加密类型和算法概念

世界上没有万无一失的安全性。

裸奔协议： http ftp smtp telent

## 安全三要素

1. 机密性：网路中是的数据不应该明文传输。
2. 数据完整性：数据在网路中传播，要确保数据的完整性，不能随意被别人篡改。
3. 身份验证：数据的完整性并不能意味一切ok，如果对方冒名顶替，就像钓鱼网站一样。就需要相互身份验证。

加密 = 转换规则（算法，数学原理）+ 秘钥（自己设置）

## 对称加密

通信双方秘钥是相同的，计算速度相对快，安全性完全依赖于秘钥因为算法是公开的。一但通信双方数量增加，秘钥数量也要增加，秘钥管理也是很头痛的事情。

存在的问题：帮我们解决了机密性问题，但是没有很好管理秘钥的方法。

常用的加密算法

DES (Data Encrption standard ) IBM提供,56bit秘钥 ,可以暴力破解，不安全了，现在没使用。

3DES : DES加密后再DES加密，在DES加密故称之为3DES。用的少。不信任他的安全性。

AES： Advance Encrption standard 高级加密标准。默认128bit秘钥，他又很多变种。如

AES192， AES256 ，AES512 ，秘钥越长加解密越慢，安全性越高。

Blowfish

## 单向加密

单向加密算法也就是信息摘要算法，用来生成特征码（指纹）。

单向加密双方就是提取数据特征码（指纹）附加在明文后面，对方接受到信息后提取特征码，然后两个特征码进行对比就知道发过来的信息有没有被篡改。如我常用的MD5, SHA1,

1. 输入一样，输出必然相同
2. 雪崩效应，微小的改变会引起结果的巨大改变
3. 定长输出，无论原始数据多大，结果大小相同
4. 不可逆，无法根据特征码还原原来的数据

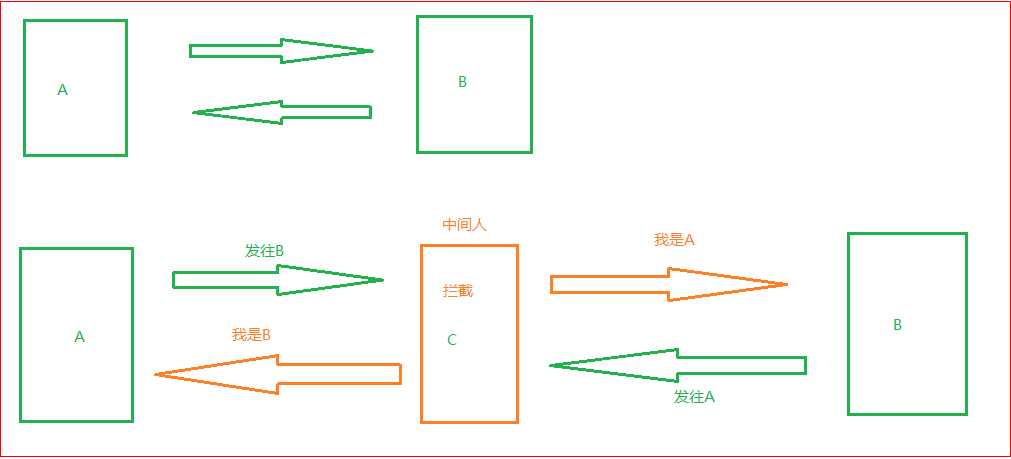
存在的问题：虽然能保证数据的完整性，但是不能应对中间人攻击，不能保证机密性，中间人获取明文后重新生成特征码，发送给接收方

算法

MD4 MD5 SHA1 SHA SHA192 SHA256 SHA384 SHA512

CRC-32 循环冗余校验码，不是为了加密，是为了校验。

这些都是定长输出。



数据机密性并不能保证机密性本身，因为C可以截获消息，重新计算特征码，发给B，B接收到后重新计算特征码。就认为是A发来的。所以数据机密性这种手段并不能保证机密性本身，还需要借助其他方法，这种方法就是使用对称加密对特征码加密。A和B使用相同秘钥，A在发送消息给B的时候，计算特征码后，使用秘钥对特征码加密，B在接收到消息后使用秘钥对特征码解密。对比特征码，如果相同，说明消息在途中没有被篡改。

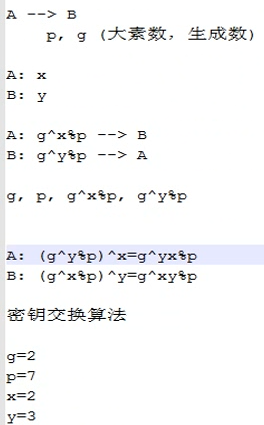
## 秘钥交换协议

问题来了，如果A和B从来没有互相通信，约定密码，怎么办？

双方可以协商生成秘钥。A和B协商生成秘钥的过程，C也不知道他们在干嘛。

协商生成密码： 秘钥交换（Internate key exchange ,IKE），需要互联网协议支撑

Diffie-Hellman协议就是著名的秘钥交换算法，离散对数原理。



A == B ，秘钥相同，就可以使用相同密码对特征码加密。这就解决了秘钥管理的问题，不需要用户自己来管理秘钥。软件本身就可以管理这些秘钥。安全可靠，管理简单。

存在的问题：秘钥交换解决了事先预定密码的问题。但是又陷入了另外一个问题。中间人C又跳出来了，A怎么知道是与B协商而不是和C协商？这就是身份验证的问题。通过秘钥A是无法验证B的身份。

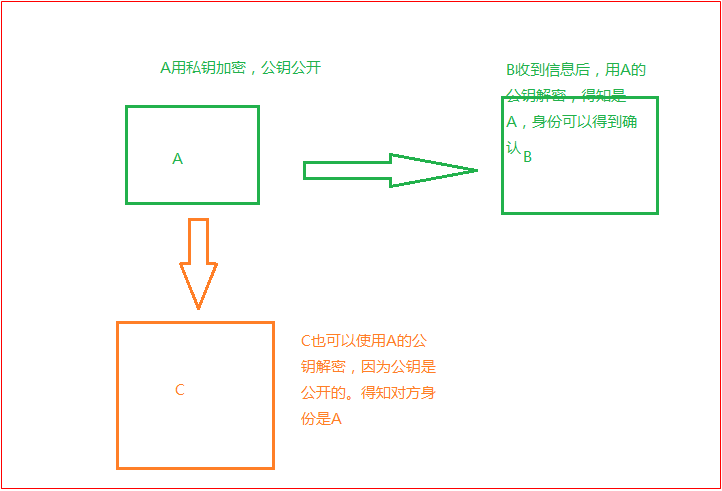
## 公钥加密

公钥加密 **非对称加密**

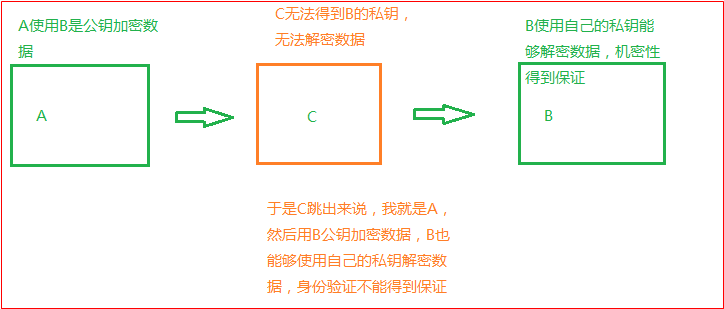
密钥对：

私钥（S）：只有自己能看到。

公钥（P）：任何人都可以看到，从私钥中提取生成。



发送方A用自己的私钥加密，B用A的公钥解密，B就知道对方是A，得到身份验证，私钥只有自己才有，就能确定身份，然而私钥加密并不能保证数据的机密性，任何持有公钥都可以解密。就如C也可也使用A的公钥解密，得知消息内容。



发送方A用对方B的公钥加密数据能保证数据的机密性，公钥加密只有私钥解密。A发送给B的信息，只有B使用自己的私钥能够解密，C无从得知B的私钥。C就无法解密数据。所以A发给B的数据机密性能够得到保证。

发送方用自己的私钥加密，可以实现身份验证。但是无法保证数据的机密性

发送方用对方的公钥加密，可以实现数据的机密性。但是无法验证对方的身份。

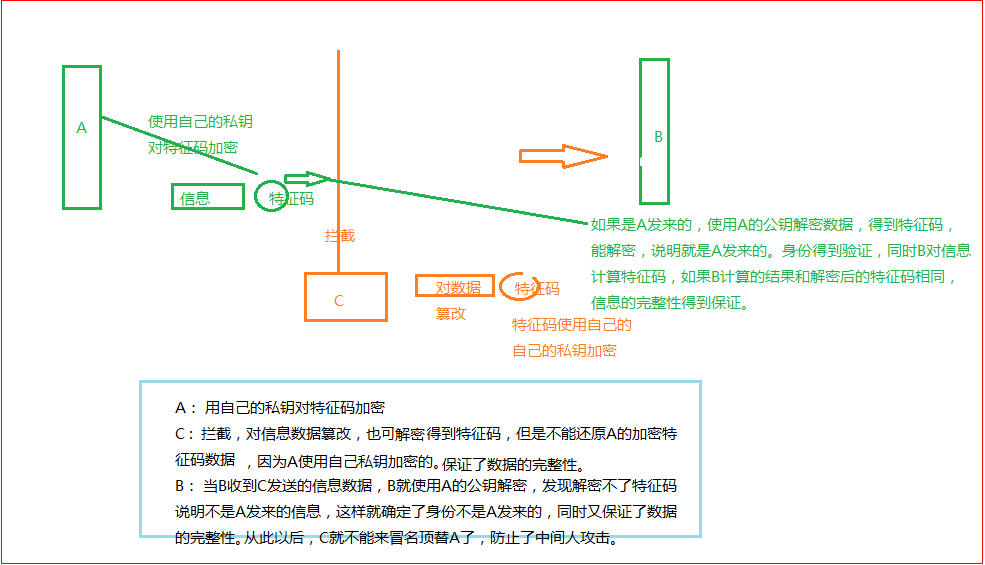
我想跟谁通信，就用谁的公钥加密。谁要和我通信，他就用我的公钥加密。反正我就记住我的一对秘钥，解决了对称加密的密码管理的问题。

有身份验证无机密性，有机密性无身份验证。

事实上公钥加密算法很少用来加密数据，密码太长，速度太慢，比对称加密要慢上3个数量级。公钥加密算法通常用来进行身份验证

问题来了：

A和B协商随机生成对称秘钥，为了加密消息的特征码的。无法事先约定，身份验证无法得到保证。

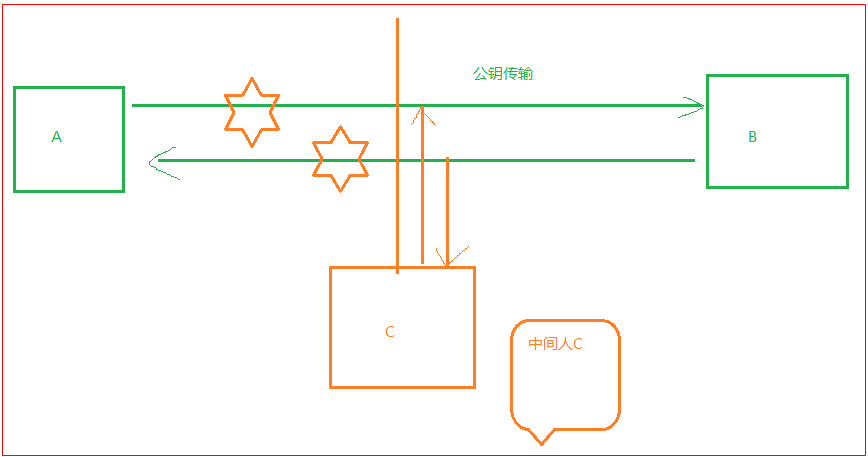


注：A 消息和加密后的特征码都使用B的公钥加密

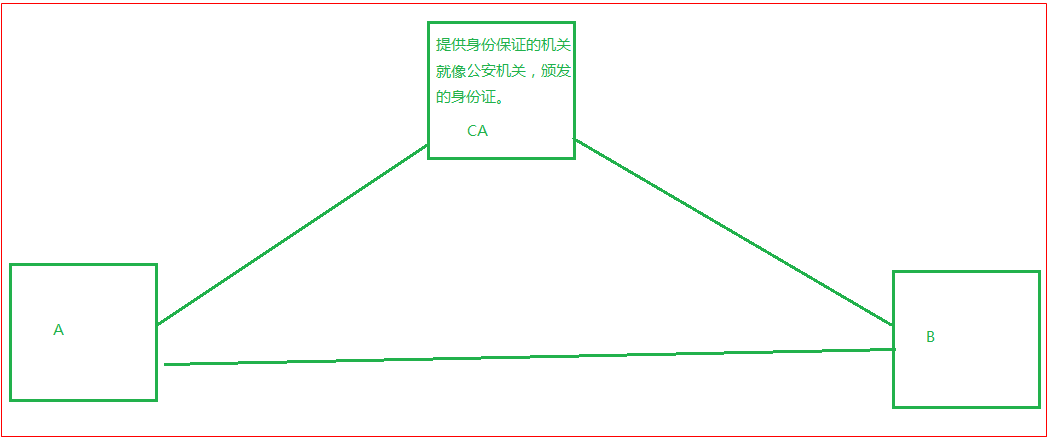
B 消息收到后使用B的私钥先解密

所以非对称加密最主要的作用就是用来进行身份验证。

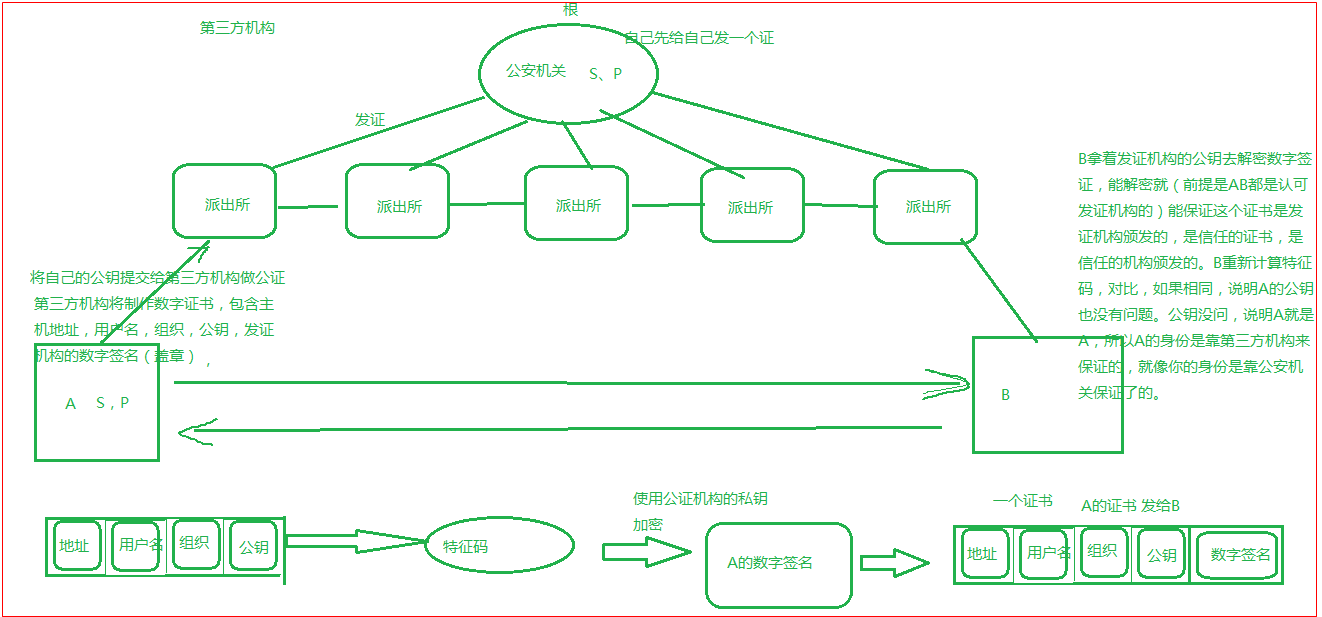
那么B如何得知A的公钥呢？A又如何得知B的公钥呢？



因此，这样是行不通的。因此出现了第三方公证机构。来证明通信双方确实是真实可靠的。就像我们的身份需要到公安机关登记发放一个身份证一样



就像公安机关，在全国各地都自己的派出机构，派出所。

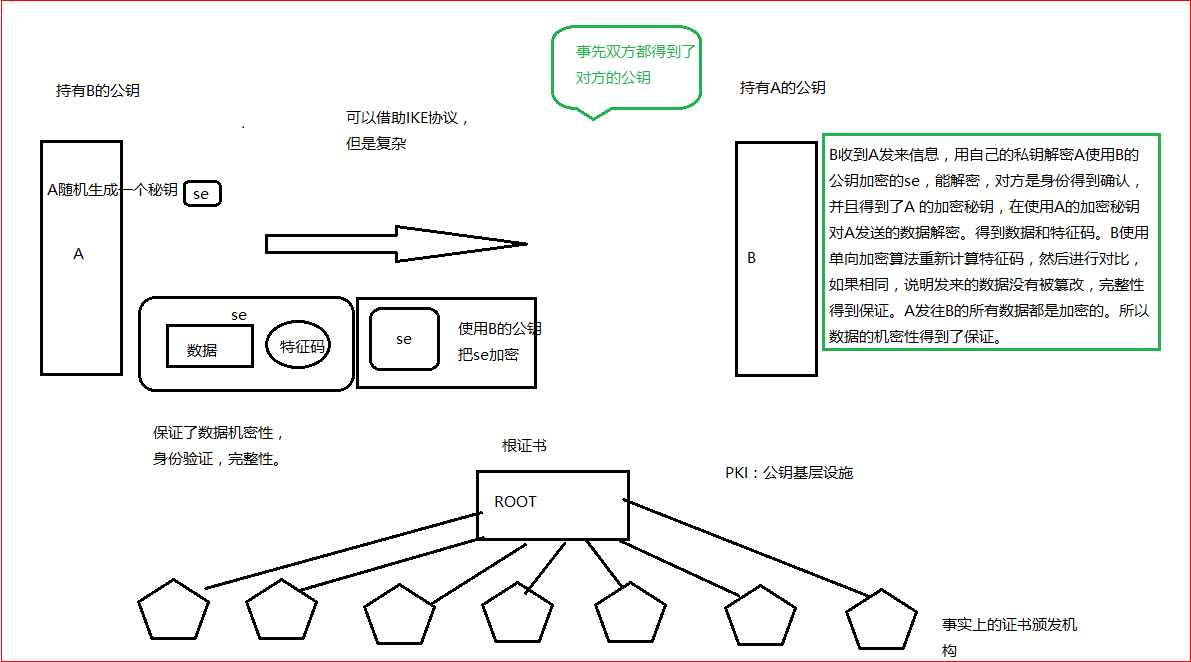


发证机关的公钥从何而来？怎么获得发证机关的证书呢？而且怎么得知发证机关的证书是可靠的？

还是自己到证书机构去拷贝一个回来，也可以送上门，需要的只是钱的问题。就类似于我们去公安机关办身份证一样。互联网身份认证不但收费，而且还不便宜。

现实情况通常是一方验证就可以了，比如：逛淘宝，京东一样，客户机验证服务器是不是淘宝，京东。所以只需要单方面验证身份就ok。

发起交易的时候需要双向验证？ 不需要，这时支付宝，银行网银，微信等等各种支付方式就出现了，客户机不需要参与交易，客户机只需得到交易的结果。淘宝，京东与第三方机构进行交易。而且交易的时候，用户与第三方机构进行验证式的交易确保了不可抵赖。



PKI：public key Infrastructrue

公钥加密两大核心功能就是**加密**和**签名**

身份认证（数字签名）

数据加密

秘钥交换

公钥加密算法有:

RSA 算法，也是一个公司的名称，也是3个创始人的名称。既能实现签名也能实现加密。加强版不公开。

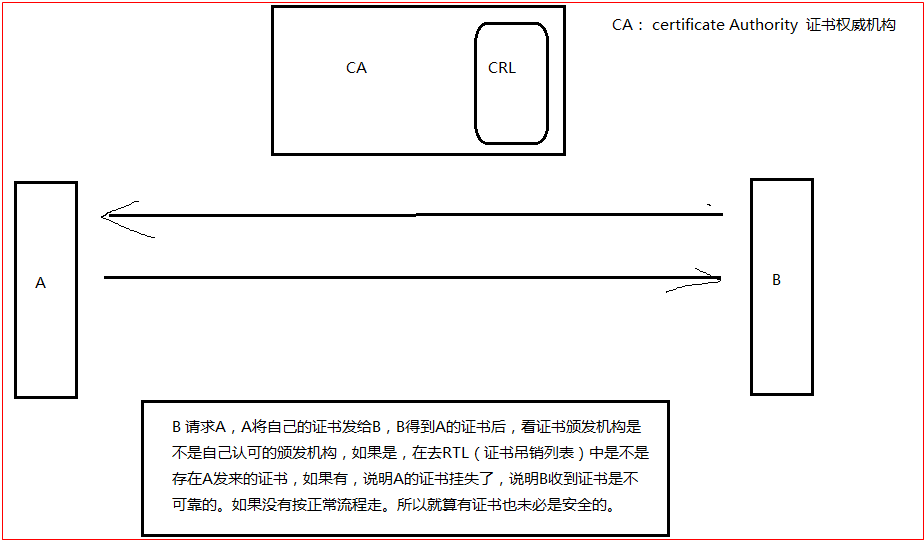
DSA 公开使用，只能实现数字签名，不能加密。

ELGamal 商业使用，需要付费。

## CA证书

CA: Certificate Authority 证书机构，主要使用来通信双方建立信任关系的。标识自己的身份的，作用就是互相传递公钥，建立信任关系。

CRL：证书吊销列表。



一个证书中包含的内容，不同标识的证书有不同的格式：

X509(标准格式) pkcs12

以X509为例

1. 公钥及有效期限
2. 证书的合法拥有者
3. 证书改如何被使用，比如拿来签名，加密等
4. CA自身的信息。
5. CA的签名（校验码）

PKI: TLS SSL

PKI: openGPG

SSL: Security Sokect Layer 安全的套接字层

TLS: Transport Layer Security 传输安全协议层

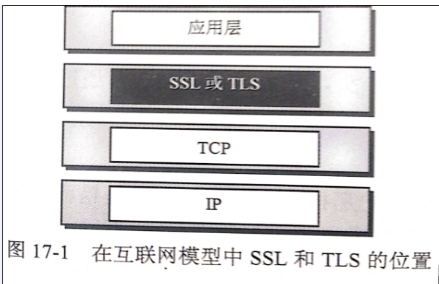
1. Libcrypto 加密库文件
2. libSSL SSL/TLS的实现,基于会话的，实现了身份认证，数据机密性和会话完整性的TLS/SSL库。
3. 多用途命令行工具

模拟实现私有证书颁发机构。

http://www.cnblogs.com/zhuqil/archive/2012/10/06/ssl\_detail.html

SSL 只是一个库，是应用层和传输层的中间层。是netscape网景公司开发的。

SSLv2 SSLv3

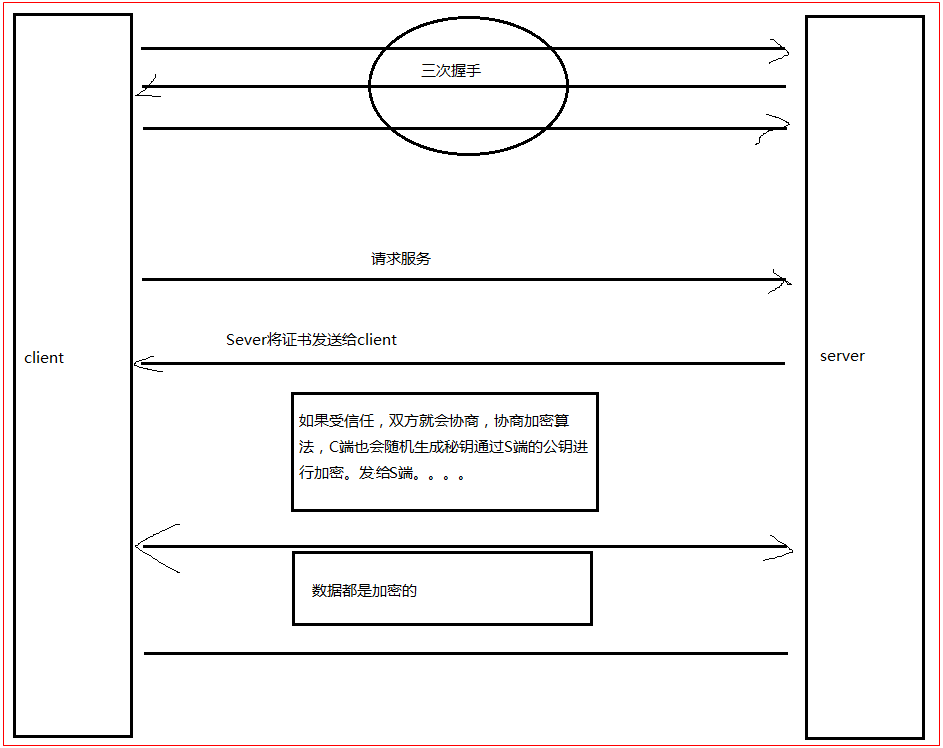


HTTPS： HTTP + SSL https走的端口：443

SSL毕竟是一个公司开发的。后来国际标准组织决定要开发一个更为通用，更为开放的安全协议。TLS就诞生了。

TLSv1 相当于 SSLv3 ,虽然协议不一样，但是功能是相同的。

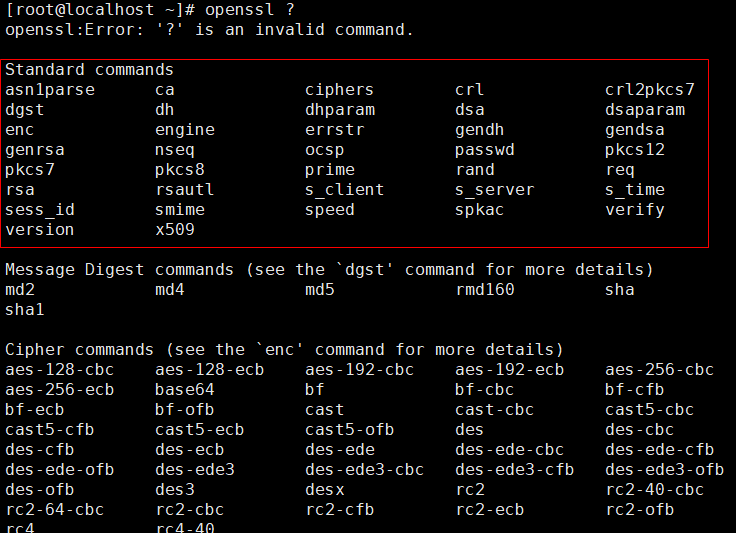
http是建立在TCP的基础上的。首先需要通过3次握手。



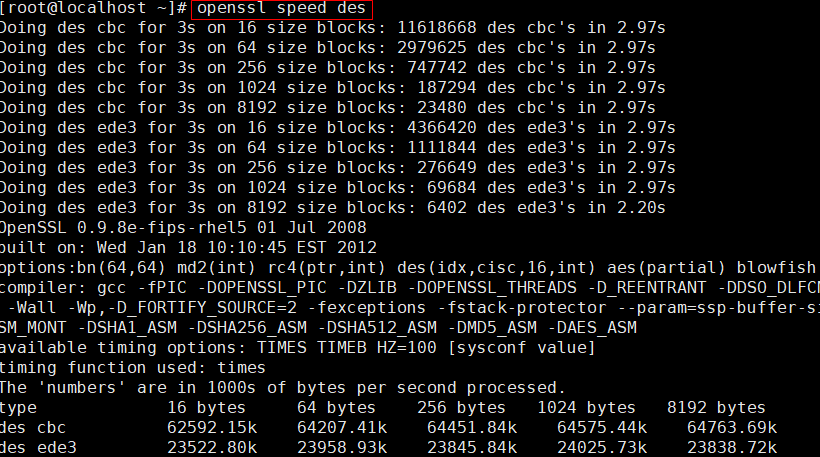
## openssl

openSSL 是SSL的开源实现，功能强大，实现了几乎所有加密算法。是一个软件。分三个部分

openssl ?



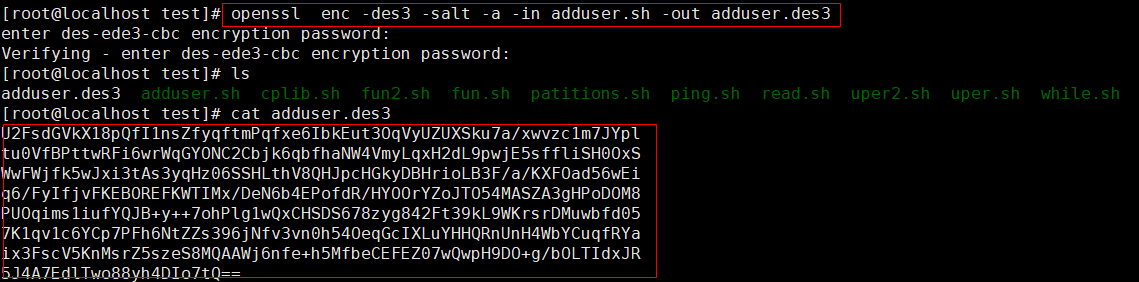
openssl speed 测试加密的速度，性能评估工具。



enc 加密

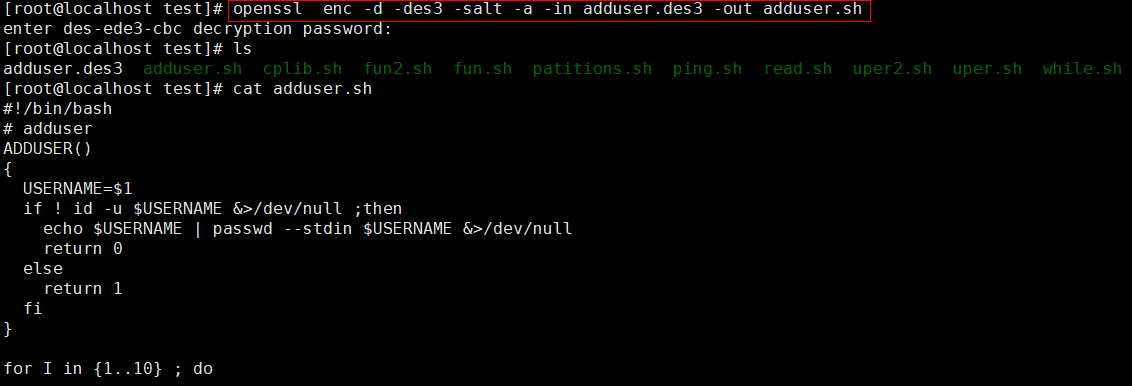
openssl enc -des3 -salt -a -in adduser.sh -out adduser.des3

-e 加密可以省略不写 -des3选定算法 –salt加点盐 –a 使用base64编码可以不加



openssl enc -d -des3 -salt -a -in adduser.des3 -out adduser.sh

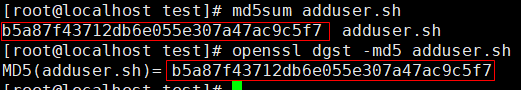
-d 解密



dgst

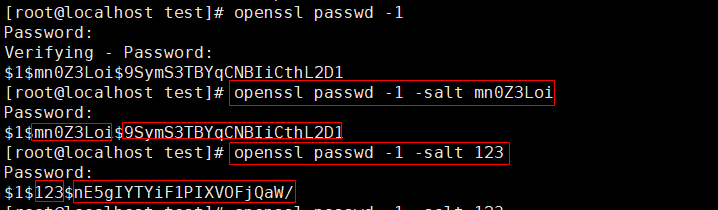
md5sum sha1sum 计算数据特征码

也可以使用 openssl dgst –md5 adduser.sh 计算数据特征码,不管使用哪种算法，计算后的特征码都相同。



openssl passwd -1 得到加密密码

openssl passwd -1 –salt xxxxx 加点盐，加点杂质。避免看到密码串一样反推密码。



设定密码都是 123456

[root@localhost test]# openssl passwd -1

Password:

Verifying - Password:

$1$mn0Z3Loi$9SymS3TBYqCNBIiCthL2D1

[root@localhost test]# openssl passwd -1 -salt mn0Z3Loi

Password:

$1$mn0Z3Loi$9SymS3TBYqCNBIiCthL2D1

[root@localhost test]# openssl passwd -1 -salt 123

Password:

$1$123$nE5gIYTYiF1PIXVOFjQaW/

密码都是123456，加入不同的盐以后，结果就发生了巨大改变。

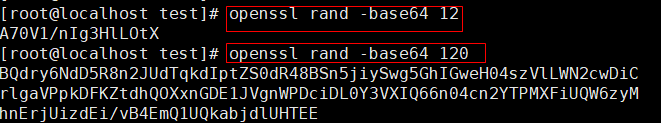
openssl 实现签名和非对称加密

dsa 只能签名，数字验证。

rsa 秘钥处理工具

rsautl rsa加密解密处理工具

rand 生成伪随机数。需要密码，就可以使用这个功能。

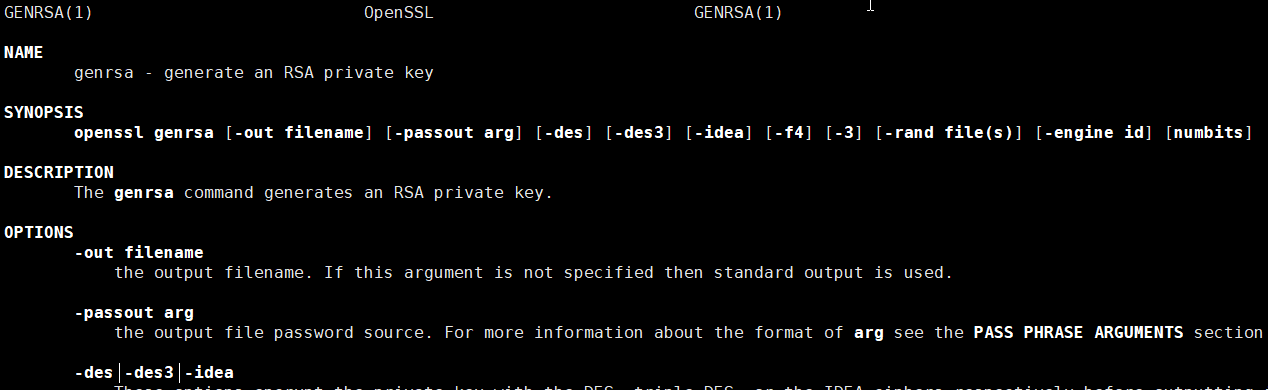


## openssl实现私有CA

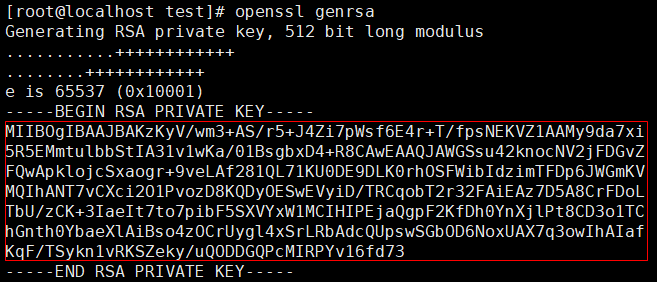
1. 生成一对秘钥
2. 生成自签证书

### 生成私钥并输出

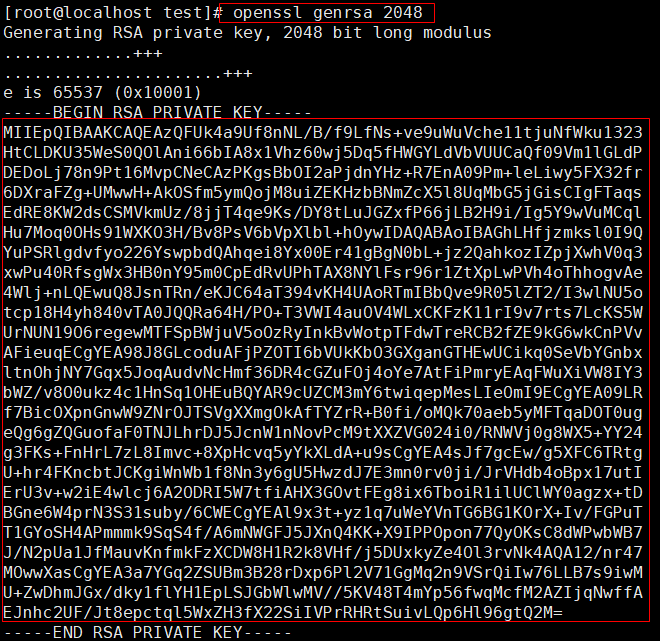
openssl genrsa



默认是512bit

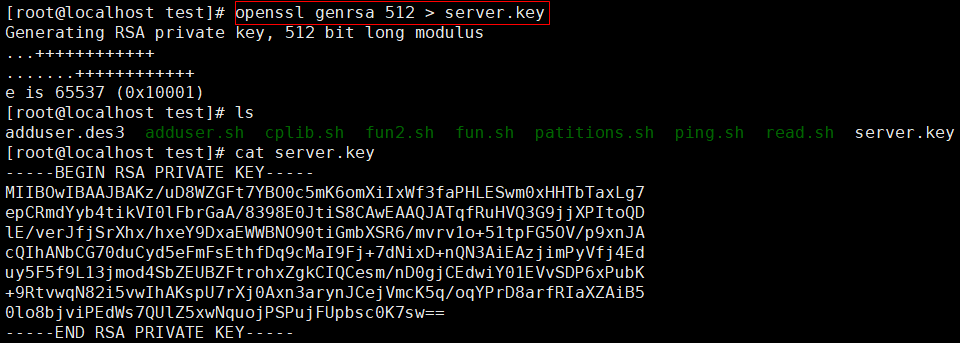


当然也可指定私钥长度



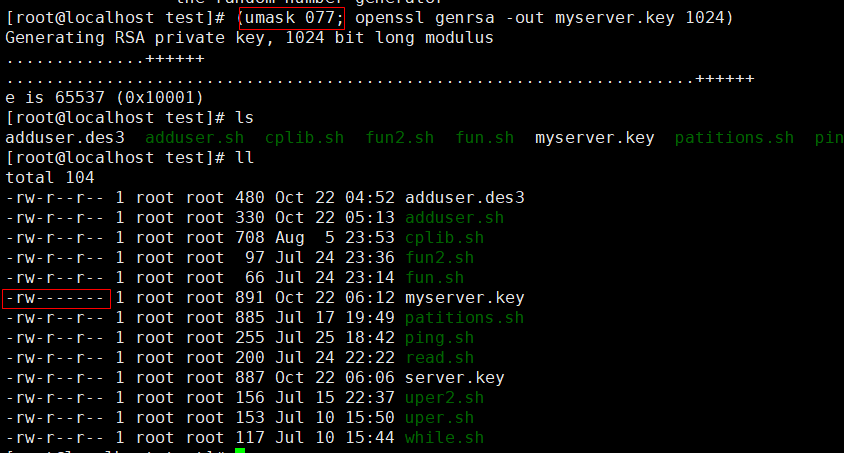
openssl genrsa 512 > server.key 重定向到server.key文件，也可也是-out指定文件

openssl genrsa –out server.key 512



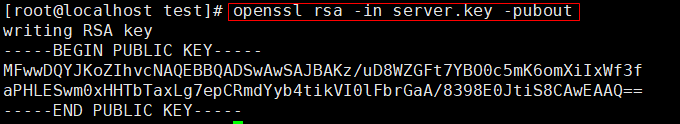
(umask 077; openssl genrsa -out myserver.key 1024)，（）表示在子shell中设定，执行完成就退出。

openssl genrsa –out myserver.key 1024



### 从私钥总提取公钥并输出

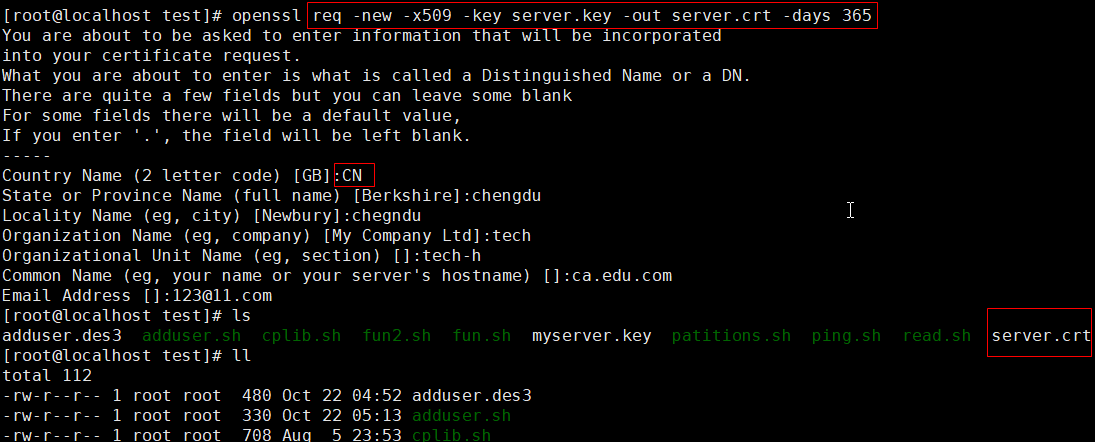
openssl rsa -in server.key -pubout



### 根证书制作

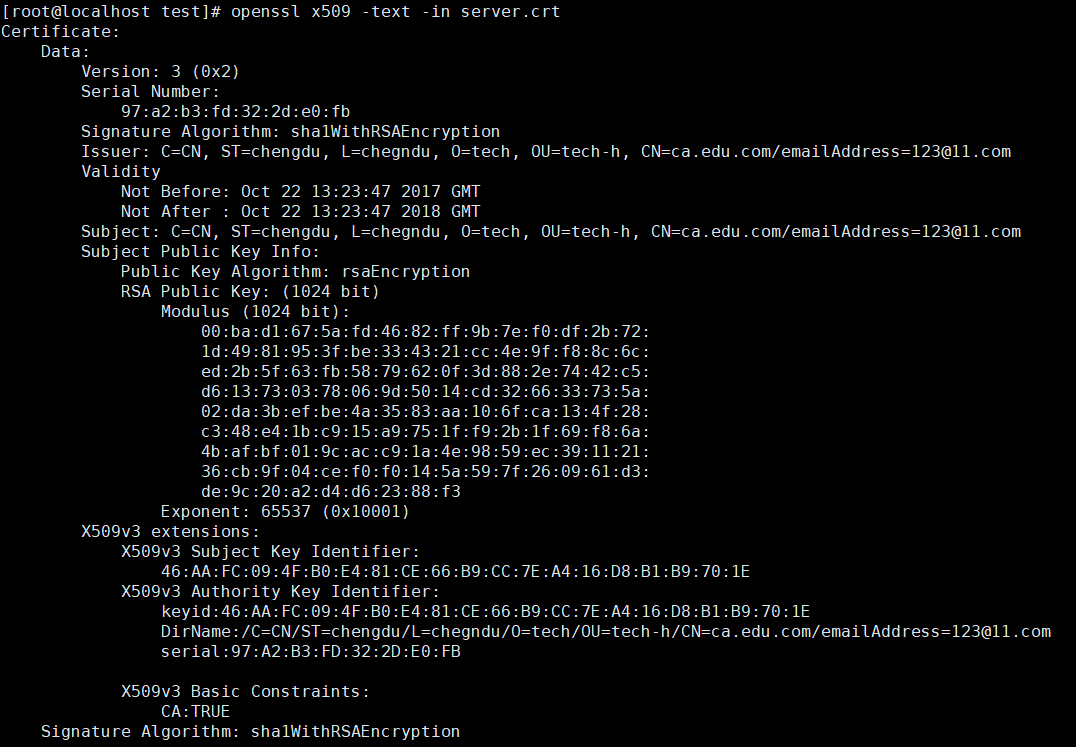
req

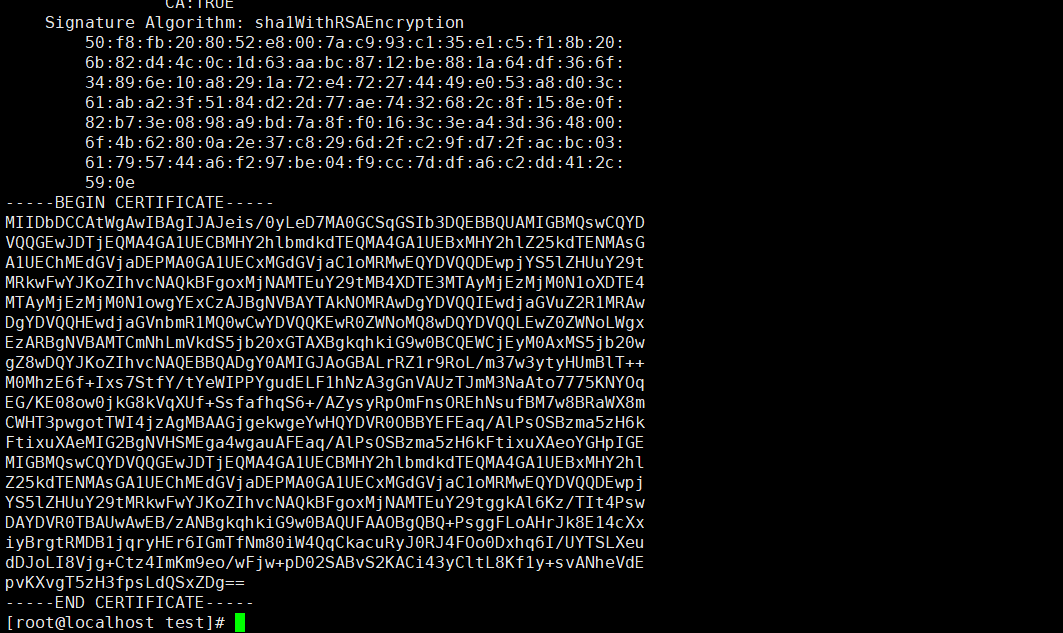
openssl req -new -x509 -key server.key -out server.crt -days 365





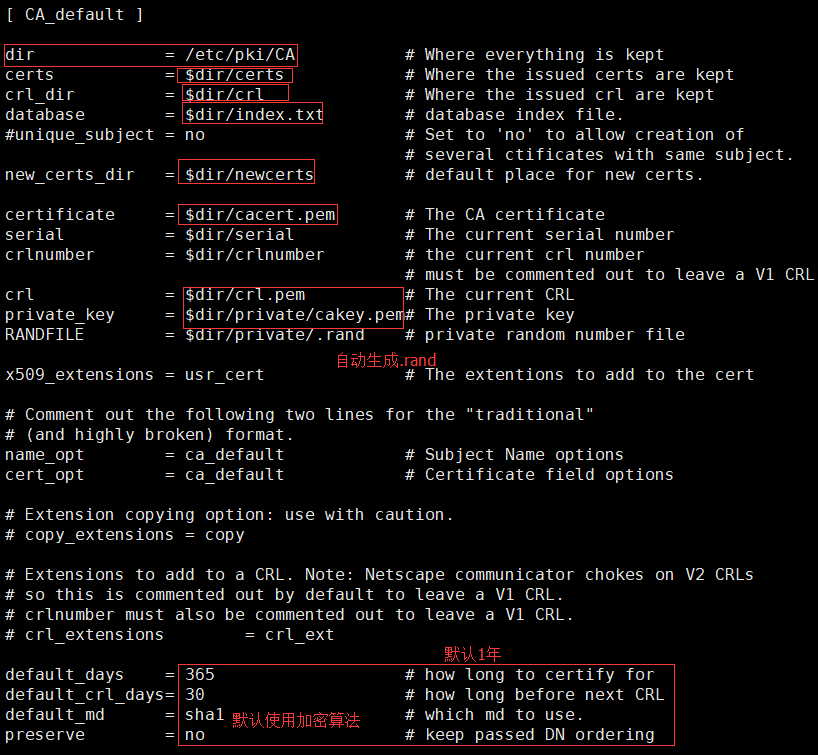
openssl x509 -text -in server.crt 输出证书的信息





### /etc/pki/tls/openssl.cnf

在linux上，CA证书和秘钥不能随便放，在配置文件 /etc/pki/tls/openssl.cnf有规定



配置好，签名信息就可以生成证书的时候就不用手动输入了

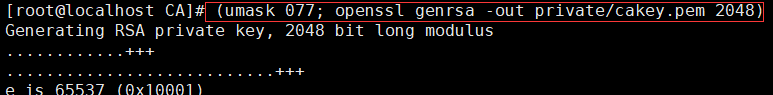


## CA

### 生成私钥

保存在private目录下

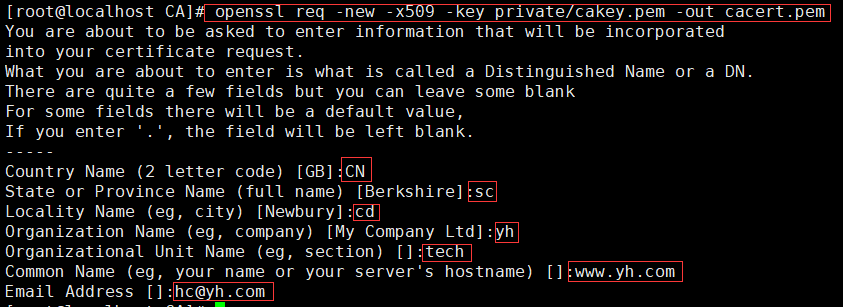
（umask 077; open）



### 从私钥中提取公钥并生成证书

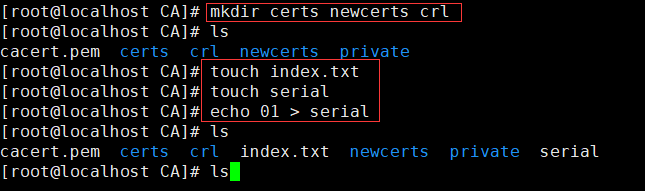
openssl req –new –x509 –key private/cakey.pem –out cacert.pem –days 3655

-x509是ca服务自签证书用的。



CA目录下新建文件夹： certs crl newcerts

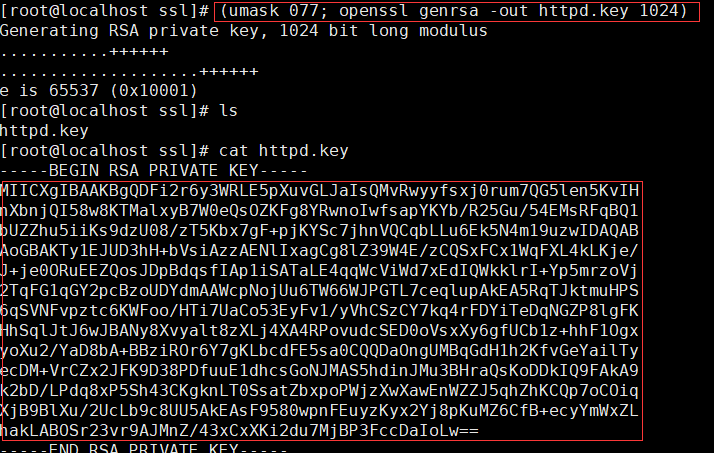
CA目录下新建文件： index.txt serial



然后就可以给其他用户签证了,其他用户先创建一个申请。让CA服务进行盖章签名，就是用CA的私钥对提交的信息（计算过后的特征码）加密并放在证书后面，签章后就可以使用了。

### 为httpd应用证书签章

#### 1.生成httpd自己的私钥

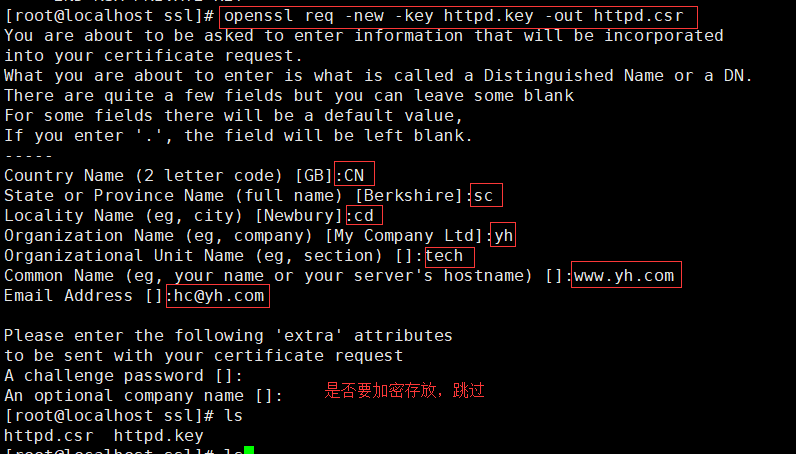


#### 2.填写申请

就像办理身份证一样，姓名，性别，住址，年龄，头像 等需要自己填写，填写好了以后，公安机关给你盖个戳。

**openssl req -new -key httpd.key -out httpd.csr**

httpd.csr (csr 证书签署请求)



### CA给httpd应用申请的证书签署

CA会提取申请的特征码用自己的私钥加密，放到申请里面。

openssl ca -in httpd.csr -out httpd.crt -days 365

