

Openssl 是一个开源程序套件 ----- 两个年轻人 为开源组织提供了对称加密 非对称加密 数个单向加密的算法的软件 功能非常强大 但是 体积非常小

这两个人工作在RSA

这两个人又为RSA提供了一套商业套件 更加强大



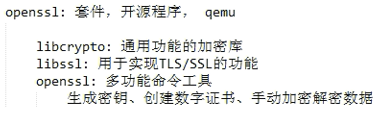
Openssl套件 有三部分组成

libcrypto ----具有通用功能的加密库 ----- 总多软件想实现加密功能 直接调用这个库就可以

libssl---- 这个实现ssl机制的 ----- 用于实现TLS/SSL功能的组件

就叫openssl --- 多功能命令行工具 ---- 能够实现加密 解密 创见证书 吊销证书 等等

生成密钥、创建数字证书、手动加密解密数据



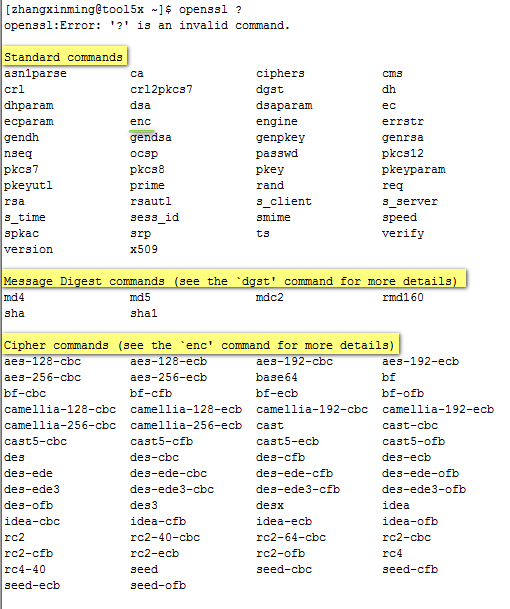
\*\*梳理一下加密解密技术中常用的功能及其算法

（1）. 对称加密 – 使用相同的密钥进行加密和解密的 ---- 加密是对数据块进行加密 很有可能前后加密的块是互相有联系的

算法：DES 3DES AES Blowfish Twofish RC6 CASTS

实现算法的工具(linux) ---- gpg和openssl enc ----指的是多功能命令行加密(enc)工具 有众多的子命令

看一下



其中 在**standard commands中 有一个子命令叫做enc** 就是实现**加密解密**的openssl的子命令 encryption



下面一组是用来实现信息摘要的命令



这里面有md2 md4 md5 sha sha1

第三个给出来的是加密的命令：

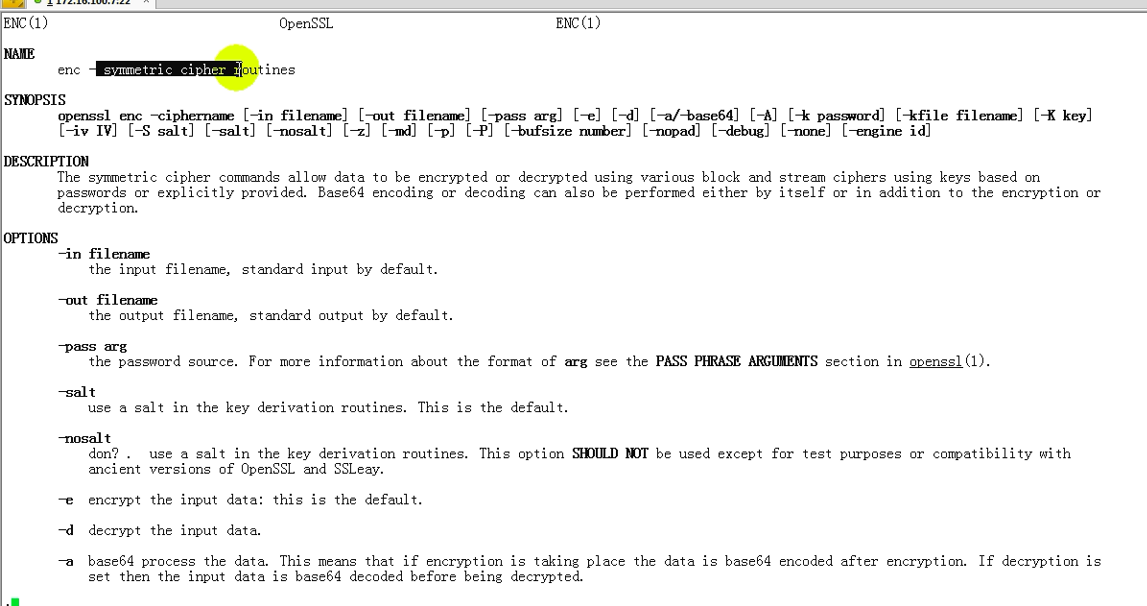


这个是**具体的加密算法**

Bf ----blow fish

Openssl enc使用

**直接man enc就可以了**

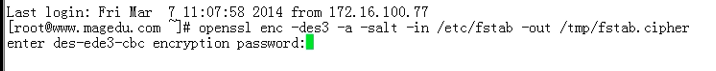


**Openssl enc是对称加密**  对称加密历程



**使得加密的过程更加安全**

表示输出的格式保存为Base64的字符串输出 --- 否则是二进制输出



这个里面让你输入密码



这就是常用加密工具

加密解密的格式 –d是解密 decrypt



**【-d表示 decrypt】**

gpg也可以实现加密解密 暂时不介绍

----- 以上是对称加密

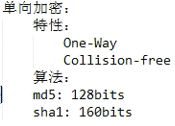
**下面是单向加密 或者hash算法**

Md5是定长128bits

sha1是定长160bits

sha512是512bits

单向加密 也成为one-way



其他算法还有很多

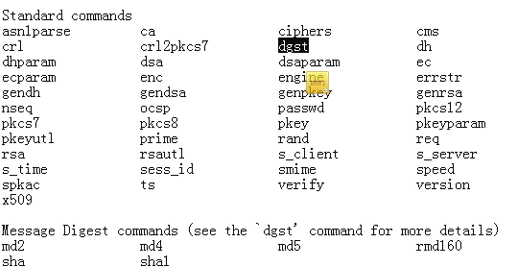
常用的工具 [比对称加密多 4种方式]

sha1sum

mdsum

cksum

openssl dgst



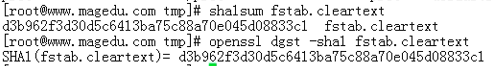
Enc是对称加密**子**命令 dgst是单向加密 也叫信息摘要子命令



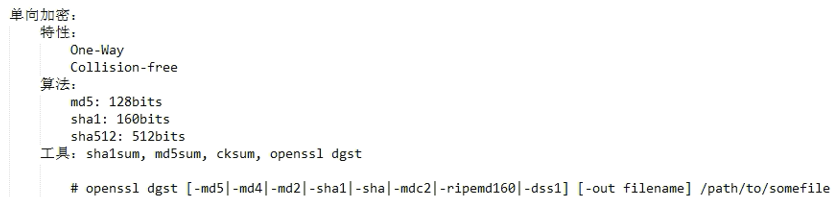
信息认证的

简单用法：





没有指定输出文件 就直接显示到控制台（因为不是定长么）



**信息摘要码--- MAC ---- message authentication codes ----单向加密的延伸应用**

应用：用于是现在网络通信中保证数据传输的数据完整性

算法：MAC的应用场景

客户端计算出信息特征码 ----- 并使用MAC进行加密 ----- 加密之后传递到服务端 ----- 所以是单项加密的延伸 或者扩展应用

MAC这种机制中 有两种：CBC-MAC 还有一种是HAMC

HMAC使用广泛 因为是使用md5和sha1算法

CBC-MAC不是特别安全

集群的时候 会用到HMAC

比如好多节点 加入一个节点 如何验证这个节点是我们要加入的呢？使用的就是HMAC

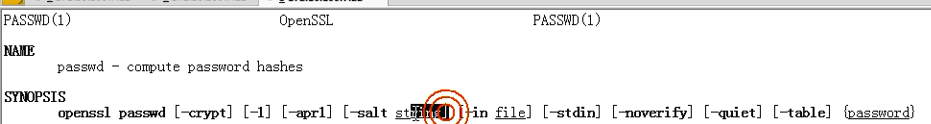
HMAC可以实现消息认证【认证就是私钥加密 公钥解密】

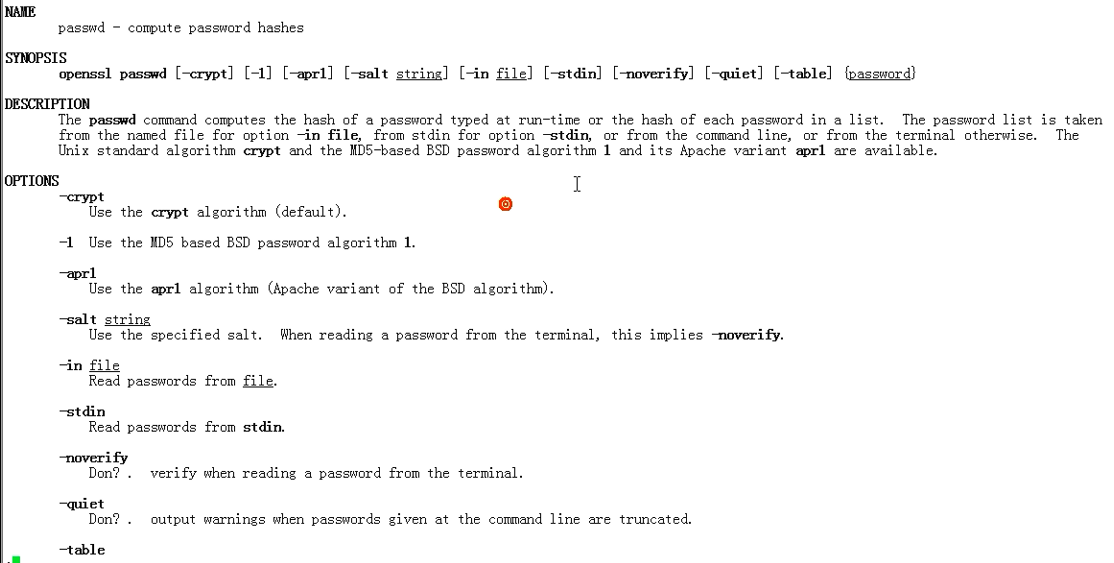
用户认证：

工具：passwd, openssl passwd这两个工具

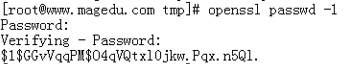
man



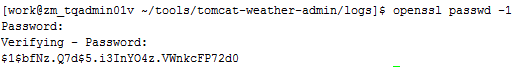




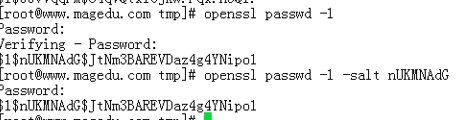
**这个openssl的passwd子命令就是计算计算密码的hash值**



-1不是-L



可以手动指定salt



这样 salt就指定进去了

这就是用户认证

======= **非对称加密** =============

公钥加密：

**密钥是成对的 pk public key 私钥sk secret key**

非对称加密

做数据传输的时候 数据本身是对称加密 对对称加密的秘钥是公钥加密 私钥解密 ---- 做电子签名的时候 是私钥加密 公钥解密

**公钥可以传给别人**

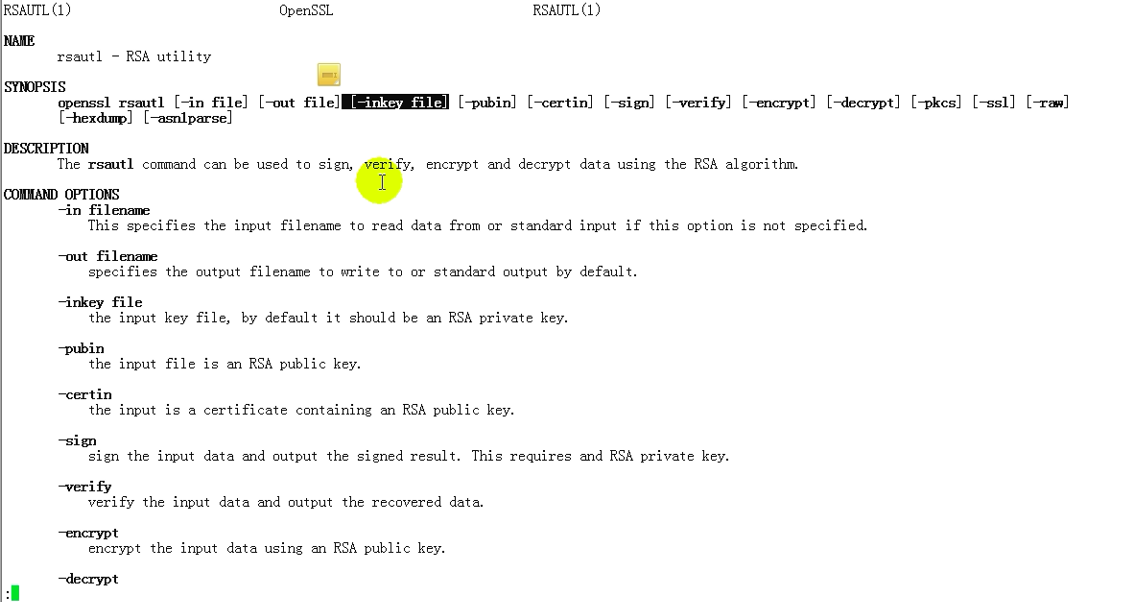
常用算法：RSA EIGamal

RSA是最为广泛的加密算法

RSA一般不用加密数据 因为非对称加密的速度比对称加密的数据慢三个数量级

所以 公钥加密用来实现密钥交换（交换的密钥是对称加密的密钥）

工具是 gpg, **openssl rsautl 【前面说过了openssl enc是对称加密 open rsautl就是rsa非对称加密的工具】**



**数字签名：使用的是私钥加密 公钥解密 --- 加密的原数据的特征码 ---- 电子签名 或者数字签名**

算法：RSA, EIGamal, DSA（DSA只能用来做签名 但是无法用来进行加密 Digital Signature Algorithm ---也称为DSS Digital Signature Standard ---- 因为只能做电子签名 所以 只能是私钥加密 公钥解密 不能反过来）

**密钥交换：IKE 互联网密钥交换**

算法： DH(diffle hellman) ---- 双方进行协商的 和上面提及的公钥加密

DH算法【略过36:57->43:24】

RSA通常可以用来做密钥交换

[之前说的 公钥加密的第一个应用 就是加密对称加密的秘钥数据]

**数字证书：公钥加密的另外的一个重要应用：就是做数字证书** ----- 就是第三方的机构 把申请者的公钥分发出去的一种机制

第三方机构就是保证公钥在互联网传输是安全的

有多个证书机构 有规范：

*规范：x509 属于x500这个标准 用于定义数字证书格式*

数字证书还有其他格式PKCS12等等

对于x509中 证书内部包含

**公钥和有效期限**

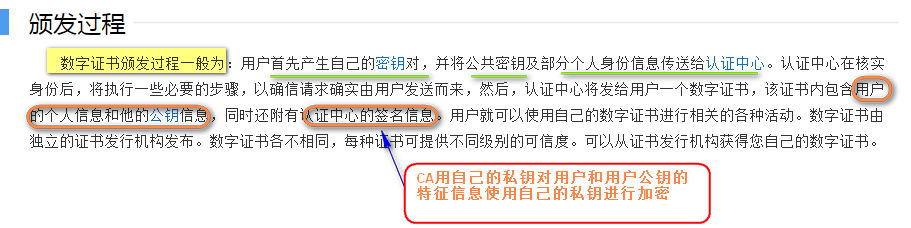
**持有者的个人合法身份信息** ： 有可能是公司或者个人 所以有组织证书和个人证书等等

---- 如果是服务器 **一定是服务器的主机名 或者被访问的域名**

证书的使用方式： 比如你的证书能拿来做什么？比如加密解密传输 主机之间的通信

CA的信息 也就是证书颁发机构的信息

CA的数字签名：用于验证这个CA是否合法 ----- 前四项信息的特征码 用CA自己的私钥加密之后



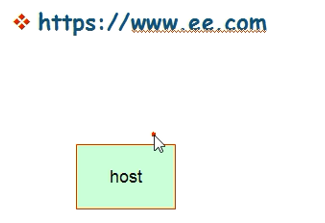
其他人要用CA的公钥对CA的信息进行验证 ----- 谁给CA发证？**自签证书**

因此 未来想做私有CA 要做私有签名 给自己先发一个证

不但能够发证书 还能吊销证书 --- **PKI 公钥基础设施**

**假如开办了一家公司 这家公司是用来做电子商务的 ----- 期望使用https来进行访问 ----- https实际上是ssl的应用 ---- 这就要求使用证书**

确保客户端访问这台服务器就是这个服务器

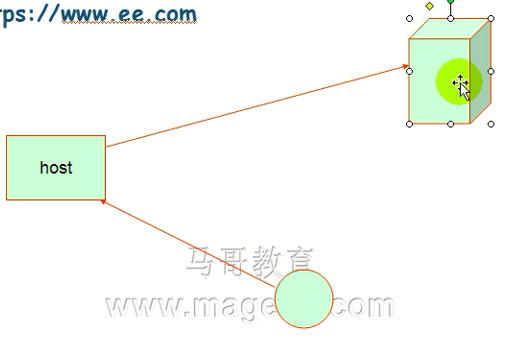
 期望用户输入的是<https://www.ee.com>

---- 这个[www.ee.com](http://www.ee.com)就是证书中“持有者的个人合法身份信息”

注意：假如 你知道www.ee.com 中有一台访问的主机的主机ip是 172.16.100.7 那么

如果你访问的是 <https://172.16.100.7> 那么 由于证书中没有这个IP的信息 你这样的使用ip进行访问 就会认证失败 【有了https 那么 ip直接访问是没有用的】

客户端访问主机 输入的**主机名** **一定和证书中申请时候的合法信息保持一致** 才能认证通过 否则不能通过



圆圈代表客户端 访问https xxxx 这样之后 服务器应该自己生成自己的私钥 公钥 让后到CA中申请证书 ----- **所以 为了能够发证**

**\*\*\*\*\*发证的流程\*\*\*\*\*\*\*\*\***

\*host本地生成一对密钥

\*把**公钥**及其**所需要的用户信息**按照**指定格式制作成证书申请**（说实在的 就是填表） ---- 申请 ---- 通常叫**证书签署请求**

CA做法

\*嘱咐你的私钥不要丢失

\*派人到你的公司检查你的营业执照 检查各信息 检查域名 服务器 IP等等 各种信息

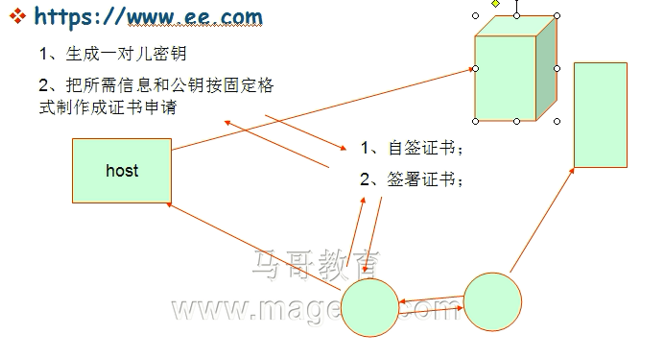
CA在发证之前 要有自己的证书 来确保CA的信息合法 ---- 所以 CA首先要有自己的证书

\*自签证书：把自己的身份合法化 把戳 章 印准备好

\*签署证书：其他的host发起了证书签署请求 ----- CA就是把证书签署就可以了---- 也就是盖个戳 ---- 生成数字签名即可

然后传输给申请人

\*维持一个吊销列表



【Security Socket Layer 加密套接字协议层】

对于openssl 功能就非常强大：

(1). 如果你是客户端 ---- openssl可以做到生成密钥对 + 构建证书申请

(2). 如果你是CA --- openssl可以做自签证书+为别人的申请签署证书+维持吊销列表

一身多能的openssl

需要注意 把openssl作为CA 测试可以 ----- 但是全球大范围使用openssl功能太单一 可以作为公司内部

这时候需要使用OpenCA ---- 同样是基于openssl 只不过是 openCA把openssl的前端做的更加易用 – 功能更加强大

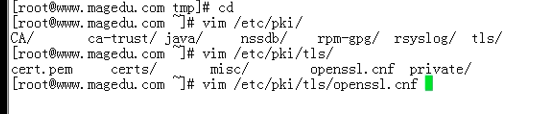
**------ 下面使用openssl来实现上面的内容**

\*\*想要在**公司内部**实现基于证书的通信 就要自建一个CA ---- 自建CA ----- 

**\*\*\*\* CA端：先生成密钥对 -> 然后生成自签证书**

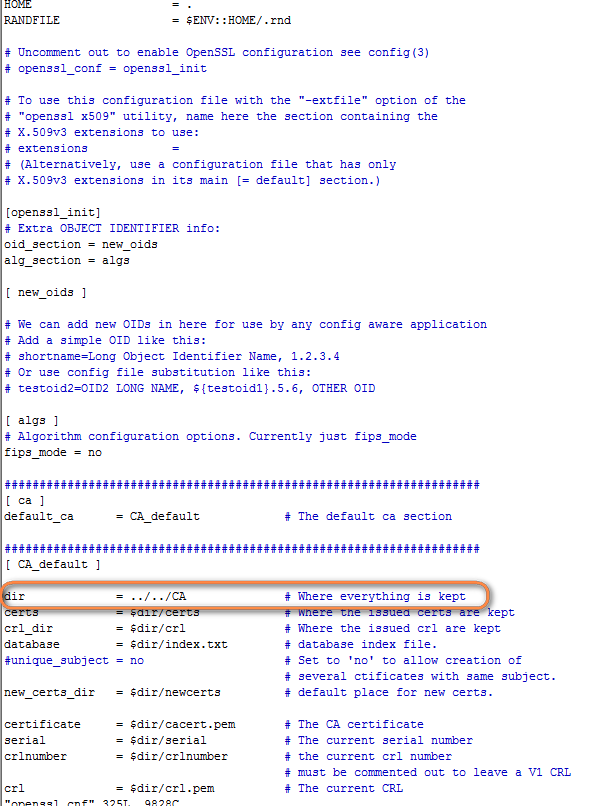
\*\*\*\* host端：生成密钥对 -> 发起证书申请（通常说的用户端）

**Openssl扮演*私有CA*或者自建CA时候的默认配置文件 ---- /etc/pki/tls/openssl.cnf**



Openssl的配置文件的位置是 **/etc/pki/tls/openssl.cnf**

这个文件大部分信息都不需要修改



**修改成 dir = /etc/pki/CA [这个是一个根目录]**



这部分和CA相关

Crl ---- certificate revoke list 认证吊销列表

***你有多少个证书 ---- 用于进行证书查询的 这个有一个索引***



***用来存放刚刚签署的证书***



***这个是CA自身的证书***

因为自身的证书 要使用自己的私钥和公钥

***发出去的证书的编号***



***已经吊销的证书的编号***



***当前正在使用的吊销列表文件***

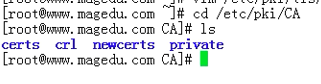


***这个是CA自己的私钥 ---安全至关重要 通常权限是600 或者400***  


***签署证书需要随机数***

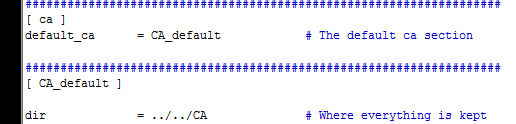
 

现在切换到

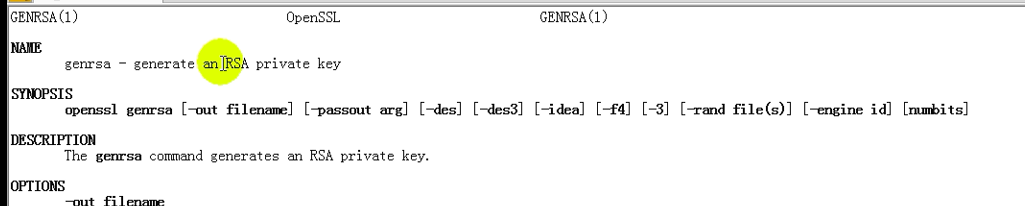


现在如何实现自建CA

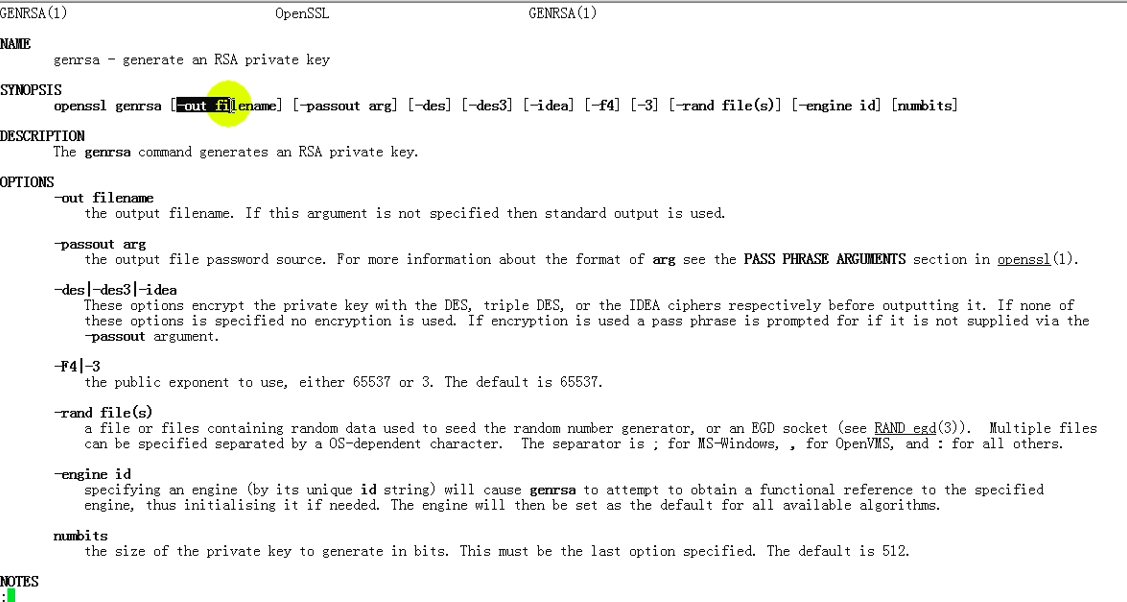
在Centos5上面 要多一步 这个使用的是相对路径

 ---- 要把这个修改为绝对路径 ---一定修改！！ **---- 这个dir是一个公共的路径**

如何生成私钥？



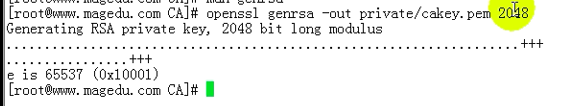
**如何生成公钥？-----二者是成对出现的 公钥是从私钥中按照某种格式提取出来的 私钥非常长 ----- 所以 没有私钥长**

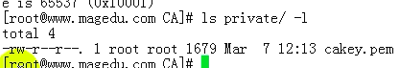


**Numbits是使用多长的密钥**

私钥放在主机上面 可能被看到 所以 私钥文件先使用对称加密进行加密 --- 所以 使用的时候 要使用密码解密

我们演示不进行加密



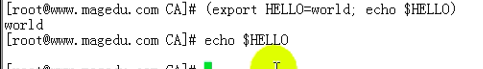


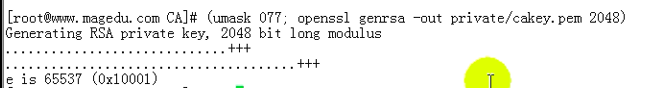
**访问权限 任何人都可以读 修改权限** ---- 另一种做法 生成时候的 给出权限

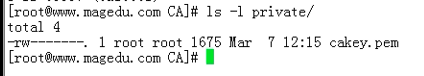
在()中执行命令 是在子shell中执行的 ---- 但是 子shell中的声明的变量在父shell中是没有效的

环境变量仅仅对当前shell和他的子shell有效 对父shell无效

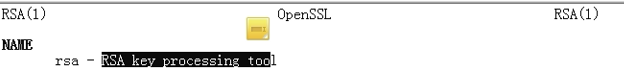


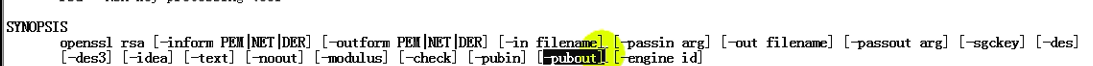




**这个生成就是600**

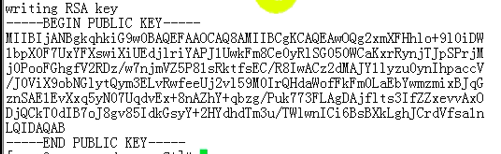
**私钥文件创建好之后 哪里获取公钥？ ---- 前面使用的子命令是genrsa 下面获取公钥的子命令是rsa子命令**



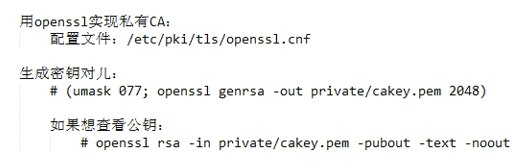


**这个子命令 可以从-in中指定的私钥的文件 以-pubout (public key out) 的字符串形式 提取出来**

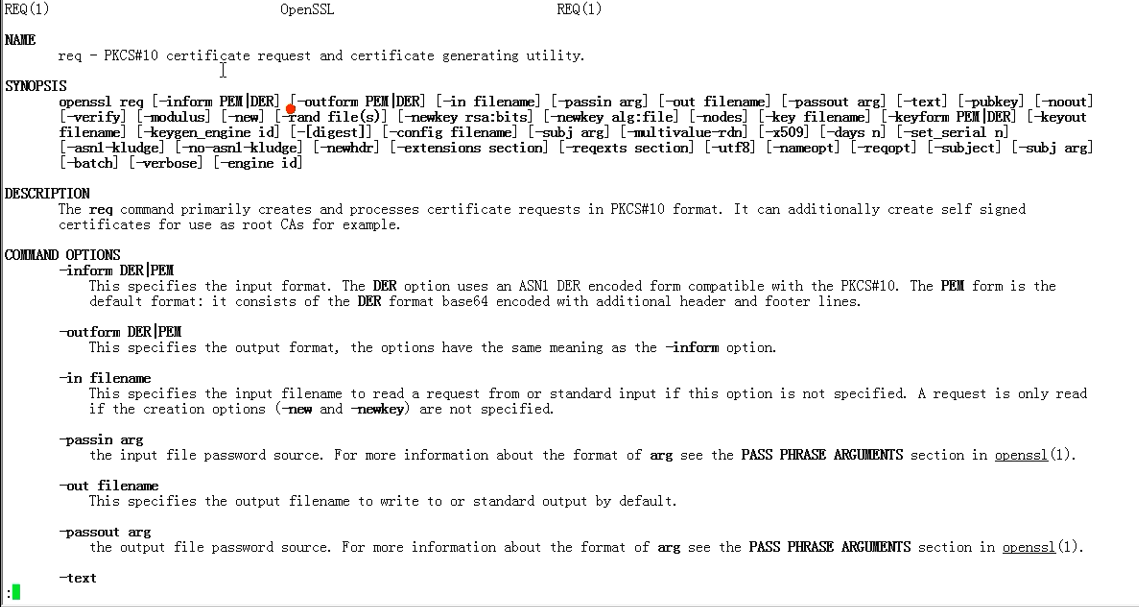




如果不想输出 就使用-noout

 ---- 这个公钥仅仅是查看

下面就是生成字签证书 将用到openssl另外一个子命令  ---- 就是自己请求自己签



-in 读进来的数据文件 –key指定私钥数据文件 –days是有效期限 –x509是字签证书的格式

req后面的是证书申请的格式

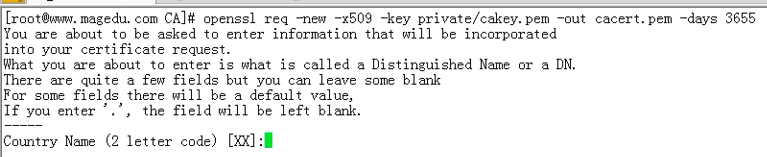
 **--- 这个req表示证书申请 和 证书生成的工具命令**

-out 申请请求放在哪里

-new 表示新发起的申请



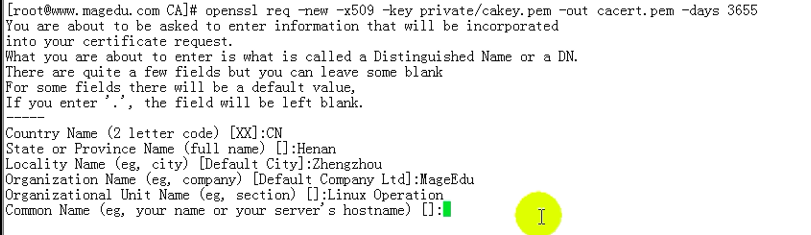




要想生成申请 必须要写明信息

默认是XX

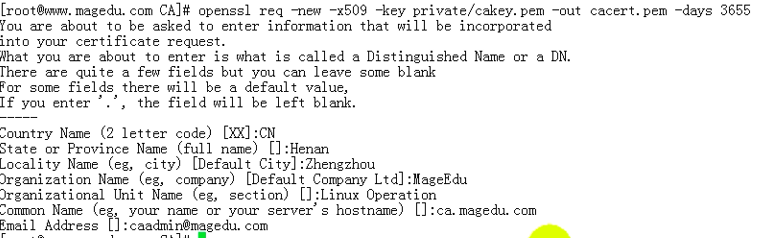
 可以配置

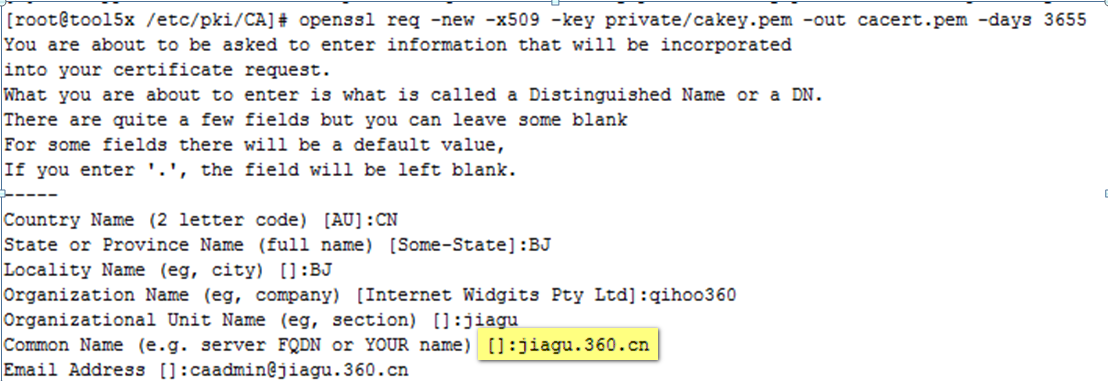


**下面的your name或者your server’s hostname这个非常重要**

**打算发给谁 这里面 是发给自己CA** 所以





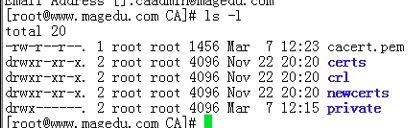
【 jiagu.360.cn】



**这样证书生成了**

这个证书是可以被访问的 --- 不建议这么做

**CA的证书 私钥都有了**

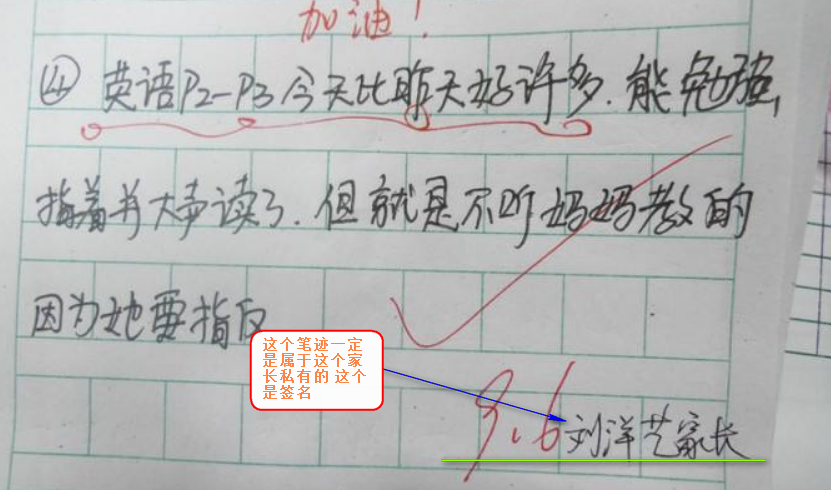


**【cakey.pem --- 实际上是CA private key cacert.pem 实际上是CA certificate】**

***还差两个文件index.txt和serial crlnumber***

【对数字签名和数字证书的理解

 这个是证书的样例

 这个是签名的样例

总结一下：

**私钥物理意义的特点：**私钥一定是不能发出去的 一定是自己留着的 ----- 所以 私钥和自己的关系最紧密的 ----- 所以 私钥能代表一个人或者主机的某种信息 是别的机器不具备的

这里面 实际生活中 无论是私人的签名 还是 官方的印章 都是分别属于个人和大学官方私有的 ---- 就相当于计算机中的私钥 ----- **实际上印章是特殊的签名**

对于国家或者国际承认的签名 比如北京大学是国家承认的大学 那么北京大学就是CA ---- 北京大学的章印到某个人的信息上 就变成了毕业证书 ==== 所以 某个人的某些信息被官方承认的章盖章了 ---- 就变成了证书

对于不是官方承认的 也就是个人 在某些内容上面的签名 就属于签名

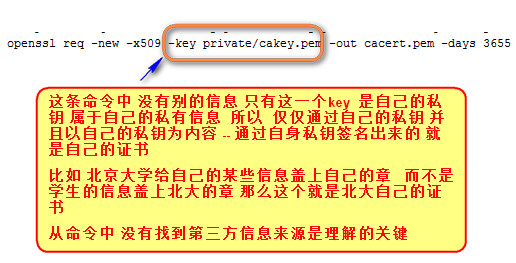
在计算机中 上面的章和个人姓名笔迹就相当于属于主机本身的内容 私有的 就是私钥 ---- 私钥不能像实际生活中 把文字写到纸上 或者章印到纸上 ---- 所以 采用私钥的特点 本身可以对数据进行加密 把自己私有的信息和原有的信息进行混合

---- 如果是CA的私钥加密 –加密出来的就是**数字证书**

---- 如果是某个主机普通的私钥加密 --- 那就是这个主机在某种信息上面的**数字签名**

**所以 无论是电子签名 还是 电子证书 --- 在形成的时候 从原始的信息 被 某个私钥加密的过程 就成为 签名的过程 ---- 这个*签名是动作***】

Serial里面记录发证的初始编号 【 所以 电子证书一定有编号 但是个人签名是没有编号 因为不被官方认可 所以 个人签名编号是没有意义的】

【所以 上面在CA根据自己的信息 这样看到到北大自己的证书 就知道这个是北大 或者 属于北大的某种信息 这样别人看到信服】



**这样准备工作基本完成**

**======== 以上就把CA的工作准备好 等待**

----- 现在 找一个客户端申请证书 --- **这个是第三方**



这个作为CA的申请者 这个是一个web ---- web域名是[www.mageedu.com](http://www.mageedu.com)

私钥放在哪？

**如果web服务器是Apache 就放在apche的配置文件中** --- 否**则放在nginx配置文件中**

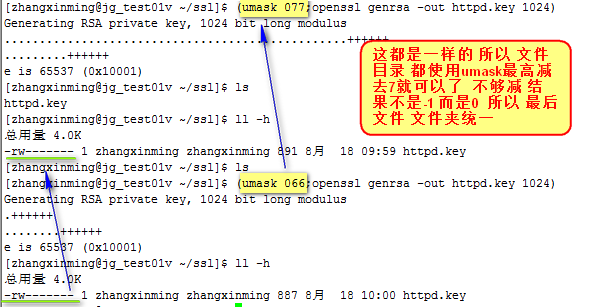
由此 找一个特别的路径

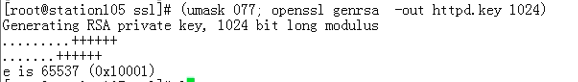
对于apache 是在httpd/conf目录



这个下面有一个ssl目录

进去

生成一个密钥对【文件满额是 666 目录满额是777 ---- 077 生成的是文件 拿走之后 666-077 就剩本组读写了 这里面 没有不够减 这里面 umask 066 对于生成的文件权限也是一样的 因为 使用umask的原因是 我们不想生成文件之后 再使用chmod修改文件权限 】





这里面要想能让CA给发证 ----- 要生成证书签署**请求**



【csr ---- certificate signature request –r表示request】

【openssl req -new -x509 -key private/cakey.pem -out cacert.pem -days 3655 这个是CA**给自己的请求**

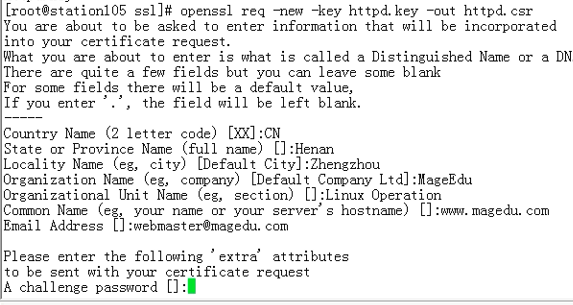
---- **自己已经就是官方认证了** 所以 自己给自己盖章的请求就是自己的证书

上面的在申请短执行的

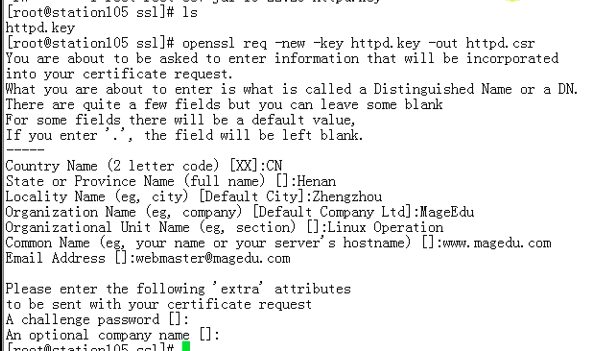
Openssl req –new –key httpd.key –out httpd.csr 这个是非CA机器利用自己的秘钥生成的签名 ---- 自签名 但是 因为不是CA 官方不承认 所以 没有格式 所以命令中没有-x509的字样 ---- 所以 这个生成的httpd.csr就是一个有着申请主机自己信息的数字签名 就是签署请求】

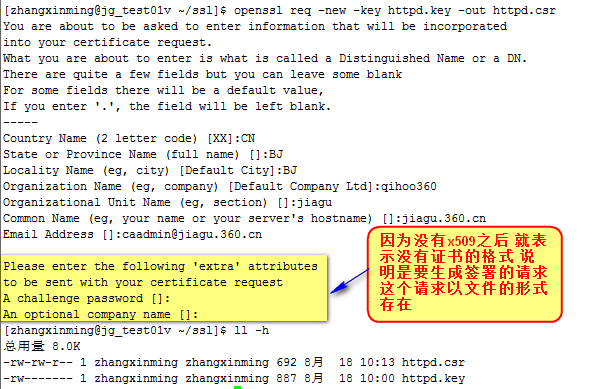
生成一个基于httpd.key的 csr --- certificate signature request 证书签署请求的缩写

这个是证书签署请求的文件

 ----- 【因为理解起来 自签证书和签署请求的实质都是一样的 所以 生成数据的过程都是一样的 ---- 因为公钥是从私钥中提取出的 并且 私钥是这个机器自身信息的象征 ---- 所以 这个里面一定有这个主机的信息 和 主机的公钥 ---- 但是 主机的私钥一定是没有包含在里面的】

**这个请求不要被别人拿去** --- 所以 这个请求 加密存放最靠谱 所以 需要一个A challenge password 需要一个密码来进行加密



最后连续敲两下回车 就表示使用空密码【演示使用的是空密码 都是为了安全起见 不让别人看到这个签署请求 因为不是生成证书 所以 会给出额外的加密信息

注意看 提示写的是 You are about to be asked to enter information that will be incorporated into your **certificate request.**

说的下面的需要填写的这些个人的信息 就会被整合到证书申请中

也可以看到 没有使用umask之后 这个默认的权限就是664 前面直接使用umask 077的结果 就是600的权限

**Umask ---- mask是面具 掩盖的意思 ---- 这样 就是掩盖掉对应位置上面的值 所以 666 掩盖掉对应位置的066之后 就剩下600的权限**

】

请求就生成了



现在把这个httpd.csr发送给CA

CA一签署 就完成了 –SCP命令



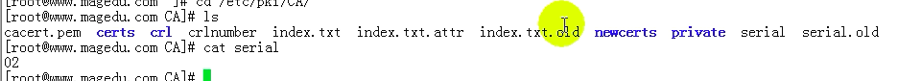
去CA下面



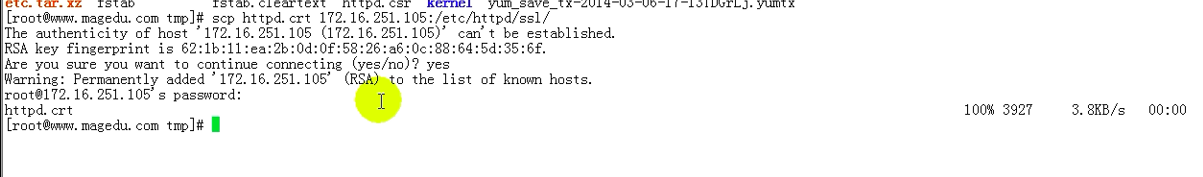
现在进行签署 使用openssl ---- 签证的命令就是ca子命令

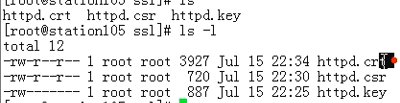


【openssl ca –in $request\_for\_ca(csr) –out $OTHER\_CERTIFICATE 这个openssl 子命令是CA 并且 有输入 输入信息就是代表CA自身信息的私钥 --- 签名其他主机的host之后 输出就是该CA对目标主机的数字证书】

serial就变成了2

现在 把证书SCP到申请者的host即可

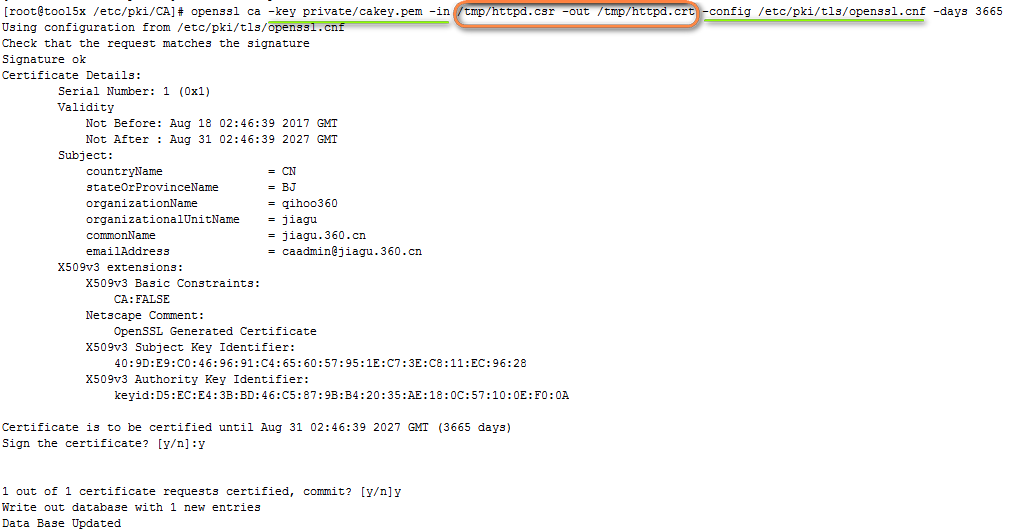




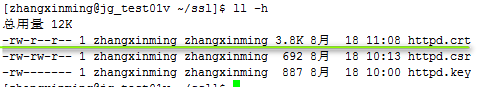
【我在生成证书的过程中出现了错误



由于暂时没找到这个默认配置的位置 所以 我们使用-config选项指定我们的证书的配置路径 同时为了避免出现其他问题 直接使用-key指定key的位置



传输过去之后 目标机器的被签发的证书如下



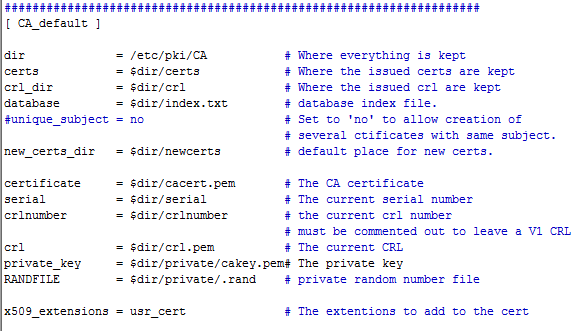
再次解释一下

最先有key ---- private key ---- httpd.key

然后有csr -----certificate request ---- httpd.csr

最后有crt ----- certificate ---- httpd.crt】

【总结



通过这个openssl.cnf的配置 知道了CA\_default的如下信息 CA默认的配置

以dir为基准的前提下 相关的目录要有

$dir/certs ---- 存放签发之后的证书的目录

$dir/crl ----被吊销的证书的位置

$dir/**cacert.pem** ---- CA字签证书 ---- 自身的招牌

$dir/serial ----- 签发证书的编号 （第一次要初始化01）

$dir/crlnumber ----被吊销的证书的编号

$dir/private/**cackey.pem** ---- CA的私钥

就知道为什么CA生成这些信息的时候 为什么证书名叫cacert.pem 为什么私钥叫cakey.pem 为什么放在这个目录下 因为CA默认的是这些

------------------------------------------------------------------------------

要想先构建一个私有CA ----

为了签名 自己要有家伙事 武器就是CA的秘钥 openssl genrsa ----- cakey.pem  
 修改自己的配置文件的基本信息 也就是CA\_default的位置 --- /etc/pki/tls/openssl.cnf

修改 **dir = /etc/pki/CA ---- 原来的位置是 ../../CA**

因为自己已经是被信任的 所以 给自己发牌照吧 让别人看到 ---- openssl req –new –x509 –key 【这个过程中 会让你主动填写CA的相关信息】

===== 这样 自己的招牌有了 也就是营业执照有了 就可以给别人签证书了

---------------------------------------------------------------------------------------------

第三方要获取CA给自己的证书 就必须知道CA需要哪些信息 ----

原理上是第三方自身的信息+第三方的公钥 ---- 公钥是通过私钥获取的

还有 就是 CA接收的信息核心是第三方自身信息 + 第三方的公钥 ----- 根据OOP的思想 第三方自身信息+ 第三方公钥 合并起来成为 **csr --- certificate signature request** ---证书签发请求

---- 第三方的核心还是私钥

\*\*先生成自身的私钥 ---- openssl genrsa ---- xxx

---- 因为公钥就来自于私钥 ---- 所以 自己机器没必要在生成公钥出来 只需要把公钥放到证书申请中就行了 所以 openssl req –new 要指定-key参数 这个后面跟的是自身的私钥

\*\*依据私钥生成证书签发申请 ---- openssl req –new –key –out ----- xxx.csr

[这个过程还会填入个人的信息 因为指定了-key 所以 –key指定的私钥会生成公钥 填写到request文件中 ---- 这样需要CA加密的信息就全了!!!]

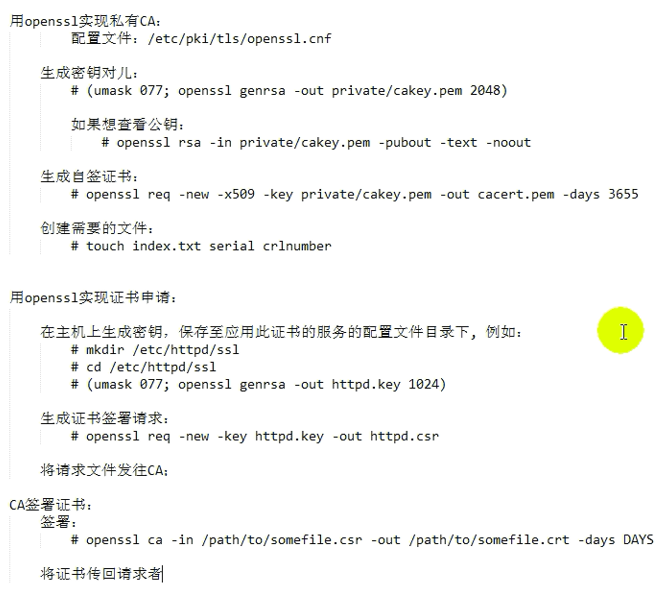
\*\*scp等任意方式发送给CA

-------------------------------------------------------------------------------------------------

CA收到xxx.csr之后 进行签发 openssl ca [–key] –in –out –days [–config]

签发之后 还给请求方】

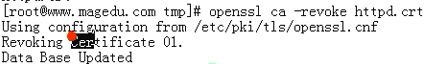
这样证书就有了 然后配置主机使用这个证书



这几个文件 貌似在Centos6上面可以不进行创建

假设需要吊销证书



 这样就吊销了



吊销之后 都要保存这个文件 会存放到吊销列表中

这样 证书的基本实现 就完成了