[OpenSSL命令详解](http://blog.csdn.net/boss666666/article/details/10284649)

这一针在农信银2代项目中有幸能碰到加密方面的东西，所以就好好学习了一下。不足之处还请前辈们见谅。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| OpenSSL功能远胜于KeyTool，可用于根证书，服务器证书和客户证书的管理 | | | | | | | |
| 这里使用的是Win32OpenSSL\_Light-1\_0\_1e.exe | | | | |  |  |  |
| http://www.slproweb.com/products/Win32OpenSSL.html | | | | | |  |  |
| 1，构建根证书 | |  |  |  |  |  |  |
| 构建根证书前，需要构建随机数文件（.rand）,完整命令如 | | | | | |  |  |
| openssl rand -out private/.rand 1000 | | | | |  |  |  |
| rand       随机数命令。这里将随机数文件输出到private目录下。 | | | | | | |  |
| -out      输出文件路径， | | |  |  |  |  |  |
| 这里的参数1000，指定来产生伪随机字节数 | | | | |  |  |  |
| 2，构建根证书私钥 | |  |  |  |  |  |  |
| openssl genrsa -aes256 -out private/ca.key.pem 2048 | | | | | |  |  |
| 3,生成根证书签发申请 | | |  |  |  |  |  |
| 完成密钥构建操作后，我们需要生成根证书签发申请文件（ca.csr）,完整命令如代码 | | | | | | | | |
| openssl req -new -key private/ca.key.pem -out private/ca.csr -subj "/C=CN/ST=BJ/L=BJ/O=lesaas/OU=lesaas/CN=\*.lesaas.cn" | | | | | | | | | | | | | |
| req          产生证书签发申请命令 | | | |  |  |  |  |  |  |  |  |  |  |
| -new         表示新请求 | | |  |  |  |  |  |  |  |  |  |  |  |
| -key         密钥,这里为private/ca.key.pem文件 | | | | | |  |  |  |  |  |  |  |  |
| -out         输出路径,这里为private/ca.csr文件 | | | | | |  |  |  |  |  |  |  |  |
| -subj        指定用户信息。这里使用泛域名"\*.lesaas.cn" | | | | | | |  |  |  |  |  |  |  |
| 得到根证书签发申请文件后，我们可以将其发生给CA机构签发，当然我们也可以自行签发根证书。 | | | | | | | | | |  |  |  |  |
| 4，签发根证书（自行签发根证书） | | | |  |  |  |  |  |  |  |  |  |  |
| openssl x509 -req -days 10000 -sha1 -extensions v3\_ca -signkey private/ca.key.pem -in private/ca.csr -out certs/ca.cer | | | | | | | | | | | | | |
| x509        签发X.509格式证书命令。 | | | |  |  |  |  |  |  |  |  |  |  |
| -req        表示证书输入请求。 | | | |  |  |  |  |  |  |  |  |  |  |
| -days       表示有效天数,这里为10000天。 | | | | |  |  |  |  |  |  |  |  |  |
| -shal       表示证书摘要算法,这里为SHA1算法。 | | | | | |  |  |  |  |  |  |  |  |
| -extensions 表示按OpenSSL配置文件v3\_ca项添加扩展。 | | | | | |  |  |  |  |  |  |  |  |
| -signkey    表示自签名密钥,这里为private/ca.key.pem。 | | | | | | |  |  |  |  |  |  |  |
| -in         表示输入文件,这里为private/ca.csr。 | | | | | |  |  |  |  |  |  |  |  |
| -out        表示输出文件,这里为certs/ca.cer。 | | | | | |  |  |  |  |  |  |  |  |
| ==================================================================================================== | | | | | | | | | | | |  |  |
| OpenSSL产生的数据证书不能再JAVA语言环境中直接使用，需要将其转化为PKCS#12编码格式。 | | | | | | | | | |  |  |  |  |
| 完整命令如代码 | |  |  |  |  |  |  |  |  |  |  |  |  |
| 5，根证书转化 | |  |  |  |  |  |  |  |  |  |  |  |  |
| openssl pkcs12 -export -cacerts -inkey private/ca.key.pem -in certs/ca.cer -out certs/ca.p12 | | | | | | | | | | |  |  |  |
| pkcs12          PKCS#12编码格式证书命令。 | | | | |  |  |  |  |  |  |  |  |  |
| -export         表示导出证书。 | | | |  |  |  |  |  |  |  |  |  |  |
| -cacerts        表示仅导出CA证书。 | | | |  |  |  |  |  |  |  |  |  |  |
| -inkey          表示输入密钥,这里为private/ca.key.pem | | | | | | |  |  |  |  |  |  |  |
| -in             表示输入文件,这里为certs/ca.cer | | | | | |  |  |  |  |  |  |  |  |
| -out            表示输出文件,这里为certs/ca.p12 | | | | | |  |  |  |  |  |  |  |  |
| 个人信息交换文件（PKCS#12） 可以作为密钥库或信任库使用，我们可以通过KeyTool查看密钥库的详细信息。 | | | | | | | | | | |  |  |  |
| 6，查看密钥库信息 | |  |  |  |  |  |  |  |  |  |  |  |  |
| keytool -list -keystore d:/CA/certs/ca.p12 -storetype pkcs12 -v -storepass 123456 | | | | | | | | | |  |  |  |  |
| 注意，这里参数-storetype值为“pkcs12”。 | | | | |  |  |  |  |  |  |  |  |  |
| 我们已经构建了根证书（ca.cer）,我们可以使用根证书签发服务器证书和客户证书。 | | | | | | | | |  |  |  |  |  |
| 7，构建服务器证书 | |  |  |  |  |  |  |  |  |  |  |  |  |
| 服务器证书的构建与根证书构建相似，首先需要构建私钥。 | | | | | |  |  |  |  |  |  |  |  |
| （1）构建服务器私钥 | | |  |  |  |  |  |  |  |  |  |  |  |
| openssl genrsa -aes256 -out private/server.key.pem 2048 | | | | | | |  |  |  |  |  |  |  |
| genrsa       产生RSA密钥命令。 | | | |  |  |  |  |  |  |  |  |  |  |
| -aes256      使用AES算法（256位密钥）对产生的私钥加密。可选算法包括DES，DESede，IDEA和AES。 | | | | | | | | | | |  |  |  |
| -out         输出路径,这里指private/server.key.pem。 | | | | | |  |  |  |  |  |  |  |  |
| 这里的参数2048，指RSA密钥长度位数，默认长度为512位。 | | | | | |  |  |  |  |  |  |  |  |
| （2）生成服务器证书签发申请 | | | |  |  |  |  |  |  |  |  |  |  |
| openssl req -new -key private/server.key.pem -out private/server.csr -subj "/C=CN/ST=BJ/L=BJ/O=lesaas/OU=lesaas/CN=www.lesaas.cn" | | | | | | | | | | | | | | |
| req          产生证书签发申请命令 | | | |  |  |  |  |  |  |  |  |  |  |  |
| -new         表示新请求。 | | |  |  |  |  |  |  |  |  |  |  |  |  |
| -key         密钥,这里为private/ca.key.pem文件 | | | | | |  |  |  |  |  |  |  |  |  |
| -out         输出路径,这里为private/ca.csr文件 | | | | | |  |  |  |  |  |  |  |  |  |
| -subj        指定用户信息,这里使用域名“www.lesaas.cn”作为用户名。 | | | | | | | |  |  |  |  |  |  |  |
| 我们已经获得了根证书，可以使用根证书签发服务器证书。 | | | | | |  |  |  |  |  |  |  |  |  |
| （3）签发服务器证书 | | |  |  |  |  |  |  |  |  |  |  |  |  |
| openssl x509 -req -days 3650 -sha1 -extensions v3\_req -CA certs/ca.cer -CAkey private/ca.key.pem -CAserial ca.srl -CAcreateserial -in private/server.csr -out | | | | | | | | | | | | | | | | | |
| certs/server.cer | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| x509           签发X.509格式证书命令。 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -req           表示证书输入请求。 | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -days          表示有效天数,这里为3650天。 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -sha1          表示证书摘要算法,这里为SHA1算法。 | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| -extensions    表示按OpenSSL配置文件v3\_req项添加扩展。 | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
| -CA            表示CA证书,这里为certs/ca.cer | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| -CAkey         表示CA证书密钥,这里为private/ca.key.pem | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
| -CAserial      表示CA证书序列号文件,这里为ca.srl | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| -CAcreateserial表示创建CA证书序列号 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -in            表示输入文件,这里为private/server.csr | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| -out           表示输出文件,这里为certs/server.cer | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| 这里我们同样需要将OpenSSL产生的数子证书转化为PKCS#12编码格式。完整命令如下 | | | | | | | | |  |  |  |  |  |  |  |  |  |
| （4）服务器证书转换 | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| openssl pkcs12 -export -clcerts -inkey private/server.key.pem -in certs/server.cer -out certs/server.p12 | | | | | | | | | | | |  |  |  |  |  |  |
| pkcs12         PKCS#12编码格式证书命令。 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -export        表示导出证书。 | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -clcerts       表示仅导出客户证书。 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -inkey         表示输入文件,这里为private/server.key.pem | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
| -in            表示输入文件,这里为certs/ca.cer | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| -out           表示输出文件,这里为certs/server.p12 | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| 我们已经构建了服务器证书（server.cer）,并可使用该证书构建基于单向认证网络 | | | | | | | | |  |  |  |  |  |  |  |  |  |
| （5）构建客户证书 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 客户证书的构建与服务器证书构建基本一致，首先需要构建私钥。 | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
| 产生客户私钥 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| openssl genrsa -aes256 -out private/client.key.pem 2048 | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
| genrsa         产生RSA密钥命令 | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -aes256        使用AES算法（256为密钥）对产生的私钥加密。可选算法包括DES,DESede,IDEA和AES。 | | | | | | | | | | |  |  |  |  |  |  |  |
| -out           输出路径,这里指private/client.key.pem | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| 这里的参数2048，指RSA密钥长度位数，默认长度为512位 | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| 完成客户证书密钥构建后，我们需要产生客户证书签发申请 | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| （6）生成客户证书签发申请 | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| openssl req -new -key private/client.key.pem -out private/client.csr -subj "/C=CN/ST=BJ/L=BJ/O=lesaas/OU=lesaas/CN=lesaas" | | | | | | | | | | | | | |  |  |  |  |
| req            产生证书签发申请命令 | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -new           表示新的请求。 | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -key           密钥,这里为private/client.csr文件 | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| -subj          指定用户信息,这里使用“lesaas”作为用户名 | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
| 我们已经获得了根证书，可以使用根证书签发客户证书（client.cer） | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
| （7）签发客户证书 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| openssl ca -days 3650 -in private/client.csr -out certs/client.cer -cert certs/ca.cer -keyfile private/ca.key.pem | | | | | | | | | | | | |  |  |  |  |  |
| ca            签发证书命令 | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -days         表示证书有效期,这里为3650天。 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -in           表示输入文件,这里为private/client.csr | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| -out          表示输出文件,这里为certs/server.cer | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| -cert         表示证书文件,这里为certs/ca.cer | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| -keyfile      表示根证书密钥文件,这里为private/ca.key.pem | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
| 最后，我们需要将获得客户证书转化Java语言可以识别的PKCS#12编码格式。 | | | | | | | |  |  |  |  |  |  |  |  |  |  |
| （8）客户证书转换 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| openssl pkcs12 -export -inkey private/client.key.pem -in certs/client.cer -out certs/client.p12 | | | | | | | | | | |  |  |  |  |  |  |  |
| pkcs12       PKCS#12编码格式证书命令、 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -export      表示导出证书 | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -clcerts     表示仅导出客户证书。 | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -inkey       表示输入密钥,这里为private/client.key.pem | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
| -in          表示输入文件,这里为certs/client.cer | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| -out         表示输出文件,这里为certs/client.p12 | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| 至此，我们完成了双向认证的所需的全部证书。 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 数字证书是公钥的载体，而密钥库可以包含公钥、私钥信息。 | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |
| JKS和PKCS#12都是比较常用的两种密钥库格式/标准。对于前者，搞Java开发，尤其是接触过HTTPS平台的朋友，并不陌生。JKS文件（通常为\*.jks或\*.keystore，扩展名无关）可以通过Java原生工具——KeyTool生成；而后者PKCS#12文件（通常为\*.p12或\*.pfx，意味个人信息交换文件），则是通过更为常用的OpenSSL工具产生。 | | | | | | | | | | | | | | | | | | | | | | | |
| 当然，这两者之间是可以通过导入/导出的方式进行转换的！当然，这种转换需要通过KeyTool工具进行！ | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 回归正题，计费同事遇到一个难题：合作方交给他们一个\*.pfx文件，需要他们从中提取密钥，然后进行加密交互。其实，通过Java直接操作密钥库文件（或个人信息交换文件）对于一般Java开发人员来说，这都是个冷门。不接触数字安全，根本不知所云。况且，Java原生的密钥库文件格式为JKS，如何操作\*.pfx文件？密钥库操作需要获知密钥库别名，\*.pfx别名是什么？！接下来就解决这些问题！ | | | | | | | | | | | | | | | | | | | | | | | |
| (PKCS#12是base64编码的) | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 方案： |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 通过keytool密钥库导入命令importkeystore，将密钥库格式由PKCS#12转换为JKS。 | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 检索新生成的密钥库文件，提取别名信息。 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 由密钥库文件导出数字证书（这里将用到别名）。 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 通过代码提取公钥/私钥、签名算法等 | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 先看格式转换： | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Cmd代码  收藏代码 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| echo 格式转换 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| keytool -importkeystore -v  -srckeystore zlex.pfx -srcstoretype pkcs12 -srcstorepass 123456 -destkeystore zlex.keystore -deststoretype jks -deststorepass 123456 | | | | | | | | | | | | | | | | | | |  |  |  |  |  |
| -importkeystore导入密钥库，通过格式设定，我们可以将PKCS#12文件转换为JKS格式。 | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -v显示详情 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -srckeystore源密钥库，这里是zlex.pfx | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -srcstoretype源密钥库格式，这里为pkcs12 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -srcstorepass源密钥库密码，这里为123456 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -destkeystore目标密钥库，这里为zlex.keystore | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -deststoretype目标密钥库格式，这里为jks，默认值也如此 | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -deststorepass目标密钥库密码，这里为123456 | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 通过这个操作，我们能够获得所需的密钥库文件zlex.keystore。 | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 这时，我们已经获得了密钥库文件，只要确定对应的别名信息，就可以提取公钥/私钥，以及数字证书，进行加密交互了！ | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |
| Cmd代码  收藏代码 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| echo 查看证书 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| keytool -list -keystore zlex.keystore -storepass 123456 -v | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -list列举密钥库 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -keystore密钥库,这里是zlex.keystore | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -storepass密钥库密码,这里是123456 | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -v显示详情 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 现在，我们把证书导出！ | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Cmd代码  收藏代码 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| echo 导出证书 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| keytool -exportcert -alias 1 -keystore zlex.keystore -file zlex.crt -storepass 123456 | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -exportcert导出证书 | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -alias别名,这里是1 | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -keystore密钥库,这里是zlex.keystore | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -file证书文件,这里是zlex.crt | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| -storepass密钥库密码,这里是123456 | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 现在证书也导出了，我们可以提取公钥/私钥，进行加密/解密，签名/验证操作了！当然，即便没有证书，我们也能够通过密钥库（JKS格式）文件获得证书，以及公钥/私钥、签名算法等。 | | | | | | | | | | | | | | | | | | |  |  |  |  |  |