## 文件列表

私钥:rsaprivatekey\_pass.pem

公钥: rsaprivatekey.cert

JAVA

私钥: ca.private.der

## openssl证书生成和使用

以OpenSSL 1.0.0l版本说明：

1）产生1024位RSA私匙，用3DES加密它，口令为123456。输出到文件rsaprivatekey\_pass.pem：

openssl genrsa -out rsaprivatekey\_pass.pem -passout pass:123456 -des3 1024

此私钥文件及其密码必须要安全保管好。

2）生成证书。days为有效天数，可以不加。输出证书文件到rsaprivatekey.cert：

openssl req -new -x509 -key rsaprivatekey\_pass.pem -out rsaprivatekey.cert -days 1095 -passin pass:123456

此证书文件中包含公钥，用于和对方进行公开的交换。

3）测试一下，使用私钥对test.txt文本内容进行数字签名，输出到test.sig：

openssl rsautl -sign -in test.txt -out test.sig -inkey rsaprivatekey\_pass.pem -passin pass:123456

4）使用公钥证书对数字签名进行验证，输出到test.vfy，此时test.vfy和test.txt的内容应完全一样：

openssl rsautl -verify -in test.sig -out test.vfy -inkey rsaprivatekey.cert -certin

**注:**

1. 私匙：(PKCS8加密码)

openssl pkcs8 -topk8 -inform PEM -outform DER -in rsaprivatekey\_pass.pem -nocrypt -passin pass:123456 -out ca.private.der

## der私匙转换

**用途：**

pkcs8格式的私钥转换工具。它处理在PKCS#8格式中的私钥文件。它可以用多样的PKCS#5 (v1.5 and v2.0)和 PKCS#12[**算法**](http://lib.csdn.net/base/datastructure)来处理没有解密的PKCS#8 PrivateKeyInfo格式和EncryptedPrivateKeyInfo格式。

**用法：**

**[cpp]** [view plain](http://blog.csdn.net/as3luyuan123/article/details/16105435) [copy](http://blog.csdn.net/as3luyuan123/article/details/16105435)

1. openssl pkcs8 [-inform PEM|DER] [-outform PEM|DER] [-in filename] [-passin arg] [-out filename]
2. [-passout arg] [-topk8] [-noiter] [-nocrypt] [-nooct] [-embed] [-nsdb] [-v2 alg] [-v1 alg] [-engine id]

**选项说明：**

-inform PEM|DER：：输入文件格式，DER或者PEM格式。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。

-outform DER|PEM：输出文件格式，DER或者PEM格式。

-in filename：输入的密钥文件，默认为标准输入。如果密钥被加密，会提示输入一个密钥口令。

-passin arg：输入文件口令保护来源。

-out filename：输出文件，默认为标准输出。如果任何加密操作已经执行，会提示输入一个密钥值。输出的文件名字不能和输入的文件名一样。

-passout arg：输出文件口令保护来源。

-topk8：通常的是输入一个pkcs8文件和传统的格式私钥文件将会被写出。设置了此选项后，位置转换过来：输入一个传统格式的私钥文件，输出一个PKCS#8格式的文件。

-noiter：MAC保护计算次数为1。

-nocrypt：PKCS#8密钥产生或输入一般用一个适当地密钥来加密PKCS#8 EncryptedPrivateKeyInfo结构。设置了此选项后，一个不加密的PrivateKeyInfo结构将会被输出。这个选项一直不加密私钥文件，在绝对必要的时候才能够使用。某些软件例如一些[**Java**](http://lib.csdn.net/base/java)代码签名软件使用不加密的私钥文件。

-nooct：这个选项产生的RSA私钥文件是一个坏的格式，一些软件将会使用。特别的是，私钥文件必须附上一个八位组字符串，但是一些软件仅仅包含本身的结构体没有使八位组字符串所环绕。不采用八位组表示私钥。

-embed：这个选项产生的RSA私钥文件是一个坏的格式。在私钥结构体中采用[**嵌入式**](http://lib.csdn.net/base/embeddeddevelopment)DSA参数格式。在这个表单中，八位组字符串包含了ASN1 SEQUENCE中的两种结构：一个SEQUENCE包含了密钥参数，一个ASN1 INTEGER包含私钥值。

-nsdb：这个选项产生的RSA私钥文件是一个坏的格式并兼容了Netscape私钥文件[**数据库**](http://lib.csdn.net/base/mysql)。采用NetscapeDB的DSA格式。

-v2 alg：采用PKCS#5 v2.0，并指定加密算法，默认的是PKCS#8私钥文件被叫做B<pbeWithMD5AndDES-CBC>（该算法用56字节的DES加密但是在PKCS#5 v1.5中有更加强壮的加密算法）的加密算法用口令进行加密。用B<-v2>选项，PKCS#5 v2.0相关的算法将会被使用，可以是des3（168字节）和rc2（128字节），推荐des3。

-v1 alg：采用PKCS#5 v1.5或pkcs12，并指定加密算法。可采用的算法见下面。

 -engine id：指定硬件引擎。

**注意：**

加密了的PEM编码PKCS#8文件表单用下面的头部和尾部：

-----BEGIN ENCRYPTED PRIVATE KEY-----

 -----END ENCRYPTED PRIVATE KEY-----

未加密的表单用：

-----BEGIN PRIVATE KEY-----

 -----END PRIVATE KEY-----

跟传统的SSLeay算法相比，用PKCS#5 v2.0系列的算法加密私钥，有更高的安全性以及迭代次数。于是附加的安全性是经过深思熟虑的。

默认的加密算法仅仅是56字节的，是因为它是PKCS#8所支持的最好的方法。

有一些软件使用PKCS#12基于密钥的加密算法来加密PKCS#8格式的私钥：它们会自动的处理但是没有选项来操作。

在PKCS#8格式中，有可能的是输出DER编码格式的经过加密的私钥文件，是因为加密的详细说明包含在DER等级中，相反的是传统的格式包含在PEM邓丽中。

**PKCS#5 v1.5和 PKCS#12 算法：**

各种各样的算法可以被选项-v1所使用。包含PKCS#5 v1.5和 PKCS#12 算法。详细描述如下：

B<PBE-MD2-DES PBE-MD5-DES>：这两个算法包含在PKCS#5 v1.5中。它们仅仅提供56字节的保护，加密算法用DES。

B<PBE-SHA1-RC2-64 PBE-MD2-RC2-64 PBE-MD5-RC2-64 PBE-SHA1-DES>：它们在传统的PKCS#5 v1.5中没有被提到，但是它们用同样地密钥引出算法，被一些软件所支持。在PKCS#5 v2.0中所提到。它们使用64字节的RC2以及56字节的DES。

B<PBE-SHA1-RC4-128 PBE-SHA1-RC4-40 PBE-SHA1-3DES PBE-SHA1-2DES PBE-SHA1-RC2-128 PBE-SHA1-RC2-40>：它们是PKCS#12基于密钥的加密算法，它们允许使用高强度的加密算法，例如3des或128位的RC2。

**实例：**

用3des算法将传统的私钥文件转换为PKCS#5 v2.0：

**[cpp]** [view plain](http://blog.csdn.net/as3luyuan123/article/details/16105435) [copy](http://blog.csdn.net/as3luyuan123/article/details/16105435)

1. openssl pkcs8 -in key.pem -topk8 -v2 des3 -out enckey.pem

用PKCS#5 1.5兼容的DES算法将私钥文件转换为pkcs8文件：

**[html]** [view plain](http://blog.csdn.net/as3luyuan123/article/details/16105435) [copy](http://blog.csdn.net/as3luyuan123/article/details/16105435)

1. openssl pkcs8 -in ocspserverkey.pem -topk8 -out ocspkcs8key.pem

用PKCS#12兼容的3DES算法将私钥文件转换为pkcs8文件：

**[html]** [view plain](http://blog.csdn.net/as3luyuan123/article/details/16105435) [copy](http://blog.csdn.net/as3luyuan123/article/details/16105435)

1. openssl pkcs8 -in key.pem -topk8 -out enckey.pem -v1 PBE-SHA1-3DES

读取一个DER格式加密了的PKCS#8格式的私钥：

**[cpp]** [view plain](http://blog.csdn.net/as3luyuan123/article/details/16105435) [copy](http://blog.csdn.net/as3luyuan123/article/details/16105435)

1. openssl pkcs8 -inform DER -nocrypt -in key.der -out key.pem

转换一个PKCS#8格式的私钥到传统的私钥：

**[cpp]** [view plain](http://blog.csdn.net/as3luyuan123/article/details/16105435) [copy](http://blog.csdn.net/as3luyuan123/article/details/16105435)

1. openssl pkcs8 -in pk8.pem -out key.pem

pkcs8中的私钥以明文存放：

**[html]** [view plain](http://blog.csdn.net/as3luyuan123/article/details/16105435) [copy](http://blog.csdn.net/as3luyuan123/article/details/16105435)

1. openssl pkcs8 -in ocspserverkey.pem -topk8  -nocrypt -out ocspkcs8key.pem

标准：

PKCS#5 v2.0的[**测试**](http://lib.csdn.net/base/softwaretest)向量的实现是以通告的形式用高强度的迭代次数算法3DES、DES和RC2来加密的。很多人要确认能够解密产生的私钥。

PKCS#8格式的DSA私钥文件没有备注文件中的：在PKCS#11 v2.01中的11.9节被隐藏了的。OpenSSL的默认DSA PKCS#8私钥格式隐藏在这个标准中。

**BUGs：**

必须有一个选项打印使用的加密算法的其他详细细节，例如迭代次数。

PKCS#8用3DES和PKCS#5 v2.0必须是默认的私钥文件：目前为了命令的兼容性。

## JAVA例子

**package** com.cheuks.bin.original.test.openssl;

**import** java.io.ByteArrayOutputStream;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.security.KeyFactory;

**import** java.security.PrivateKey;

**import** java.security.PublicKey;

**import** java.security.spec.KeySpec;

**import** java.security.spec.PKCS8EncodedKeySpec;

**import** java.security.spec.X509EncodedKeySpec;

**import** sun.misc.BASE64Decoder;

**import** javax.crypto.Cipher;

**import** org.bouncycastle.jce.provider.BouncyCastleProvider;

**public** **class** der {

**public** **void** a() **throws** Exception {

FileInputStream in = **new** FileInputStream(**new** File("d:/ca.private.der"));

ByteArrayOutputStream out = **new** ByteArrayOutputStream();

**byte**[] buff = **new** **byte**[512];

**int** len;

**while** (-1 != (len = in.read(buff))) {

out.write(buff, 0, len);

}

//加密

KeySpec keySpec = **new** PKCS8EncodedKeySpec(out.toByteArray());

KeyFactory keyFactory = KeyFactory.*getInstance*("RSA");

PrivateKey key = keyFactory.generatePrivate(keySpec);

// PublicKey key1 = keyFactory.generatePublic(keySpec);

Cipher cipher = Cipher.*getInstance*("RSA", **new** BouncyCastleProvider());

cipher.init(Cipher.***ENCRYPT\_MODE***, key);

**byte**[] output = cipher.doFinal("你好吗？".getBytes());

System.***err***.println(**new** String(output));

//解码

BASE64Decoder base64Decoder = **new** BASE64Decoder();

String a = "MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDdzpR7Qhrq27fnlbNaZ5XN1ChD" + "pfb2hLTd8loRyrbij/imnOolvaHIb5dXAB6LcrUd4Vg2KbB5+PtJS1qA3TPphmLc" + "2I411o9xL5VuTzbzQATE8VQaiX91hsenysJWJb7/Xr2jf8vGWu1UiMzsz0QG347P" + "JxE78b+cSNCEu14C5QIDAQAB";

**byte**[] buffer = base64Decoder.decodeBuffer(a);

//####

KeySpec keySpec1 = **new** X509EncodedKeySpec(buffer);

PublicKey publicKey = keyFactory.generatePublic(keySpec1);

cipher.init(Cipher.***DECRYPT\_MODE***, publicKey);

output = cipher.doFinal(output);

System.***err***.println(**new** String(output));

}

**public** **static** **void** main(String[] args) **throws** Exception {

**new** der().a();

}

}