Linux 下的工具们通常使用 base64 编码的文本格式，相关常用后缀如下：  
\* 证书：.crt, .pem  
\* 私钥：.key  
\* 证书请求：.csr  
  
.cer 好像是二进制的证书。当然你也可以把证书和 key 放到同一个文件里边。这时候扩展名通常叫 .pem。Java 的 keystore 什么的都是二进制的，好像是自有格式。

公钥和私钥一般都是用PEM方式保存，但是公钥文件还不足以成为证书，还需要CA的签名  
CSR是证书签名请求，CA用自己的私钥文件签名之后生成CRT文件就是完整的证书了  
CER和CRT其实是一样的，只是一般Linux下面叫CRT多，Windows下面叫CER多

ssh-keygen 生成密钥,默认路径/c/Users/DELL-13/.ssh/id\_rsa

Your identification has been saved in /c/Users/DELL-13/.ssh/id\_rsa.

Your public key has been saved in /c/Users/DELL-13/.ssh/id\_rsa.pub.

The key fingerprint is:

SHA256:jrD4WXfcW0uzzi58xVEFhLerRc0+prSihOMUO6X7vY8 DELL-13@Jiangli-PC

The key's randomart image is:

+---[RSA 2048]----+

| oo.o|

| . . .|

| . +.|

| o.o|

| . .S. . +.|

| . o o\*. . .oo+|

| . . oBo.o.ooB..|

| . oo.=...=B.+ |

| o o.o.E=\*\* |

keytool 是个密钥和证书管理工具。它使用户能够管理自己的公钥/私钥对及相关证书，用于（通过数字签名）自我认证（用户向别的用户/服务认证自己）或数据完整性以及认证服务。

在JDK 1.4以后的版本中都包含了这一工具，它的位置为%JAVA\_HOME%\bin\keytool.exe

创建证书主要是使用" -genkeypair"，该命令的可用参数如下：

范例：生成一个名称为test1的证书  
　　Cmd代码  
　　1 keytool -genkeypair -alias "test1" -keyalg "RSA" -keystore "test.keystore"

功能：  
　　创建一个别名为test1的证书，该证书存放在名为test.keystore的密钥库中，若test.keystore密钥库不存在则创建。

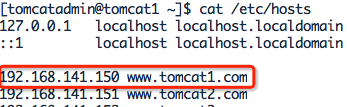
参数说明：  
　　-genkeypair：生成一对非对称密钥;  
　　-alias：指定密钥对的别名，该别名是公开的;  
-keyalg：指定加密算法，本例中的采用通用的RAS加密算法;  
　　-keystore:密钥库的路径及名称，不指定的话，默认在操作系统的用户目录下生成一个".keystore"的文件

注意：  
　　1.密钥库的密码至少必须6个字符，可以是纯数字或者字母或者数字和字母的组合等等  
　　2."名字与姓氏"应该是输入域名，而不是我们的个人姓名，其他的可以不填  
　　执行完上述命令后，在操作系统的用户目录下生成了一个"test.keystore"的文件，如下图所示：

在机器生成证书：

jasig CAS实现单点登录(数据库认证)

这时会在用户主目录下生成.keystore文件，这个文件也可在上述命令种指定，其中生成过程会填入一些信息，注意输入第一个时(名字与姓氏)就输入你本机器的域名(不能时IP), 这里我的是www.tomcat1.com:



jasig CAS实现单点登录(数据库认证)

　四、查看密钥库里面的证书  
　　范例：查看test.keystore这个密钥库里面的所有证书  
　　Cmd代码  
　　1 keytool -list -keystore test.keystore

　五、导出到证书文件  
　　范例：将名为test.keystore的证书库中别名为test1的证书条目导出到证书文件test.crt中  
　　Cmd代码  
　　1 keytool -export -alias test1 -file test.crt -keystore test.keystore

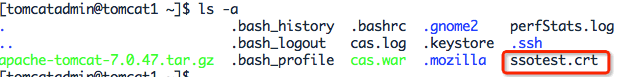
运行结果：在操作系统的用户目录(gacl)下生成了一个"test.crt"的文件，如下图所示：

导出证书

jasig CAS实现单点登录(数据库认证)



于是在用户主目录下就有了ssotest.crt证书文件(保留着，待会客户端配置要用)，



配置Tomcat SSL: ${TOMCAT\_HOME}/conf/server.xml中83-93行修改为:

<!-- Define a SSL HTTP/1.1 Connector on port 8443

This connector uses the JSSE configuration, when using APR, the

connector should be using the OpenSSL style configuration

described in the APR documentation -->

<!-- configure ssl -->

<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"

maxThreads="150" scheme="https" secure="true"

clientAuth="false" sslProtocol="TLS"

keystoreFile="/home/tomcatadmin/.keystore"

keystorePass="ssotest"/>

　导入证书  
　　范例：将证书文件test.crt导入到名为test\_cacerts的证书库中  
　　Cmd代码：  
　　1 keytool -import -keystore test\_cacerts -file test.crt

服务端导入证书

keytool -import -keystore $JAVA\_HOME/jre/lib/security/cacerts -file ~/ssotest.crt -alias ssotest

# NOTE: 有可能会有异常:java.io.IOException: Keystore was tampered with, or password was incorrect. 那就先删除本机上述的cacerts文件。

删除密钥库中的条目  
　　范例：删除密钥库test.keystore中别名为test1的证书条目  
　　Cmd代码：  
　　1 keytool -delete -keystore test.keystore -alias test1

修改证书条目的口令  
　　范例：将密钥库test.keystore中别名为test2的证书条目的密码修改为xdp123456  
　　Cmd代码：  
　　1 keytool -keypasswd -alias test2 -keystore test.keystore

作者：刘长元  
链接：https://www.zhihu.com/question/29620953/answer/45012411  
来源：知乎  
著作权归作者所有，转载请联系作者获得授权。

纯粹个人理解，因为SSL本身确实比较复杂，欢迎纠正。  
我把SSL系统比喻为工商局系统。  
首先有SSL就有CA，certificate authority。证书局，用于制作、认证证书的第三方机构，我们假设营业执照非常难制作，就像身份证一样，需要有制证公司来提供，并且提供技术帮助工商局验证执照的真伪。  
然后CA是可以有多个的，也就是可以有多个制证公司，但工商局就只有一个，它来说那个制证公司是可信的，那些是假的，需要打击。在SSL的世界中，微软、Google和Mozilla扮演了一部分这个角色。也就是说，IE、Chrome、Firefox中内置有一些CA，经过这些CA颁发，验证过的证书都是可以信的，否则就会提示你不安全。  
这也是为什么前几天Chrome决定屏蔽CNNIC的CA时，CNNIC那么遗憾了。  
也因为内置的CA是相对固定的，所以当你决定要新建网站时，就需要购买这些内置CA颁发的证书来让用户看到你的域名前面是绿色的，而不是红色。而这个最大的卖证书的公司就是VeriSign如果你听说过的话，当然它被卖给了Symantec，这家伙不只出Ghost，还是个卖证书的公司。  
  
要开店的老板去申请营业执照的时候是需要交他的身份证的，然后办出来的营业执照上也会有他的照片和名字。身份证相当于私钥，营业执照就是证书，Ceritficate，.cer文件。  
  
然后关于私钥和公钥如何解释我没想好，而它们在数据加密层面，数据的流向是这样的。

消息-->[公钥]-->加密后的信息-->[私钥]-->消息

公钥是可以随便扔给谁的，他把消息加了密传给我。对了，可以这样理解，我有一个箱子，一把锁和一把钥匙，我把箱子和开着的锁给别人，他写了信放箱子里，锁上，然后传递回我手上的途中谁都是打不开箱子的，只有我可以用原来的钥匙打开，这就是SSL，公钥，私钥传递加密消息的方式。这里的密钥就是key文件。  
  
于是我们就有了.cer和.key文件。接下来说keystore  
  
不同的语言、工具序列SSL相关文件的格式和扩展名是不一样的。  
其中Java系喜欢用keystore, truststore来干活，你看它的名字，Store，仓库，它里面存放着key和信任的CA，key和CA可以有多个。  
这里的truststore就像你自己电脑的证书管理器一样，如果你打开Chrome的设置，找到HTTP SSL，就可以看到里面有很多CA，truststore就是干这个活儿的，它也里面也是存一个或多个CA让Tomcat或Java程序来调用。  
而keystore就是用来存密钥文件的，可以存放多个。  
  
然后是PEM，它是由RFC1421至1424定义的一种数据格式。其实前面的.cert和.key文件都是PEM格式的，只不过在有些系统中（比如Windows）会根据扩展名不同而做不同的事。所以当你看到.pem文件时，它里面的内容可能是certificate也可能是key，也可能两个都有，要看具体情况。可以通过openssl查看。  
  
RFC1421的第一节是这样说的

1. Executive Summary

This document defines message encryption and authentication

procedures, in order to provide privacy-enhanced mail (PEM) services

for electronic mail transfer in the Internet. It is intended to

become one member of a related set of four RFCs. The procedures

defined in the current document are intended to be compatible with a

wide range of key management approaches, including both symmetric

(secret-key) and asymmetric (public-key) approaches for encryption of

data encrypting keys. Use of symmetric cryptography for message text

encryption and/or integrity check computation is anticipated. RFC1422 specifies supporting key management mechanisms based on the use

of public-key certificates. RFC 1423 specifies algorithms, modes,

and associated identifiers relevant to the current RFC and to RFC1422. RFC 1424 provides details of paper and electronic formats and

procedures for the key management infrastructure being established in

support of these services.

本文档定义了Internet中消息加密和身份认证的流程，它用于为电子邮件传输提供增强的私密邮件服务(PEM)。它是4个相关RFC组中的一个，当前文档中定义的流程打算与各种密钥管理方法兼容，包含对称和不对称数据加密协议。使用对称方式来加密文本消息和完整性检查是可以预期的。RFC1422描述了基于公钥证书和密钥管理机制，RFC1423描述了算法，模式和与RFC1421/1422相关的身份认证的内容，RFC1424提供了详尽的纸制/电子格式和流程，来描述如何构建支持这些协议的密钥管理基础设施。

Privacy enhancement services (confidentiality, authentication,

message integrity assurance, and non-repudiation of origin) are

offered through the use of end-to-end cryptography between originator

and recipient processes at or above the User Agent level. No special

processing requirements are imposed on the Message Transfer System at

endpoints or at intermediate relay sites. This approach allows

privacy enhancement facilities to be incorporated selectively on a

site-by-site or user-by-user basis without impact on other Internet

entities. Interoperability among heterogeneous components and mail

transport facilities is supported.

The current specification's scope is confined to PEM processing

procedures for the RFC-822 textual mail environment, and defines the

Content-Domain indicator value "RFC822" to signify this usage.

Follow-on work in integration of PEM capabilities with other

messaging environments (e.g., MIME) is anticipated and will be

addressed in separate and/or successor documents, at which point

additional Content-Domain indicator values will be defined.

至于CAS4.0这个东西，网上一堆的教程，中文的、英文的，先自己做吧，遇到问题时把你的操作过程，在什么步骤遇到什么问题，出现什么错误，你怎么理解都写出来，成长比你成功部署一个CAS要大多了。  
  
最后，看到一篇文章写的不错，在这里翻译一部分。  
[What is SSL and what are Certificates?](//link.zhihu.com/?target=http%3A//www.tldp.org/HOWTO/SSL-Certificates-HOWTO/x64.html)

The Secure Socket Layer protocol was created by Netscape to ensure secure transactions between web servers and browsers. The protocol uses a third party, a Certificate Authority (CA), to identify one end or both end of the transactions. This is in short how it works.

1. A browser requests a secure page (usually https://).
2. The web server sends its public key with its certificate.
3. The browser checks that the certificate was issued by a trusted party (usually a trusted root CA), that the certificate is still valid and that the certificate is related to the site contacted.
4. The browser then uses the public key, to encrypt a random symmetric encryption key and sends it to the server with the encrypted URL required as well as other encrypted http data.
5. The web server decrypts the symmetric encryption key using its private key and uses the symmetric key to decrypt the URL and http data.
6. The web server sends back the requested html document and http data encrypted with the symmetric key.
7. The browser decrypts the http data and html document using the symmetric key and displays the information.

安全套接层协议是由Netscape创建的，它用来保证在WEB服务器和浏览器间的数据被安全传输。  
协议使用一个第三方的证书局（CA）来验证传输的一方或双方的身份。下面是简单的描述它如何工作：

1. 浏览器请求一个安全的页面(通过以https://开头)
2. WEB服务器返回它的公钥和证书
3. 浏览检查证书是由可信的机构颁发的（通过是可信的根CA），证书仍然有效并且证书与被访问的网站相关
4. 浏览器使用公钥来加密一个随机的对称密钥，加上加密后的URL和其它加密后的http数据一起发回至服务器。
5. WEB服务器使用私钥解密对称密钥，并用它来解密在浏览器上加密了的URL和http数据
6. WEB服务器使用对称密钥加密请求的HTML文档和http数据并发回至浏览器
7. 浏览器使用对称密钥解密HTML文档和http数据并展示给用户