果一致则信任。

opsnssl + 对称加密算法

对称加密算法的特点

> **① 加密/解密使用相同的密钥。**
> **② 加密和解密的过程是可逆的。**
> **③ 效率高、性能好，但是存在密钥传输安全问题。**

经典加密算法 DES、3DES 和 AES 等

001 AES-ECB加密的过程

```
wendingding$ mkdir openssl
wendingding$ cd openssl/
wendingding$ echo "测试的字符串-des加密" >> 123.txt
wendingding$ cat 123.txt
测试的字符串-des加密
wendingding$ ls -a
```

```
.       ..      123.txt
wendingding$ openssl enc -des-ecb -K 616263 -nosalt -in 123.txt -out 123.bin
wendingding:openssl wendingding$ ls -a
.       ..      123.bin 123.txt
```

002 AES-ECB解密的过程

```
wendingding$ openssl enc -des-ecb -K 616263 -nosalt -in 123.bin -out 123_new.txt -d
wendingding$ cat 123_new.txt
测试的字符串-des加密
```

> 密码算法可以分为分组密码和流密码两种。

分组密码 的特点是在解密和解密时，每次只能处理特定长度的一组数据，一个分组的比特数量就称之为分组长度。DES 和 3DES 的分组长度都是64比特，也就是每次只能加密64比特的明文，并生成64比特的密文。AES 的分组长度有128比特、192比特和256比特可以选择。

流密码 的特点是加密和解密时会对数据流进行连续处理。流密码中一般以1比特、8比特或者是32比特等作为单位俩进行加密和解密。

> **备注** 分组密码算法在具体实现的时候，又有很多的分组模式可以选择。常见的分组模式有 ECB 和 CBC 等。ECB模式的全称为 Electronic CodeBook 模式，CBC模式全称为 Cipher Block Chainning 模式，它的特点是分组后先将明文分组与前一个密文分组进行XOR运算，然后再进行加密。关于CBC和ECB的具体细节可以参考数据安全系列 术语这篇文章。

003 AES-CBC加密的过程

```
$ echo "测试aes-cbc" >> a.txt
$ ls -a
.               123.bin         123_new.txt
..              123.txt         a.txt
$ openssl enc -des-cbc -K 616263 -iv 0102030405060708 -nosalt -in a.txt -out a.bin
$ ls -a
.               123.bin         123_new.txt     a.txt
..              123.txt         a.bin
```

004 AES-CBC解密的过程

```
$ openssl enc -des-cbc -K 616263 -iv 0102030405060708 -nosalt -in a.bin -out a_new.txt -d
$ ls -a
.               123.bin         123_new.txt     a.txt
..              123.txt         a.bin           a_new.txt
$ cat a_new.txt
测试aes-cbc
```

说明 上文命令行中的 `-iv 0102030405060708` 是CBC分组模式需要使用到的初始向量值。

opsnssl + 非对称加密算法

非对称加密算法的特点

> **① 加密的时候使用公钥，解密的时候使用私钥。**
> **② 公钥是公开的，私钥是绝对保密的。**
> **③ 效率不高，性能不好，可能会遭遇中间人攻击。**

经典加密算法 RSA

这里简单介绍RSA算法的原理和小示例

```
[1] RSA 原理
      (1) 求N, 准备两个质数p和q,N = p x q
      (2) 求L,L是p-1和q-1的最小公倍数。L = lcm (p-1,q-1)
      (3) 求E, E和L的最大公约数为1 (E和L互质)
      (4) 求D, E x D mode L = 1

[2] RSA加密小实践
      (1) p = 17,q = 19 =>N = 323
      (2) lcm (p-1,q-1) =>lcm (16, 18) =>L= 144
      (3) gcd (E,L) =1 =>E=5
      (4) E乘以几可以mode L =1? D=29可以满足
      (5) 得到公钥为: E=5,N=323
      (6) 得到私钥为: D=29,N=323
      (7) 加密 明文的E次方 mod N = 123的5次方  mod 323 = 225 (密文)
      (8) 解密 密文的D次方 mod N = 225的29次方 mod 323 = 123 (明文)
```

非对称加密命令行介绍

001 生成512位的RSA私钥

```
wendingding$ openssl genrsa -out private.pem 512
Generating RSA private key, 512 bit long modulus
.........++++++++++++++++++++++++++
..++++++++++++++++++++++++++
e is 65537 (0x10001)
```

002 以明文输出私钥内容

```
wendingding$ openssl rsa -in private.pem -text -out private.txt
writing RSA key
```

003 校验私钥文件

```
wendingding$ openssl rsa -in private.pem -check
RSA key ok
```

```
RSA key ok
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIIBOwIBAAJBAKfq2SR+2arBoS1c+Jc8Pt7N6Jus4IaqvFttbBaBriVm/EFGY5c0
eS4lDVwLWr0ld2bee720855YkFEsa6j9IlECAwEAAQJBAJwiSOogcTDPfpj5m8S0
JhlyCNnH87YER8QTiK+cwVrQV7AVMNaNWMa4hKZh/rsXTD5oPemUsTTjEWNNNjbE
o+kCIQDZypEycPx1knONDQSVbJwhEsFfnC81BPRUglAyLLK1NwIhAMVgWK6q5vI5
ZKR73XhoHMC3OLE3alvx43JnD0TE+yi3AiAU618tACtPw8RznPd+vtL0Xk/StjbW
meY6I9Y+K4ry2wIhAIPI9q15MsZch9RDV/aDHV+XdmTtKTjG4ySJp2o1U7U1AiBG

IM0D897qkeovClo3kw/mFDA9lgfJu1ir0W3OLSmvXg==
-----END RSA PRIVATE KEY-----
```

004 从私钥中提取公钥

```
wendingding$ openssl rsa -in private.pem -out public.pem -outform PEM -pubout
writing RSA key
```

005 以明文输出公钥内容

```
wendingding$ openssl rsa -in public.pem -out public.txt -pubin -pubout -text
writing RSA key

wendingding$ cat public.txt
Public-Key: (512 bit)
Modulus:
    00:a7:ea:d9:24:7e:d9:aa:c1:a1:2d:5c:f8:97:3c:
    3e:de:cd:e8:9b:ac:e0:86:aa:bc:5b:6d:6c:16:81:
    ae:25:66:fc:41:46:63:97:34:79:2e:25:0d:5c:0b:
    5a:bd:25:77:66:de:7b:bd:b4:f3:9e:58:90:51:2c:
    6b:a8:fd:22:51
Exponent: 65537 (0x10001)
-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKfq2SR+2arBoS1c+Jc8Pt7N6Jus4Iaq
vFttbBaBriVm/EFGY5c0eS4lDVwLWr0ld2bee720855YkFEsa6j9IlECAwEAAQ==
-----END PUBLIC KEY-----
```

006 使用公钥加密文件

```
wendingding$ echo "rsa算法测试" >> msg.txt
wendingding$ openssl rsautl -encrypt -pubin -inkey public.pem -in msg.txt -out msg.bin
wendingding$ ls -l
total 48
-rw-r--r--  1 wendingding  staff    64  2 24 18:20 msg.bin
-rw-r--r--  1 wendingding  staff    16  2 24 18:20 msg.txt
-rw-r--r--  1 wendingding  staff   497  2 24 18:19 private.pem
-rw-r--r--  1 wendingding  staff  1626  2 24 18:19 private.txt
-rw-r--r--  1 wendingding  staff   182  2 24 18:19 public.pem
-rw-r--r--  1 wendingding  staff   458  2 24 18:19 public.txt
```

007 对加密后的文件进行解密操作

```
wendingding$ openssl rsautl -decrypt -inkey private.pem -in msg.bin -out a.txt
```

```
wendingding$ cat a.txt
rsa算法测试

wendingding$ md5 a.txt
MD5 (a.txt) = f0f64bc1852acfd133e27567bd71c92e



wendingding$ md5 msg.txt
MD5 (msg.txt) = f0f64bc1852acfd133e27567bd71c92e
```