

声明：

1.此文档中，证书生成时口令、密码一律是：**12345**.方便排查；

```
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

```
[dong@ubuntu /data/SSL/apps/demoCA/mySSAtest]$sudo openssl
OpenSSL> pkcs12 -export -inkey CA_key.pem -in CAreq_cert.pem -out mypkcs12.pfx
Enter pass phrase for CA_key.pem:
Enter Export Password:
Verifying - Enter Export Password:
```

2.另外，所有命令均可在 [Openssl 在线文档](#)中查询；

3.本文档，支持命令检索，所有命令首字母大写；

# 一、证书类

## 1.1 证书申请：Req

1)Req

生成和处理 PKCS10 证书，证书申请文件；

用法：

```
Openssl req [-inform PEM|DER ] [-in filenames] [-passin arg] [-config filename] [batch]
[-newkey rsa:bits] [-new] [-rand file:rand dat] [-text] [-days] [-x509]
[-subj arg /CN=china/OU=tt/O=ab/CN=fxx] [-utf8] [-noout] [-keyout] [-pubkey]
```

其中，

-inform 指定输入格式；

-newkey rsa:bits 用于生成新的 rsa 密钥以及证书请求；默认名称为 privkey.pem；

-new 生成新的证书请求以及私钥，默认 1024bits

-rand 随机数种子文件； -rand file: rand.dat；

-config file 指定证书请求模板，默认为 openssl.cnf；

-subj arg 制定证书请求的用户信息；不指定，则要求输入； -subj /CN=cn/OU=t/O=a；

-utf8 指定为 utf8，默认为 ASCII 编码；

-batch 不询问用户请求；

-newhdr 在生成的 PEM 证书请求文件头尾包含 NEW 字样；有些 CA 需要；

-pubkey 获取证书请求中的公钥信息；

-modulus 输出请求中的模数；

-text 打印证书信息；

-verify 验证证书请求；

-x509 生成自签名证书；

-md5/4/2/sha1/mdc2 生成自签名证书时，指定摘要算法；

-days 生成自签名证书的有效期；

例子：

#生成证书申请和私钥

> req -new -batch -out myreq.pem #只有 -new 需要输入 subj 等；加入 -batch 则不

需要输入；

```
> req -newkey rsa:1024 -out myreq1.pem -keyout reqkey.pem
> req -new -subj /CN=china/OU=tt/O=ab/CN=dzr
#查看公钥信息；可见其中包含证书请求；
> req -in myreq.pem -pubkey -out pubkey.pem
> req -in myreq.pem -pubkey -text
> req -in myreq.pem -text
> req -in myreq.pem -modulus #所谓模数
> req -in pubkey.pem -modulus
> req -in pubkey.pem -text
#生成自签名证书
> req -in myreq.pem -x509 -key reqkey.pem -out mycert.pem -days 300 -text -md5
```

## 1.2 消息格式： PKCS7

1)pkcs7

加密消息语法，各种消息存放的格式标准；包括数据、签名、摘要、数字信封等；

用于处理 DER 或者 PEM 格式的 pkcs7 文件

用法：

Openssl pkcs7 [-inform PEM][--outform] [-in ][-print\_certs][--text][noout]

其中，

-print\_certs 打印证书或 CRL，每行打出持有者和颁布者；

-text 打印相关信息；

例子：

#没有 pkcs7 文件，可通过 crltopkcs7 生成；

```
Openssl> pkcs7 -in myreq.pem -outform DER -out myreq_pkcs7.der
```

```
Openssl> pkcs7 -in file.pem print_certs -out certs.pem
```

## 1.3 个人证书： PKCS12

1) pkcs12

工具，用于生成和分析 pkcs12 文件

用法：

Openssl pkcs12 [export][chain][inkey][cedrtfile][CApath][CAfile][name][out]

[aes128][des3][password][rand]

其中，

-export 输出 pkcs12 文件；

-chain 添加证书链；

-inkey 指定私钥文件；

-certfile file 添加 file 中所有证书文件；

-CApath arg 指定 CA 文件目录；

-CAfile arg 指定 CA 文件；

-name 指定证书和私钥的友好名；

-in 指定私钥和证书读取的文件，为 PEM 格式；  
-out 指定输出的 pkcs12 文件，默认为标准输出；  
-nomacver 读取时不验证 MAC；  
-clcerts 只输出客户证书，不包含 CA 证书；  
-nokeys 不输出私钥；  
-info 输出 pkcs12 结构信息；  
-des3/aes128 私钥加密算法；  
-nodes 不对私钥加密；  
-maciter 加强完整性保护，多次计算 MAC；  
-descert 用 3DES 加密 pkcs12 文件，默认 RC2-40；  
-keysig 设置私钥只能用于签名；

例子：

#生成 pkcs12，不带 CA 证书

> pkcs12 -export -inkey reqkey.pem -in mycert.pem -out mypkcs12.pfx

> pkcs12 -export -inkey CA\_key.pem -in CAreq\_cert.pem -out myPkcs12.pfx

#生成，带 CA 证书

> pkcs12 -export -inkey CA\_key.pem -in CAreq\_cert.pem -CAfile ../cacert.pem -chain -out omsp1.pfx

#信息分离出来写入文件；

> pkcs12 -in myPkcs12.pfx -out certandkey.pem

```
[dongq@ubuntu /dcrSSL/apps/demoCA/myPkcsTest]$cat certandkey.pem
Bag Attributes
    localKeyID: AA 85 E5 E8 DE 07 12 58 F5 46 EE F5 41 55 A1 6D
subject=/C=AU
issuer=/C=AU
-----BEGIN CERTIFICATE-----
MIIB6DCCAVGgAwIBAgIJAKixpAoElRYuMA0GCSqGSIb3DQEBAUAMA0xCzAJBgN
BAYTAKFVMB4XDTEyMDQwNzAzMDc1NVowDTEyMDc1NVowDTElMAkGA1U
```

#显示 pkcs12 信息

> pkcs12 -in myPkcs12.pfx -info

## 1.3 转换证书： PKCS8

10.pkcs8

私钥转换工具， pkcs8 格式

用法：

Openssl pkcs8 [inform ][outform][in ][passin][out][topk8][noiter][v2][embed]

其中，

-topk8 输出 pkcs8 文件；

-noiter MAC 保护计算次数为 1；

-nooct 不采用 8 位组表示私钥；

-embed 采用嵌入式 DSA 参数格式；

-v2 alg 采用 PKCS5 v2.0 并指定加密算法；

例子：

#私钥转换为 pkcs8 文件；比较前后文件内容，的确不同的格式；

Openssl> pkcs8 -in CA\_key.pem -topk8 -out mypkcs8key.pem

#以明文存放

```
> pkcs8 -in CA_key.pem -topk8 -nocrypt -out my8key_plain.pem
```

## 1.5 证书中心： Ca

1)Ca

一个小型的 CA 系统，签发证书请求和生成 CRL，维护一个已签发证书状态的文本数据库。

用法：

```
Openssl ca [verbose][config][name][gendcrl][revoke][crl_reason][crl_hold][crl_compromise]
[crl_CA_compromise][subj][crl_days][crl_housrs][days][md][policy]
```

其中，

-verbose 打印附加信息；

-config 指定配置文件；默认安装，配置文件在 /usr/local/ssl 路径下；

-name section 替换配置文件 default\_ca 所代表的内容；

-gencrl 生成 CRL 文件；

-revoke file 撤销证书，file 文件中包含了证书；

-crl\_reason reason 设置 CRLv2 撤销原因；

-policy arg 指定 CA 策略；

-utf8 表明输入必须是 utf8 编码；

-keyfile 指定签发证书的私钥文件；

-key 指定私钥解密口令；

-cert file 指定 CA 文件；

-in 输入证书请求文件；

-out 输出文件名；

-notext 不输出文本格式证书信息；

-outdir 设置输出路径；

-infiles 处理多个路径请求文件，必须放在最后；

-ss\_cert file 指定需要由 CA 签发的自签名证书；

-preserveDN 将证书中的 DN 顺序由配置文件来决定；如设置此项，则顺序与请求文件一致；

-noemailDN 设置此项，则生成证书不会包含持有者 DN，而是放在了扩展项里；

-batch 批处理，不询问客户信息；

-md arg 设置摘要算法，md5/sha/sha1/mdc2；

-utf8 表示输入必须为 utf8 编码，默认为 ASCII 编码；

例子：

#/以下所有命令 必须在 apps 目录下运行；（不看清，后面有教训！）

### #1. 建立 CA

CA.sh -newca，默认，配置文件在 /usr/local/ssl，可先在 apps 目录的 CA.sh 建立环境

#可以生成的 demoCA/ 目录结构，包含项目较多；

#其中 CA index:

```
[deng@ubuntu /usr/local/ssl/apps/demoCA]$ cat index.txt
V          150409093241Z          80F0676A66D4803B          unknown /C=gq/ST=ww/O=rr/OU=tt/CN=yy/emailAddress=uu
```

### #2. 生成证书请求

#用-batch 属性，导致后来签发失败；且看；（以后务必在 apps 目录进行）

```
Openssl> req -new -batch -out CA_req.pem -keyout CA_key.pem
```

#参考 Req 一节，执行如下

```
> req -in CA_req.pem -text
```

#证书申请自签名后的样子；比较可以发现，自签名证书和证书请求间的区别；

```
> req -in CA_req.pem -x509 -key CA_key.pem -out CAreq_cert.pem -days 300 -text -md5
```

### #3 签发证书

```
Openssl> ca -config /usr/local/ssl/openssl.cnf -name CA_default -days 365 -md sha1 -policy  
policy_anything -cert demoCA/cacert.pem -in CA_req.pem -out CA_cert.pem -preserveDN  
-noemailDN -subj /CN=Cn /O=JS/OU=WmX/cn=michaDong -extensions myexts (好长)
```

#简洁版：

#正确执行签发；

```
> ca -in CA_req1.pem -cert ./demoCA/cacert.pem -out CA_cert.pem
```

```
demoCA> openssl x509 -text -in CA_cert.pem  
Certificate:  
Data:  
Version: 3 (0x2)  
Serial Number:  
96:63:2b:2e:26:6e:64:14  
Signature Algorithm: sha1withRSAEncryption  
Issuer: C=qq, ST=ww, O=rr, OU=tt, CN=yy/emailAddress=uu  
Validity  
Not Before: Apr  9 12:08:33 2012 GMT  
Not After : Apr  9 12:08:33 2013 GMT  
Subject: C=qq, ST=ww, O=rr, OU=tt, CN=yyy/emailAddress=uu  
Subject Public Key Info:  
Public Key Algorithm: rsaEncryption  
Public-Key: (1024 bit)
```

#比较签发前的证书请求，可见签发加入了 X509v3 extensions；和自签名证书类似；

### #4. 撤销证书

#按上面步骤，在生成一个 CA\_cert2.pem 证书；

```
> req -new -out CA_req2.pem -keyout CA_key2.pem
```

```
> ca -in CA_req2.pem -out CA_cert2.pem
```

#可以查看文本数据库 index.txt，比较前后变化

```
> ca -revoke CA_cert2.pem
```

#生成 CRL，设置原因、挂起处理方法；

```
> ca -gencrl -out CA_crl.crl
```

## 1.6 证书吊销： Crl/Crl2pkcs7

1)Crl

工具，用于处理 PEM 或 DER 格式的 CRL 文件

用法：

```
Openssl crl [inform] [text][in ] [hash][ issuer][lastupdate][CAfile][CApath dir]
```

其中，

-noout 不打印 CRL 文件内容；

-hash 打印值；

-issuer 打印颁布者 DN；

-lastupdate 上次发布时间；

-CAfile 指定 CA 文件；

-CApath dir 指定多个 CA 文件路径，文件名为 xxx.0，xxx 为特有摘要值；

例子：

#参考 CA，生成 CRL 文件继续

#在 apps 目录下进行；

```

> ca -gencrl -out CA_crl.pem
-----
#没有文件 demoCA/crlnumber ; CA 自动建立时，并没有生成该文件；
#手动创建一个试试；
$ sudo touch demoCA/crlnumber
$ chmod a+w demoCA/crlnumber
$ sudo cat 2 > demoCA/crl number
> ca -gencrl -out CA_crl.pem
#设置 2 太短，出错，最好是两位数如 23 就可以了（有待更多探索）；还有格式是 crl；
-----
> ca -gencrl -out CA_crl1.crl
#显示 CRL 信息
Openssl> crl -in CA_crl1.crl -text -issuer -hash -lastupdate -nextupdate
#验证 CRL
> crl -in CA_crl.crl -CAfile demoCA/cacert.pem -text
#通过指定 CA 路径验证；在 demoCA 中建立 CAfiles；
> x509 -in demoCA/cacert.pem -hash
#在 CAfiles 下建立 b07cc20a.0 文件，内容为 demoCA/cacert.pem 内容；
#验证 CRL
> crl -in CA_crl.crl -CApath demoCA/CAfiles -noout
2)Crl2pkcs7
根据 CRL 或者证书生成 pkcs7 消息；
用法：
Openssl crl2pkcs7 [inform][in ] [certfile][nocrl]
其中，
-certfile filename 可以指定证书， PEM 证书可以包含多个证书；可多次使用次选项；
-nocrl 不处理 crl；设置此项，读取时忽略 CRL 信息；
例子：
#只有 CRL 信息；
> crl2pkcs7 -in CA_crl1.crl -out CA_crlpkcs7.pem
#包含 CRL 信息和证书信息；
> crl2pkcs7 -in CA_crl1.crl -certfile demoCA/cacert.pem -out CA_crlcertpkcs7.pem
#只包含证书信息；
> crl2pkcs7 -in CA_crl1.crl -certfile demoCA/cacert.pem -out CA_certpkcs7.pem -nocrl

```

## 1.7 强大证书工具： X509

### 1)X509

多用途的证书工具，显示证书信息、转换证书格式、签名证书请求及改变证书信任设置；

用法：

```

Openssl x509 [ -inform DER|PEM|NET ] [ -CAform ] [-serial] [-hash] [-subject] [-out] [-issuer]
[-email] [-enddate] [-pubkey] [-ocspid] [-trustout] [-clrrreject] [-addtrust] [-signkey filename]
[-x509toreq] [-req] [-CA] [-set_serial] [-text] [-nameopt option]
[-CAcreateserial]

```

其中，



-subject\_hash 显示证书摘要号，同 -hash；  
 -subject 显示证书持有者 DN；  
 -issuer 显示证书颁布者的 DN；  
 #还有打印输出相关，  
 -serial 显示序列号；  
 -enddate 到期时间；  
 -purpose 证书用途；  
 -modulus 证书公钥模数；  
 -email 显示邮件地址；  
 -pubkey 输出公钥；  
 -fingerprint 打印微缩图；  
 -C 以 C 语言格式显示信息；  
 -trustout 输出可信任证书；  
 -text 打印证书信息；  
 #特殊处理  
 -req 输入为证书请求，需要进行处理！  
 -x509toreq 根据证书来生成证书请求，需要指定签名私钥；和 req 是逆过程；  
 -exfile filename 指定包含证书扩展名的文件；  
 -CA 设置 CA 文件；必须为 PEM 格式；  
 -CAkey 设置 CA 私钥文件，必须为 PEM 格式；  
 -CAserial 指定序列号文件；  
 -set\_serial 设置证书序列号；  
 -md2/5/sha1/mdc2；

例子：

#转换为证书请求

```
Openssl> x509 -in CA_cert.pem -x509toreq -signkey CA_key.pem -out CA_certoreq.pem
```

#控制打印格式

```
> x509 -in CA_cert.pem -nameopt RFC2253 -C
```

#扩展项

```
> x509 -req -in CA_req.pem -extfile openssl.cnf -extensions v3_usr -CA casert.pem
```

```
-CAkey key.pem -CAcreateserial
```

#打印微缩图

```
> x509 -in CA_cert.pem -text -fingerprint
```

```

-----END X509 CRL-----
OpenSSL> x509 -in CA_cert.pem -fingerprint
SHA1 Fingerprint=49:7B:E2:9C:3C:0E:3D:75:69:FE:15:E4:46:51:AA:F1:E8:7
-----BEGIN CERTIFICATE-----
MIICojCCAgugAwIBAgIJAJZjKywmbmQUMA0GCSqGSIb3DQEBBQUAMFQxCzAJBgNV
BAYTANFxmQswCQYDVQQIDAJ3dzELMAkGA1UECgwCcnIxCzAJBgNVBAsMANR0MQsw
CQYDVQQDDAJ5eTERMA8GCSqGSIb3DQEJARYCdXUwHhcNMTIwNDA5MTIwODMzWhcN
-----END CERTIFICATE-----

```

## 1.8 证书验证：Verify

1)Verify

证书验证工具

用法：

```
Openssl verify [CApath][CAfile][purpose][untrusted][help][issuer_checks][verbose]
```

[crl\_check] **certificates**

其中，

-CApath 信任的 CA 证书存放目录，其中文件名为 xxx.0，为证书持有者摘要值，通过 openssl  
>x509 -in cacert1.pem 可以获取；

-CAfile CA 证书，格式为 PEM，可有多 CA 证书；

-untrusted file 不信任的 CA 的证书，一个文件里可有多不信任的 CA 证书；

-purpose 证书的用途；不设置，不会验证证书链；该值可以是：sslclient/sslsrvr/nssslserver/  
Smimesign 和 smimeencrypt；

-help

-verbose 打印详细信息；

-issuer\_checks 打印被验证证书与 CA 证书间的关系；

-crl\_checks 验证 CRL，可将 CRL 内容写到 CAfile 指定的 PEM 文件中；

**certificates** 待验证的证书；

例子：

#1 ) 验证 CRL

#需要将 crl 文件 CA\_crl.crl 的内容拷贝到 demoCA/cacert.pem 的结尾，然后对 CA\_cert.pem  
验证 crl. 否则会出现异常结果；

\$ chmod a+w demoCA/cacert.pem

\$ cat CA\_crl.crl >> demoCA/cacert.pem

> verify -CAfile demoCA/cacert.pem -verbose -crl\_check CA\_cert.pem CA\_cert2.pem

#正确执行验证的结果，可见，CA\_cert2.crl 已经在撤销列表了；

```
OpenSSL> verify -CAfile demoCA/cacert.pem -verbose -crl_check CA_cert.pem CA_cert2.pem
CA_cert.pem: OK
CA_cert2.pem: C = qq, ST = ww, O = rr, OU = tt, CN = y, emailAddress = uu
error 23 at 0 depth lookup:certificate revoked
```

#2 ) 验证证书

#验证证书是否可以通过 CA 证书验证；

> verify -CAfile demoCA/cacert.pem -purpose sslclient CA\_cert.pem CA\_cert2.pem

## 二、算法类

包含摘要算法，对称非对称算法

### 2.1 摘要算法：Gendh/Dhparam/Dh

1)Gendh

用于生成 DH 参数

用法：

Openssl>gendh [-out file ] [-rand ] [-engine e ]

例子：

Openssl> gendh -5 -out CA\_dh.pem 1024

2)dhparam 和 dh



DH 参数及生成操作

用法：

Openssl dhparam [inform ][outform][in][out][dsaparam][noout][check][text][numbits]

-dsaparam 生成 DSA 参数，并转换为 DH 格式；

-check 检查 dh 参数；

-C 以 C 语言风格打印；

-numbits 指定素数 bit 数；默认为 512 位；

例子：

Openssl> dhparam -out dhparam.pem -text 512 #生成 DH 参数

#检查 DH 参数

> dhparam -in dhparam.pem -text -check

## 2.2 摘要算法集： Dgst

1)Dgst

用于数据摘要

用法：

Openssl>dgst [-md5|-sha|mdc2/ripemd160/dss1] [-hex] [ -binary][ -out filename] [-engine] [-sign privatekeyfile]

[-prverify privatekeyfile][ -verify publickeyfile]

其中，

-d 打印调试；

-sign 用私钥签名；

-verify 用公钥验证签名；

-keyform PEM |ENGINE 密钥格式， pem，或者采用 Engine;

-hex 显示十六进制结果，默认；

-binary 显示二进制；

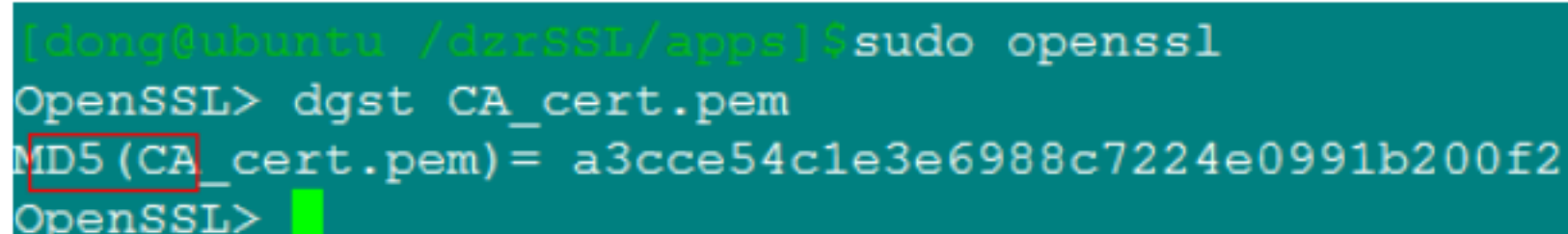
-engine e 采用引擎 e 来运算；

-md5 默认，用来摘要；

-md4/2/sha1/sha256/sha512/mdc2/ripemd160 各种摘要可选；

例子：

Openssl> dgst CA\_cert.pem



```
[dong@ubuntu /dztSSL/apps]$sudo openssl
OpenSSL> dgst CA_cert.pem
MD5(CA_cert.pem) = a3cce54c1e3e6988c7224e0991b200f2
OpenSSL>
```

#摘要结果用私钥签名；

Openssl> dgst -sign CA\_key.pem -sha512 -out CA\_sigDgst.pem CA\_cert.pem

## 2.3 非对称加密： Genrsa/Rsa/Rsautl

1)Genrsa

生成 RSA 密钥

用法：

Openssl >genrsa [-out filename] [-passout arg] [-des3] [-rand files] [-engine id] [-numbits]

其中，

-des/des3/idea/aes128/aes192/aes256 以 des/des3/idea cbc 模式加密；

-out file 输出文件；

-f4/3 指定 E 为 f4 、 3；

-rand file 指定随机数种子文件；

-numbits 密钥长度，默认为 512；

例子：

Openssl> genrsa -aes256 -out CA\_rsa\_key.pem -f4 1024

2)Rsa

处理 RSA 密钥、格式转换和打印信息

用法：

Openssl rsa [-inform PEM|NET|DER][-in ][-passout arg][-des3][-check][-pubout]

例子：

#生成私钥明文文件；

Openssl>genrsa -out prikey.pem

#转换为 DER 编码；

Openssl>rsa -in prikey.pem -outform der -out prikey.der -text

#明文私钥转换为密码保护；

Openssl>rsa -inform der -in prikey.der -des3 -out encprikey.pem

#明文私钥转换为密码保护；用 prikey.pem 试试；

Openssl>rsa -inform pem -in prikey.pem -des3 -out encprikey.pem

#将公钥写入文件

Openssl>rsa -in prikey.pem -pubout -out pubkey.pem

#打印公钥信息，其中 modulus 打印模数；

Openssl>rsa -pubin -in pubkey.pem -text -modulus

#显示私钥信息，保护密钥写在 pwd.txt 中；

Openssl>rsa -in encprikey.pem (-passin file:pwd.txt)

3)Rsautl

RSA 工具，使用 RSA 算法签名，验证身份，加解密数据

用法：

Openssl rsautl [-in ][inkey][pubin][certin][sign][verify][encrypt][decrypt][pkcs]

[ssl][raw][hexdump][passin]

其中，

-pubin 表示输入的是公钥文件，默认为私钥；

-sign 给输入的数据签名；

-verify 对输入数据验证；

-encrypt 用公钥对输入数据加密；

-decrypt 用私钥对输入数据解密；

-pkcs/oaep/ssl/raw 指定填充方式；

-hexdump 用 16 进制输出数据；

-passin arg 指定私钥保护口令的来源，如 -passin file :pwd.txt

例子：

#生成 RSA 密钥

```

> genrsa -des3 -out prikey.pem
#分离出公钥
> rsa -in prikey.pem -pubout -out pubkey.pem
#对文件签名；文件不能太长，删改 a.txt；
> rsautl -sign -inkey prikey.pem -in a.txt -out sig.dat -hexdump
#验证签名；相当于消除签名，还原内容；
> rsautl -verify -inkey prikey.pem -in sig.dat
#私钥签名，用公钥查看；
> rsautl -verify -pubin -inkey pubkey.pem -in sig.dat # 验证签名；用公钥，发现也可以；
OpenSSL> rsautl -verify -pubin -inkey pubkey.pem -in sig.dat
signed by prikey.pem and will be
#公钥加密；私钥解密；也可以；
> rsautl -encrypt -pubin -inkey pubkey.pem -in a.txt -out b.txt
> rsautl -decrypt -inkey prikey.pem -in b.txt
#用证书中的公钥加密
> rsautl -encrypt -certin -inkey CAreq_cert.pem -in a.txt -out enca.dat
#用私钥解密，看下能否恢复；
#私钥不对为证书 CAreq 的私钥 CA_key.pem 文件，修改后成功；
> rsautl -decrypt -inkey CA_key.pem -in enca.dat -out deca.txt

```

## 2.4 椭圆曲线算法： Ecparam/Ec

### 1)Ecparam

椭圆曲线参数及生成操作

用法：

Openssl ecparam [inform][outform][in][param\_enc][no\_seed][genkey]

其中，

-check 检查参数；

-name arg 采用短名字；

-list\_curves 打印所有可用的短名字；

-conv\_form arg 指定信息存放的方式，可以是 compressed/uncompressed/bybrid,默认为 compressed；

-no\_seed 如-param\_enc 指定为 explicit，不采用随机种子；

-rand files 指定随机种子；

-genkey 生成密钥；

例子：

Openssl> ecparam -list\_curves #打印所有可用的短名字

```

OpenSSL> ecparam -list_curves
secp112r1 : SECG/WTLS curve over a 112 bit prime field
secp112r2 : SECG curve over a 112 bit prime field
secp128r1 : SECG curve over a 128 bit prime field
secp128r2 : SECG curve over a 128 bit prime field

```

#生成椭圆曲线密钥；

> ecparam -name secp112r2 -genkey -text

> ecparam -genkey -name sect163r1 -out ec163.pem

#更新生成密钥

> req -newkey ec:ec163.pem

## 2)Ec

椭圆曲线密钥处理工具

用法：

```
Openssl>ec [inform] [in][passin][out][des][conv]][param]
```

例子：

```
Openssl>ecparam -genkey -name secp112r1 -out ekey.pem -text          #生成 ec 私钥
```

```
>ec -outform der -in ekey.pem -out ekey.der          #转换为 DER 编码
```

```
>ec -in ekey.pem -des -out encekey.pem              #私钥进行口令保护
```

```
>ec -in ekey.pem -pubout -out ecpubkey.pem          #公钥写入文件
```

```
>ec -in ekey.pem -text          #显示密钥信息
```

```
>pkcs8 -topk8 -in ekey .pem -out ekeypk8.pem        #转换为 pkcs8 格式
```

## 2.5 数字签名： Dsa/Dsaparam/Gendsa

### 1)Dsa

处理 DSA 密钥、格式转换和打印信息

用法：

```
Openssl dsa [inform ][outform][in ] [des3][text][modulus]
```

其中，

-passin arg 指定私钥包含口令存放方式，如 -passin file :pwd txt ；

-des 指定私钥保护加密算法；

-modulus 打印公钥信息；

例子：

```
Openssl> dsaparam -out dsaparam.pem 1024          #生成 dsa 参数文件
```

```
> gensa -out dsakey.pem dsaparam.pem              #生成 dsa 密钥
```

```
> dsa -in dsakey.pem -outform DER -out dsakeyder.pem          #转换为 DER 格式
```

```
> dsa -in dsakey.pem -modulus          #打印公钥信
```

```
> dsa -in dsakey.pem -des -out enckey.pem          #加密存放
```

### 2)Dsaparam

用于生成和操作 dsa 证书参数

用法：

```
Openssl dsaparam [-inform DER|PEM][-in ] [text][ -genkey][ -C][ -numbits]
```

其中，

-genkey 生成 dsa 密钥；

-nubmber 制定密钥大小；

例子：

```
Openssl>dsaparam -genkey 512 -out dsa.pem -C
```

### 3)Gendsa

生成 DSA 密钥，密钥参数用 dsaparam 生成；

用法：

```
Openssl gensa [ out ][ des] paramfile 必选
```

其中，

-des3 指定私钥口令保护算法；不指定，则明文存放；

**paramfile** 制定 DSA 密钥参数文件；

例子：

```
Openssl>dsaparam -genkey 512 -out dsa.pem
Openssl>gendsa -des3 -out encdsa.pem dsa.pem
```

## 2.5 对称算法： Enc

1)Enc

对称加解密工具，可以进行 base64 编码转换；

用法：

```
Openssl enc -ciphername [-in ][pass][bufsize] [nopad][kfile] [e d a A p P ] [debug]
```

其中，

-ciphername 对称算法名字

-pass arg 指定密码保护来源；

-e 加密

-d 解密

-a 自动在加密后，解密前，进行 base64 编解码；

-p 打印出使用的 salt、口令以及初始化变量 iv；

-P 同 p，只不做加解密操作；

-bufsize 设置 IO 擦左缓冲区大小；

-debug 调试信息；

例子：

```
Openssl>enc -des3 -e -in a.txt -out b.txt
```

```
OpenSSL> enc -des3 -e -in a.txt -out b.txt
enter des-ede3-cbc encryption password:
Verifying - enter des-ede3-cbc encryption password:
```

# 三、工具类

## 3.1 诊断工具： ASN.1

1)Aasn1parse

用于诊断 ASN.1 结构的工具，也可从 ASN.1 数据中提取数据；

用法：

```
Openssl >asn1parse [-inform PEM|DER] [-in filename] [-out filename] [-length num] [-i] [-oid]
```

其中，

-out 默认为 PEM，可以为 DER 文件；

-offset num 数据分析字节偏移量；

-length 分析数据长度，默认为全部；

-dump 显示 16 进制；

-i 标记实体，输出有所进；

-oid file 指定外部的 oid 文件；

例子：

```
Openssl>asn1parse -in CA_req.pem -inform pem -i
```

```
>asn1parse -in CA_key.pem -i
```

```
#其中 n : d=x h1=x l=x cons/prim: SEQUENCE 格式里 ,
-n 表示偏移量 ;
-d 表示此项的深度 ;
-hl 表示 asn1 头长度 ;
-l 表示内容长度 ;
Prim:OBJECT 表示 ASN1 类型 ;
#其他例子
#显示内容 , 并生成 der 编码文件 ;
> asn1parse -in CA_req.pem -out CA_req.der
#偏移量开始分析
> asn1parse -in CA_req.pem -offset 54
```

## 3.2 序列转换 : Nseq

```
1)Nseq
多证书与 netscape 证书序列间相互转化
用法 :
Openssl nseq [in] [out][toseq]
例子 :
#多个证书写入同一个文件
$ cat CA_cert.pem >CA_1.pem
$ cat CA_cert2.pem >>CA_1.pem
#多个证书转化为 netscape 证书序列
> nseq -in CA_1.pem -toseq -out CA_2.pem
#netscape 证书序列转化为多个证书
>nseq -in 2.pem -out 3.pem
```

## 3.3 在线证书状态 : OCSP

```
1)Ocsp
在线证书状态工具
用法 :
Openssl ocsp [out ][issuer][cert][serial][req][url][CA][no_certs][req_text][host][CApath
dir][CAfile][trust]
例子 :
#用 req 和 ca 命令生成 OCSP 服务器证书和私钥
#生成 OCSP 请求
Openssl>ocsp -issuer demoCA/casert.pem -cert cert.pem -cert -cert2.pem -reqout ocspreq.der
>ocsp -reqin ocspreq.der -text# 打印 OCSP 请求消息
#启动 OCSP 服务
>ocsp -ndays 1 -index/demoCA /index.txt -port 3904 -CA demoCA/cacert.pem -text
-rkey ocspserverkey.pem -rsigner ocspservercert.pem#
#请求 OCSP 响应
>ocsp -issuer demoCA/cacert.pem -url https:#127.0.0.1:3904 -reqin ocspreq.der
-VAfile ocspservercert.pem -respout resp.der
```



### 3.4 加密套件： Ciphers

1)Ciphers

显示支持的加密套件

Openssl ciphers [-v ][-ssl3][tls1][cipherlist]

其中，

-v 详细列出所有加密套件，包括 ssl 版本、密钥交换算法、身份验证算法、对称算法；摘要算法等；

-ssl3 只列出 sslv3 使用的加密套件；

-tls1 只列出 TLSv1 使用的；

-cipherlist 规则字符串；

例子：

Openssl>ciphers -v

Openssl>ciphers -v 'ALL:eNULL'

Openssl>ciphers -v '3DES:+RSA'

```
OpenSSL> ciphers -v '3DES:+RSA'
ECDHE-RSA-DES-CBC3-SHA SSLv3 Kx=ECDH Au=RSA Enc=3DES(168) Mac=SHA1
ECDHE-ECDSA-DES-CBC3-SHA SSLv3 Kx=ECDH Au=ECDSA Enc=3DES(168) Mac=SHA1
EDH-RSA-DES-CBC3-SHA SSLv3 Kx=DH Au=RSA Enc=3DES(168) Mac=SHA1
EDH-DSS-DES-CBC3-SHA SSLv3 Kx=DH Au=DSS Enc=3DES(168) Mac=SHA1
AECDH-DES-CBC3-SHA SSLv3 Kx=ECDH Au=None Enc=3DES(168) Mac=SHA1
ADH-DES-CBC3-SHA SSLv3 Kx=DH Au=None Enc=3DES(168) Mac=SHA1
ECDH-RSA-DES-CBC3-SHA SSLv3 Kx=ECDH/RSA Au=ECDH Enc=3DES(168) Mac=SHA1
ECDH-ECDSA-DES-CBC3-SHA SSLv3 Kx=ECDH/ECDSA Au=ECDH Enc=3DES(168) Mac=SHA1
PSK-3DES-EDE-CBC-SHA SSLv3 Kx=PSK Au=PSK Enc=3DES(168) Mac=SHA1
DES-CBC3-SHA SSLv3 Kx=RSA Au=RSA Enc=3DES(168) Mac=SHA1
OpenSSL>
```

### 3.4 性能工具： Speed/S\_time

1)Speed

调整测试库的性能

用法：

Openssl [-engine id] [md5][sha1][rsa 1024][dsa2048][des][blowfish][-slapsed]

其中，

-elapsed 测量采用实时时间，不是 CPU 时间；

-mr 生成机器可读显示；

-multi n 并行允许 n 个测试；

例子：

> speed rsa4096 -mr -elapsed

> speed blowfish

> speed dsa2048

> speed rmd160

> speed md5

#从数值比较来看， md5 的执行速度相比 rm160 较快；

2)S\_time

提供的 SSL/TLS 性能测试工具，测试服务；

用法：

Openssl s\_time [connect host:port][www][cert][key][CApath][CAfile][new][time][ssl2][bugs]

-connect host:port 指定服务，默认为 4433 端口；

-www 指定获取的 web 网页；

-cert 指定证书；  
-CApath dir 指定 CA 文件目录；  
-CAfile 指定 CA 文件；  
-new 新建连接；  
-verify depth 设置验证深度；  
-nbio 不采用 BIO；  
-ssl2/3 采用 SSL 协议；  
-bugs 开启 SSL bug 兼容；  
-cipher 指定加密套件；

例子：

```
Openssl>s_server -cert sslservercert.pem -key sslserver.pem -ssl3#启动 s_server 服务
>s_time -cert sslclientcert.pem -key sslclientkey.pem -CAfile dmeoCA/cacert.pem -ssl3#启动
s_time
```

### 3.5 协议会话工具： Sess\_id/S\_server/S\_client

1) sess\_id

SSL/TLS 协议的 session 处理工具

用法：

```
Openssl sess_id [-inform PEM|DER][--outform ] [-in ] [-text][--context ID][--cert]
```

其中，

-cert 打印数字证书；  
-text 打印信息；

需要有 session 文件，将 SSL\_SESSION 写入文件即可，然后可以分析；

例子：

【空缺案例】

2) s\_server

#生成各种服务器证书，参考 CA 一节；

SSL 服务程序，前提是需要生成各种证书；可测试客户端，如各浏览器 https 协议支持；

用法：

```
Openssl s_server [-accept port][--context id] [-verify depth][--cert filename]
```

```
[-debug][--msg][--www][--WWW][--HTTP][--crlf][ q Q r R P S]
```

其中，

-accept arg 监听 TCP 端口，缺省为 4433；  
-contextarg 设置 ssl 上下文，不设置采用默认；  
-cert certname 服务使用的证书文件名；  
-certform 证书格式；  
-keyform 私钥格式；  
-pass arg 私钥口令来源；  
-msg 打印协议内容；  
-timeout 设置超时；  
-key file 服务使用的私钥文件；  
-no\_tmp\_rsa 不生成临时 RSA 密钥；  
-verify depth 设置证书验证深度；  
-Verify arg 设置为 1，服务端必须验证客户身份；

-CAfile 指定 CA 证书文件；  
-state 打印 SSL 握手状态；  
-nbio 不采用 bio；  
-quiet 不打印输出信息；  
-www 返回用户一个网页；内容为 SSL 握手的内容；  
-WWW -HTTP 将某文件作为网页返回客户端；URL 请求为 https://myhost/page.html，则把 ./page.html 返回给 client；不设置时，客户端终端输入什么，服务返回相同字符；  
-crlf 将用户在终端输入的换行回车转化为 /r/n；

连接命令有，

-q 中断当前连接，但不关闭服务；  
-Q 中断连接，退出程序；  
-r 重新协商；  
-R 重新协商，并要求客户端证书；  
-P 在 TCP 直接送明文，造成客户端握手错误并断开连接；  
-S 打印缓存的 SESSION 信息；

例子：

```
> s_server -state -msg -CAfile demoCA/cacert_bak.pem -key demoCA/private/cakey.pem
OpenSSL> s_server -state -msg -CAfile demoCA/cacert.pem -key demoCA/private/cakey.pem
Enter pass phrase for demoCA/private/cakey.pem:
Using default temp DH parameters
Using default temp ECDH parameters
error setting private key
3077646488:error:0B080074:x509 certificate routines:X509_check_private_key:key values mismatch:x509_cmp.c:318:
error in s_server
OpenSSL>
```

[求解]

难道 CA 证书和 CA 私钥不匹配？

#监听服务端口 4433 默认；

```
> s_server -accept 4433 -msg -state
```

3)S\_client

为 SSL/TLS 客户端程序，与 s\_server 对应，可通信

用法：

```
Openssl s_client [ -connect host:port ][ -verify depth ][ -cert filename ]
```

```
[ -pause ][ -debug ][ -ssl3 ][ -tls1 ][ cipher cipherlist ][ -rand
```

其中，

-host host 设置服务地址；  
-port 设置端口；  
-connect host:port 设置服务地址和端口；  
-verify 设置证书验证深度；  
-cert arg 设置握手采用的证书；  
-certform arg 设置证书格式；  
-key arg 指定客户私钥名；  
-CApath 设置信任 CA 文件路径，ca 文件名采用特殊 xxx.0 形式，通过 x509 -hash 获得；  
-CAfile 指定 CA 文件名；  
-reconnect 重新连接，进行 session 重用；

-pause 每当读写数据时， sleep 1s；  
-debug 调试  
-showcerts 显示证书链；  
-msg 打印协议信息；  
-state 打印 SSL 状态；  
-quiet 不显示客户端数据；  
-cipher 指定加密套件；  
-crlf 将用户在中断输入的换行回车转化成 /r/n ；

例子：

#连接服务器 200.200.145.201

> s\_client -connect 200.200.145.201:4433 -state -msg

> s\_client -showcerts

## 3.6 邮件验证工具： Smime

1)Smime

处理 S/MIME 邮件，加密、解密、签名和验证

用法：

Openssl smime [encrypt][decrypt][sign][verify][pk7out][des][subject]

其中，

-encrypt 加密；

-decrypt 解密；

例子：

>smime -encrypt -in mail.pem -out enced.pem -des newcert.pem#用对方的证书加密消息

>smime -decrypt -in enced.pem -out mymail.pem -inkey newkey.pem# 私钥解密消息

## 3.7 其他工具： Prime/Errstr/Version/Passwd

1)Prime

检查一个数是否是素数

例子：

Openssl> prime 79

2)Errstr

查询错误代码；

用法：

Openssl errstr [stats]<errno>

例子：

Openssl req -config no.txt# 输入错误 ,提示错误 02001002

>errstr -stats 02001002

3)version

打印版本信息

用法：

Version -[avbofp]

-a 所有；

-b 编译；

-o 加密算法和机器字节；

-f 编译选项；

-p 平台；

例子：

> version -a -b -o -f -p

```
OpenSSL> version -a -b -o -f -p
OpenSSL 1.0.0e 6 Sep 2011
built on: Thu Feb  9 00:57:05 UTC 2012
platform: debian-i386
options: bn(64,32) rc4(idx,int) des(ptr,risc1,16,long) blowf
compiler: cc -fPIC -DOPENSSL_PIC -DZLIB -DOPENSSL_THREADS -D_
a,--noexecstack -g -Wall
OPENSSLDIR: "/usr/lib/ssl"
OpenSSL> version -a
```

4)Passwd

生成各种口令密文

用法：

Openssl> passwd [-crypt] [-in file ] [-table] [-noverify]

其中，

-crypt 默认，显示 Unix 密文；

-stdin 默认，从标准输入口令；

-noverify 不验证口令；

-quiet 无警告；

-in file 从文件读取口令；

-reverse 输入口令和结果用缩进隔开，输出内容顺序颠倒；

例子：

Openssl> passwd -crypt -noverify -reverse

1.5)Rand

生成随机数

用法：

Openssl rand [-out file] [-rand files] [-base64] num

其中，

-base 64 输出结果为 BASE64 编码；

-num 随机数长度；

例子：

Openssl> rand 100 -base64 -out CA\_rand.dat

# 花絮

小编在写作此文档时，犯过一些有趣的错误；

分享一个在 Openssl 协议中“生成 CA 签发证书”的故事；

#修改签发证书命令，关于 Subj 的探索很纠结；（此处是试验，结果证明是 错的 ！）

-----START-----

Openssl> ca -in CA\_req.pem -batch -out CA\_cert.pem



#重新生成证书请求，参考上文

> req -new -out CA\_req.pem -keyout CA\_key.pem

```
OpenSSL> req -new -out CA_req.pem -keyout CA_key.pem
Generating a 1024 bit RSA private key
.....+++++
.....+++++
writing new private key to 'CA_key.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
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) [AU]:qq
State or Province Name (full name) [Some-State]:ww
Locality Name (eg, city) []:ee
Organization Name (eg, company) [Internet Widgits Pty Ltd]:xx
Organizational Unit Name (eg, section) []:tt
Common Name (eg, YOUR name) []:yy
Email Address []:uu

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:12345
An optional company name []:
```

> req -in CA\_req.pem -text

```
OpenSSL> req -in CA_req.pem -text
Certificate Request:
Data:
  Version: 0 (0x0)
  Subject: C=qq, ST=ww, L=ee, O=rr, OU=tt, CN=yy/emailAddress=uu
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    Public-Key: (1024 bit)
    Modulus:
      00:bf:26:7b:72:f5:47:ac:4b:b7:db:cc:03:27:6d:
```

#比较根证书内容

```
[dong@ubuntu /usr/local/apps]$ cat demoCA/cacert.pem
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      80:f0:67:6a:66:d4:80:3b
    Signature Algorithm: sha1WithRSAEncryption
    Issuer: C=qq, ST=ww, O=rr, OU=tt, CN=yy/emailAddress=uu
    Validity
      Not Before: Apr  9 09:32:41 2012 GMT
```

#发现一致，进行签发证书应该不会错；

> ca -in CA\_req.pem -out CA\_cert.pem

```
OpenSSL> ca -in CA_req.pem -out CA_cert.pem
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrases for ./demoCA/private/cakey.pem:
Check that the request matches the signature
Signature OK
Certificate Details:
  Serial Number:
    80:f0:67:6a:66:d4:80:3b
  Validity
    Not Before: Apr  9 11:14:51 2012 GMT
    Not After : Apr  9 11:14:51 2013 GMT
  Subject:
    countryName           = qq
    stateOrProvinceName   = ww
    organisationName      = rr
    organisationalUnitName = tt
    commonName            = yy
    emailAddress          = uu
  X509v3 extensions:
    X509v3 Basic Constraints:
      CA:FALSE
  Netscape Comment:
    OpenSSL Generated Certificate
  X509v3 Subject Key Identifier:
    07:03:1A:71:4C:17:13:1F:6A:1E:BD:94:1B:96:65:AD:00:3B:0B:D4:66
  X509v3 Authority Key Identifier:
    keyid:2A:1B:1F:6A:1E:BD:94:1B:96:65:AD:00:3B:0B:D4:66
Certificate is to be certified until Apr  9 11:14:51 2013 GMT (365 days)
Sign the certificate? [y/N]:y
Failed to update database
TXT_DB error number 2
error:10
error:10
```

#数据库更新失败？求解；

#重新生成 CA，结果如下；

```
Check that the request matches the signature
Signature OK
Certificate Details:
  Serial Number:
    90:03:12b:2c:26:6e:64:13
  Validity
    Not Before: Apr  9 11:19:16 2012 GMT
    Not After : Apr  9 11:19:16 2013 GMT
  Subject:
    countryName           = qq
    stateOrProvinceName   = ww
    organisationName      = rr
    organisationalUnitName = tt
    commonName            = yy
    emailAddress          = uu
  X509v3 extensions:
    X509v3 Subject Key Identifier:
      CC:3B:02:143:6B:26:0E:EB:3A:6F:C3:EF:2A:D1:DE:0B:3B:27:5D:1C
    X509v3 Authority Key Identifier:
      keyid:CC:3B:02:143:6B:26:0E:EB:3A:6F:C3:EF:2A:D1:DE:0B:3B:27:5D:1C
  X509v3 Basic Constraints:
    CA:TRUE
Certificate is to be certified until Apr  9 11:19:16 2013 GMT (1095 days)
Write out database with 1 new entries
Data Base Updated
```

#结果还是数据库更新失败！



#所以，原因是什么呢？是因为没有指定 CA 文件么？试试

```
> ca -in CA_req.pem -cert ./demoCA/cacert.pem -out CA_cert.pem
```

```
Certificate is to be certified until Apr  9 11:29:05 2013 GMT (365 days)
Sign the certificate? [y/n]:y
failed to update database
TXT_DB error number 2
```

#还是失败，是不是 Subj 不能完全一样呢？试试

```
> req -new -out CA_req1.pem -keyout CA_key1.pem
```

```
> ca -in CA_req1.pem -cert ./demoCA/cacert.pem -out CA_cert.pem
```

```
Enter pass phrase for ./demoCA/private/cacert.pem:
Check that the request matches the signature
signature ok
Certificate Details:
  Serial Number:
    96:6312B12a:26:6a:64:14
  Validity
    Not Before: Apr  9 12:08:33 2013 GMT
    Not After : Apr  9 12:08:33 2013 GMT
  Subject:
    CountryName           = QQ
    StateOrProvinceName   = YY
    OrganizationName       = EE
    OrganizationalUnitName = LL
    CommonName             = VVV
    emailAddress           = GG
  X509v3 extensions:
    CA:FALSE
    Netscape Comment:
      OpenSSL Generated Certificate
    X509v3 Subject Key Identifier:
      DB:DE1001B51CA10A10F1B211210A1C71CB1BD12D14210F1171251971A6
    X509v3 Authority Key Identifier:
      keyid:CC:3810214316B12610E1E813A18F1C31EF12A1D11DE1CB13B12715D11C
Certificate is to be certified until Apr  9 12:08:33 2013 GMT (365 days)
Sign the certificate? [y/n]:y
1 out of 1 certificate requests certified, commit? [y/n]:y
Write out database with 1 new entries
Data Base Updated
```

#成功了！有木有 ^\_^

#所以，问题在于文本数据库的 Key 基于 Subj 的，不能完全一样！

-----The End-----