openssl 简介

BBS 水木清华站 (Fri Nov 10 20:19:30 2000)

前言

不久前接到有关 ssl 的活， 结果找遍中文网站资料实在奇缺。感觉是好象现在国内做这个技术的人不多所有有兴趣写点东西来介绍一下。

我使用的 ssl 的 toolkit 是 openssl 就用 openssl 做例子来讲解

openssl 实在太大了，指令也多，API 也多，更严重的是 它的 API 没有说明。我打算漫漫说清楚其主要指令的用法，主要 API 的中文说明，以及使用/编程的方法。

工作量很大，因为我接触它也没几个月，现在大概 完成了 1/10 吧， 先把目前自己的一些心得，找到的资料 和一些翻译出来的东西贴出来，希望对研究 ssl 的人有帮助

openssl 简介－证书
http://bbs.chinaunix.net/forum/viewtopic.php?p=3161562#3161585
openssl 简介－加密算法
http://bbs.chinaunix.net/forum/viewtopic.php?p=3161562#3161685
openssl 简介－协议
http://bbs.chinaunix.net/forum/viewtopic.php?p=3161562#3161727
openssl 简介－入门
http://bbs.chinaunix.net/forum/viewtopic.php?p=3161562#3162073
openssl 简介－指令 verify
http://bbs.chinaunix.net/forum/viewtopic.php?p=3161562#3173096
openssl 简介－指令 asn1parse
http://bbs.chinaunix.net/forum/viewtopic.php?p=3161562#3173120
openssl 简介－指令 ca
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173126#3173126
openssl 简介－指令 cipher
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173132#3173132
openssl 简介－指令 dgst
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173136#3173136
openssl 简介－指令 dhparam
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173142#3173142
openssl 简介－指令 enc
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173146#3173146
openssl 简介－指令 gendsa
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173149#3173149
openssl 简介－指令 genrsa
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173152#3173152
openssl 简介－指令 passwd

http://bbs.chinaunix.net/forum/viewtopic.php?p=3173154#3173154
openssl 简介－指令 pkcs7
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173156#3173156
openssl 简介－指令 rand
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173158#3173158
openssl 简介－指令 req
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173162#3173162
openssl 简介－指令 rsa
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173164#3173164
openssl 简介－指令 rsautl
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173168#3173168
openssl 简介－指令 s_client
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173171#3173171
openssl 简介－指令 s_server
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173175#3173175
openssl 简介－指令 sess_id
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173176#3173176
openssl 简介－指令 speed
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173178#3173178
openssl 简介－指令 version
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173181#3173181
openssl 简介－指令 x509
http://bbs.chinaunix.net/forum/viewtopic.php?p=3173188#3173188

证书

证书就是数字化的文件，里面有一个实体(网站，个人等)的公共密钥和其他的属性，如名称等。该公共密钥只属于某一个特定的实体，它的作用是防止一个实体假装成另外一个实体。

证书用来保证不对称加密算法的合理性。想想吧，如果没有证书记录，那么假设某俩人 A 与 B 的通话过程如下：

这里假设 A 的 publickey 是 K1,privatekey 是 K2，B 的 publickey 是 K3,privatekey 是 K4

xxxxxx(kn)表示用 kn 加密过的一段文字 xxxxxx

A-----〉hello(plaintext)----------〉B
A〈---------hello(plaintext)〈---------B
A〈---------Bspublickey〈----------B
A---------〉spublickey(K1)--------〉B
.....

如果 C 想假装成 B,那么步骤就和上面一样。
A-----〉hello(plaintext)----------〉C
A〈---------hello(plaintext)〈---------C

注意下一步，因为 A 没有怀疑 C 的身份，所以他理所当然的接受了 C 的 publickey,并且使用这个 key 来继续下面的通信。

A〈---------Cspublickey〈----------C
A---------〉Aspublickey(K1)--------〉C
.....

这样的情况下 A 是没有办法发觉 C 是假的。如果 A 在通话过程中要求取得 B 的证书，并且验证证书里面记录的名字，如果名字和 B 的名字不符合，就可以发现对方不是 B.验证 B 的名字通过再从证书里面提取 B 的公用密钥，继续通信过程。

那么，如果证书是假的怎么办？或者证书被修改过了怎么办？慢慢看下来吧。

证书最简单的形式就是只包含有证书拥有者的名字和公用密钥。当然现在用的证书没这么简单，里面至少还有证书过期的 deadline,颁发证书的机构名称，证书系列号，和一些其他可选的信息。最重要的是，它包含了证书颁发机构(certificationauthority 简称 CA)的签名信息。

我们现在常用的证书是采用 X.509 结构的，这是一个国际标准证书结构。任何遵循该标准的应用程序都可以读，写 X509 结构的证书。

通过检查证书里面的 CA 的名字，和 CA 的签名，就知道这个证书的确是由该 CA 签发的然后，你就可以简单证书里面的接收证书者的名字，然后提取公共密钥。这样做建立的基础是，你信任该 CA,认为该 CA 没有颁发错误的证书。

CA 是第三方机构，被你信任，由它保证证书的确发给了应该得到该证书的人。CA 自己有一个庞大的 publickey 数据库，用来颁发给不同的实体。

这里有必要解释一下，CA 也是一个实体，它也有自己的公共密钥和私有密钥，否则怎么做数字签名？它也有自己的证书，你可以去它的站点 down 它的证书得到它的公共密钥。

一般 CA 的证书都内嵌在应用程序中间。不信你打开你的 IE,在 internet 选项里面选中"内容",点击"证书",看看那个"中间证书发行机构"和"委托根目录发行机构",是不是有一大堆 CA 的名称？也有时 CA 的证书放在安全的数据库里面。

当你接受到对方的证书的时候，你首先会去看该证书的 CA,然后去查找自己的 CA 证书数据库，看看是否找的到，找不到就表示自己不信任该 CA,那么就告吹本次连接。找到了的话就用该 CA 的证书里面的公用密钥去检查 CA 在证书上的签名。

这里又有个连环的问题，我怎么知道那个 CA 的证书是属于那个 CA 的？人家不能造假吗？

解释一下吧。CA 也是分级别的。最高级别的 CA 叫 RootCAs,其他 cheap 一点的 CA 的证书由他们来颁发和签名。这样的话，最后的保证就是：我们信任 RootCAs.那些有 RootCAs 签名过的证书的 CA 就可以来颁发证书给实体或者其他 CA 了。

你不信任 RootCAs?人民币由中国人民银行发行，运到各个大银行，再运到地方银行，你从地方银行取人民币的时候不信任发行它的中国人民银行吗？RootCAs 都是很权威的机构，没有必要担心他们的信用。

那 RootCAs 谁给签名?他们自己给自己签名,叫自签名.

说了这么多，举个 certificate 的例子吧,对一些必要的 item 解释一下。

CertificateExample
Certificate:
Data:
Version:1(0x0)
SerialNumber://系列号
02:41:00:00:16
SignatureAlgorithm:md2WithRSAEncryption//CA 同志的数字签名的算法
Issuer:C=US,O=RSADataSecurity,Inc.,OU=Commercial//CA 自报家门
Certification
Authority
Validity
NotBefore:Nov418:58:341994GMT//证书的有效期
NotAfter:Nov318:58:341999GMT
Subject:C=US,O=RSADataSecurity,Inc.,OU=Commercial
CertificationAuthority
SubjectPublicKeyInfo:
PublicKeyAlgorithm:rsaEncryption
RSAPublicKey:(1000bit)
Modulus(1000bit):
00:a4:fb:81:62:7b:ce:10:27:dd:e8:f7:be:6c:6e:
c6:70:99:db:b8:d5:05:03:69:28:82:9c:72:7f:96:
3f:8e:ec:ac:29:92:3f:8a:14:f8:42:76:be:bd:5d:
03:b9:90:d4:d0:bc:06:b2:51:33:5f:c4:c2:bf:b6:
8b:8f:99:b6:62:22:60:dd:db:df:20:82:b4:ca:a2:
2f:2d:50:ed:94:32:de:e0:55:8d:d4:68:e2:e0:4c:
d2:cd:05:16:2e:95:66:5c:61:52:38:1e:51:a8:82:
a1:c4:ef:25:e9:0a:e6:8b:2b:8e:31:66:d9:f8:d9:
fd:bd:3b:69:d9:eb
Exponent:65537(0x10001)
SignatureAlgorithm:md2WithRSAEncryption

76:b5:b6:10:fe:23:f7:f7:59:62:4b:b0:5f:9c:c1:68:bc:49:
bb:b3:49:6f:21:47:5d:2b:9d:54:c4:00:28:3f:98:b9:f2:8a:
83:9b:60:7f:eb:50:c7:ab:05:10:2d:3d:ed:38:02:c1:a5:48:
d2:fe:65:a0:c0:bc:ea:a6:23:16:66:6c:1b:24:a9:f3:ec:79:
35:18:4f:26:c8:e3:af:50:4a:c7:a7:31:6b:d0:7c:18:9d:50:
bf:a9:26:fa:26:2b:46:9c:14:a9:bb:5b:30:98:42:28:b5:4b:
53:bb:43:09:92:40:ba:a8:aa:5a:a4:c6:b6:8b:57:4d:c5

其实这是我们看的懂的格式的证书内容，真正的证书都是加密过了的，如下：

-----BEGINCERTIFICATE-----

MIIDcTCCAtqgAwIBAgIBADANBgkqhkiG9w0BAQQFADCBiDELMAkGA1UEBhMCQ0gx
EjAQBgNVBAgTCWd1YW5nZG9uZzESMBAGA1UEBxMJZ3Vhbmd6aG91MREwDwYDVQQK
Ewhhc2lhaW5mbzELMAkGA1UECxMCc3cxDjAMBgNVBAMTBWhlbnJ5MSEwHwYJKoZI
hvcNAQkBFhJmb3JkZXNpZ25AMjFjbi5jb20wHhcNMDAwODMwMDc0MTU1WhcNMDEw
ODMwMDc0MTU1WjCBiDELMAkGA1UEBhMCQ0gxEjAQBgNVBAgTCWd1YW5nZG9uZzES
MBAGA1UEBxMJZ3Vhbmd6aG91MREwDwYDVQQKEwhhc2lhaW5mbzELMAkGA1UECxMC
c3cxDjAMBgNVBAMTBWhlbnJ5MSEwHwYJKoZIhvcNAQkBFhJmb3JkZXNpZ25AMjFj
bi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMDYArTAhLIFacYZwP30
Zu63mAkgpAjVHaIsIEJ6wySIZl2THEHjJ0kS3i8lyMqcl7dUFcAXlLYi2+rdktoG
jBQMOtOHv1/cmo0vzuf38+NrAZSZT9ZweJfllp8W9uyz8Dv5hekQgXFg/l3L+HSx
wNvQalaOEw2nyf45/np/QhNpAgMBAAGjgegwgeUwHQYDVR0OBBYEFKBL7xGeHQSm
ICH5wBrOiqNFiildMIG1BgNVHSMEga0wgaqAFKBL7xGeHQSmICH5wBrOiqNFiild
oYGOpIGLMIGIMQswCQYDVQQGEwJDSDESMBAGA1UECBMJZ3Vhbmdkb25nMRIwEAYD
VQQHEwlndWFuZ3pob3UxETAPBgNVBAoTCGFzaWFpbmZvMQswCQYDVQQLEwJzdzEO
MAwGA1UEAxMFaGVucnkxITAfBgkqhkiG9w0BCQEWEmZvcmRlc2lnbkAyMWNuLmNv
bYIBADAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBBAUAA4GBAGQa9HK2mixM7ML7
0jZr1QJUHrBoabX2AbDchb4Lt3qAgPOktTc3F+K7NgB3WSVbdqC9r3YpS23RexU1
aFcHihDn73s+PfhVjpT8arC1RQDg9bDPvUUYphdQC0U+HF72/CvxGCTqpnWiqsgw
xqeog0A8H3doDrffw8Zb7408+Iqf

-----ENDCERTIFICATE-----

证书都是有寿命的。就是上面的那个 NotBefore 和 NotAfter 之间的日子。过期的证书，如果没有特殊原因，都要摆在证书回收列(certificaterevocationlist)里面.证书回收列，英文缩写是 CRL.比如一个证书的 key 已经被破了，或者证书拥有者没有权力再使用该证书，该证书就要考虑作废。CRL 详细记录了所有作废的证书。

CRL 的缺省格式是 PEM 格式。当然也可以输出成我们可以读的文本格式。下面有个 CRL 的例子。

-----BEGINX509CRL-----

MIICjTCCAfowDQYJKoZIhvcNAQECBQAwXzELMAkGA1UEBhMCVVMxIDAeBgNVBAoT
F1JTQSBEYXRhIFNlY3VyaXR5LCBJbmMuMS4wLAYDVQQLEyVTZWN1cmUgU2VydmVy
IENlcnRpZmljYXRpb24gQXV0aG9yaXR5Fw05NTA1MDIwMjEyMjZaFw05NTA2MDEw
MDAxNDlaMIIBaDAWAgUCQQAABBcNOTUwMjAxMTcyNDI2WjAWAgUCQQAACRcNOTUw
MjEwMDIxNjM5WjAWAgUCQQAADxcNOTUwMjI0MDAxMjQ5WjAWAgUCQQAADBcNOTUw
MjI1MDA0NjQ0WjAWAgUCQQAAGxcNOTUwMzEzMTg0MDQ5WjAWAgUCQQAAFhcNOTUw
MzE1MTkxNjU0WjAWAgUCQQAAGhcNOTUwMzE1MTk0MDQxWjAWAgUCQQAAHxcNOTUw
MzI0MTk0NDMzWjAWAgUCcgAABRcNOTUwMzI5MjAwNzExWjAWAgUCcgAAERcNOTUw
MzMwMDIzNDI2WjAWAgUCQQAAIBcNOTUwNDA3MDExMzIxWjAWAgUCcgAAHhcNOTUw
NDA4MDAwMjU5WjAWAgUCcgAAQRcNOTUwNDI4MTcxNzI0WjAWAgUCcgAAOBcNOTUw
NDI4MTcyNzIxWjAWAgUCcgAATBcNOTUwNTAyMDIxMjI2WjANBgkqhkiG9w0BAQIF
AAN+AHqOEJXSDejYy0UwxxrH/9+N2z5xu/if0J6qQmK92W0hW158wpJg+ovV3+wQ
wvIEPRL2rocL0tKfAsVq1IawSJzSNgxG0lrcla3MrJBnZ4GaZDu4FutZh72MR3Gt
JaAL3iTJHJD55kK2D/VoyY1djlsPuNh6AEgdVwFAyp0v

-----ENDX509CRL-----

下面是文本格式的 CRL 的例子。

ThefollowingisanexampleofaCRLintextformat:

issuer=/C=US/O=RSADataSecurity,Inc./OU=SecureServerCertification
Authority
lastUpdate=May202:12:261995GMT
nextUpdate=Jun100:01:491995GMT
revoked:serialNumber=027200004CrevocationDate=May202:12:261995GMT
revoked:serialNumber=0272000038revocationDate=Apr2817:27:211995GMT
revoked:serialNumber=0272000041revocationDate=Apr2817:17:241995GMT
revoked:serialNumber=027200001ErevocationDate=Apr800:02:591995GMT
revoked:serialNumber=0241000020revocationDate=Apr701:13:211995GMT
revoked:serialNumber=0272000011revocationDate=Mar3002:34:261995GMT
revoked:serialNumber=0272000005revocationDate=Mar2920:07:111995GMT
revoked:serialNumber=024100001FrevocationDate=Mar2419:44:331995GMT
revoked:serialNumber=024100001ArevocationDate=Mar1519:40:411995GMT
revoked:serialNumber=0241000016revocationDate=Mar1519:16:541995GMT
revoked:serialNumber=024100001BrevocationDate=Mar1318:40:491995GMT
revoked:serialNumber=024100000CrevocationDate=Feb2500:46:441995GMT
revoked:serialNumber=024100000FrevocationDate=Feb2400:12:491995GMT
revoked:serialNumber=0241000009revocationDate=Feb1002:16:391995GMT
revoked:serialNumber=0241000004revocationDate=Feb117:24:261995GMT

总结一下 X.509 证书是个什么东东吧。它实际上是建立了公共密钥和某个实体之间联系的数字化的文件。它包含的内容有：

版本信息,X.509 也是有三个版本的。

系列号
证书接受者名称
颁发者名称
证书有效期
公共密钥
一大堆的可选的其他信息
CA 的数字签名

证书由 CA 颁发，由 CA 决定该证书的有效期，由该 CA 签名。每个证书都有唯一的系列号。证书的系列号和证书颁发者来决定某证书的唯一身份。

openssl 有四个验证证书的模式。你还可以指定一个 callback 函数，在验证证书的时候会自动调用该 callback 函数。这样可以自己根据验证结果来决定应用程序的行为。具体的东西在以后的章节会详细介绍的。

openssl 的四个验证证书模式分别是：

SSL_VERIFY_NONE:完全忽略验证证书的结果。当你觉得握手必须完成的话，就选用这个选项。其实真正有证书的人很少,尤其在中国。那么如果 SSL 运用于一些免费的服务，比如 email 的时候，我觉得 server 端最好采用这个模式。

SSL_VERIFY_PEER:希望验证对方的证书。不用说这个是最一般的模式了.对 client 来说，如果设置了这样的模式，验证 server 的证书出了任何错误，SSL 握手都告吹.对 server 来说,如果设置了这样的模式,client 倒不一定要把自己的证书交出去。如果 client 没有交出证书，server 自己决定下一步怎么做。

SSL_VERIFY_FAIL_IF_NO_PEER_CERT:这是 server 使用的一种模式，在这种模式下，server 会向 client 要证书。如果 client 不给，SSL 握手告吹。

SSL_VERIFY_CLIENT_ONCE：这是仅能使用在 sslsessionrenegotiation 阶段的一种方式。什么是 SSLsessionrenegotiation?以后的章节再解释。我英文差点，觉得这个词组也很难翻译成相应的中文。以后的文章里，我觉得很难直接翻译的单词或词组，都会直接用英文写出来。如果不是用这个模式的话,那么在 regegotiation 的时候，client 都要把自己的证书送给 server,然后做一番分析。这个过程很消耗 cpu 时间的，而这个模式则不需要 client 在 regotiation 的时候重复送自己的证书了。

加密算法

要理解 ssl 先要知道一些加密算法的常识.

加密算法很容易理解啦，就是把明文变成人家看不懂的东西，然后送给自己想要的送到的地方，接收方用配套的解密算法又把密文解开成明文，这样就不怕在路世上如果密文给人家截获而泄密。

加密算法有俩大类，第一种是不基于 KEY 的，举个简单的例子，我要加密"fordesign"这么一串字符，就把每个字符都变成它的后一个字符，那么就是"gpseftjhm"了，这样的东西人家当然看不明白，接收方用相反的方法就可以得到原文。当然这只是个例子，现在应该没人用这么搞笑的加密算法了吧。

不基于 KEY 的加密算法好象一直用到了计算机出现。我记得古中国军事机密都是用这种方式加密的。打战的时候好象军队那些电报员也要带着密码本，也应该是用这种方式加密的。这种算法的安全性以保持算法的保密为前提。

这种加密算法的缺点太明显了，就是一旦你的加密算法给人家知道，就肯定挂。日本中途岛惨败好象就是密码给老米破了。设计一种算法是很麻烦的，一旦给人破了就没用了，这也忒浪费。

我们现在使用的加密算法一般是基于 key 的，也就是说在加密过程中需要一个 key,用这个 key 来对明文进行加密。这样的算法即使一次被破，下次改个 key,还可以继续用。

key 是一个什么东西呢？随便你，可以是一个随机产生的数字，或者一个单词，啥都行，只要你用的算法认为你选来做 key 的那玩意合法就行。

这样的算法最重要的是：其安全性取决于 key,一般来说取决于 key 的长度。也就是说应该保证人家在知道这个算法而不知道 key 的情况下，破解也相当困难。其实现在常用的基于 KEY 的加密算法在网络上都可以找到，很多革命同志(都是老外)都在想办法破解基于 key 的加密算法又包括俩类：对称加密和不对称加密。对称加密指的是双方使用完全相同的 key，最常见的是 DES.DES3,RC4 等。对称加密算法的原理很容易理解，通信一方用 KEK 加密明文，另一方收到之后用同样的 KEY 来解密就可以得到明文。

不对称加密指双方用不同的 KEY 加密和解密明文，通信双方都要有自己的公共密钥和私有密钥。举个例子比较容易理解，我们们假设通信双方分别是 A,B.
A,拥有 KEY_A1,KEY_A2,其中 KEY_A1 是 A 的私有密钥，KEY_A2 是 A 的公共密钥。
B,拥有 KEY_B1,KEY_B2,其中 KEY_B1 是 B 的私有密钥，KEY_B2 是 B 的公共密钥。

公共密钥和私有密钥的特点是，经过其中任何一把加密过的明文，只能用另外一把才能够解开。也就是说经过 KEY_A1 加密过的明文，只有 KEY_A2 才能够解密，反之亦然。

通信过程如下：

A-------->;KEY_A2---------->;B
A<--------KEY_B2<----------A

这个过程叫做公共密钥交换，老外管这叫 keyexchange.之后 A 和 B 就分别用对方的公共密钥加密，用自己的私有密钥解密。

一般公共密钥是要发布出去的，然后你通过自己的私有密钥加密明文，人家用你的公共密钥解密，如果能解开，那么说明你是加密人，这就是 SSL 使用的验证机制。

常用的不对称加密一般有 RSA,DSA,DH 等。我们一般使用 RSA.

数字签名也是不对称加密算法的一个重要应用，理解它对于理解 SSL 很重要的，放在这里一起介绍一下。

签名是什么大家都很熟悉吧？证明该东西是你写的，是你发布的，你就用签名搞定。看看那些重要文件都要头头签名。数字签名就是数字化的签名了。记得公用密钥和私有密钥的特征吗？只有你一个人有你自己的私有密钥。而你的公用密钥是其他人都知道的了。那么你在写完一封邮件之后，用自己的私有密钥加密自己的名字，接收人用你的公共密钥解开一看，哦，是你发的。这就是你的数字签名过程了。

上面的解释是很简化的了，其实数字签名比这个复杂多了，但我们没有了解的必要，知道数字签名是这么一回事就可以了。

还有一种我们需要知道的加密算法，其实我不觉得那是加密算法，应该叫哈希算法，英文是 messagedigest,是用来把任何长度的一串明文以一定规则变成固定长度的一串字符串。它在 SSL 中的作用也很重要，以后会慢慢提及的。一般使用的是 MD5,SHA.

base64 不是加密算法，但也是 SSL 经常使用的一种算法，它是编码方式，用来把 asc 码和二进制码转来转去的。

具体的加密解密过程我们不需要了解，因为 SSL 根本不关心。但了解加密算法的一些基本原理是必要的，否则很难理解 SSL。

对加密算法的细节有兴趣的同志，可以去网络上找这些加密算法的原理的文章和实现的程序来研究，不过先学数论吧。

协议

SSL(SecureSocketLayer)是 netscape 公司提出的主要用于 web 的安全通信标准,分为 2.0 版和 3.0 版.TLS(TransportLayerSecurity)是 IETF 的 TLS 工作组在 SSL3.0 基础之上提出的安全通信标准,目前版本是 1.0,即 RFC2246.SSL/TLS 提供的安全机制可以保证应用层数据在互联网络传输不被监听,

伪造和窜改.

一般情况下的网络协议应用中，数据在机器中经过简单的由上到下的几次包装，就进入网络，如果这些包被截获的话，那么可以很容易的根据网络协议得到里面的数据.由网络监听工具可以很容易的做到这一点。

SSL 就是为了加密这些数据而产生的协议，可以这么理解，它是位与应用层和 TCP/IP 之间的一层，数据经过它流出的时候被加密，再往 TCP/IP 送，而数据从 TCP/IP 流入之后先进入它这一层被解密，同时它也能够验证网络连接俩端的身份。

它的主要功能就是俩个：
一：加密解密在网络中传输的数据包,同时保护这些数据不被修改,和伪造。
二：验证网络对话中双方的身份

SSL 协议包含俩个子协议，一个是包协议，一个是握手协议。包协议是说明 SSL 的数据包应该如何封装的。握手协议则是说明通信双方如何协商共同决定使用什么算法以及算法使用的 key。很明显包协议位于握手协议更下一层。我们暂时对包协议的内容没有兴趣。

SSL 握手过程说简单点就是：通信双方通过不对称加密算法来协商好一个对称加密算法以及使用的 key,然后用这个算法加密以后所有的数据完成应用层协议的数据交换。

握手一般都是由 client 发起的，SSL 也不例外。

1、client 送给 server 它自己本身使用的 ssl 的 version(ssl 一共有三个 version),加密算法的一些配置，和一些随机产生的数据，以及其他在 SSL 协议中需要用到的信息。

2、server 送给 client 它自己的 SSL 的 version,加密算法的配置，随机产生的数据，还会用自己的私有密钥加密 SERVER-HELLO 信息。Server 还同时把自己的证书文件给送过去。同时有个可选的项目，就是 server 可以要求需要客户的 certificate。

3、client 就用 server 送过来的 certificate 来验证 server 的身份。如果 server 身份验证没通过，本次通信结束。通过证书验证之后，得到 server 的公共密钥，解开 server 送来的被其用私有密钥加密过的 SERVER-HELLO 信息，看看对头与否。如果不对，说明对方只有该 server 的公共密钥而没有私有密钥，必是假的。通信告吹。

4、client 使用到目前为止所有产生了的随机数据(sharedsecret),client 产生本次握手中的 premastersecret(这个步骤是有可能有 server 的参与的，由他们使用的加密算法决定),并且把这个用 server 的公共密钥加密，送回给 server.如果 server 要求需要验证 client,那么 client 也需要自己把自己的证书送过去，同时送一些自己签过名的数据过去。

SSL 协议有俩种技术来产生 sharedsecret(真不好意思，又是一个很难意译的词组),
一种是 RSA,一种是 EDH.

RSA 就是我们上一章说过的一种不对称加密算法。首先 server 把自己的 RSA 公共密钥送给 client,client 于是用这个 key 加密一个随机产生的值(这个随机产生的值就是 sharedsecret)，再把结果送给 server.

EDH 也是一种不对称加密算法，但它与 RSA 不同的是，它好象没有自己固定的公共密钥和私有密钥，都是在程序跑起来的时候产生的，用完就 K 掉。其他的步骤俩者就差不多了。

RSA,DSA,DH 三种不对称加密算法的区别也就在这里。RSA 的密钥固定，后俩个需要一个参数来临时生成 key.DH 甚至要求双方使用同样的参数，这个参数要事先指定。如果 SSL 库没有 load 进这个参数，DH 算法就没办法用。DSA 没研究过。

5、Server 验证完 client 的身份之后，然后用自己的私有密钥解密得到 premastersecret 然后双方利用这个 premastersecret 来共同协商，得到 mastersecret.

6、双方用 master 一起产生真正的 sessionkey,着就是他们在剩下的过程中的对称加密的 key 了。这个 key 还可以用来验证数据完整性。双方再交换结束信息。握手结束。

接下来双方就可以用协商好的算法和 key 来用对称加密算法继续下面的过程了。

很简单吧？其实要复杂一些的，我简化了很多来说。

不过还是有个问题，喜欢捣蛋的人虽然看不懂他们在交流些什么，但篡改总可以吧？
记得我们在加密算法里面介绍过的哈希算法吗？就是为了对付这种捣蛋者的。在每次送信息的时候，附带把整条信息的哈希值也送过去，接收方收到信息的时候，也把收到的内容哈希一把，然后和对方送来的哈希值对比一下，看看是否正确。捣蛋者如果乱改通信内容，哈希出来的值是不同的，那么就很容易被发现了。

但这样子，捣蛋者至少可以学舌。他可以把之前监听到的内容重复发给某一方，而这些内容肯定是正确的,无法验证出有问题的。哎，SSL 是怎么对付这种人的我还没看出来。有篇文章说：多放点随机数在信息里可以对付，我也没去研究这句话是什么意思。

入门

实现了 SSL 的软件不多，但都蛮优秀的。首先，netscape 自己提出来的概念，当然自己会实现一套了。netscape 的技术蛮优秀的，不过我没用过他们的 ssl-toolkit.甚至连名字都没搞清楚。

1995 年，eric.young 开始开发 openssl, 那时候叫 ssleay.一直到现在，openssl 还在不停的修改和新版本的发行之中。openssl 真够大的，我真佩服 eric 的水平和兴趣。这些 open/free 的斗士的精神是我写这个系列的主要动力，虽然写的挺烦的。
ps: eric 现在去了 RSA 公司做，做了一个叫 SSL-C 的 toolkit, 其实和 openssl 差不多。估计应该比 openssl 稳定，区别是这个是要银子的，而且几乎所有低层的函数都不提供直接调用了。那多没意思。

去 www.openssl.org down openssl 吧，最新的是 0.9.6 版。

安装是很简单的。我一直用的是 sun sparc 的机器，所以用 sun sparc 的机器做例子。

gunzip -d openssl.0.9.6.tar.gz

tar -xf openssl.0.9.6.tar

mv openssl.0.9.6 openssl

cd openssl

./configure --prefix=XXXXX --openssldir=XXXXXXXX

(这里 prefix 是你想安装 openssl 的地方， openssldir 就是你 tar 开的 openssl 源码的地方。好象所有的出名点的 free software 都是这个操行，configure, make , make test, make install, 搞定。)

./make(如果机器慢，这一步的时候可以去洗个澡，换套衣服)

./make test

./make install

OK, 如果路上没有什么问题的话，搞定。

经常有人报 bug, 在 hp-ux, sgi 上装 openssl 出问题，我没试过，没发言权。

现在可以开始玩 openssl 了。

注意： 我估计 openssl 最开始是在 linux 下开发的。大家可以看一看在 linxu 下有这么一个文件：/dev/urandom, 在 sparc 下没有。这个文件有什么用？你可以随时找它要一个随机数。在加密算法产生 key 的时候，我们需要一颗种子：seed。这个 seed 就是找/dev/urandom 要的那个随机数。那么在 sparc 下，由于没有这么一个设备，很多 openssl 的函数会报错："PRNG not seeded". 解决方法是：在你的~/.profile 里面添加一个变量$RANDFILE， 设置如下：

$RANDFILE=$HOME/.rnd

然后在$HOME 下 vi .rnd, 随便往里面乱输入一些字符，起码俩行。

很多 openssl 的函数都会把这个文件当 seed, 除了 openssl rsa, 希望 openssl 尽快修改这个 bug.

如果用 openssl 做 toolkit 编程， 则有其他不太安全的解决方法。以后讲到 openssl 编程的章节会详细介绍。

先生成自己的私有密钥文件，比如叫 server.key

openssl genrsa -des3 -out server.key 1024

genras 表示生成 RSA 私有密钥文件，-des3 表示用 DES3 加密该文件，1024 是我们的 key 的长度。一般用 512 就可以了，784 可用于商业行为了，1024 可以用于军事用途了。

当然，这是基于现在的计算机的速度而言，可能没过几年 1024 是用于开发测试，2048 用于一般用途了。

生成 server.key 的时候会要你输入一个密码，这个密钥用来保护你的 server.key 文件，这样即使人家偷走你的 server.key 文件，也打不开，拿不到你的私有密钥。

openssl rsa -noout -text -in server.key

可以用来看看这个 key 文件里面到底有些什么东西(不过还是看不懂)

如果你觉得 server.key 的保护密码太麻烦想去掉的话：

openssl rsa -in server.key -out server.key.unsecure

不过不推荐这么做

下一步要得到证书了。得到证书之前我们要生成一个 Certificate Signing Request.

CA 只对 CSR 进行处理。

openssl req -new -key server.key -out server.csr

生成 CSR 的时候屏幕上将有提示,依照其指示一步一步输入要求的信息即可.

生成的 csr 文件交给 CA 签名后形成服务端自己的证书.怎么交给 CA 签名？

自己去 www.verisign.com 慢慢看吧。

如果是自己玩下，那么自己来做 CA 吧。openssl 有很简单的方法做 CA.但一般只好在开发的时候或者自己玩的时候用，真的做出产品，还是使用正规的 CA 签发给你的证书吧

在你 make install 之后，会发现有个 misc 的目录，进去，运行 CA.sh -newca，他会找你要 CA 需要的一个 CA 自己的私有密钥密码文件。没有这个文件？按回车创建，输入密码来保护这个密码文件。之后会要你的一个公司信息来做 CA.crt 文件。最后在当前目录下多了一个./demoCA 这样的目录../demoCA/private/cakey.pem 就是 CA 的 key 文件啦，

./demoCA/cacert.pem 就是 CA 的 crt 文件了。把自己创建出来的 server.crt 文件 copy 到 misc 目录下，mv 成 newreq.pem,然后执行 CA.sh -sign, ok,

得到回来的证书我们命名为 server.crt.

看看我们的证书里面有些什么吧

openssl x509 -noout -text -in server.crt

玩是玩过了，openssl 的指令繁多，就象天上的星星。慢慢一个一个解释吧。

指令 verify

用法：

openssl verify【-CApath directory】【-CAfile file】【-purpose purpose】【-untrusted file】【-help】【-issuer_checks】 【-verbose】 【-】 【certificates】

说明： 证书验证工具。

选项

-CApath directory

我们信任的 CA 的证书存放目录。这些证书的名称应该是这样的格式：

xxxxxxxx.0(xxxxxxxx 代表证书的哈希值。 参看 x509 指令的-hash)

你也可以在目录里 touch 一些这样格式文件名的文件，符号连接到真正的证书。

那么这个 xxxxxxxx 我怎么知道怎么得到？x509 指令有说明。

其实这样子就可以了：

openssl x509 -hash -in server.crt

-CAfile file

我们信任的 CA 的证书，里面可以有多个 CA 的证书。

-untrusted file

我们不信任的 CA 的证书。

-purpose purpose

证书的用途。如果这个 option 没有设置，那么不会对证书的 CA 链进行验证。

现在这个 option 的参数有以下几个：
sslclinet
sslserver
nssslserver
smimesign
smimeencrypt
等下会详细解释的。

-help
打印帮助信息。

-verbose
打印出详细的操作信息。

-issuer_checks
打印出我们验证的证书的签发 CA 的证书的之间的联系。
要一次验证多个证书，把那些证书名都写在后面就好了。

验证操作解释：
S/MIME 和本指令使用完全相同的函数进行验证。
我们进行的验证和真正的验证有个根本的区别：
在我们对整个证书链进行验证的时候，即使中途有问题，我们也会验证到最后，而真实的验证一旦有一个环节出问题，那么整个验证过程就告吹。
验证操作包括几个独立的步骤。
首先建立证书链，从我们目前的证书为基础，一直上溯到 Root CA 的证书.
如果中间有任何问题，比如找不到某个证书的颁发者的证书，那么这个步骤就挂。有任何一个证书是字签名的，就被认为是 Root CA 的证书。
寻找一个证书的颁发 CA 也包过几个步骤。在 openssl0.9.5a 之前的版本，如果一个证书的颁发者和另一个证书的拥有着相同，就认为后一个证书的拥有者就是前一个证书的签名 CA.
openssl0.9.6 及其以后的版本中，即使上一个条件成立，还要进行更多步骤的检验。包括验证系列号等。到底有哪几个我也没看明白。
得到 CA 的名称之后首先去看看是否是不信任的 CA, 如果不是，那么才去看看是否是信任的 CA. 尤其是 Root CA, 更是必须是在信任 CA 列表里面。
现在得到链条上所有 CA 的名称和证书了，下一步是去检查第一个证书的用途是否和签发时候批准的一样。其他的证书则必须都是作为 CA 证书而颁发的。
证书的用途在 x509 指令里会详细解释。
过了第二步，现在就是检查对 Root CA 的信任了。可能 Root CA 也是每个都负责不同领域的证书签发。缺省的认为任何一个 Root CA 都是对任何用途的证书有签发权。
最后一步，检查整条证书链的合法性。比如是否有任何一个证书过期了？签名是否是正确的？是否真的是由该证书的颁发者签名的？
任何一步出问题，所有该证书值得怀疑，否则，证书检验通过。

如果验证操作有问题了，那么打印出来的结果可能会让人有点模糊。
一般如果出问题的话，会有类似这样子的结果打印出来：

server.pem: /C=AU/ST=Queensland/O=CryptSoft Pty Ltd/CN=Test CA (1024 bit)
error 24 at 1 depth lookup:invalid CA certificate
第一行说明哪个证书出问题，后面是其拥有者的名字，包括几个字段。
第二行说明错误号，验证出错在第几层的证书，以及错误描述。
下面是错误号及其描述的详细说明,注意，有的错误虽然有定义，
但真正使用的时候永远不会出现。用 unused 标志.
0 X509_V_OK
验证操作没有问题
2 X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT
找不到该证书的颁发 CA 的证书。
3 X509_V_ERR_UNABLE_TO_GET_CRL (unused)
找不到和该证书相关的 CRL
4 X509_V_ERR_UNABLE_TO_DECRYPT_CERT_SIGNATURE
无法解开证书里的签名。
5 X509_V_ERR_UNABLE_TO_DECRYPT_CRL_SIGNATURE (unused)
无法解开 CRLs 的签名。
6 X509_V_ERR_UNABLE_TO_DECODE_ISSUER_PUBLIC_KEY
无法得到证书里的公共密钥信息。
7 X509_V_ERR_CERT_SIGNATURE_FAILURE
证书签名无效
8 X509_V_ERR_CRL_SIGNATURE_FAILURE (unused)
证书相关的 CRL 签名无效
9 X509_V_ERR_CERT_NOT_YET_VALID
证书还没有到有效开始时间
10 X509_V_ERR_CRL_NOT_YET_VALID (unused)
与证书相关的 CRL 还没有到有效开始时间
11 X509_V_ERR_CERT_HAS_EXPIRED
证书过期
12 X509_V_ERR_CRL_HAS_EXPIRED (unused)
与证书相关的 CRL 过期
13 X509_V_ERR_ERROR_IN_CERT_NOT_BEFORE_FIELD
证书的 notBefore 字段格式不对,就是说那个时间是非法格式。
14 X509_V_ERR_ERROR_IN_CERT_NOT_AFTER_FIELD
证书的 notAfter 字段格式不对,就是说那个时间是非法格式。
15 X509_V_ERR_ERROR_IN_CRL_LAST_UPDATE_FIELD (unused)
CRL 的 lastUpdate 字段格式不对。
16 X509_V_ERR_ERROR_IN_CRL_NEXT_UPDATE_FIELD (unused)
CRL 的 nextUpdate 字段格式不对
17 X509_V_ERR_OUT_OF_MEM
操作时候内存不够。这和证书本身没有关系。
18 X509_V_ERR_DEPTH_ZERO_SELF_SIGNED_CERT
需要验证的第一个证书就是字签名证书，而且不在信任 CA 证书列表中。
19 X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN
可以建立证书链，但在本地找不到他们的根？？

: self signed certificate in certificate chain
the certificate chain could be built up using the untrusted certificates
but the root could not be found locally.
20 X509_V_ERR_UNABLE_TO_GET_ISSUER_CERT_LOCALLY
有一个证书的签发 CA 的证书找不到。这说明可能是你的 Root CA 的证书列表不齐全。
21 X509_V_ERR_UNABLE_TO_VERIFY_LEAF_SIGNATURE
证书链只有一个 item, 但又不是字签名的证书。
22 X509_V_ERR_CERT_CHAIN_TOO_LONG (unused)
证书链太长。
23 X509_V_ERR_CERT_REVOKED (unused)
证书已经被 CA 宣布收回。
24 X509_V_ERR_INVALID_CA
某 CA 的证书无效。
25 X509_V_ERR_PATH_LENGTH_EXCEEDED
参数 basicConstraints pathlentgh 超过规定长度
26 X509_V_ERR_INVALID_PURPOSE
提供的证书不能用于请求的用途。
比如链条中某个证书应该是用来做 CA 证书的，但证书里面的该字段说明该证书不是用做 CA 证书的，就是这样子的情况。
27 X509_V_ERR_CERT_UNTRUSTED
Root CA 的证书如果用在请求的用途是不被信任的。
28 X509_V_ERR_CERT_REJECTED
CA 的证书根本不可以用做请求的用途。
29 X509_V_ERR_SUBJECT_ISSUER_MISMATCH
证书颁发者名称和其 CA 拥有者名称不相同。-issuer_checks 被 set 的时候可以检验出来。
30 X509_V_ERR_AKID_SKID_MISMATCH
证书的密钥标志和其颁发 CA 为其指定的密钥标志不同.
31 X509_V_ERR_AKID_ISSUER_SERIAL_MISMATCH
证书系列号与起颁发 CA 为其指定的系列号不同。
32 X509_V_ERR_KEYUSAGE_NO_CERTSIGN
某 CA 的证书用途不包括为其他证书签名。
50 X509_V_ERR_APPLICATION_VERIFICATION
应用程序验证出错。

指令 asn1parse

用法：openssl asn1parse [-inform PEM|DER] [-in filename] [-out filename]
[-noout] [-offset number] [-length number] [-i] [- structure filename]
[-strparse offset]
用途：一个诊断工具，可以对 ASN1 结构的东东进行分析。

ASN1 是什么？一个用来描述对象的标准。要解释的话，文章可以比解释 openssl 结构的文章更长。有兴趣的话自己去网络上找来看吧。

-inform DER|PEM|TXT

输入的格式，DER 是二进制格式，PEM 是 base64 编码格式,TXT 不用解释了吧

-in filename

输入文件的名称，缺省为标准输入。

-out filename

输入文件的名称，输入一般都是 DER 数据。如果没这个项，就没有东西输入咯。该项一般都要和-strparse 一起使用。

-noout

不要输出任何东西(不明白有什么用)

-offset number

从文件的那里开始分析，看到 offset 就应该知道是什么意思了吧。

-length number

一共分析输入文件的长度的多少，缺省是一直分析到文件结束。

-i

根据输出的数据自动缩进。

- structure filename

当你输入的文件包含有附加的对象标志符的时候，使用这个。

这种文件的格式在后面会介绍。

-strparse offset

从由 offset 指定的偏移量开始分析 ASN1 对象。当你碰到一个嵌套的对象时，可以反复使用这个项来一直进到里面的结构捏出你需要的东东。

一般分析完之后输入的东东如下：

```
openssl asn1parse -out temp.ans -i -inform pem < server.crt
    0:d=0 hl=4 l= 881 cons: SEQUENCE
    4:d=1 hl=4 l= 730 cons: SEQUENCE
    ... ....
    172:d=3 hl=2 l= 13 prim: UTCTIME :000830074155Z
    187:d=3 hl=2 l= 13 prim: UTCTIME :010830074155Z
    202:d=2 hl=3 l= 136 cons: SEQUENCE
    205:d=3 hl=2 l= 11 cons: SET
    ... ...
    359:d=3 hl=3 l= 141 prim: BIT STRING
    ... ...
```

本例是一个自签名的证书。每一行的开始是对象在文件里的偏移量。d=xx 是结构嵌套的深度。知道 ASN1 结构的人应该知道，每一个 SET 或者 SEQUENCE 都会让嵌套深度增加 1.

hl=xx 表示当前类型的 header 的长度。l=xx 表示内容的八进制的长度。

-i 可以让输出的东西容易懂一点。

如果没有 ASN.1 的知识，可以省略看这一章。

本例中 359 行就是证书里的公共密钥。可以用-strparse 来看看

openssl asn1parse -out temp.ans -i -inform pem -strparse 359 < server.crt

0:d=0 hl=3 l= 137 cons: SEQUENCE

3:d=1 hl=3 l= 129 prim: INTEGER :C0D802B4C084B20569C619C0FDF
466EEB7980920A408D51DA22C20427AC32488665D931C41E3274912DE2F25C8CA9C97B75
415C01794B622DBEADD92DA068C140C3AD387BF5FDC9A8D2FCEE7F7F3E36B0194994FD67
07897C8969F16F6ECB3F03BF985E910817160FE5DCBF874B1C0DBD06A568E130DA7C9FE3
9FE7A7F421369

135:d=1 hl=2 l= 3 prim: INTEGER :010001

不要试图去看 temp.ans 的内容，是二进制来的，看不懂的。

指令 ca

用途：

模拟 CA 行为的工具.有了它，你就是一个 CA,不过估计是 nobody trusted CA.可以用来给各种格式的 CSR 签名，用来产生和维护 CRL(不记得 CRL 是什么了？去看证书那一章).他还维护着一个文本数据库，记录了所有经手颁发的证书及那些证书的状态。

用法：

openssl ca [-verbose] [-config filename] [-name section] [-gencrl]
[-revoke file] [-crldays days] [-crlhours hours] [-crlexts section]
[-startdate date] [-enddate date] [-days arg] [-md arg] [-policy arg]
[-keyfile arg] [-key arg] [-passin arg] [-cert file] [-in file]
[-out file] [-notext] [-outdir dir] [-infiles] [-spkac file]
[-ss_cert file] [-preserveDN] [-batch] [-msie_hack] [-extensions section]

哇噻，好复杂也。不过用过 GCC 的人应该觉得这么点 flag 还是小 case.

-config filename

指定使用的 configure 文件。

-in filename

要签名的 CSR 文件。

-ss_cert filename

一个有自签名的证书，需要我们 CA 签名，就从这里输入文件名。

-spkac filename

这一段实在没有看懂，也没兴趣，估计和 SPKAC 打交道可能性不大，奉送上英文原文。

a file containing a single Netscape signed public key and challenge and additional field values to be signed by the CA.

SPKAC FORMAT

The input to the -spkac command line option is a Netscape signed public key and challenge. This will usually come from the KEYGEN tag in an HTML form to create a new private key. It is however possible to create SPKACs using the spkac utility.

The file should contain the variable SPKAC set to the value of the SPKAC and also the required DN components as name value pairs. If you need to include the same component twice then it can be preceded by a number and a .

-infiles

如果你一次要给几个 CSR 签名，就用这个来输入，但记得这个选项一定要放在最后。这个项后面的所有东东都被认为是 CSR 文件名参数。

-out filename

签名后的证书文件名。证书的细节也会给写进去。

-outdir directory

摆证书文件的目录。证书名就是该证书的系列号，后缀是.pem

-cert

CA 本身的证书文件名

-keyfile filename

CA 自己的私有密钥文件

-key password

CA 的私有密钥文件的保护密码。

在有的系统上，可以用 ps 看到你输入的指令，所以这个参数要小心点用。

-passin arg

也是一个输入私有密钥保护文件的保护密码的一种方式，可以是文件名，设备名或者是有名管道。程序会把该文件的第一行作为密码读入。（也蛮危险的）。

-verbose

操作过程被详细 printf 出来

-notext

不要把证书文件的明文内容输出到文件中去。

-startdate date

指明证书的有效开始日期。格式是 YYMMDDHHMMSSZ, 同 ASN1 的 UTCTime 结构相同。

-enddate date

指明证书的有效截止日期，格式同上。

-days arg

指明给证书的有效时间，比如 365 天。

-md alg

签名用的哈希算法，比如 MD2, MD5 等。

-policy arg

指定 CA 使用的策略。其实很简单，就是决定在你填写信息生成 CSR 的时候，哪些信息是我们必须的，哪些不是。看看 config 文件里面的 policy 这个 item 就明白了。

-msie_hack

为了和及其古老的证书版本兼容而做出的牺牲品，估计没人会用的，不解释了。

-preserveDN

和-msie_hack 差不多的一个选项。

-batch

设置为批处理的模式，所有的 CSR 会被自动处理。

-extensions section

我们知道一般我们都用 X509 格式的证书，X509 也有几个版本的。如果你在这个选项后面带的那个参数在 config 文件里有同样名称的 key,那么就颁发 X509V3 证书，否则颁发 X509v1 证书。

还有几个关于 CRL 的选项，但我想一般很少人会去用。我自己也没兴趣去研究。

有兴趣的自己看看英文吧。

CRL OPTIONS

-gencrl

this option generates a CRL based on information in the index file.

-crldays num

the number of days before the next CRL is due. That is the days from now to place in the CRL nextUpdate field.

-crlhours num

the number of hours before the next CRL is due.

-revoke filename

a filename containing a certificate to revoke.

-crlexts section

the section of the configuration file containing CRL extensions to include. If no CRL extension section is present then a V1 CRL is created, if the CRL extension section is present (even if it is empty) then a V2 CRL is created. The CRL extensions specified are CRL extensions and not CRL entry extensions. It should be noted that some software (for example Netscape) can't handle V2 CRLs.

相信刚才大家都看到很多选项都和 config 文件有关,那么我们来解释一下 config 文件 make install 之后，openssl 会生成一个全是缺省值的 config 文件:openssl.cnf.也长的很，贴出来有赚篇幅之嫌，xgh 不屑。简单解释一下其中与 CA 有关的 key.

与 CA 有关的 key 都在 ca 这个 section 之中。

```
[ ca ]
default_ca = CA_default
[ CA_default ]
dir = ./demoCA # Where everything is kept
certs = $dir/certs # Where the issued certs are kept
crl_dir = $dir/crl # Where the issued crl are kept
database = $dir/index.txt # database index file.
new_certs_dir = $dir/newcerts # default place for new certs.
certificate = $dir/cacert.pem # The CA certificate
serial = $dir/serial # The current serial number
crl = $dir/crl.pem # The current CRL
private_key = $dir/private/cakey.pem# The private key
RANDFILE = $dir/private/.rand # private random number file
x509_extensions = usr_cert # The extentions to add to the cert
# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs
```

```
# so this is commented out by default to leave a V1 CRL.
# crl_extensions = crl_ext
default_days = 365 # how long to certify for
default_crl_days= 30 # how long before next CRL
default_md = md5 # which md to use.
preserve = no # keep passed DN ordering
# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy = policy_match
# For the CA policy
[ policy_match ]
countryName = match
stateOrProvinceName = match
organizationName = match
organizationalUnitName = optional
commonName = supplied
emailAddress = optional
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName = optional
stateOrProvinceName = optional
localityName = optional
organizationName = optional
organizationalUnitName = optional
commonName = supplied
emailAddress = optional
```

config 文件里 CA section 里面的很多 key 都和命令行参数是对应的。

如果某个 key 后面标明 mandatory,那就说明这个参数是必须提供的，无论你通过命令行还是通过 config 文件去提供。

new_certs_dir

本 key 同命令行的 -outdir 意义相同。(mandatory)

certificate

同命令行的 -cert 意义相同。(mandatory)

private_key

同命令行-keyfile 意义相同.(mandatory)

RANDFILE

指明一个用来读写时候产生 random key 的 seed 文件。具体意义在以后的 RAND 的 API 再给出解释。(不是我摆谱，我觉得重复解释没有必要)

default_days

意义和命令行的 -days 相同。

default_startdate

意义同命令行的 -startdate 相同。如果没有的话那么就使用产生证书的时间。

default_enddate

意义同命令行的 -enddate 相同。(mandatory).

crl_extensions

preserve

default_crl_hours default_crl_days

CRL 的东西.....自己都没弄懂.....

default_md

同命令行的-md 意义相同. (mandatory)

database

记得 index.txt 是什么文件吗？不记得自己往前找。这个 key 就是指定 index.txt 的。初始化是空文件。

serialfile

指明一个 txt 文件，里面必须包含下一个可用的 16 进制数字，用来给下一个证书做系列号。(mandatory)

x509_extensions

意义同 -extensions 相同。

msie_hack

意义同-msie_hack 相同。

policy

意义同-policy 相同。自己看看这一块是怎么回事。(mandatory)

[policy_match]

countryName = match

stateOrProvinceName = match

organizationName = match

organizationalUnitName = optional

commonName = supplied

emailAddress = optional

其实如果你做过 CSR 就会明白，这些项就是你做 CSR 时候填写的那些东西麻。

后面的"match", "supplied"等又是什么意思呢？"match"表示说明你填写的这一栏一定要和 CA 本身的证书里面的这一栏相同。supplied 表示本栏必须，optional 就表示本栏可以不填写。

举例时间到了：

注意，本例中我们先要在 $OPENSSL/misc 下面运行过 CA.sh -newca，建立好相应的目录，所有需要的文件，包括 CA 的私有密钥文件，证书文件，系列号文件，和一个空的 index 文件。并且文件都已经各就各位。放心把，产生文件和文件就位都由 CA.sh 搞定，你要做的就是运行 CA.sh -nweca 就行了，甚至在你的系列号文件中还有个 01,用来给下一个证书做系列号。

给一个 CSR 签名：

openssl ca -in req.pem -out newcert.pem

给一个 CSR 签名, 产生 x509v3 证书：

openssl ca -in req.pem -extensions v3_ca -out newcert.pem

同时给数个 CSR 签名：

openssl ca -infiles req1.pem req2.pem req3.pem

注意：

index.txt 文件是整个处理过程中很重要的一部分，如果这玩意坏了，很难修复。理论上根据

已经颁发的证书和当前的 CRL 当然是有办法修复的啦，但 openssl 没提供这个功能。:(

openssl 还有俩大类指令: crl, crl2pkcs7, 都是和 CRL 有关的，

由于我们对这个没有兴趣，所以这俩大类不做翻译和解释。

指令 cipher

说明：cipher 就是加密算法的意思。ssl 的 cipher 主要是对称加密算法和不对称加密算法的组合。本指令是用来展示用于 SSL 加密算法的工具。它能够把所有 openssl 支持的加密算法按照一定规律排列（一般是加密强度）。这样可以用来做测试工具，决定使用什么加密算法。

用法：

openssl ciphers [-v] [-ssl2] [-ssl3] [-tls1] [cipherlist]

COMMAND OPTIONS

-v

详细列出所有符合的 cipher 的所有细节。列出该 cipher 使用的 ssl 的版本，公共密钥交换算法，身份验证方法，对称加密算法以及哈希算法。还列出该算法是否可以出口。

算法出口？ 趁这个机会可以给大家来点革命教育。米国的加密算法研究是世界上最先进的，其国家安全局(NSA)在这方面的研究水平已经多次证明比"最先进水平"领先 10 到 15 年。他们的预算据说是每年 200 亿美圆。他们的数学家比你知道的还多，他们还是全世界最大的计算机硬件买家。DES 就是他们最先弄出来的。到了 70 年代，IBM 也有人在实现室弄出这个算法。都弄出来 30 年了，还使用的这么广泛。

该算法的最隐蔽的是一个叫 S 匣的东西，是一个常数矩阵。研究 DES 你就会知道这玩意。因为 NSA 和 IBM 都没有给出这个 S 匣的解释，所以大家都怀疑使用这个东西是否是 NSA 和 IBM 搞出来的后门？

一直到了 90 年代，才有俩个以色列人发现了原因，这个是为了对付一种叫什么微分密码分析的破解法而如此设置的，对 S 匣的任何改动都将使微分密码分析比较容易的将 DES 给 K 掉。S 匣不仅不是后门，还是最大限度的增加了加密强度。

说远了，大意就是：老米在这方面领先的可怕。但他们怕他们的研究成果给其他国家的人用，搞的自己也破解不了，那就麻烦了。所以他们用法律规定了，一定强度以上的加密算法禁止给其他国家用。那些加密强度很弱的就可以出口。

这个故事教育我们，为了中国的崛起，还有很多路要走呐。

如果没有-v 这个参数， 很多 cipher 可能重复出现，因为他们可以同时被不同版本的 SSL 协议使用。

-ssl3

只列出 SSLv3 使用的 ciphers

-ssl2

只列出 SSLv2 使用的 ciphers

-tls1

只列出 TLSv1 使用的 ciphers

-h, -?

打印帮助信息

cipherlist

列出一个 cipher list 的详细内容。一般都这么用：

openssl -v XXXXX

这个 XXXXX 就是 cipher list.如果是空的话，那么 XXXXX 代表所有的 cipher.

CIPHER LIST 的格式

cipher list 由许多 cipher string 组成，由冒号，逗号或者空格分隔开。但一般最常用的是用冒号。

cipher string 又是什么？

它可以仅仅包含一个 cipher, 比如 RC4-SHA.

它也可以仅仅包含一个加密算法，比如 SHA, 那就表示所有用到 SHA 的 cipher 都得列出来。

你还可以使用三个符号来捏合各种不同的 cipher,做出 cipher string.这三个符号是 +, -, !。我想这个很好理解吧，MD5+DES 表示同时使用了这俩种算法的 cipher，!SHA 就表示所有没有有用到 SHA 的 cipher， IDEA-CBC 就表示使用了 IDEA 而没有使用 CBC 的所有 cipher.

openssl 还缺省的定义了一些通用的 cipher string, 有：

DEFAULT: 缺省的 cipher list.

ALL: 所有的 cipher

HIGH, LOW, MEDIUM: 分别代表 高强度，中等强度和底强度的 cipher list.具体一点就是对称加密算法的 key 的长度分别是 >;128bit <128bit 和 ==128bit 的 cipher.

EXP, EXPORT, EXPORT40: 老米的垄断体现，前俩者代表法律允许出口的加密算法，包括 40bit, 56bit 长度的 key 的算法，后者表示只有 40bit 长度的 key 的加密算法。

eNULL, NULL: 表示不加密的算法。(那也叫加密算法吗？)

aNULL: 不提供身份验证的加密算法。目前只有 DH 一种。该算法很容易被监听者，路由器等中间设备攻击，所以不提倡使用。

下表列出了 SSL/TLS 使用的 cipher, 以及 openssl 里面如何表示这些 cipher.

SSL v3.0 cipher suites OPENLLS 表示方法

SSL_RSA_WITH_NULL_MD5 NULL-MD5

SSL_RSA_WITH_NULL_SHA NULL-SHA

SSL_RSA_EXPORT_WITH_RC4_40_MD5 EXP-RC4-MD5

SSL_RSA_WITH_RC4_128_MD5 RC4-MD5

SSL_RSA_WITH_RC4_128_SHA RC4-SHA

SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 EXP-RC2-CBC-MD5

SSL_RSA_WITH_IDEA_CBC_SHA IDEA-CBC-SHA

SSL_RSA_EXPORT_WITH_DES40_CBC_SHA EXP-DES-CBC-SHA

SSL_RSA_WITH_DES_CBC_SHA DES-CBC-SHA

SSL_RSA_WITH_3DES_EDE_CBC_SHA DES-CBC3-SHA

SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA Not implemented.

SSL_DH_DSS_WITH_DES_CBC_SHA Not implemented.

SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA Not implemented.

SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA Not implemented.

SSL_DH_RSA_WITH_DES_CBC_SHA Not implemented.

SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA Not implemented.

SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA EXP-EDH-DSS-DES-CBC-SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA EDH-DSS-CBC-SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA EDH-DSS-DES-CBC3-SHA
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA EXP-EDH-RSA-DES-CBC-SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA EDH-RSA-DES-CBC-SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA EDH-RSA-DES-CBC3-SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5 EXP-ADH-RC4-MD5
SSL_DH_anon_WITH_RC4_128_MD5 ADH-RC4-MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA EXP-ADH-DES-CBC-SHA
SSL_DH_anon_WITH_DES_CBC_SHA ADH-DES-CBC-SHA
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA ADH-DES-CBC3-SHA
SSL_FORTEZZA_KEA_WITH_NULL_SHA Not implemented.
SSL_FORTEZZA_KEA_WITH_FORTEZZA_CBC_SHA Not implemented.
SSL_FORTEZZA_KEA_WITH_RC4_128_SHA Not implemented.

TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA EXP1024-DES-CBC-SHA
TLS_RSA_EXPORT1024_WITH_RC4_56_SHA EXP1024-RC4-SHA
TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA
EXP1024-DHE-DSS-DES-CBC-SHA
TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA EXP1024-DHE-DSS-RC4-SHA
TLS_DHE_DSS_WITH_RC4_128_SHA DHE-DSS-RC4-SHA

TLS v1.0 cipher suites.
TLS_RSA_WITH_NULL_MD5 NULL-MD5
TLS_RSA_WITH_NULL_SHA NULL-SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5 EXP-RC4-MD5
TLS_RSA_WITH_RC4_128_MD5 RC4-MD5
TLS_RSA_WITH_RC4_128_SHA RC4-SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 EXP-RC2-CBC-MD5
TLS_RSA_WITH_IDEA_CBC_SHA IDEA-CBC-SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA EXP-DES-CBC-SHA
TLS_RSA_WITH_DES_CBC_SHA DES-CBC-SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA DES-CBC3-SHA
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA Not implemented.
TLS_DH_DSS_WITH_DES_CBC_SHA Not implemented.
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA Not implemented.
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA Not implemented.
TLS_DH_RSA_WITH_DES_CBC_SHA Not implemented.
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA Not implemented.
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA EXP-EDH-DSS-DES-CBC-SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA EDH-DSS-CBC-SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA EDH-DSS-DES-CBC3-SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA EXP-EDH-RSA-DES-CBC-SHA

TLS_DHE_RSA_WITH_DES_CBC_SHA EDH-RSA-DES-CBC-SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA EDH-RSA-DES-CBC3-SHA

TLS_DH_anon_EXPORT_WITH_RC4_40_MD5 EXP-ADH-RC4-MD5
TLS_DH_anon_WITH_RC4_128_MD5 ADH-RC4-MD5
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA EXP-ADH-DES-CBC-SHA
TLS_DH_anon_WITH_DES_CBC_SHA ADH-DES-CBC-SHA
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA ADH-DES-CBC3-SHA

NOTES

DH 算法由于老米没有允许人家使用，所有 openssl 都没有实现之。

举例时间：

详细列出所有 openssl 支持的 ciphers,包括那些 eNULL ciphers:

openssl ciphers -v 'ALL:eNULL'

按加密强度列出所有加密算法:

openssl ciphers -v 'ALL:!ADH:@STRENGTH'

详细列出所有同时使用了 3DES 和 RSA 的 ciphers

openssl ciphers -v '3DES:+RSA'

指令 dgst

用法：

openssl dgst [md5|md2|sha1|sha|mdc2|ripemd160] [-c] [-d] [file...]

说明：这个指令可以用来哈希某个文件内容的，以前的版本还可以用来做数字签名和认证。这个工具本来有很多选项的，可是不知道为什么，现在版本的 openssl 删掉了很多。表示你用什么算法来哈希该文件内容

OPTIONS

-md5 -sha 那些就不用结实了吧，都是一些哈希算法的名称

-c

打印出哈希结果的时候用冒号来分隔开。

-d

详细打印出调试信息

file...

你要哈希的文件，如果没有指定，就使用标准输入。

举例时间：

要哈希一个叫 fordesign.txt 文件的内容，使用 SHA 算法

openssl dgst -sha -c fordesign.txt

SHA(fordesign.txt)=

57:37:dc:a5:8c:bd:12:aa:43:45:fe:2a:19:f5:05:a3:be:e9:08:cc

---

wingger 回复于：2005-01-10 23:44:05

openssl 简介－指令 dhparam

用法：

openssl dhparam [-inform DER|PEM] [-outform DER|PEM] [-in filename]
[-out filename] [-dsaparam] [-noout] [-text] [-C] [-2] [-5]
[-rand file(s)] [numbits]

描述：
本指令用来维护 DH 的参数文件。

OPTIONS:
-inform DER|PEM
指定输入的格式是 DEM 还是 DER. DER 格式采用 ASN1 的 DER 标准格式。一般用的多的都是 PEM 格式，就是 base64 编码格式.你去看看你做出来的那些.key, .crt 文件一般都是 PEM 格式的，第一行和最后一行指明内容，中间就是经过编码的东西。
-outform DER|PEM
和上一个差不多，不同的是指定输出格式
-in filename
要分析的文件名称。
-out filename
要输出的文件名。
-dsaparam
如果本 option 被 set, 那么无论输入还是输入都会当做 DSA 的参数。它们再被转化成 DH 的参数格式。这样子产生 DH 参数和 DH key 都会块很多。会使 SSL 握手的时间缩短。当然时间是以安全性做牺牲的，所以如果这样子最好每次使用不同的参数，以免给人 K 破你的 key.
-2, -5
使用哪个版本的 DH 参数产生器。版本 2 是缺省的。如果这俩个 option 有一个被 set, 那么将忽略输入文件。

-rand file(s)
产生 key 的时候用过 seed 的文件，可以把多个文件用冒号分开一起做 seed.
numbits
指明产生的参数的长度。必须是本指令的最后一个参数。如果没有指明，则产生 512bit 长的参数。
-noout
不打印参数编码的版本信息。
-text
将 DH 参数以可读方式打印出来。

-C
将参数转换成 C 代码方式。这样可以用 get_dhnumbits()函数调用这些参数。

openssl 还有俩个指令， dh, gendh, 现在都过时了，全部功能由 dhparam 实现。
现在 dh, gendh 这俩个指令还保留，但在将来可能会用做其他用途。

指令 enc

用法：

openssl enc -ciphername [-in filename] [-out filename] [-pass arg] [-e]
[-d] [-a] [-k password] [-kfile filename] [-K key] [-iv IV] [-p]
[-P] [-bufsize number] [-debug]

说明：
对称加密算法工具。它能够把数据用不同对称加密算法来加/解密。还能够把加密/接密,还可以把结果进行 base64 编码。

OPTIONS
-in filename
要加密/解密的输入文件，缺省为标准输入。
-out filename
要加密/解密的输出文件，缺省为标准输出。
-pass arg
输入文件如果有密码保护，在这里输入密码。
-salt
为了和 openssl0.9.5 以后的版本兼容，必须 set 这个 option.salt 大概又是密码学里的一个术语，具体是做什么的我也没弄的很明白。就我的理解,这是加密过后放在密码最前面的一段字符串, 用途也是为了让破解更难.如果理解错了,请密码学高手指正.
-nosalt
想和 openssl0.9.5 以前的版本兼容，就 set 这个 option
-e
一个缺省会 set 的 option, 把输入数据加密。
-d
解密输入数据。
-a
用 base64 编码处理数据。set 了这个 option 表示在加密之后的数据还要用 base64 编码捏一次，解密之前则先用 base64 编码解码。
-k password
一个过时了的项，为了和以前版本兼容。现在用-key 代替了。

-kfile filename
同上，被 passin 代替。
-K key
以 16 进制表示的密码。
-iv IV
作用完全同上。
-p
打印出使用的密码。
-P
作用同上，但打印完之后马上退出。
-bufsize number
设置 I/O 操作的缓冲区大小
-debug
打印调试信息。

注意事项：

0.9.5 以后的版本，使用这个指令，-salt 是必须被 set 的。否则很容易用字典攻击法破你的密码，流加密算法也容易被破。(加密算法中有块加密算法和流加密算法俩种，块加密算法是一次加密固定长度的数据，一般是 8Bytes, 流加密算法则加密大量数据)。为什么我也弄不清楚。研究加密算法实在麻烦，也不是我们程序员的责任本指令可以用不同加密算法，那么哪些好，哪些坏呢？如果你使用不当，高强度的加密算法也变脆弱了。一般推荐新手门使用 des3-cbc。

本指令支持的加密算法
base64 Base 64
bf-cbc Blowfish in CBC mode
bf Alias for bf-cbc
bf-cfb Blowfish in CFB mode
bf-ecb Blowfish in ECB mode
bf-ofb Blowfish in OFB mode

cast-cbc CAST in CBC mode
cast Alias for cast-cbc
cast5-cbc CAST5 in CBC mode
cast5-cfb CAST5 in CFB mode
cast5-ecb CAST5 in ECB mode
cast5-ofb CAST5 in OFB mode

des-cbc DES in CBC mode
des Alias for des-cbc
des-cfb DES in CBC mode
des-ofb DES in OFB mode
des-ecb DES in ECB mode

des-ede-cbc Two key triple DES EDE in CBC mode

des-ede Alias for des-ede
des-ede-cfb Two key triple DES EDE in CFB mode
des-ede-ofb Two key triple DES EDE in OFB mode

des-ede3-cbc Three key triple DES EDE in CBC mode
des-ede3 Alias for des-ede3-cbc
des3 Alias for des-ede3-cbc
des-ede3-cfb Three key triple DES EDE CFB mode
des-ede3-ofb Three key triple DES EDE in OFB mode
desx DESX algorithm.
idea-cbc IDEA algorithm in CBC mode
idea same as idea-cbc
idea-cfb IDEA in CFB mode
idea-ecb IDEA in ECB mode
idea-ofb IDEA in OFB mode

rc2-cbc 128 bit RC2 in CBC mode
rc2 Alias for rc2-cbc
rc2-cfb 128 bit RC2 in CBC mode
rc2-ecb 128 bit RC2 in CBC mode
rc2-ofb 128 bit RC2 in CBC mode
rc2-64-cbc 64 bit RC2 in CBC mode
rc2-40-cbc 40 bit RC2 in CBC mode
rc4 128 bit RC4
rc4-64 64 bit RC4
rc4-40 40 bit RC4
rc5-cbc RC5 cipher in CBC mode
rc5 Alias for rc5-cbc
rc5-cfb RC5 cipher in CBC mode
rc5-ecb RC5 cipher in CBC mode
rc5-ofb RC5 cipher in CBC mode

大家可能看到 DES 都分 des-ecb, des-cbc, des-cfb 这些。简单解释一下。

ecb 就是说每来 8bytes,就加密 8bytes 送出去。各个不同的数据块之间没有任何联系。cbc 和 cfb 则每次加密一个 8bytes 的时候都和上一个 8bytes 加密的结果有一个运算法则。各个数据块之间是有联系的。

举例时间：

把某二进制文件转换成 base64 编码方式：

openssl base64 -in file.bin -out file.b64

把某 base64 编码文件转换成二进制文件。

openssl base64 -d -in file.b64 -out file.bin

把某文件用 DES-CBC 方式加密。加密过程中会提示你输入保护密码。

openssl des3 -salt -in file.txt -out file.des3

解密该文件， 密码通过-k 来输入
openssl des3 -d -salt -in file.des3 -out file.txt -k mypassword
加密某文件，并且把加密结果进行 base64 编码。用 bf+cbc 算法加密
openssl bf -a -salt -in file.txt -out file.bf
先用 base64 解码某文件，再解密
openssl bf -d -salt -a -in file.bf -out file.txt

指令 gendsa

用法：

openssl gendsa [-out filename] [-des] [-des3] [-idea]
[-rand file(s)] [paramfile]

描述：

本指令由 DSA 参数来产生 DSA 的一对密钥。dsa 参数可以用 dsaparam 来产生。

OPTIONS
-des|-des3|-idea
采用什么加密算法来加密我们的密钥。一般会要你输入保护密码。
如果这三个中一个也没 set, 我们的密钥将不被加密而输入。
-rand file(s)
产生 key 的时候用过 seed 的文件，可以把多个文件用冒号分开一起做 seed.
paramfile
指定使用的 DSA 参数文件。

指令 genrsa

用法：

openssl genrsa [-out filename] [-passout arg] [-des] [-des3] [-idea]
[-f4] [-3] [-rand file(s)] [numbits]

DESCRIPTION
生成 RSA 私有密钥的工具。

OPTIONS

-out filename

私有密钥输入文件名，缺省为标准输出。

the output filename. If this argument is not specified then standard output is uused.

-passout arg

参看指令 dsa 里面的 passout 参数说明

-des|-des3|-idea

采用什么加密算法来加密我们的密钥。一般会要你输入保护密码。

如果这三个中一个也没 set, 我们的密钥将不被加密而输入。

-F4|-3

使用的公共组件，一种是 3, 一种是 F4, 我也没弄懂这个 option 是什么意思。

-rand file(s)

产生 key 的时候用过 seed 的文件，可以把多个文件用冒号分开一起做 seed.

numbits

指明产生的参数的长度。必须是本指令的最后一个参数。如果没有指明，则产生 512bit 长的参数。

研究过 RSA 算法的人肯定知道，RSA 的私有密钥其实就是三个数字，其中俩个是质数。这俩个呢，就叫 prime numbers.产生 RSA 私有密钥的关键就是产生这俩。还有一些其他的参数，引导着整个私有密钥产生的过程。因为产生私有密钥过程需要很多随机数，这个过程的时间是不固定的。

产生 prime numbers 的算法有个 bug, 它不能产生短的 primes. key 的 bits 起码要有 64 位。一般我们都用 1024bit 的 key.

指令 passwd

SYNOPSIS

openssl passwd [-crypt] [-1] [-apr1] [-salt string] [-in file] [-stdin]
[-quiet] [-table] {password}

说明：

本指令计算用来哈希某个密码，也可以用来哈希文件内容。

本指令支持三种哈希算法：

UNIX 系统的标准哈希算法(crypt)

MD5-based BSD(1)

OPTIONS

-crypt -1 -apr1

这三个 option 中任意选择一个作为哈希算法，缺省的是-crypt

-salt string

输入作为 salt 的字符串。

-in file

要哈希的文件名称

-stdin

从标准输入读入密码

-quiet

当从标准输入读密码，输入的密码太长的时候，程序将自动解短它。这个 option 的 set 将不在情况下发出警告。

-table

在输出列的时候,先输出明文的密码,然后输出一个 TAB,再输出哈希值.

举例时间:

openssl passwd -crypt -salt xx password xxj31ZMTZzkVA.

openssl passwd -1 -salt xxxxxxxx password 1xxxxxxxx$8XJIcl6ZXqBMCK0qFevqT1.

openssl passwd -apr1 -salt xxxxxxxx password $apr1$xxxxxxxx$dxHfLAsjHkDRmG83UXe8K0

指令 pkcs7

用法:

openssl pkcs7 [-inform PEM|DER] [-outform PEM|DER] [-in filename]
[-out filename] [-print_certs] [-text] [-noout]

说明:

处理 PKCS#7 文件的工具,

OPTIONS

-inform DER|PEM

指定输入的格式是 DEM 还是 DER. DER 格式采用 ASN1 的 DER 标准格式。一般用的多的都是 PEM 格式，就是 base64 编码格式.你去看看你做出来的那些.key, .crt 文件一般都是 PEM 格式的，第一行和最后一行指明内容，中间就是经过编码的东西。

-outform DER|PEM

和上一个差不多，不同的是指定输出格式

-in filename

要分析的文件名称, 缺省是标准输入.

-out filename

要输出的文件名, 缺省是标准输出.

write to or standard output by default.

-print_certs

打印出该文件内的任何证书或者 CRL.

-text

打印出证书的细节.
-noout
不要打印出 PKCS#7 结构的编码版本信息.
举例时间:
把一个 PKCS#7 文件从 PEM 格式转换成 DER 格式
openssl pkcs7 -in file.pem -outform DER -out file.der
打印出文件内所有的证书
openssl pkcs7 -in file.pem -print_certs -out certs.pem
PCKS#7 文件的开始和结束俩行是这样子的:
-----BEGIN PKCS7-----
-----END PKCS7-----
为了和某些猥琐 CA 兼容,这样子的格式也可以接受
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
好象我们还没有解释 pkcs#7 是什么东西. 有兴趣的可以看看 rfc2315, 估计看完目录还没有阵亡的同学不会超过 1/10.

指令 rand

用法:

openssl rand [-out file] [-rand file(s)] [-base64] num

描述:
用来产生伪随机字节. 随机数字产生器需要一个 seed, 先已经说过了,在没有/dev/srandom 系统下的解决方法是自己做一个~/.rnd 文件.如果该程序能让随机数字产生器很满意的被 seeded,程序写回一些怪怪的东西回该文件.

OPTIONS
-out file
输出文件.
-rand file(s)
产生随机数字的时候用过 seed 的文件，可以把多个文件用冒号分开一起做 seed.
-base64
对产生的东西进行 base64 编码
num
指明产生多少字节随机数.

指令 req

用法:

openssl req [-inform PEM|DER] [-outform PEM|DER] [-in filename]
[-passin arg] [-out filename] [-passout arg] [-text] [-noout]
[-verify] [-modulus] [-new] [-rand file(s)] [-newkey rsa]
[-newkey dsa] [-nodes] [-key filename] [-keyform PEM|DER]
[-keyout filename] [-[md5|sha1|md2|mdc2]] [-config filename]
[-x509] [-days n] [-asn1-kludge] [-newhdr] [-extensions section]
[-reqexts section]

描述:
本指令用来创建和处理 PKCS#10 格式的证书.它还能够建立自签名证书,做 Root CA.

OPTIONS
-inform DER|PEM
指定输入的格式是 DEM 还是 DER. DER 格式采用 ASN1 的 DER 标准格式。一般用的多的都是 PEM 格式，就是 base64 编码格式.你去看看你做出来的那些.key, .crt 文件一般都是 PEM 格式的，第一行和最后一行指明内容，中间就是经过编码的东西。
-outform DER|PEM
和上一个差不多，不同的是指定输出格式
-in filename
要处理的 CSR 的文件名称,只有-new 和-newkey 俩个 option 没有被 set,本 option 才有效
-passin arg
去看看 CA 那一章关于这个 option 的解释吧。
-out filename
要输出的文件名
-passout arg
参看 dsa 指令里的 passout 这个 option 的解释吧.
-text
将 CSR 文件里的内容以可读方式打印出来
-noout
不要打印 CSR 文件的编码版本信息.
-modulus
将 CSR 里面的包含的公共米要的系数打印出来.
-verify
检验请求文件里的签名信息.
-new
本 option 产生一个新的 CSR, 它会要用户输入创建 CSR 的一些必须的信息.至于需要哪些信息,是在 config 文件里面定义好了的.如果-key 没有被 set, 那么就将根据 config 文件里的信息先产生一

对新的 RSA 密钥

-rand file(s)

产生 key 的时候用过 seed 的文件，可以把多个文件用冒号分开一起做 seed.

-newkey arg

同时生成新的私有密钥文件和 CSR 文件. 本 option 是带参数的.如果是产生 RSA 的私有密钥文件,参数是一个数字, 指明私有密钥 bit 的长度. 如果是产生 DSA 的私有密钥文件,参数是 DSA 密钥参数文件的文件名.

-key filename

参数 filename 指明我们的私有密钥文件名.允许该文件的格式是 PKCS#8.

-keyform DER|PEM

指定输入的私有密钥文件的格式是 DEM 还是 DER. DER 格式采用 ASN1 的 DER 标准格式。一般用的多的都是 PEM 格式，就是 base64 编码格式.你去看看你做出来的那些.key, .crt 文件一般都是 PEM 格式的，第一行和最后一行指明内容，中间就是经过编码的东西。

-outform DER|PEM

和上一个差不多，不同的是指定输出格式

-keyform PEM|DER

私有密钥文件的格式, 缺省是 PEM

-keyout filename

指明创建的新的私有密钥文件的文件名. 如果该 option 没有被 set, 将使用 config 文件里面指定的文件名.

-nodes

本 option 被 set 的话,生成的私有密钥文件将不会被加密.

-[md5|sha1|md2|mdc2]

指明签发的证书使用什么哈希算法.如果没有被 set, 将使用 config 文件里的相应 item 的设置. 但 DSA 的 CSR 将忽略这个 option, 而采用 SHA1 哈希算法.

-config filename

使用的 config 文件的名称. 本 option 如果没有 set, 将使用缺省的 config 文件.

-x509

本 option 将产生自签名的证书. 一般用来错测试用,或者自己玩下做个 Root CA.证书的扩展项在 config 文件里面指定.

-days n

如果-509 被 set, 那么这个 option 的参数指定我们自己的 CA 给人家签证书的有效期.缺省是 30 天.

-extensions section -reqexts section

这俩个 option 指定 config 文件里面的与证书扩展和证书请求扩展有关的俩个 section 的名字(如果-x509 这个 option 被 set).这样你可以在 config 文件里弄几个不同的与证书扩展有关的 section, 然后为了不同的目的给 CSR 签名的时候指明不同的 section 来控制签名的行为.

-asn1-kludge

缺省的 req 指令输出完全符合 PKCS10 格式的 CSR, 但有的 CA 仅仅接受一种非正常格式的 CSR, 这个 option 的 set 就可以输出那种格式的 CSR. 要解释这俩种格式有点麻烦, 需要用到 ASN1 和 PKCS 的知识,而且现在这样子怪的 CA 几乎没有,所以省略解释

-newhdr

在 CSR 问的第一行和最后一行中加一个单词"NEW", 有的软件(netscape certificate server)和有的 CA 就有这样子的怪癖嗜好.如果那些必须要的 option 的参数没有在命令行给出，那么就会到

config 文件里去查看是否有缺省值， 然后时候。config 文件中相关的一些 KEY 的解释与本指令有关的 KEY 都在[req]这个 section 里面.

input_password output_password

私有密钥文件的密码和把密码输出的文件名.同指令的 passin, passout 的意义相同.

default_bits

指定产生的私有密钥长度, 如果为空,那么就用 512.只有-new 被 set, 这个设置才起作用,意义同-newkey 相同.

default_keyfile

指定输出私有密钥文件名,如果为空, 将输出到标准输入,意义同-keyout 相同.

oid_file

oid_section

与 oid 文件有关的项, oid 不清楚是什么东西来的.

RANDFILE

产生随机数字的时候用过 seed 的文件，可以把多个文件用冒号分开一起做 seed.

encrypt_key

如果本 KEY 设置为 no, 那么如果生成一个私有密钥文件,将不被加密.同命令行的-nodes 的意义相同.

default_md

指定签名的时候使用的哈希算法,缺省为 MD5. 命令行里有同样的功能输入.

string_mask

屏蔽掉某些类型的字符格式. 不要乱改这个 KEY 的值!!有的字符格式 netscape 不支持,所以乱改这个 KEY 很危险.

req_extensions

指明证书请求扩展 section, 然后由那个 secion 指明扩展的特性. openssl 的缺省 config 文件里, 扩展的是 X509v3, 不扩展的是 x509v1.这个 KEY 的意义和命令行里-reqexts 相同.

x509_extensions

同命令行的-extension 的意义相同.指明证书扩展的 sesion, 由那个 section 指明证书扩展的特性.

prompt

如果这个 KEY 设置为 no, 那么在生成证书的时候需要的那些信息将从 config 文件里读入,而不是从标准输入由用户去输入, 同时改变下俩个 KEY 所指明的 section 的格式.

attributes

一个过时了的东西, 不知道也罢. 不过它的意义和下一个 KEY 有点类似,

格式则完全相同.

distinguished_name

指定一个 section, 由那个 section 指定在生成证书或者 CRS 的时候需要的资料.该 section 的格式如下:

其格式有俩种, 如果 KEY prompt 被 set 成 no(看看 prompt 的解释), 那么这个 secion 的格式看起来就是这样子的:

CN=My Name

OU=My Organization

emailAddress=someone@somewhere.org

就说只包括了字段和值。这样子可以可以让其他外部程序生成一个摸板文件,包含所有字段和

值, 把这些值提出来.等下举例时间会有详细说明.如果 prompt 没有被 set 成 no, 那么这个 section 的格式则如下:

fieldName="please input ur name"
fieldName_default="fordesign"
fieldName_min= 3
fieldName_max= 18

"fieldname"就是字段名, 比如 commonName(或者 CN). fieldName(本例中是"prompt")是用来提示用户输入相关的资料的.如果用户什么都不输, 那么就使用确省值.如果没有缺省值, 那么该字段被忽略.用户如果输入 '.' ,也可以让该字段被忽略.

用户输入的字节数必须在 fieldName_min 和 fieldName_max 之间. 不同的 section 还可能对输入的字符有特殊规定,比如必须是可打印字符.那么在本例里面, 程序的表现将如下:

首先把 fieldName 打印出来给用户以提示

please input ur name:

之后如果用户必须输入 3 到 18 之间的一个长度的字符串, 如果用户什么也不输入,那么就把 fieldName_default 里面的值"fordesign"作为该字段的值添入.

有的字段可以被重复使用.这就产生了一个问题, config 文件是不允许同样的 section 文件里面有多于一个的相同的 key 的.其实这很容易解决,比如把它们的名字分别叫做 "1.organizationName", "2.organizationName"

openssl 在编译的时候封装了最必须的几个字段， 比如 commonName, countryName, localityName, organizationName,organizationUnitName, stateOrPrivinceName 还增加了 emailAddress surname, givenName initials 和 dnQualifier.

举例时间:

就使用确省值.如果没有缺省值, 那么该字段被忽略.用户如果输入 '.' ,也可以让该字段被忽略. 用户输入的字节数必须在 fieldName_min 和 fieldName_max 之间. 不同的 section 还可能对输入的字符有特殊规定,比如必须是可打印字符.那么在本例里面, 程序的表现将如下:

首先把 fieldName 打印出来给用户以提示

please input ur name:

之后如果用户必须输入 3 到 18 之间的一个长度的字符串, 如果用户什么也不输入,那么就把 fieldName_default 里面的值"fordesign"作为该字段的值添入.

有的字段可以被重复使用.这就产生了一个问题, config 文件是不允许同样的 section 文件里面有多于一个的相同的 key 的.其实这很容易解决,比如把它们的名字分别叫做 "1.organizationName", "2.organizationName" openssl 在编译的时候封装了最必须的几个字段， 比如 commonName,countryName,localityName, organizationName,organizationUnitName, stateOrPrivinceName 还增加了 emailAddress surname, givenName initials 和 dnQualifier.

举例时间:

Examine and verify certificate request:

检查和验证 CSR 文件.

openssl req -in req.pem -text -verify -noout

做自己的私有密钥文件, 然后用这个文件生成 CSR 文件.

openssl genrsa -out key.pem 1024

openssl req -new -key key.pem -out req.pem

也可以一步就搞定:

openssl req -newkey rsa:1024 -keyout key.pem -out req.pem

做一个自签名的给 Root CA 用的证书:

openssl req -x509 -newkey rsa:1024 -keyout key.pem -out crt.pem

下面是与本指令有关的 config 文件中相关的部分的一个例子:

```
[ req ]
default_bits = 1024
default_keyfile = privkey.pem
distinguished_name = req_distinguished_name
attributes = req_attributes
x509_extensions = v3_ca
dirstring_type = nobmp
[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = AU
countryName_min = 2
countryName_max = 2
localityName = Locality Name (eg, city)
organizationalUnitName = Organizational Unit Name (eg, section)
commonName = Common Name (eg, YOUR name)
commonName_max = 64
emailAddress = Email Address
emailAddress_max = 40
[ req_attributes ]
challengePassword = A challenge password
challengePassword_min = 4
challengePassword_max = 20
 [ v3_ca ]
 subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer:always
basicConstraints = CA:true
RANDFILE = $ENV::HOME/.rnd
[ req ]
default_bits = 1024
default_keyfile = keyfile.pem
distinguished_name = req_distinguished_name
attributes = req_attributes
prompt = no
output_password = mypass
[ req_distinguished_name ]
C = GB
ST = Test State or Province
L = Test Locality
O = Organization Name
OU = Organizational Unit Name
CN = Common Name
```

emailAddress = test@email.address
[req_attributes]
challengePassword = A challenge password
一般的 PEM 格式的 CSR 文件的开头和结尾一行如下
-----BEGIN CERTIFICATE REQUEST----
-----END CERTIFICATE REQUEST----
但个把变态软件和 CA 硬是需要 CSR 的文件要这样子:
-----BEGIN NEW CERTIFICATE REQUEST----
-----END NEW CERTIFICATE REQUEST----
用-newhdr 就可以啦, 或者你自己手工加也中.openssl 对俩种格式都承认.
openssl 的 config 文件也可以用环境变量 OPENSSL_CONF 或者 SSLEAY_CONF 来指定.

指令 rsa

用法

openssl rsa [-inform PEM|NET|DER] [-outform PEM|NET|DER] [-in filename]
[-passin arg] [-out filename] [-passout arg] [-sgckey] [-des] [-des3]
[-idea] [-text] [-noout] [-modulus] [-check] [-pubin] [-pubout]

说明:
rsa 指令专门处理 RSA 密钥.其实其用法和 dsa 的差不多.

OPTIONS
-inform DER|PEM|NET
指定输入的格式是 DEM 还是 DER 还是 NET.注意, 这里多了一种格式,就是 NET,DER 格式采用 ASN1 的 DER 标准格式。一般用的多的都是 PEM 格式，就是 base64 编码格式.你去看看你做出来的那些.key, .crt 文件一般都是 PEM 格式的，第一行和最后一行指明内容，中间就是经过编码的东西。NET 格式在本章后面会详细解释.
-outform DER|PEM|NET
和上一个差不多，不同的是指定输出格式
-in filename
要分析的文件名称。如果文件有密码保护,会要你输入的.
-passin arg
去看看 CA 那一章关于这个 option 的解释吧。
-out filename
要输出的文件名。
-passout arg
没什么用的一个选项，用来把保护 key 文件的密码输出的，意义和 passin 差不多。
-sgckey

配合 NET 格式的私有密钥文件的一个 option, 没有必要去深入知道了。

-des|-des3|-idea

指明用什么加密算法把我们的私有密钥加密。加密的时候会需要我们输入密码来保护该文件的。如果这仨一个都没有选，那么你的私有密钥就以明文写进你的 key 文件。该选项只能输出 PEM 格式的文件。

-text

打印出私有密钥的各个组成部分.

-noout

不打印出 key 的编码版本信息。

-modulus

把其公共密钥的值也打印出来

-pubin

缺省的来说是从输入文件里读到私有密钥，这个就可以从输入文件里去读公共密钥.

-pubout

缺省的来说是打印出私有密钥，这个就可以打印公共密钥.如果上面那个选项有 set 那么这个选项也自动被 set.

-check

检查 RSA 的私有密钥是否被破坏了这个指令实在和 dsa 太相似了。copy 的我手软。

现在解释一下 NET 是一种什么格式。它是为了和老的 netscape server 以及 IIS 兼容才弄出来的。他使用没有被 salt 过的 RC4 做加密算法，加密强度很底，如果不是一定要用就别用。

举例时间：

把 RSA 私有密钥文件的保护密码去掉（最好别这么做）

openssl rsa -in key.pem -out keyout.pem

用 DES3 算法加密我们的私有密码文件：

openssl rsa -in key.pem -des3 -out keyout.pem

把一个私有密钥文件从 PEM 格式转化成 DER 格式：

openssl rsa -in key.pem -outform DER -out keyout.der

把私有密钥的所有内容详细的打印出来：

openssl rsa -in key.pem -text -noout

只打印出公共密钥部分：

openssl rsa -in key.pem -pubout -out pubkey.pem

指令 rsautl

用法：

openssl rsautl [-in file] [-out file] [-inkey file] [-pubin] [-certin]
[-sign] [-verify] [-encrypt] [-decrypt] [-pkcs] [-ssl] [-raw] [-hexdump]
[-asn1parse]

描述：

本指令能够使用 RSA 算法签名，验证身份， 加密/解密数据。

OPTIONS

-in filename

指定输入文件名。缺省为标准输入。

-out filename

指定输入文件名， 缺省为标准输出。

-inkey file

指定我们的私有密钥文件， 格式必须是 RSA 私有密钥文件。

-pubin

指定我们的公共密钥文件。说真的我还真不知道 RSA 的公共密钥文件有什么用，一般公共密钥都是放在证书里面的。

-certin

指定我们的证书文件了。

-sign

给输入的数据签名。需要我们的私有密钥文件。

-verify

对输入的数据进行验证。

-encrypt

用我们的公共密钥对输入的数据进行加密。

-decrypt

用 RSA 的私有密钥对输入的数据进行解密。

-pkcs, -oaep, -ssl, -raw

采用的填充模式， 上述四个值分别代表：PKCS#1.5(缺省值), PKCS#1 OAEP, SSLv2 里面特定的填充模式，或者不填充。如果要签名，只有-pkcs 和-raw 可以使用.

-hexdump

用十六进制输出数据。

-asn1parse

对输出的数据进行 ASN1 分析。看看指令 asn1parse 吧。该指令一般和-verify 一起用的时候威力大。

本指令加密数据的时候只能加密少量数据，要加密大量数据，估计要调 API.我也没试过写 RSA 加密解密的程序来玩。

举例时间：

用私有密钥对某文件签名：

openssl rsautl -sign -in file -inkey key.pem -out sig

注意哦， 文件真的不能太大， 这个不能太大意思是必须很小。

文件大小最好不要大过 73。绝对不能多过 150，多了就会出错。

这个工具真是用来玩的

对签名过的数据进行验证，得到原来的数据。

openssl rsautl -verify -in sig -inkey key.pem

检查原始的签名过的数据：

openssl rsautl -verify -in sig -inkey key.pem -raw -hexdump

0000 - 00 01 ff ff ff ff ff ff-ff ff ff ff ff ff ff ff .....
0010 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff .....
0020 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff .....
0030 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff .....
0040 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff .....
0050 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff .....
0060 - ff ff ff ff ff ff ff ff-ff ff ff ff ff ff ff ff .....
0070 - ff ff ff ff 00 68 65 6c-6c 6f 20 77 6f 72 6c 64 .....hello world

很明显，这是 PKCS#1 结构：使用 0xff 填充模式。

配合指令 asn1parse,可以分析签名的证书，我们在 req 指令里说了怎么做自签名的证书了，现在来分析一下先。

openssl asn1parse -in pca-cert.pem
0:d=0 hl=4 l= 742 cons: SEQUENCE
4:d=1 hl=4 l= 591 cons: SEQUENCE
8:d=2 hl=2 l= 3 cons: cont [0]
10:d=3 hl=2 l= 1 prim: INTEGER :02
13:d=2 hl=2 l= 1 prim: INTEGER :00
16:d=2 hl=2 l= 13 cons: SEQUENCE
18:d=3 hl=2 l= 9 prim: OBJECT :md5WithRSAEncryption
29:d=3 hl=2 l= 0 prim: NULL
31:d=2 hl=2 l= 92 cons: SEQUENCE
33:d=3 hl=2 l= 11 cons: SET
35:d=4 hl=2 l= 9 cons: SEQUENCE
37:d=5 hl=2 l= 3 prim: OBJECT :countryName
42:d=5 hl=2 l= 2 prim: PRINTABLESTRING :AU
....
599:d=1 hl=2 l= 13 cons: SEQUENCE
601:d=2 hl=2 l= 9 prim: OBJECT :md5WithRSAEncryption
612:d=2 hl=2 l= 0 prim: NULL
614:d=1 hl=3 l= 129 prim: BIT STRING

最后一行 BIT STRING 就是实际的签名。我们可以这样子捏它出来：

openssl asn1parse -in pca-cert.pem -out sig -noout -strparse 614

还可以这样子把公共密钥给弄出来：

openssl x509 -in test/testx509.pem -pubkey -noout >;pubkey.pem

我们也可以这样子分析签名：

openssl rsautl -in sig -verify -asn1parse -inkey pubkey.pem -pubin
0:d=0 hl=2 l= 32 cons: SEQUENCE
2:d=1 hl=2 l= 12 cons: SEQUENCE
4:d=2 hl=2 l= 8 prim: OBJECT :md5
14:d=2 hl=2 l= 0 prim: NULL
16:d=1 hl=2 l= 16 prim: OCTET STRING
0000 - f3 46 9e aa 1a 4a 73 c9-37 ea 93 00 48 25 08 b5 .F...Js.7...H%..

这是经过分析后的 ASN1 结构。可以看出来使用的哈希算法是 md5. (很抱歉，我自己试这一行的时候输出结果却完全不同。

0:d=0 hl=2 l= 120 cons: appl [24]

length is greater than 18

完全没有办法看出那里有写哈希算法。)

证书里面的签名部分可以这么捏出来：

openssl asn1parse -in pca-cert.pem -out tbs -noout -strparse 4

这样得到他的哈希算法的细节：

openssl md5 -c tbs

MD5(tbs)= f3:46:9e:aa:1a:4a:73:c9:37:ea:93:00:48:25:08:b5

指令 s_client

用法：

openssl s_client [-connect host:port>;] [-verify depth] [-cert filename]
[-key filename] [-CApath directory] [-CAfile filename] [-reconnect]
[-pause] [-showcerts] [-debug] [-nbio_test] [-state] [-nbio] [-crlf]
[-ign_eof] [-quiet] [-ssl2] [-ssl3] [-tls1] [-no_ssl2] [-no_ssl3]
[-no_tls1] [-bugs] [-cipher cipherlist]

描述：

用于模拟一个普通的 SSL/TLS client, 对于调试和诊断 SSL server 很有用。

OPTIONS

-connect host:port

这个不用解释了吧， 连接的 ip:port.

-cert certname

使用的证书文件。如果 server 不要求要证书，这个可以省略。

-key keyfile

使用的私有密钥文件

-verify depth

指定验证深度。记得 CA 也是分层次的吧？如果对方的证书的签名 CA 不是 Root CA,那么你可以再去验证给该 CA 的证书签名的 CA，一直到 Root CA. 目前的验证操作即使这条 CA 链上的某一个证书验证有问题也不会影响对更深层的 CA 的身份的验证。所以整个 CA 链上的问题都可以检查出来。当然 CA 的验证出问题并不会直接造成连接马上断开，好的应用程序可以让你根据验证结果决定下一步怎么走。

-CApath directory

一个目录。里面全是 CA 的验证资料，该目录必须是"哈希结构". verify 指令里会详细说明。在建立 client 的证书链的时候也有用到这个指令。

-CAfile file

某文件，里面是所有你信任的 CA 的证书的内容。当你要建立 client 的证书链的时候也需要用到这个文件。

-reconnect

使用同样的 session-id 连接同一个 server 五次，用来测试 server 的 session 缓冲功能是否有问题。

-pause

每次读写操作后都挺顿一秒。

-showcerts

显示整条 server 的证书的 CA 的证书链。否则只显示 server 的证书。

-prexit

当程序退出的时候打印 session 的信息。即使连接失败，也会打印出调试信息。一般如果连接成功的话，调试信息将只被打出来一次。本 option 比较有用，因为在一次 SSL 连接中，cipher 也可能改变，或者连接可能失败。要注意的是：有时候打印出来的东西并不一定准确。(这样也行？？eric, 言重了.)

-state

打印 SSL session 的状态， ssl 也是一个协议，当然有状态。

-debug

打印所有的调试信息。

-nbio_test

检查非阻塞 socket 的 I/O 运行情况。

-nbio

使用非阻塞 socket

-crlf

回把你在终端输入的换行回车转化成/r/n 送出去。

-ign_eof

当输入文件到达文件尾的时候并不断开连接。

-quiet

不打印出 session 和证书的信息。同时会打开-ign_eof 这个 option.

-ssl2, -ssl3, -tls1, -no_ssl2, -no_ssl3, -no_tls1

选择用什么版本的协议。很容易理解，不用多解释了吧。

注意，有些很古老的 server 就是不能处理 TLS1, 所以这个时候要关掉 TLS1.n.

-bugs

SSL/TLS 有几处众所周知的 bug, set 了这个 option 使出错的可能性缩小。

-cipher cipherlist

由我们自己来决定选用什么 cipher，尽管是由 server 来决定使用什么 cipher,但它一般都会采用我们送过去的 cipher 列表里的第一个 cipher.

有哪些 cipher 可用？指令 cipher 对这个解释的更清楚。

一旦和某个 SSL server 建立连接之后，所有从 server 得到的数据都会被打印出来，所有你在终端上输入的东西也会被送给 server. 这是人机交互式的。这时候不能 set -quiet 和 -ign_eof 这俩个 option。如果输入的某行开头字母是 R,那么在这里 session 会 renegociate, 如果输入的某行开头是 Q, 那么连接会被断开。你完成整个输入之后连接也会被断开。

If a connection is established with an SSL server then any data received from the server is displayed and any key presses will be sent to the server. When used interactively (which means neither -quiet nor

-ign_eof have been given), the session will be renegociated if the line begins with an R, and if the line begins with a Q or if end of file is reached, the connection will be closed down.

本指令主要是来 debug 一个 SSL server 的。如果想连接某个 SSL HTTP server,输入下一条指令：

openssl s_client -connect servername:443

如果连接成功，你可以用 HTTP 的指令，比如"GET /"什么的去获得网页了。

如果握手失败，原因可能有以下几种：

1. server 需要验证你的证书，但你没有证书

2.如果肯定不是原因 1, 那么就慢慢一个一个 set 以下几个 option

-bugs, -ssl2, -ssl3, -tls1, -no_ssl2, -no_ssl3, -no_tls1

这可能是因为对方的 server 处理 SSL 有 bug.

有的时候，client 会报错：没有证书可以使用，或者供选择的证书列表是空的。这一般是因为 Server 没有把给你签名的 CA 的名字列进它自己认为可以信任的 CA 列表,你可以用检查一下 server 的信任 CA 列表。有的 http server 只在 client 给出了一个 URL 之后才验证 client 的证书，这中情况下要 set -prexit 这个 option, 并且送给 server 一个页面请求。

即使使用-cert 指明使用的证书，如果 server 不要求验证 client 的证书，那么该证书也不会被验证。所以不要以为在命令行里加了-cert 的参数又连接成功就代表你的证书没有问题。

如果验证 server 的证书没有问题，就可以 set -showcerts 来看看 server 的证书的 CA 链了。

其实这个工具并不好用， 自己写一个 client 的会方便很多。

举例时间：

注意，中间的 pop3 协议的指令是我通过终端输入的。其他都是程序输出的对话

过程。具体的每行意义不用解释了。

```
openssl s_client -key server.key -verify 1 -showcerts -prexit -state \
    -crlf -connect 127.0.0.1:5995
    verify depth is 1
    CONNECTED(00000003)
    SSL_connect:before/connect initialization
    SSL_connect:SSLv2/v3 write client hello A
    SSL_connect:SSLv3 read server hello A
    depth=0 /C=AU/ST=Some-State/L=gz/O=ai ltd/OU=sw/CN=fordesign/
    Email=xxx@xxx.xom
    verify error:num=20:unable to get local issuer certificate
    verify return:1
    depth=0 /C=AU/ST=Some-State/L=gz/O=ai ltd/OU=sw/CN=fordesign/
    Email=xxx@xxx.xom
    verify error:num=27:certificate not trusted
    verify return:1
    depth=0 /C=AU/ST=Some-State/L=gz/O=ai ltd/OU=sw/CN=fordesign/
    Email=xxx@xxx.xom
    verify error:num=21:unable to verify the first certificate
    verify return:1
    SSL_connect:SSLv3 read server certificate A
    SSL_connect:SSLv3 read server done A
```

SSL_connect:SSLv3 write client key exchange A
SSL_connect:SSLv3 write change cipher spec A
SSL_connect:SSLv3 write finished A
SSL_connect:SSLv3 flush data
SSL_connect:SSLv3 read finished A
Certificate chain
0 s:/C=AU/ST=Some-State/L=gz/O=ai ltd/OU=sw/CN=fordesign/Email=xxx@xxx.xom
i:/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd/CN=fordesign/
Email=fordeisgn@21cn.com
----BEGIN CERTIFICATE-----
MIIDdzCCAuCgAwIBAgIBATANBgkqhkiG9w0BAQQFADB8MQswCQYDVQQGEwJBVTET
MBEGA1UECBMKU29tZS1TdGF0ZTEhMB8GA1UEChMYSW50ZXJuZXQgV2lkZ2l0cyBQ
dHkgTHRkMRIwEAYDVQQDEwlmb3JkZXNpZ24xITAfBgkqhkiG9w0BCQEWEmZvcmRl

aXNnbkAyMWNuLmNvbTAeFw0wMDExMTIwNjE5MDNaFw0wMTExMTIwNjE5MDNaMH0x
CzAJBgNVBAYTAkFVMRMwEQYDVQQIEwpTb21lLVN0YXRlMQswCQYDVQQHEwJnejEP

MA0GA1UEChMGYWkgbHRkMQswCQYDVQQLEwJzdzESMBAGA1UEAxMJZm9yZGVzaWdu
MRowGAYJKoZIhvcNAQkBFgt4eHhAeHh4LnhvbTCBnzANBgkqhkiG9w0BAQEFAAOB
jQAwgYkCgYEAuQVRVaCyF+a8/927cA9CjlrSEGOL17+Fk1U6rqZ8fJ6UR+kvhUUk
fgyMmzrw4bhnZlk2NV5afZEhiiNdRri9f8loklGRXRkDfmhyUWtjiFWUDtzkuQoT
6jhWfoqGNCKh/92cjq2wicJpp40wZGlfwTwSnmjN9/eNVwEoXigSy5ECAwEAAaOC

AQYwggECMAkGA1UdEwQCMAAwLAYJYIZIAYb4QgENBB8WHU9wZW5TU0wgR2VuZXJh
dGVkIENlcnRpZmljYXRlMB0GA1UdDgQWBBS+WovE66PrvCAtojYMV5pEUYZtjzCB
pwYDVR0jBIGfMIGcgBRpQYdVvVKZ0PXsEX8KAVNYTgt896GBgKR+MHwxCzAJBgNV
BAYTAkFVMRMwEQYDVQQIEwpTb21lLVN0YXRlMSEwHwYDVQQKExhJbnRlcm5ldCBX
aWRnaXRzIFB0eSBMdGQxEjAQBgNVBAMTCWZvcmRlc2lnbjEhMB8GCSqGSIb3DQEJ
ARYSZm9yZGVpc2duQDIxY24uY29tggEAMA0GCSqGSIb3DQEBBAUAA4GBADDOp/O/
o3mBZV4vc3mm2C6CcnB7rRSYEoGm6T6OZsi8mxyF5w1NOK5oI5fJU8xcf8aYFVoi
0i4LlsiQw+EwpnjUXfUBxp/g4Cazlv57mSS6h1t4a/BPOIwzcZGpo/R3g/fOPwsF
F/2RC++81s6k78iezFrTs9vnsm/G4vRjngLI
-----END CERTIFICATE-----

Server certificate
subject=/C=AU/ST=Some-State/L=gz/O=ai ltd/OU=sw/CN=fordesign/
Email=xxx@xxx.xom
issuer=/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd/CN=fordesign/
Email=fordeisgn@21cn.com

No client certificate CA names sent

SSL handshake has read 1069 bytes and written 342 bytes

New, TLSv1/SSLv3, Cipher is DES-CBC3-SHA
Server public key is 1024 bit
SSL-Session:
Protocol : SSLv3
Cipher : DES-CBC3-SHA
Session-ID: E1EC3B051F5DB8E2E3D3CD10E4C0412501DDD6641ACA932B65
DC25DCD0A3A86E
Session-ID-ctx:
Master-Key: 47DB3A86375DB2E99982AFD8F5B382B4316385694B01B74BFC3
FA26C7DBD489CABE0EE1B20CE8E95E4ABF930099084B0
Key-Arg : None
Start Time: 974010506
Timeout : 300 (sec)
Verify return code: 0 (ok)

+OK AIMC POP service (sol7.gzai.com) is ready.
user ssltest0
+OK Please enter password for user <ssltest0>;.
pass ssltest0
+OK ssltest0 has 12 message (282948 octets)
list
+OK 12 messages (282948 octets)
1 21230
2 21230
3 21230
4 21230
5 21229
6 21230
7 21230
8 21230
9 111511
10 136
11 141
12 1321
.
quit
+OK Pop server at (sol7.gzai.com) signing off.
read:errno=0
SSL3 alert write:warning:close notify

Certificate chain
0 s:/C=AU/ST=Some-State/L=gz/O=ai ltd/OU=sw/CN=fordesign/
Email=xxx@xxx.xom
i:/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd/CN=fordesign/

Email=fordeisgn@21cn.com
-----BEGIN CERTIFICATE-----
MIIDdzCCAuCgAwIBAgIBATANBgkqhkiG9w0BAQQFADB8MQswCQYDVQQGEwJBVTET
MBEGA1UECBMKU29tZS1TdGF0ZTEhMB8GA1UEChMYSW50ZXJuZXQgV2lkZ2l0cyBQ
dHkgTHRkMRIwEAYDVQQDEwlmb3JkZXNpZ24xITAfBgkqhkiG9w0BCQEWEmZvcmRl

aXNnbkAyMWNuLmNvbTAeFw0wMDExMTIwNjE5MDNaFw0wMTExMTIwNjE5MDNaMH0x
CzAJBgNVBAYTAkFVMRMwEQYDVQQIEwpTb21lLVN0YXRlMQswCQYDVQQHEwJnejEP

MA0GA1UEChMGYWkgbHRkMQswCQYDVQQLEwJzdzESMBAGA1UEAxMJZm9yZGVzaWdu
MRowGAYJKoZIhvcNAQkBFgt4eHhAeHh4LnhvbTCBnzANBgkqhkiG9w0BAQEFAAOB
jQAwgYkCgYEAuQVRVaCyF+a8/927cA9CjlrSEGOL17+Fk1U6rqZ8fJ6UR+kvhUUk
fgyMmzrw4bhnZlk2NV5afZEhiiNdRri9f8loklGRXRkDfmhyUWtjiFWUDtzkuQoT
6jhWfoqGNCKh/92cjq2wicJpp40wZGlfwTwSnmjN9/eNVwEoXigSy5ECAwEAAaOC

AQYwggECMAkGA1UdEwQCMAAwLAYJYIZIAYb4QgENBB8WHU9wZW5TU0wgR2VuZXJh
dGVkIENlcnRpZmljYXRlMB0GA1UdDgQWBBS+WovE66PrvCAtojYMV5pEUYZtjzCB
pwYDVR0jBIGfMIGcgBRpQYdVvVKZ0PXsEX8KAVNYTgt896GBgKR+MHwxCzAJBgNV
BAYTAkFVMRMwEQYDVQQIEwpTb21lLVN0YXRlMSEwHwYDVQQKExhJbnRlcm5ldCBX
aWRnaXRzIFB0eSBMdGQxEjAQBgNVBAMTCWZvcmRlc2lnbjEhMB8GCSqGSIb3DQEJ
ARYSZm9yZGVpc2duQDIxY24uY29tggEAMA0GCSqGSIb3DQEBBAUAA4GBADDOp/O/
o3mBZV4vc3mm2C6CcnB7rRSYEoGm6T6OZsi8mxyF5w1NOK5oI5fJU8xcf8aYFVoi
0i4LlsiQw+EwpnjUXfUBxp/g4Cazlv57mSS6h1t4a/BPOIwzcZGpo/R3g/fOPwsF
F/2RC++81s6k78iezFrTs9vnsm/G4vRjngLI
-----END CERTIFICATE-----

Server certificate
subject=/C=AU/ST=Some-State/L=gz/O=ai ltd/OU=sw/CN=fordesign/
Email=xxx@xxx.xom
issuer=/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd/CN=fordesign/
Email=fordeisgn@21cn.com

No client certificate CA names sent

SSL handshake has read 1579 bytes and written 535 bytes

New, TLSv1/SSLv3, Cipher is DES-CBC3-SHA
Server public key is 1024 bit
SSL-Session:
Protocol : SSLv3
Cipher : DES-CBC3-SHA
Session-ID: E1EC3B051F5DB8E2E3D3CD10E4C0412501DDD6641ACA932B65DC2
5DCD0A3A86E
Session-ID-ctx:

Master-Key: 47DB3A86375DB2E99982AFD8F5B382B4316385694B01B74BFC3FA
26C7DBD489CABE0EE1B20CE8E95E4ABF930099084B0
Key-Arg : None
Start Time: 974010506
Timeout : 300 (sec)
Verify return code: 0 (ok)

指令 s_server

用法：

openssl s_server [-accept port] [-context id] [-verify depth]
[-Verify depth] [-cert filename] [-key keyfile] [-dcert filename]
[-dkey keyfile] [-dhparam filename] [-nbio] [-nbio_test] [-crlf]
[-debug] [-state] [-CApath directory] [-CAfile filename] [-nocert]
[-cipher cipherlist] [-quiet] [-no_tmp_rsa] [-ssl2] [-ssl3] [-tls1]
[-no_ssl2] [-no_ssl3] [-no_tls1] [-no_dhe] [-bugs] [-hack] [-www]
[-WWW] [-engine id]

说明：
和 s_client 是反义词， 模拟一个实现了 SSL 的 server.

OPTIONS
-accept port
监听的 TCP 端口。缺省为 4433.
-context id
设置 SSL context 的 id, 可以设置为任何值。SSL context 是什么？编程的章节会详细介绍的。你也可以不 set 这个 option, 有缺省的给你用的。
-cert certname
使用的证书文件名。缺省使用 ./server.pem
-key keyfile
使用的私有密钥文件。如果没有指定，那么证书文件会被使用。????
The private key to use. If not specified then the certificate
file will be used.
-dcert filename, -dkey keyname
指定一个附加的证书文件和私有密钥文件。不同的 cipher 需要不同的证书和 私有密钥文件。这个不同的 cipher 主要指 cipher 里面的不对称加密算法不同 比如基于 RSA 的 cipher 需要的是 RSA 的私有密钥文件和证书,而基于 DSA 的算法 则需要的是 DSA 的私有密钥文件和证书.这个 option 可以让这样我们的 server 同时支持俩种算法的 cipher 成为可能。

-nocert

如果 server 不想使用任何证书，set 这个 option.

目前只有 anonymous DH 算法有需要这么做。

-dhparam filename

使用的 DH 参数文件名。如果没有 set, 那么 server 会试图去从证书文件里面获得这些参数。如果证书里面没有这么参数，一些 hard code 的参数就被调用。

-nodhe

禁止使用基于 EDH 的 cipher.

-no_tmp_rsa

现在的出口 cipher 有时会使用临时 RSA 密钥。那就是说每次对话的时候临时生成密钥对。本 optio 就是用来禁止这种情况的。

-verify depth, -Verify depth

意义和 s_client 的这个 option 一样，但同时表示必须验证 client 的证书。不记得 server 对 client 的证书验证是可以选的吗？-verify 表示向 client 要求证书，但 client 还是可以选择不发送证书，-Verify 表示一定要 client 的证书验证，否则握手告吹。

-CApath directory

-CAfile file

-state

-debug

-nbio_test

-nbio

-crlf

-quiet

-ssl2, -ssl3, -tls1, -no_ssl2, -no_ssl3, -no_tls1

-bugs

-cipher cipherlist

这些 option 于 s_client 的同名 option 意义相同。

下面俩个指令模拟一个简单的 http server.

-www

当 client 连接上来的时候，发回一个网页，内容就是 SSL 握手的一些内容。

-WWW

用来把具体某个文件当网页发回给 client 的请求。比如 client 的 URL 请求是 https://myhost/page.html ,就把 ./page.html 发回给 client.如果没有 set -www, -WWW 这俩个 option, 当一个 ssl client 连接上来的话它所发过来的任何东西都会显示出来，你在终端输入的任何东西都会发回 给 client.你可以通过在终端输入的行的第一个字母控制一些行为

q:

中断当前连接，但不关闭 server.

Q

中断当前连接，退出程序。

r

进行 renegotiate 行为。

R

进行 renegotiate 行为, 并且要求 client 的证书 。

P

在 TCP 层直接送一些明文。这会使 client 认为我们没有按协议的游戏规则进行通信而断开连接。

S

打印出 session-cache 的状态信息。session-cache 在编程章节会详细介绍。

NOTES

用于调试 ssl client.

下一条指令用来模拟一个小的 http server, 监听 443 端口。

openssl s_server -accept 443 -www

session 的参数可以用 sess_id 指令打印。

我对这条指令实在没有兴趣，一般使用 openssl 都是用做 server, 没有机会调试 client.我甚至没有用过这个指令。

指令 sess_id

用法：

openssl sess_id [-inform PEM|DER] [-outform PEM|DER] [-in filename]
[-out filename] [-text] [-noout] [-context ID]

说明：

本指令是处理 SSL_SESSION 结构的，可以打印出其中的细节。这也是一个调试工具。

-inform DER|PEM

指定输入格式是 DER 还是 PEM.

-outform DER|PEM

指定输出格式是 DER 还是 PEM

-in filename

指定输入的含有 session 信息的文件名，可以通过标准输入得到。

-out filename

指定输出 session 信息的文件名

-text

打印出明文的密钥的各个部件。

-cert

set 本 option 将会把 session 中使用的证书打印出来。如果-text 也被 set, 那么将会把其用文本格式打印出来。

-noout

不打印出 session 的编码版本。

-context ID

设置 session id. 不常用的一个 option.

本指令的典型的输出是：

SSL-Session:
Protocol : TLSv1
Cipher : 0016
Session-ID: 871E62626C554CE95488823752CBD5F3673A3EF3DCE9
C67BD916C809914B40ED
Session-ID-ctx: 01000000
Master-Key: A7CEFC571974BE02CAC305269DC59F76EA9F0B180CB66
42697A68251F2D2BB57E51DBBB4C7885573192AE9AEE220FACD
Key-Arg : None
Start Time: 948459261
Timeout : 300 (sec)
Verify return code 0 (ok)
Protocol
使用的协议版本信息。
Cipher
使用的 cipher, 这里是原始的 SSL/TLS 里定义的代码。
Session-ID
16 进制的 session id
Session-ID-ctx
session-id-ctx 的 16 进制格式。
Master-Key
ssl session master key.
Key-Arg
key 的参数，只用于 SSLv2
Start Time
session 开始的时间。标准的 unix 格式。
Timeout
session-timeout 时间。
Verify return code
证书验证返回值.
ssl session 文件的 pem 标准格式的第一行和最后一行是：
---BEGIN SSL SESSION PARAMETERS-----
-----END SSL SESSION PARAMETERS-----

因为 ssl session 输出包含握手的重要信息：master key, 所以一定要用一定的加密算法把起输出加密。一般是禁止在实际应用中把 session 的信息输出。我没用过这个工具。研究 source 的时候这个可能有点用。

指令 speed

用法：

openssl speed [-elapsed] [md2] [mdc2] [md5] [hmac] [sha1] [rmd160]
[idea-cbc] [rc2-cbc] [rc5-cbc] [bf-cbc] [des-cbc] [des-ede3]
[rc4] [rsa512] [rsa1024] [rsa2048] [rsa4096] [dsa512]
[dsa1024] [dsa2048] [idea] [rc2] [des] [rsa] [blowfish]

说明：
算法在你的机器上的测试工具。

OPTIONS
-elapsed
set 了这个 option 将使测试结果是我们比较容易懂的时间格式，否则将是和 time 指令那样子显示的 cpu 时间。
其他的 option 都是算法了。

指令 version

用法：

openssl version [-a] [-v] [-b] [-o] [-f] [-p]

说明：
用来打印版本信息的。最没用的指令和最简单的指令。

OPTIONS
-a
打印所有信息， 相当于把其他 option 全 set 起来。
当你向 openssl 官方站点报 bug 的时候,需要把这个指令列出来的东西也告诉他们
-v
打印当前 openssl 的版本信息。
-b
打印当前版本的 openssl 是什么时候弄出来的
-o
建立库的时候的各种于加密算法和机器字节有关的信息。
-c
编译时候的编译其的参数
-p
平台信息

指令 x509

用法：

openssl x509 [-inform DER|PEM|NET] [-outform DER|PEM|NET]
[-keyform DER|PEM][-CAform DER|PEM] [-CAkeyform DER|PEM]
[-in filename][-out filename] [-serial] [-hash] [-subject]
[-issuer] [-nameopt option] [-email] [-startdate] [-enddate]
[-purpose] [-dates] [-modulus] [-fingerprint] [-alias]
[-noout] [-trustout] [-clrtrust] [-clrreject] [-addtrust arg]
[-addreject arg] [-setalias arg] [-days arg]
[-signkey filename][-x509toreq] [-req] [-CA filename]
[-CAkey filename] [-CAcreateserial] [-CAserial filename]
[-text] [-C] [-md2|-md5|-sha1|-mdc2] [-clrext]
[-extfile filename] [-extensions section]

说明：

本指令是一个功能很丰富的证书处理工具。可以用来显示证书的内容，转换其格式，给 CSR 签名等等。由于功能太多，我们按功能分成几部分来讲。

输入，输出等一些一般性的 option

-inform DER|PEM|NET

指定输入文件的格式。

-outform DER|PEM|NET

指定输出文件格式

-in filename

指定输入文件名

-out filename

指定输出文件名

-md2|-md5|-sha1|-mdc2

指定使用的哈希算法。缺省的是 MD5 于打印有关的 option

-text

用文本方式详细打印出该证书的所有细节。

-noout

不打印出请求的编码版本信息。

-modulus

打印出公共密钥的系数值。没研究过 RSA 就别用这个了。

-serial

打印出证书的系列号。

-hash

把证书的拥有者名称的哈希值给打印出来。

-subject

打印出证书拥有者的名字。

-issuer

打印证书颁发者名字。

-nameopt option

指定用什么格式打印上俩个 option 的输出。

后面有详细的介绍。

-email

如果有，打印出证书申请者的 email 地址

-startdate

打印证书的起始有效时间

-enddate

打印证书的到期时间

-dates

把上俩个 option 都给打印出来

-fingerprint

打印 DER 格式的证书的 DER 版本信息。

-C

用 C 代码风格打印结果。

与证书信任有关的 option

一个可以信任的证书的就是一个普通证书，但有一些附加项指定其可以用于哪些用途和不可以用于哪些用途, 该证书还应该有一个"别名"。

一般来说验证一个证书的合法性的时候，相关的证书链上至少有一个证书必须是一个可以信任的证书。缺省的认为如果该证书链上的 Root CA 的证书可以信任，那么整条链上其他证书都可以用于任何用途。

以下的几个 option 只用来验证 Root CA 的证书。CA 在颁发证书的时候可以控制该证书的用途，比如颁发可以用于 SSL client 而不能用于 SSL server 的证书。

-trustout

打印出可以信任的证书。

-setalias arg

设置证书别名。比如你可以把一个证书叫"fordesign's certificate", 那么以后就可以使用这个别名来引用这个证书。

-alias

打印证书别名。

-clrtrust

清除证书附加项里所有有关用途允许的内容。

-clrreject

清除证书附加项里所有有关用途禁止的内容。

-addtrust arg

添加证书附加项里所有有关用途允许的内容。

-addreject arg

添加证书附加项里所有有关用途禁止的内容。

-purpose

打印出证书附加项里所有有关用途允许和用途禁止的内容。

与签名有关的 otpion

本指令可以用来处理 CSR 和给证书签名，就象一个 CA

-signkey filename

使用这个 option 同时必须提供私有密钥文件。这样把输入的文件变成字签名的证书。

如果输入的文件是一个证书，那么它的颁发者会被 set 成其拥有者.其他相关的项也会被改成符合自签名特征的证书项。

如果输入的文件是 CSR, 那么就生成自签名文件。

-clrext

把证书的扩展项删除。

-keyform PEM|DER

指定使用的私有密钥文件格式。

-days arg

指定证书的有效时间长短。缺省为 30 天。

-x509toreq

把一个证书转化成 CSR.用-signkey 指定私有密钥文件

-req

缺省的认为输入文件是证书文件，set 了这个 option 说明输入文件是 CSR.

-CA filename

指定签名用的 CA 的证书文件名。

-CAkey filename

指定 CA 私有密钥文件。如果这个 option 没有参数输入，那么缺省认为私有密钥在 CA 证书文件里有。

-CAserial filename

指定 CA 的证书系列号文件。证书系列号文件在前面介绍过，这里不重复了。

-CAcreateserial filename

如果没有 CA 系列号文件，那么本 option 将生成一个。

-extfile filename

指定包含证书扩展项的文件名。如果没有，那么生成的证书将没有任何扩展项。

-extensions section

指定文件中包含要增加的扩展项的 section

上面俩个 option 有点难明白是吧？后面有举例。

与名字有关的 option.这些 option 决定证书拥有者/颁发者的打印方式。缺省方式是印在一行中。

这里有必要解释一下这个证书拥有者/颁发者是什么回事。它不是我们常识里的一个名字，而是一个结构，包含很多字段。

英文分别叫 subject name/issuer name.下面是一个 subject name 的例子

subject=

countryName = AU

stateOrProvinceName = Some-State

localityName = gz

organizationName = ai ltd

organizationalUnitName = sw

commonName = fordesign

emailAddress = xxx@xxx.xom

-nameopt

这个 option 后面的参数就是决定打印的方式，其参数有以下可选：

compat

使用以前版本的格式，等于没有设置任何以下 option

RFC2253

使用 RFC2253 规定的格式。

oneline

所有名字打印在一行里面。

multiline

名字里的各个字段用多行打印出来。

上面那几个是最常用的了，下面的这些我怎么用怎么不对，也许以后研究 source 在完善这里了。

esc_2253 esc_ctrl esc_msb use_quote utf8 no_type show_type dump_der

dump_nostr dump_all dump_unknown sep_comma_plus sep_comma_plus_space

sep_semi_plus_space sep_multiline dn_rev nofname, sname, lname, oid spc_eq

举例时间：

打印出证书的内容：

openssl x509 -in cert.pem -noout -text

打印出证书的系列号

openssl x509 -in cert.pem -noout -serial

打印出证书的拥有者名字

openssl x509 -in cert.pem -noout -subject

以 RFC2253 规定的格式打印出证书的拥有者名字

openssl x509 -in cert.pem -noout -subject -nameopt RFC2253

在支持 UTF8 的终端一行过打印出证书的拥有者名字

openssl x509 -in cert.pem -noout -subject -nameopt oneline -nameopt -escmsb

打印出证书的 MD5 特征参数

openssl x509 -in cert.pem -noout -fingerprint

打印出证书的 SHA 特征参数

openssl x509 -sha1 -in cert.pem -noout -fingerprint

把 PEM 格式的证书转化成 DER 格式

openssl x509 -in cert.pem -inform PEM -out cert.der -outform DER

把一个证书转化成 CSR

openssl x509 -x509toreq -in cert.pem -out req.pem -signkey key.pem

给一个 CSR 进行处理，颁发字签名证书，增加 CA 扩展项

openssl x509 -req -in careq.pem -extfile openssl.cnf -extensions v3_ca -signkey key.pem -out cacert.pem

给一个 CSR 签名，增加用户证书扩展项

openssl x509 -req -in req.pem -extfile openssl.cnf -extensions v3_usr -CA cacert.pem -CAkey key.pem -CAcreateserial

把某证书转化成用于 SSL client 可信任证书, 增加别名 alias

openssl x509 -in cert.pem -addtrust sslclient -alias "Steve's Class 1 CA" -out trust.pem

上面有很多地方涉及到证书扩展/证书用途，这里解释一下：

我们知道一个证书是包含很多内容的，除了基本的那几个之外，还有很多扩展的项。比如证书用途，其实也只是证书扩展项中的一个。

我们来看看一个 CA 证书的内容：
openssl x509 -in ca.crt -noout -text
Certificate:
Data:
Version: 3 (0x2)
Serial Number: 0 (0x0)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=AU, ST=Some-State, O=Internet Widgits Pty Ltd,
CN=fordesign/Email=fordeisgn@21cn.com
Validity
Not Before: Nov 9 04:02:07 2000 GMT
Not After : Nov 9 04:02:07 2001 GMT
Subject: C=AU, ST=Some-State, O=Internet Widgits Pty Ltd,
CN=fordesign/Email=fordeisgn@21cn.com
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (1024 bit)
Modulus (1024 bit):
00:e7:62:1b:fb:78:33:d7:fa:c4:83:fb:2c:65:c1:
08:03:1f:3b:79:b9:66:bb:31:aa:77:d4:47:ac:be:
e5:20:ce:ed:1f:b2:b5:4c:79:c9:9b:ad:1d:0b:7f:
84:49:03:6b:79:1a:fd:05:ca:36:b3:90:b8:5c:c0:
26:93:c0:02:eb:78:d6:8b:e1:91:df:85:39:33:fc:
3d:59:e9:7f:58:34:bf:be:ef:bd:22:a5:be:26:c0:
16:9b:41:36:45:05:fe:f9:b2:05:42:04:c9:3b:28:
c1:0a:48:f4:c7:d6:a8:8c:f9:2c:c1:1e:f5:8b:dc:
19:59:7c:47:f7:91:cc:5d:75
Exponent: 65537 (0x10001)
X509v3 extensions:
X509v3 Subject Key Identifier:
69:41:87:55:BD:52:99:D0:F5:EC:11:7F:0A:01:53:58:4E:0B:7C:F7
X509v3 Authority Key Identifier:
keyid:69:41:87:55:BD:52:99:D0:F5:EC:11:7F:0A:01:53:58:
4E:0B:7C:F7
DirName:/C=AU/ST=Some-State/O=Internet Widgits Pty
Ltd/CN=fordesign/Email=fordeisgn@21cn.com
serial:00
X509v3 Basic Constraints:
CA:TRUE
Signature Algorithm: md5WithRSAEncryption
79:14:99:4a:8f:64:63:ab:fb:ad:fe:bc:ba:df:53:97:c6:92:
41:4d:de:fc:59:98:39:36:36:8e:c6:05:8d:0a:bc:49:d6:20:
02:9d:a2:5f:0f:03:12:1b:f2:af:23:90:7f:b1:6a:86:e8:3e:
0b:2c:fd:11:89:86:c3:21:3c:25:e2:9c:de:64:7a:14:82:32:

22:e1:35:be:39:90:f5:41:60:1a:77:2e:9f:d9:50:f4:81:a4:
67:b5:3e:12:e5:06:da:1f:d9:e3:93:2d:fe:a1:2f:a9:f3:25:
05:03:00:24:00:f1:5d:1f:d7:77:8b:c8:db:62:82:32:66:fd:
10:fa

是否看到我们先提到过的 subject name/issuer name.本证书中这俩个字段是一样的，明显是自签名证书，是一个 Root CA 的证书。从 X509v3 extension 开始就是证书扩展项了。

这里有个 X509v3 Basic constraints. 里面的 CA 字段决定该证书是否可以做 CA 的证书，这里是 TURE。如果这个字段没有，那么会根据其他内容决定该证书是否可以做 CA 证书。

如果是 X509v1 证书，又没有这个扩展项，那么只要 subject name 和 issuer name 相同，就认为是 Root CA 证书了。

本例的证书没有证书用途扩展项，它是一个叫 keyUseage 的字段。

举个例子就可以看出该字段目前可以有以下值

openssl x509 -purpose -in server.crt
Certificate purposes:
SSL client : Yes
SSL client CA : No
SSL server : Yes
SSL server CA : No
Netscape SSL server : Yes
Netscape SSL server CA : No
S/MIME signing : Yes
S/MIME signing CA : No
S/MIME encryption : Yes
S/MIME encryption CA : No
CRL signing : Yes
CRL signing CA : No
Any Purpose : Yes
Any Purpose CA : Yes
SSL Client
SSL Client CA

每个值的具体意义应该可以看名字就知道了吧？

X509 指令在转化证书成 CSR 的时候没有办法把证书里的扩展项转化过去给 CSR 签名做证书的时候，如果使用了多个 option,应该自己保证这些 option 没有冲突。

编后语

注意： 我估计 openssl 最开始是在 linux 下开发的。大家可以看一看在 linxu 下有这么一个文件：/dev/urandom, 在 sparc 下没有。这个文件有什么用？你可以随时找它要一个随机数。在加密算法产生 key 的时候，我们需要一颗种子：seed。这个 seed 就是找/dev/urandom 要的那个随机数。那么在 sparc 下，由于没有这么一个设备，很多 openssl 的函数会报错："PRNG not seeded". 解决方法是：.....

有个补充：关于 sparc 系统下无/dev/urandom 的问题，可以考虑安装 patch112438-03(sol8 : http://sunsolve.sun.com/search/advsearch.do?collection=PATCH&type=collections&max=50&language=en&queryKey5=112438&toDocument=yes),需要 reboot.