**文件控制信息**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 文件名： | NETCA PKI JAVA API Cookbook | | | | | | | |
| 编 号： |  | | | 版本号 | | | V1.0 | |
| 文件密级： | 公开 限制√ 企业秘密 企业机密 | | | | | | | |
| 适用范围： |  | | | | | | | |
| 起 草 人： | 开发部 | | | | | | | |
| 批 准 人： |  | | | | | | | |
| 批准日期： |  | 年 |  | | 月 |  | | 日 |
| 生效日期： |  | 年 |  | | 月 |  | | 日 |
| 状 态： | 待批准 已批准 已实施 参考 无效 | | | | | | | |

**文件版权说明**

　　1）本文件是广东省电子商务认证有限公司的内部管理文件，其版权归广东省电子商务认证有限公司所有，版权内容包括文件格式及内容。

　　2）任何人或组织未经广东省电子商务认证有限公司授权，不得对本文件进行复制、传播，更不得用于商业用途。

**文件修改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **段落** | **修改者** | **主要修改内容说明** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 文件编号： | | 生效日期： | | 受控编号： | |
| 密级：普通 | | 版次：1.0 | | 修改状态： | |
| 总页数 |  | 正文 |  | 附录 |  |
| 编制： | | 审核： | | 批准： | |

广东省电子商务认证有限公司

（版权所有，翻版必究）

目 录

[1 安装 1](#_Toc41665715)

[1.1 需要的JAVA版本 1](#_Toc41665716)

[1.2 安装 1](#_Toc41665717)

[1.3 在tomcat下安装 2](#_Toc41665718)

[1.4 如何使用log4j来写日志 3](#_Toc41665719)

[1.5 如何禁止日志 3](#_Toc41665720)

[2 设备 4](#_Toc41665721)

[2.1 获取内置的软件设备[JNI] 4](#_Toc41665722)

[2.2 获取当前连接的设备[JNI] 4](#_Toc41665723)

[2.3 获取指定的设备[JNI] 5](#_Toc41665724)

[2.4 获取指定的密钥对[JNI] 5](#_Toc41665725)

[2.5 获取CSP实现的密钥对[JNI] 6](#_Toc41665726)

[2.6 从密钥对获取设备对象[JNI] 6](#_Toc41665727)

[2.7 Key的PIN码是否锁死[JNI] 6](#_Toc41665728)

[2.8 获取PIN码重试次数[JNI] 7](#_Toc41665729)

[2.9 进程密码缓存[JNI] 7](#_Toc41665730)

[2.10 持久数据的权限[JNI] 8](#_Toc41665731)

[3 基本的加密运算 8](#_Toc41665732)

[3.1 获取随机数 8](#_Toc41665733)

[3.2 Hash运算 9](#_Toc41665734)

[3.3 复制Hash对象[JNI] 10](#_Toc41665735)

[3.4 Mac运算 11](#_Toc41665736)

[3.5 产生随机的对称密钥 11](#_Toc41665737)

[3.6 序列密码的对称加解密[JNI] 12](#_Toc41665738)

[3.7 ECB模式的对称加解密[JNI] 12](#_Toc41665739)

[3.8 CBC模式的对称加解密 14](#_Toc41665740)

[3.9 CTR模式的对称加解密[JNI] 15](#_Toc41665741)

[3.10 CCM模式的对称加解密[JNI] 15](#_Toc41665742)

[3.11 GCM模式的对称加解密[JNI] 16](#_Toc41665743)

[3.12 XTS模式的对称加解密[JNI] 17](#_Toc41665744)

[3.13 SM1加解密[JNI] 17](#_Toc41665745)

[3.14 产生密钥对[JNI] 18](#_Toc41665746)

[3.15 产生临时密钥对[JNI] 18](#_Toc41665747)

[3.16 获取公钥[JNI] 19](#_Toc41665748)

[3.17 RSA加密 19](#_Toc41665749)

[3.18 RSA解密 20](#_Toc41665750)

[3.19 SM2加密 20](#_Toc41665751)

[3.20 SM2解密 21](#_Toc41665752)

[3.21 RSA签名 21](#_Toc41665753)

[3.22 RSA验证签名 22](#_Toc41665754)

[3.23 SM2签名 23](#_Toc41665755)

[3.24 SM2验证签名 24](#_Toc41665756)

[3.25 ECDSA签名 25](#_Toc41665757)

[3.26 ECDSA验证签名 26](#_Toc41665758)

[3.27 ECDH密钥协商[JNI] 27](#_Toc41665759)

[3.28 使用口令导出密钥 27](#_Toc41665760)

[4 证书 28](#_Toc41665761)

[4.1 已知证书编码，获取证书对象 28](#_Toc41665762)

[4.2 获取设备中的所有证书[JNI] 30](#_Toc41665763)

[4.3 选择签名证书 31](#_Toc41665764)

[4.4 选择加密证书[JNI] 32](#_Toc41665765)

[4.5 证书的微缩图 32](#_Toc41665766)

[4.6 双证书的类型 32](#_Toc41665767)

[4.7 证书的公钥[JNI] 33](#_Toc41665768)

[4.8 证书的密钥对[JNI] 33](#_Toc41665769)

[4.9 获取证书的扩展 34](#_Toc41665770)

[4.10 RSA证书 35](#_Toc41665771)

[4.11 SM2证书 35](#_Toc41665772)

[4.12 ECC证书 36](#_Toc41665773)

[4.13 证书在有效期内 36](#_Toc41665774)

[4.14 证书是否存在私钥[JNI] 37](#_Toc41665775)

[4.15 证书对应的设备是否插入[JNI] 37](#_Toc41665776)

[4.16 个人证书库的注册[JNI] 38](#_Toc41665777)

[4.17 使用其他CA的证书[JNI] 38](#_Toc41665778)

[4.18 使用PFX 39](#_Toc41665779)

[4.19 获取证书链 40](#_Toc41665780)

[4.20 验证证书 40](#_Toc41665781)

[4.21 验证过期证书 42](#_Toc41665782)

[4.22 限制获取指定算法的证书[JNI] 42](#_Toc41665783)

[5 证书请求 43](#_Toc41665784)

[5.1 验证证书请求 43](#_Toc41665785)

[5.2 产生证书请求 43](#_Toc41665786)

[5.3 证书请求的主体采用UTF8String编码[JNI] 44](#_Toc41665787)

[6 数字信封 44](#_Toc41665788)

[6.1 构造数字信封 44](#_Toc41665789)

[6.2 不带ContentInfo的数字信封 46](#_Toc41665790)

[6.3 构造SM1的数字信封 46](#_Toc41665791)

[6.4 符合国密标准的SM2数字信封 47](#_Toc41665792)

[6.5 解开数字信封 47](#_Toc41665793)

[6.6 加密机解开数字信封 49](#_Toc41665794)

[6.7 获取数字信封的所有接收者信息[JNI] 49](#_Toc41665795)

[7 SignedData 50](#_Toc41665796)

[7.1 带原文的签名 50](#_Toc41665797)

[7.2 不带原文的签名 51](#_Toc41665798)

[7.3 验证带原文的签名 52](#_Toc41665799)

[7.4 验证不带原文的签名 53](#_Toc41665800)

[7.5 验证不带证书的签名 54](#_Toc41665801)

[7.6 签名是否带原文 55](#_Toc41665802)

[7.7 过期证书的签名 56](#_Toc41665803)

[7.8 证书过期后验证签名 56](#_Toc41665804)

[7.9 不带ContentInfo的SignedData 56](#_Toc41665805)

[7.10 证书的标识 57](#_Toc41665806)

[7.11 SignedData的证书 57](#_Toc41665807)

[7.12 使用加密机进行签名 58](#_Toc41665808)

[7.13 SM2签名 58](#_Toc41665809)

[7.14 符合国密标准的SM2签名 59](#_Toc41665810)

[7.15 不带签名属性 60](#_Toc41665811)

[7.16 产生CAdES-BES 60](#_Toc41665812)

[7.17 验证CAdES-BES 61](#_Toc41665813)

[7.18 CAdES-EPES[JNI] 61](#_Toc41665814)

[7.19 SignedData的签名值 62](#_Toc41665815)

[7.20 产生带时间戳的签名 62](#_Toc41665816)

[7.21 验证带时间戳的签名 63](#_Toc41665817)

[7.22 签名后追加时间戳 64](#_Toc41665818)

[7.23 PKCS#7的签名[JNI] 64](#_Toc41665819)

[7.24 多人签名[JNI] 65](#_Toc41665820)

[7.25 联署签名[JNI] 65](#_Toc41665821)

[7.26 验证联署签名[JNI] 66](#_Toc41665822)

[7.27 P7b文件 66](#_Toc41665823)

[7.28 验证SignedData的时候，同时验证证书 67](#_Toc41665824)

[7.29 CAdES-T 68](#_Toc41665825)

[7.30 CAdES-C[JNI] 68](#_Toc41665826)

[7.31 CAdES-X Long[JNI] 69](#_Toc41665827)

[7.32 CAdES-X Type1[JNI] 69](#_Toc41665828)

[7.33 CAdES-X Type2[JNI] 70](#_Toc41665829)

[7.34 使用ATSv2属性的CAdES-A[JNI] 70](#_Toc41665830)

[7.35 使用ATSv3属性的CAdES-A[JNI] 71](#_Toc41665831)

[7.36 带加密证书的SignedData[JNI] 72](#_Toc41665832)

[7.37 在SignedData中添加作废信息 72](#_Toc41665833)

[7.38 CAdES-B-B 73](#_Toc41665834)

[7.39 CAdES-B-T 74](#_Toc41665835)

[7.40 CAdES-B-LT 74](#_Toc41665836)

[7.41 CAdES-B-LTA 75](#_Toc41665837)

[7.42 CAdES-E-A[JNI] 76](#_Toc41665838)

[8 时间戳 76](#_Toc41665839)

[8.1 获取时间戳 76](#_Toc41665840)

[8.2 验证时间戳Token 78](#_Toc41665841)

[8.3 手工获取时间戳[纯JAVA] 79](#_Toc41665842)

[8.4 获取不带签名证书的时间戳 79](#_Toc41665843)

[9 S/MIME 80](#_Toc41665844)

[9.1 简单的MIME[JNI] 80](#_Toc41665845)

[9.2 MIME头使用中文[JNI] 80](#_Toc41665846)

[9.3 MIME体使用中文[JNI] 81](#_Toc41665847)

[9.4 多部分的MIME[JNI] 81](#_Toc41665848)

[9.5 包含附件的MIME[JNI] 82](#_Toc41665849)

[9.6 获取MIME的内容[JNI] 82](#_Toc41665850)

[9.7 创建签名的S/MIME[JNI] 83](#_Toc41665851)

[9.8 是否签名的S/MIME[JNI] 83](#_Toc41665852)

[9.9 验证签名的S/MIME[JNI] 84](#_Toc41665853)

[9.10 产生加密的S/MIME[JNI] 84](#_Toc41665854)

[9.11 是否加密的S/MIME[JNI] 84](#_Toc41665855)

[9.12 解密加密的S/MIME[JNI] 85](#_Toc41665856)

[10 JOSE 85](#_Toc41665857)

[10.1 创建JWS签名 85](#_Toc41665858)

[10.2 创建不带有效载荷的JWS 86](#_Toc41665859)

[10.3 创建不对有效载荷进行Base64Url编码的JWS 86](#_Toc41665860)

[10.4 创建JWS的MAC 87](#_Toc41665861)

[10.5 验证JWS 88](#_Toc41665862)

[10.6 多人JWS[JNI] 89](#_Toc41665863)

[10.7 创建JWE 89](#_Toc41665864)

[10.8 解密JWE 90](#_Toc41665865)

[10.9 创建签名的JWT 91](#_Toc41665866)

[10.10 验证JWT签名 92](#_Toc41665867)

[10.11 创建加密的JWT 93](#_Toc41665868)

[10.12 解密JWT 94](#_Toc41665869)

[10.13 嵌套的JWT 94](#_Toc41665870)

[11 SignedAndEnvelopedData 95](#_Toc41665871)

[11.1 创建SignedAndEnvelopedData 95](#_Toc41665872)

[11.2 不带ContentInfo的SignedAndEnvelopedData 96](#_Toc41665873)

[11.3 如何验证SignedAndEnvelopedData 96](#_Toc41665874)

[12 电子印章和电子签章 97](#_Toc41665875)

[12.1 创建电子印章 97](#_Toc41665876)

[12.2 验证电子印章 99](#_Toc41665877)

[12.3 产生电子签章 100](#_Toc41665878)

[12.4 验证电子签章 101](#_Toc41665879)

[12.5 追加电子签章签名时间戳 102](#_Toc41665880)

[12.6 验证带签名时间戳的电子签章 102](#_Toc41665881)

[13 杂项 103](#_Toc41665882)

[13.1 日志[JNI] 103](#_Toc41665883)

NETCA PKI JAVA API包括两部分：主体部分和JCE。主体部分有两个版本：一个是作为一个整体的NetcaJCrypto.jar，另一个是拆分版本，包括：netcapki-cmp.jar、netcapki-common.jar、netcapki-core.jar、netcapki-core-jni.jar、netcapki-globalplatform.jar、netcapki-jni.jar、netcapki-jose.jar、netcapki-jose-jni.jar、netcapki-scvp.jar、netcapki-seseal.jar和netcapki-simple.jar。JCE为netcajce.jar，是通过调用主体部分来实现的。

主体部分主要包括两部分：JNI部分和纯JAVA部分。JNI部分是对Netca\_Crypto.dll的封装，使用slf4j来写日志。纯JAVA部分完全使用JAVA来实现，密码运算可以使用JCE或者JNI部分。

# 安装

## 需要的JAVA版本

问题

NETCA Crypto的JAVA包支持JAVA的版本？

解决方案

NetcaJCrypto.jar和JCE部分需要JRE 6及其以上的版本

拆分版本版本需要JRE 8及其以上的版本

讨论

当前版本的NetcaJCrypto.jar仅仅支持JRE 6及其以上的版本。以前的某个版本可以支持JRE1.3或1.4。

拆分版本和NetcaJCrypto.jar基本一致，只是某些地方使用了一些新版本的功能，比如net.netca.pki.Freeable接口继承了AutoCloseable。

## 安装

问题

NETCA Crypto的JAVA包如何安装？

解决方案

先安装NETCA Crypto

把NetcaJCrypto.jar或者拆分版本的jar包拷到CLASSPATH路径之下

把slf4j-api-1.7.x.jar拷到CLASSPATH路径之下，如果需要写日志还需要相关的后端日志包

把NetcaJCrypto.dll（Windows）或libNetcaJCrypto.so（Linux）拷到java.library.path属性的路径之下。

讨论

NetcaJCrypto.jar主要包括两部分：JNI部分和纯JAVA部分，以及在这两部份之上的一个简化封装。如果你使用JNI部分，还需要JNI相关的库和日志库；如果你仅仅使用纯JAVA部分，则一个NetcaJCrypto.jar或者拆分版本相应的jar包就可以了，可以把它放到JAVA能调用到的地方就行了；如果你两者都要使用和使用JNI部分的方式安装相同

JNI部分的相关安装主要步骤：

1. NetcaJCrypto.jar放在JAVA能调用到的地方，通常是CLASSPATH路径之下。不同的应用环境不一定会相同
2. JNI库，在Windows下是NetcaJCrypto.dll，在Linux下是libNetcaJCrypto.so，需要放在JAVA能调用到的地方，通常是java.library.path属性的路径之下。一般来说，Windows可以放在System32目录下，Linux可以放在LD\_LIBRARY\_PATH的路径之下。不同的应用环境不一定会相同。也可以使用net.netca.pki.jnipath系统属性直接指定JNI库的全路径名(包括绝对路径的目录部分和文件名)
3. JNI库依赖的库，也就是NETCA Crypto。安装NETCA Crypto包就行了
4. 日志库使用slf4j。slf4j包括两部分：slf4j-api-1.7.x.jar和相关的后端日志适配包（及其日志包）。把slf4j-api-1.7.x.jar放在JAVA能调用到的地方，通常是CLASSPATH路径之下。如果你不需要真正写日志，这样就可以了。如果你需要写日志，还需要对应的日志包和日志适配包放在JAVA能调用到的地方。不同的应用环境不一定会相同。参见1.4

其实和一般的使用JNI的包的安装是相同的。

参见1.3

## 在tomcat下安装

问题

如何在tomcat下安装NETCA Crypto的JAVA包？

解决方案

先安装NETCA Crypto

把NetcaJCrypto.jar拷到${CATALINA\_HOME}/lib路径之下

把slf4j-api-1.7.x.jar拷到${CATALINA\_HOME}/lib路径之下，如果需要写日志还需要相关的后端日志包

把NetcaJCrypto.dll（Windows）或libNetcaJCrypto.so（Linux）拷到java.library.path属性的路径之下。

讨论

如果仅仅使用纯JAVA部分，仅仅需要把NetcaJCrypto.jar拷到${CATALINA\_HOME}/lib目录下或者应用的lib目录下即可。

如果使用JNI部分，NetcaJCrypto.jar是不能放在应用的lib目录下的，因为这样在多次装载NetcaJCrypto.jar的时候会导致JNI不能调用。NetcaJCrypto.jar是需要放在${CATALINA\_HOME}/lib目录下，相关的日志相关库也应该放在${CATALINA\_HOME}/lib目录下。其他的参见1.2。也可以使用net.netca.pki.jnipath系统属性直接指定JNI库的全路径名，这样jar包可以放在任何能访问到的位置。

## 如何使用log4j来写日志

问题

如何使用log4j来为NETCA Crypto的JAVA包写日志？

解决方案

1. 把slf4j-api-1.7.x.jar、slf4j-log4j12-1.7.x.jar和log4j.jar放到CLASSPATH路径之下
2. 在log4j的配置文件log4j.properties进行相关的设置

讨论

纯JAVA部分是没有日志的。

JNI部分是使用slf4j来写日志的。要使用log4j，需要四个文件：slf4j-api-1.7.x.jar这个是slf4j基础包，它本身只是个门面，不做任何实质的日志记录；slf4j-log4j12-1.7.x.jar这个是一个适配包，是告诉slf4j后端使用slf4j来写日志；log4j.jar和log4j.properties是log4j本身，用来实际进行写日志。slf4j-api-1.7.x.jar、slf4j-log4j12-1.7.x.jar和log4j.jar放到放在JAVA能调用到的地方，通常是CLASSPATH路径之下，如果是Web应用参见1.3。log4j.properties根据log4j的进行放置。

## 如何禁止日志

问题

如何设置使得NETCA Crypto的JAVA包不写日志？

解决方案

设置net.netca.pki.log.enabled系统属性为false

讨论

纯JAVA部分是没有日志的。

JNI部分是使用slf4j写日志的。5.3版本后可以使用net.netca.pki.log.enabled系统属性来控制是否启动日志。如果在Android中默认不启用日志，其他平台默认启用日志

# 设备

## 获取内置的软件设备[JNI]

问题

如何获取内置的软件设备对象？

解决方案

net.netca.pki.Device.getPseudoDevice()

讨论

内置的软件设备提供了可以软件实现的基本的加密算法的实现，通常在不需要使用特定硬件设备的私钥的情况下，可以方便使用。由于它的实现是内置在NETCA\_CRYPTO.DLL中的，因此，无需像其它的加密设备那样需要安装和配置，可以直接使用。使用了加密设备的对象进行某些加密运算，如果加密设备本身不支持，而且可以用软件实现的话，是可以转换到内置的软件设备来进行的，这样可以简化设备DLL的开发。默认情况下，内置的软件设备里的密钥对是在内存保存的，释放后会从内存中删掉，这个和其他的加密设备是不同的。

## 获取当前连接的设备[JNI]

问题

如何获取当前连接的设备？

解决方案

使用net.netca.pki.DeviceSet类

1. new DeviceSet(type,flag)获取设备集对象
2. 使用DeviceSet对象的count()方法获取设备集中设备数量
3. 循环调用DeviceSet对象的get(index)来获取设备对象
4. 使用DeviceSet对象的free()来释放

讨论

DeviceSet有好几个构造方法，可以根据不同的情况来获取设备集对象的，DeviceSet对象使用后需要使用free方法来释放。DeviceSet对象是不能获取到内置的软件设备。

## 获取指定的设备[JNI]

问题

已知一个加密设备，如何才能获取其设备对象？

解决方案

加密设备由其设备类型和序列号来唯一标识。使用net.netca.pki.Device.findDevice(type,String sn,int flag)方法可以根据设备类型和设备序列号来获取指定的设备对象，如果没有该设备返回null。

讨论

根据设备类型和设备序列号可以确定一个设备。但是，根据net.netca.pki.Device.findDevice方法获取到的设备对象，再用getSerialNumber方法获取序列号，对某些设备来说得到的序列号可能和输入的序列号不一样。这是因为findDevice方法对某些设备来说是会忽略序列号或者序列号的一部分。比如，内置的软件设备就是前者，普华加密机就是后者。getSerialNumber方法获取到的序列号是规范化的序列号，是唯一的。建议使用规范化的序列号来调用findDevice方法。

另外，要注意的是，某些加密设备获取到设备对象后，并不能真正使用，必须先调用verifyPwd方法成功验证用户密码才能使用。这类加密设备的verifyPwd方法方法返回true，典型的例子是得安的加密机。

## 获取指定的密钥对[JNI]

问题

已知一个加密设备里的密钥对，如何才能获取其密钥对对象？

解决方案

net.netca.pki.KeyPair.findKeyPair(deviceType,sn,flag,pwd,keypairType,label)

讨论

根据设备类型和设备序列号可以确定一个设备。在同一设备中，密钥对类型和密钥对标签可以唯一确定一对密钥对。因此，先使用net.netca.pki.Device.findDevice先获取设备对象，再使用设备对象的findKeyPair方法可以获取到密钥对对象。

net.netca.pki.KeyPair.findKeyPair是把上面两者结合起来，并对同一物理设备映射成多个逻辑设备的情况进行了处理。也就是如果是一个设备映射为多少物理设备的情况下，序列号可以仅仅包含前缀部分，而不用包含完整的序列号。例如：对于普华加密机来说，net.netca.pki.KeyPair.findKeyPair(Device.SJY03B\_DEVICE, ”SJJ1011”,0,null, KeyPair.ENCRYPT,”1”)和net.netca.pki.KeyPair.findKeyPair(Device.SJY03B\_DEVICE, ”SJJ1011::1”,0,null, KeyPair.ENCRYPT,”1”)的效果是一样的。

也可以直接使用net.netca.pki.getKeyPair(str,flag,pwd)方法来根据密钥对信息字符串表示来获取密钥对对象。

## 获取CSP实现的密钥对[JNI]

问题

如何获取CSP实现的密钥对？

解决方案

使用net.netca.pki.Certificate对象的getKeyPair (Certificate.SEARCH\_KEYPAIR\_FLAG\_CURRENT\_USER, purpose,null)

讨论

NETCA CRYPTO支持两种类型的密钥对对象：一种是使用设备接口实现的密钥对，另外一种是CSP实现的密钥对。2.4的方法可以获取到第一种类型的密钥对。对于CSP实现的密钥对，当然仅仅支持RSA类型的CSP，不支持其他类型的CSP，也不支持CNG。CSP实现的密钥对通常只能从个人证书库中，根据证书来获取。getKeyPair方法的flag参数设置为SEARCH\_KEYPAIR\_FLAG\_CURRENT\_USER则是从当前用户的个人证书库里搜索，设置为SEARCH\_KEYPAIR\_FLAG\_LOCAL\_MACHINE 则是从本机的个人证书库里搜索。如果cert本身就是从个人证书库里获取的证书，flag参数可以设置为0。

另外，orcale有可以调用CSP的JCE，也可以考虑使用纯JAVA部分来使用。

## 从密钥对获取设备对象[JNI]

问题

已知密钥对对象，怎么获取设备对象？

解决方案

getDevice方法

讨论

getDevice方法可以获取密钥对相关联的设备对象。对于使用设备接口实现的密钥对来说，通常是成功的。对于CSP实现的密钥对来说，总是返回null

## Key的PIN码是否锁死[JNI]

问题

如何知道USB Key的PIN码是否已经锁死？

解决方案

使用verifyPwd (type, pwd)，如果返回false。接着调用getPwdRetryNumber(type,true)，如果返回0则表示USB Key已经锁死。

讨论

如果PIN码的重试次数为0，则表示PIN码已经锁死。在PIN码已经锁死的情况下，即使再次输入正确的PIN码也不能成功验证PIN码了。对于用户PIN来说，如果PIN码已经锁死，可以使用unlockPwd方法来解锁用户PIN。

在使用verifyPwd、changePwd和unlockPwd方法的时候，如果返回值为false，接着调用getPwdRetryNumber(type,true)获取密码重试次数如果返回0则表示USB Key已经锁死。

不是所有的设备都支持这样使用的。有些设备的用户密码是不会锁死的。

## 获取PIN码重试次数[JNI]

问题

验证PIN码失败，如何获取PIN码重试次数？

解决方案

使用verifyPwd (type, pwd)，如果返回false，表示验证PIN码错误。这时，可以使用getPwdRetryNumber(type,true)来获取PIN码重试次数。如果retryNum为0表示PIN码已经锁死，如果retryNum为-1表示不支持获取PIN码重试次数。

讨论

通常在使用verifyPwd、changePwd和unlockPwd方法的时候，如果返回值为false，可以接着调用getPwdRetryNumber(type,true)获取密码重试次数。这些函数如果不是密码错误或者密码锁死，会抛出异常的。某些时候，不验证密码的也想先获取密码重试次数，可以使用getPwdRetryNumber(type,false)。

## 进程密码缓存[JNI]

问题

在同一进程中，验证用户密码成功后，如何使得以后使用私钥都不需要再次验证用户密码？

解决方案

连接设备的时候，flag参数的第1、2位的值（最末位从算起）为Device.DEVICE\_FLAG\_CACHE\_PIN\_IN\_PROCESS即可。比如：new DeviceSet(type, Device.DEVICE\_FLAG\_CACHE\_PIN\_IN\_PROCESS)

讨论

密码缓存是在NETCA CRYPTO层做的，只针对用户密码。当为.DEVICE\_FLAG\_CACHE\_PIN\_IN\_PROCESS时，会把验证成功的用户密码保存在内存中，当同一进程获取不同进程调用某些操作（比如需要使用私钥签名）发现权限不足，需要再次验证用户密码时，返回NETCA\_PKI\_ACCESSDENIED，如果检查发现可以缓存密码，会自动重新验证用户密码，再次重试该操作。只会重试一次，不会不断的重试。

是否可以缓存密码主要根据USB Key的插拔信息和密码缓存的时间来决定。当前USB Key的插拔信息是保存在共享内存映射文件里的，这个信息通常是由KeyX来维护的，NETCA CRYPTO只读取里面的信息进行判断。

要注意的是只有密码运算（包括证书的处理）才会使用缓存的密码，对加密设备里的持久数据的操作是不会自动使用缓存的密码。

## 持久数据的权限[JNI]

问题

在加密设备中，创建持久数据的时候，如果指定其访问权限？

解决方案

createData方法的mode参数表示访问权限，mode采用Hex编码。其含义为：从低到高，第1个字节表示长度，第2字节表示没有校验密码时的权限，第3字节表示校验用户密码时的权限，第4字节表示校验管理员密码时的权限，后面每两个字节为一组，其第1个字节为应用PIN码的编号，第2个字节为校验该PIN码成功时的权限。每个字节从低到高，第0位为读权限，第1位为写权限，第2位为删除权限。任何人可读可写的权限为：”04070707”，校验用户密码才可读可写的权限为：”04000700”，任何人可读校验用户密码可读可写的权限为：”04010700”， 任何人可读校验管理员密码可读可写的权限为：”04010107”。

讨论

持久数据的访问权限和USB Key本身的特性有关。不是所有的权限类型都可以创建，有时候实现会把指定的权限转换为USB Key的一种类似但不等价的权限，这是要注意的问题。使用PKCS#11的实现，只能创建任何人可读可写、校验用户密码才可读可写，不能创建任何人可读校验用户密码可读可写和任何人可读校验管理员密码可读可写。有些USB Key没有对应的删除权限的，可能会直接忽略删除。

# 基本的加密运算

## 获取随机数

问题

如何获取随机数？

解决方案

使用net.netca.pki.Device 对象的generateRandom (len)方法

讨论

[JNI]获取随机数，直接使用net.netca.pki.Device 对象的generateRandom方法即可。generateRandom方法会根据平台的不同使用平台的随机数来做种子，如果使用的设备支持产生随机数的话，还会使用设备产生的随机数作为种子。默认情况下，不管设备是否支持产生随机数或者产生随机数是否失败，通常generateRandom方法都不会失败。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.SecureRandomGenerator接口的，generate方法可以产生随机数。有两个SecureRandomGenerator接口的实现，net.netca.pki.encoding.asn1.pki.JCESecureRandomGenerator是直接使用JCE来实现；net.netca.pki.encoding.asn1.pki.NetcaSecureRandomGenerator是使用JNI来实现。

[简化封装] 使用net.netca.pki.global.Pki类的generateRandom方法

## Hash运算

问题

如何进行Hash运算？

解决方案

Hash运算是使用net.netca.pki.Hash类来实现的

通常进行Hash运算的步骤如下：

1. new Hash(algo)创建Hash对象
2. 多次调用update方法，输入要进行Hash的数据，直到所有的数据输入完毕
3. 调用doFinal 方法，得到Hash值
4. 调用free方法释放Hash对象

讨论

[JNI]通常new Hash(algo)方法通常会使用内置软件设备来进行Hash运算。如果加密设备本身支持Hash算法，且其速度比软件实现的快或者与之相当，则也可以使用Device对象的newHash方法来创建Hash对象。如果加密设备本身不支持Hash算法，使用newHash方法来创建Hash对象其实也是使用内置软件设备来进行Hash运算的（假设内置的软件设备支持的话）。

如果数据能够一次获取到，可以直接使用Hash.computeHash方法。

[纯JAVA] net.netca.pki.encoding.asn1.pki. Hashable接口可以实现Hash运算。步骤如下：

1. init初始化Hash运算
2. 多次调用update方法，输入要进行Hash的数据，直到所有的数据输入完毕
3. digest方法得到Hash值

如果数据能够一次获取到，可以直接使用hash方法。

有两个Hashable接口的实现，net.netca.pki.encoding.asn1.pki.JCEHasher是直接使用JCE来实现，无论是否存在支持SM3的JCE，这个类总是支持SM3的；net.netca.pki.encoding.asn1.pki.NetcaHasher是使用JNI来实现。

[简化封装] 使用net.netca.pki.global.IHash接口可以实现Hash运算。步骤如下：

1. 使用net.netca.pki.global.Pki对象的getHashObject方法来获取IHash接口
2. 多次调用IHash接口的update方法，输入要进行Hash的数据，直到所有的数据输入完毕
3. 调用IHash接口的doFinal方法得到Hash值

如果数据能够一次获取到，可以直接使用net.netca.pki.global.Pki对象的hash方法。

Hash算法的对应长度，如下表所示：

|  |  |
| --- | --- |
| Hash算法 | Hash值长度（位数） |
| MD5 | 128 |
| SHA-1 | 160 |
| SHA-224 | 224 |
| SHA-256 | 256 |
| SHA-384 | 384 |
| SHA-512 | 512 |
| SHA-512/224 | 224 |
| SHA-512/256 | 256 |
| SHA3-224 | 224 |
| SHA3-256 | 256 |
| SHA3-384 | 384 |
| SHA3-512 | 512 |
| SM3 | 256 |

MD5和SHA-1的冲突已经找到，不再安全，不要用于完整性和签名的场合

## 复制Hash对象[JNI]

问题

如何复制Hash对象？

解决方案

使用Hash对象的clone方法

讨论

大部分的对象复制采用的是增加引用计数的方式，Hash对象的复制不是那样的，是直接复制内部的状态。这样的话，原先的对象和复制后的对象是独立的。

## Mac运算

问题

如何进行Mac运算？

解决方案

Hash运算是使用net.netca.pki.Mac类来实现的

通常进行Mac运算的步骤如下：

1. 使用new Mac(algo, key)创建Mac对象
2. 多次调用update方法，输入要计算Mac的数据，直到所有的数据输入完毕
3. 调用doFinal方法，获得Mac值
4. 调用free方法来释放Mac对象

讨论

[JNI]通常Mac的构造函数会使用内置软件设备来进行Mac运算。如果加密设备本身支持Mac算法，且其速度比软件实现的快或者与之相当，则也可以使用Device对象的newMac方法来创建Mac对象。如果加密设备本身不支持Mac算法，使用newMac方法来创建Mac对象其实也是使用内置软件设备来进行Mac运算的（假设内置的软件设备支持的话）。

[纯JAVA] net.netca.pki.encoding.asn1.pki.IHmac接口可以实现HMAC运算。但仅支持一步运算，使用hmac方法。net.netca.pki.encoding.asn1.pki.SoftwareHmac实现了IHmac接口，使用net.netca.pki.encoding.asn1.pki.Hashable来实现HMAC，当前支持HMAC-MD5、HMAC-SHA1、HMAC-SHA224、HMAC-SHA256、HMAC-SHA384、HMAC-SHA512、HMAC-SHA512/224、HMAC-SHA512/256、HMAC-SHA3-224、HMAC-SHA3-256、HMAC-SHA3-384、HMAC-SHA3-512和HMAC-SM3。

## 产生随机的对称密钥

问题

如何产生随机的对称密钥？

解决方案

直接使用产生随机数，对于DES和3DES还需要设置校验位

讨论

通常产生随机的对称密钥就是产生对应长度的随机数而已。对应某些算法，需要对密钥进行规范化以及去除弱密钥。除DES和3DES外，其他的算法可以直接产生随机数就可以了（参见3.1）。对于DES和3DES，还需要设置校验位。如果不需要对密钥进行完整性检查，很多实现是可以直接忽略DES和3DES的校验位的。

## 序列密码的对称加解密[JNI]

问题

如何使用序列密码进行加解密？

解决方案

序列密码加解密使用net.netca.pki.Cipher类来处理

使用序列密码进行加解密的步骤如下：

1. 使用new Cipher创建一个Cipher对象。
2. 使用setKey方法设置对称密钥
3. 使用init方法初始化加密或者解密
4. 多次调用update方法进行加密或解密，同时可以得到加密或解密的结果
5. 使用doFinal得到加密或解密最后部分的结果。
6. 使用free方法来释放Cipher对象

讨论

如果内置的软件设备已经实现了该序列密码则Cipher的构造函数通常使用的是内置的软件设备的实现。如果加密设备本身支持序列密码算法，且其速度比软件实现的快或者与之相当，则也可以使用Device类的newCipher方法来创建Cipher对象。如果加密设备本身不支持序列密码算法，使用newCipher方法来创建加解密对象其实也是使用内置软件设备来进行加解密运算的（假设内置的软件设备支持的话）。

序列密码一次可以加密一位或者一个字节，因此是不需要填充的。

## ECB模式的对称加解密[JNI]

问题

如何使用ECB模式进行加解密？

解决方案

使用ECB模式进行加解密使用net.netca.pki.Cipher类来处理

使用ECB模式进行加解密的步骤如下：

1. 使用new Cipher创建一个Cipher对象。
2. 使用setKey方法设置对称密钥
3. 使用setPadding方法设置填充方式
4. 使用init方法初始化加密或者解密
5. 多次调用update方法进行加密或解密，同时可以得到加密或解密的结果
6. 使用doFinal得到加密或解密最后部分的结果。
7. 使用free方法来释放Cipher对象

讨论

NETCA CRYPTO中对称加密算法是包含模式的。如果内置的软件设备已经实现了该分组密码的ECB模式则Cipher的构造函数通常使用的是内置的软件设备的实现。如果加密设备本身支持分组密码的ECB模式，且其速度比软件实现的快或者与之相当，则也可以使用Device类的newCipher方法来创建Cipher对象。如果加密设备本身不支持分组密码的ECB模式，使用newCipher方法来创建加解密对象其实也是使用内置软件设备来进行加解密运算的（假设内置的软件设备支持的话）。

分组密码一个基本操作只能对一个分组进行处理，因此需要进行填充的，除非数据刚好是分组长度的整数倍。ECB模式的加解密安全性不高，通常不建议使用。

RC2算法需要有效密钥长度，其ECB的加解密，则还需要在init之前，使用setEffectiveKeyBits方法来设置密钥的有效密钥长度。

RC2算法安全性不高，通常不建议使用。

分组加密算法的密钥长度如下表所示：

|  |  |
| --- | --- |
| 分组加密算法 | 密钥长度（位数） |
| DES | 64（存在校验位，实际使用56位） |
| 3DES | 有两种形式，使用两个密钥的3DES和使用三个密钥的3DES  128/192（存在校验位，实际使用112/168） |
| RC2 | 通常40-128 |
| AES | 128、192、256 |
| SM1 | 128 |
| SM4 | 128 |
| SSF33 | 128 |

分组加密算法的分组长度如下表所示：

|  |  |
| --- | --- |
| 分组加密算法 | 分组长度（位数） |
| DES | 64 |
| 3DES | 64 |
| RC2 | 64 |
| AES | 128 |
| SM1 | 128 |
| SM4 | 128 |
| SSF33 | 128 |

DES和RC2不安全，不要使用。3DES的安全性也不高，尽量避免使用。

## CBC模式的对称加解密

问题

如何使用CBC模式进行加解密？

解决方案

使用CBC模式进行加解密使用net.netca.pki.Cipher类来处理

使用CBC模式进行加解密的步骤如下：

1. 使用new Cipher创建一个Cipher对象。
2. 使用setKey方法设置对称密钥
3. 使用setIV方法设置IV值
4. 使用setPadding方法设置填充方式
5. 使用init方法初始化加密或者解密
6. 多次调用update方法进行加密或解密，同时可以得到加密或解密的结果
7. 使用doFinal得到加密或解密最后部分的结果。
8. 使用free方法来释放Cipher对象

讨论

[JNI]NETCA CRYPTO中对称加密算法是包含模式的。如果内置的软件设备已经实现了该分组密码的CBC模式则Cipher的构造函数通常使用的是内置的软件设备的实现。如果加密设备本身支持分组密码的CBC模式，且其速度比软件实现的快或者与之相当，则也可以使用Device类的newCipher方法来创建加解密对象。如果加密设备本身不支持分组密码的CBC模式，但是支持ECB模式，可以配置使用ECB模式来实现CBC模式，不过这时通常会很慢。如果连ECB模式也不支持，使用newCipher方法来创建加解密对象其实也是使用内置软件设备来进行加解密运算的（假设内置的软件设备支持的话）。

分组密码一个基本操作只能对一个分组进行处理，因此需要进行填充的，除非数据刚好是分组长度的整数倍。

创建加解密对象时，只需要输入IV值，而不需要IV值的长度，因为IV的长度是等于分组的长度（参见3.7），是由加密算法决定的。IV不需要保密，但不要重复。

RC2算法需要有效密钥长度，其CBC的加解密，则还需要在init之前，使用setEffectiveKeyBits方法来设置密钥的有效密钥长度。

RC2算法安全性不高，通常不建议使用。

[纯JAVA]可以使用net.netca.pki.encoding.asn1.pki.SymEncrypter接口的cipher方法来实现CBC加解密。SymEncrypter接口有两个实现：net.netca.pki.encoding.asn1.pki.JCESymEncrypter类是JCE的实现；net.netca.pki.encoding.asn1.NetcaSymEncrypter类是JNI的实现

## CTR模式的对称加解密[JNI]

问题

如何使用CTR模式进行加解密？

解决方案

使用CTR模式进行加解密使用net.netca.pki.Cipher类来处理

使用CTR模式进行加解密的步骤如下：

1. 使用new Cipher创建一个Cipher对象。
2. 使用setKey方法设置对称密钥
3. 使用setCtrParam方法设置CTR的参数
4. 使用init方法初始化加密或者解密
5. 多次调用update方法进行加密或解密，同时可以得到加密或解密的结果
6. 使用doFinal得到加密或解密最后部分的结果。
7. 使用free方法来释放Cipher对象

讨论

NETCA CRYPTO中对称加密算法是包含模式的。如果内置的软件设备已经实现了该分组密码的CTR模式则Cipher的构造函数通常使用的是内置的软件设备的实现。如果加密设备本身支持分组密码的CTR模式，且其速度比软件实现的快或者与之相当，则也可以使用Device类的newCipher方法来创建Cipher对象。如果加密设备本身不支持分组密码的CTR模式，但是支持ECB模式，可以配置使用ECB模式来实现CTR模式，不过这时速度和ECB模式基本一样。如果连ECB模式也不支持，使用newCipher方法来创建Cipher对象其实也是使用内置软件设备来进行加解密运算的（假设内置的软件设备支持的话）。

CTR模式是使用一个增量序列，对这个增量序列进行加密，然后与明文进行异或得到密文。由于可以丢弃最后分组不要的部分，因此不需要进行填充。setCtrParam方法的iv只取一个分组的长度（参见3.7），countBits表示计数器的位数，也就是iv只有countBits来产生增1的增量序列。如果位数取得太小，则处理很少的数据就可以让计数器重复，这时加解密会失败。CTR模式能处理的数据长度为分组长度\*2countBits。CTR的IV值绝对不能重复。

## CCM模式的对称加解密[JNI]

问题

如何使用CCM模式进行加解密？

解决方案

使用CCM模式进行加解密使用net.netca.pki.Cipher类来处理

使用CCM模式进行加解密的步骤如下：

1. 使用new Cipher创建一个Cipher对象。
2. 使用setKey方法设置对称密钥
3. 使用setCcmParam方法设置CCM的参数
4. 使用init方法初始化加密或者解密
5. 使用cipher方法进行加密或解密。这个方法只能一次调用不能多次调用。
6. 使用free方法来释放Cipher对象

讨论

CCM模式需要提前知道要加密明文的数据的长度的，因此要保证setCcmParam方法的pLen和加密时cipher的输入一致。下层的设备接口是使用这种一步完成的方法的，因此，直接使用cipher方法为宜。

CCM是带认证的加密模式，本身就带有Mac，无需再另外使用Mac来保证完整性。setCcmParam方法的tLen只能是4、6、8、10、12、14和16，nonce长度在7到13之间，包括7和13。CCM模式不需要进行填充。

## GCM模式的对称加解密[JNI]

问题

如何使用GCM模式进行加解密？

解决方案

使用GCM模式进行加解密使用net.netca.pki.Cipher类来处理

使用GCM模式进行加解密的步骤如下：

1. 使用new Cipher创建一个Cipher对象。
2. 使用setKey方法设置对称密钥
3. 使用setGcmParam方法设置GCM的参数
4. 使用init方法初始化加密或者解密
5. 使用cipher方法进行加密或解密。这个方法只能一次调用不能多次调用。

使用free方法来释放Cipher对象

讨论

GCM模式下层的设备接口是使用这种一步完成的方法的，因此，直接使用cipher方法为宜。

GCM是带认证的加密模式，本身就带有Mac，无需再另外使用Mac来保证完整性。setGcmParam方法的tLen通常只能是12、13、14、15和16，某些情况下还可以为4和8。GCM模式不需要进行填充。GCM的IV值的长度范围为1到264-1位，推荐使用96位，IV值必须唯一；明文长度小于等于239-256位；附加的认证数据aad长度小于等于264-1位。

## XTS模式的对称加解密[JNI]

问题

如何使用XTS模式进行加解密？

解决方案

使用XTS模式进行加解密使用net.netca.pki.Cipher类来处理

使用XTS模式进行加解密的步骤如下：

1. 使用new Cipher创建一个Cipher对象。
2. 使用setKey方法设置对称密钥，主要这里是需要两个对称密钥的
3. 使用setTweak方法设置Tweak值
4. 使用init方法初始化加密或者解密
5. 使用cipher方法进行加密或解密。这个方法只能一次调用不能多次调用。

使用free方法来释放Cipher对象

讨论

当前只支持AES-XTS，而AES-XTS只支持128位和256位的AES，因为XTS使用两个密钥的，因此，128位的AES-XTS的密钥长度为32，256位的AES-XTS的密钥长度为64

XTS模式下层的设备接口是使用这种一步完成的方法的，因此，直接使用cipher方法为宜。

XTS模式不需要进行填充。

## SM1加解密[JNI]

问题

如何使用SM1或者SSF33进行加解密？

解决方案

使用SM1或者SSF33进行加解密和其他的算法没什么不同，只是要使用Device类的newCipher方法来创建Cipher对象，其中，使用的Device对象必须支持对应的算法。

讨论

SM1和SSF33这些算法没有公开，没有软件实现，只能用硬件加密设备来实现。因此需要先获取支持的硬件设备的Device对象再用newCipher方法创建Cipher对象。同时，由于一般的加密设备都不支持CTR的模式，因此需要配置加密设备的ECB2CTR为1，使之可以使用ECB模式来实现CTR模式。

## 产生密钥对[JNI]

问题

如何产生密钥对？

解决方案

产生密钥对需要net.netca.pki.Devcie类。

产生RSA密钥对： generateRSAKeyPair(keyBits,type,label)

产生ECC密钥对：generateECCKeyPair(curve,type,label)

讨论

不是所有的加密设备都支持产生密钥对。

产生ECC密钥对，需要指定椭圆曲线号，如果curve为Device.ECC\_CURVE\_SM2则产生的是SM2的密钥对。

本条产生的密钥对都是在设备中保存的密钥对。如果需要产生临时的密钥对，参见3.15。对于内置的软件设备密钥对是在内存里的，不能跨进程使用。

## 产生临时密钥对[JNI]

问题

如何产生临时密钥对？

解决方案

产生临时的密钥对需要net.netca.pki.Devcie类。

产生临时的RSA密钥对：generateTempRSAKeyPair(keyBits)

产生临时的ECC密钥对：generateTempECCKeyPair (curve)

讨论

不是所有的加密设备都支持产生临时密钥对。

产生临时的ECC密钥对，需要指定椭圆曲线号，如果curve为Device.ECC\_CURVE\_SM2则产生的是SM2的密钥对。

临时密钥对释放后，将不存在。临时密钥对没有标识，不能从设备中获取。

## 获取公钥[JNI]

问题

如何获取公钥？

解决方案

获取公钥主要有以下方法：

1. 从证书里获取，使用net.netca.pki.Certificate类的getPublicKey方法
2. 从密钥对里获取，使用net.netca.pki.KeyPair类的getPublicKey方法
3. 根据编码获取，使用net.netca.pki.Device类的importSubjectPublicKeyInfo方法

讨论

公钥对象仅仅是内存的结构，不会持久化到设备中的。

从证书里获取会受到证书密钥用法和有效期的限制，参见4.6。

Device类的importSubjectPublicKeyInfo方法需要的编码是SubjectPublicKeyInfo的DER编码。如果需要NETCA PKI API内部使用的编码，则需要使用importPublicKey方法。

此外，还可以从证书请求中获取公钥（使用net.netca.pki.CertReq 类的getPublicKey方法），验证证书路径后获取公钥（使用net.netca.pki.CertVerify类的getPublicKey方法）。

## RSA加密

问题

如何使用RSA进行加密？

解决方案

使用net.netca.pki.PublicKey类的encrypt(KeyPair.RSA\_PKCS1\_V1\_5\_ENC,clear)方法

讨论

[JNI]通常使用RSA加密是使用PKCS#1 V1.5方式的加密，它只能加密不大于（模长-11字节）长度的明文。由于加密过程的填充使用了随机数，因此，每次加密的结果是不同的。加密得到的密文的长度为模长，数据采用大端字节序。

此外，还可以使用OAEP方式的加密这时需要使用rsaOaepEncrypt方法，RSAOAEPParam类的其中包含Hash算法，MGF算法和Label。OAEP加密只能加密不大于（模长-2\*Hash长度-2字节）的明文。同样地，每次加密的结果也是不同的，密文的长度也为模长，数据采用大端字节序。

注：RSA的密钥长度为模长。

[纯JAVA] net.netca.pki.encoding.asn1.pki.PublicKeyEncrypter接口的encrypt方法可以进行RSA加密。它有两个实现：net.netca.pki.encoding.asn1.pki.JCEPublicKeyEncrypter是JCE的实现，net.netca.pki.encoding.asn1.pki.NetcaPublicKeyEncrypter是JNI的实现。加密得到的密文为大端字节序。

[简化封装] net.netca.pki.global.X509Certificate类的encrypt方法可以进行RSA加密，只要证书是RSA证书，当前的实现都不支持RSA OAEP加密。加密得到的密文为大端字节序。

## RSA解密

问题

如何使用RSA进行解密？

解决方案

使用net.netca.pki.KeyPair类的decrypt (KeyPair.RSA\_PKCS1\_V1\_5\_ENC, cipher)方法

讨论

[JNI]通常使用RSA加密是使用PKCS#1 V1.5方式的加密。此外，还可以使用OAEP方式的加密，这时需要使用rsaOaepDecrypt方法，其中包含Hash算法，MGF算法和Label。

RSA的解密输入的密文的长度必须等于模长，数据必须采用大端字节序

[纯JAVA] net.netca.pki.encoding.asn1.pki.PrivateKeyDecrypter接口的decrypt方法可以进行RSA解密。它有两个实现：net.netca.pki.encoding.asn1.pki.JCEPrivateKeyDecrypter是JCE的实现，net.netca.pki.encoding.asn1.pki.NetcaPrivateKeyDecrypter是JNI的实现。RSA的解密输入的密文必须采用大端字节序

[简化封装] net.netca.pki.global.X509Certificate类的decrypt方法可以进行RSA解密，只要证书是RSA证书，当前的实现都不支持RSA OAEP解密。RSA的解密输入的密文必须采用大端字节序

## SM2加密

问题

如何使用SM2进行加密？

解决方案

1、使用net.netca.pki.PublicKey类的encrypt(KeyPair.SM2\_ENC,clear)方法

2、使用net.netca.pki.Util类的encodeSM2Cipher(cipher)方法转换为标准的DER编码，其中cipher是第一步的结果

讨论

[JNI]SM2加密的明文长度必须大于0，没有其他的限制，不过加密设备的实现可能有自己的限制 。对于同样的明文，SM2加密每次得到的结果是不同的。SM2加密得到的密文是很容易看出明文的长度的。

SM2公钥通常根据证书来获取。

PublicKey类encrypt方法得到的密文的格式为C1||C2||C3。C1采用为压缩编码，也就是04||X||Y，密文长度为明文长度+97。根据《GM/T 0009-2012 SM2密码使用规范》，SM2加密的结果应该是SM2Cipher的DER编码结构，因此需要使用Util类的encodeSM2Cipher方法来进行转换。

[纯JAVA] net.netca.pki.encoding.asn1.pki.PublicKeyEncrypter接口的encrypt方法可以进行SM2加密。它有两个实现：net.netca.pki.encoding.asn1.pki.JCEPublicKeyEncrypter是JCE的实现(无论JCE是否支持SM2加密都可以进行SM2加密)，net.netca.pki.encoding.asn1.pki.NetcaPublicKeyEncrypter是JNI的实现。加密得到的密文为SM2Cipher的DER编码结构。

[简化封装] net.netca.pki.global.X509Certificate类的encrypt方法可以进行RSA加密，只要证书是SM2证书。加密得到的密文为SM2Cipher的DER编码结构。

## SM2解密

问题

如何使用SM2进行解密？

解决方案

1. 使用net.netca.pki.Util类的decodeSM2Cipher方法对密文进行转换
2. 使用net.netca.pki.KeyPair类的decrypt (KeyPair. SM2\_ENC, cipher)进行解密，其中cipher为第一步的结果

讨论

[JNI]SM2加密的明文长度必须大于0，没有其他的限制，不过加密设备的实现可能有自己的限制。

SM2解密需要先把DER结构的SM2Cipher进行转换，然后再进行解密处理。参见3.19

[纯JAVA] net.netca.pki.encoding.asn1.pki.PrivateKeyDecrypter接口的decrypt方法可以进行RSA解密。它有两个实现：net.netca.pki.encoding.asn1.pki.JCEPrivateKeyDecrypter是JCE的实现(需要JCE支持才能实现SM2解密)，net.netca.pki.encoding.asn1.pki.NetcaPrivateKeyDecrypter是JNI的实现。SM2的解密输入的密文为SM2Cipher的DER编码结构

[简化封装] net.netca.pki.global.X509Certificate类的decrypt方法可以进行SM2解密，只要证书是SM2证书。SM2的解密输入的密文为SM2Cipher的DER编码结构

## RSA签名

问题

如何使用RSA进行签名？

解决方案

RSA签名使用net.netca.pki.Signature类

使用RSA进行签名的步骤如下：

1. 调用new Signature(algo, keypair)创建Signature对象，用于签名
2. 多次调用update 方法，输入要进行签名的数据，直到输完所有的数据
3. 调用sign方法进行签名，得到签名值。
4. 调用free方法释放Signature对象

讨论

[JNI]通常的RSA签名是使用PKCS#1 V1.5方式填充的，通常实现的话，对同样的数据和私钥，签名值是一样的。如果使用RSA PSS的签名的话，由于带有随机数，每次签名的签名值是不一样的。RSA PSS的签名需要使用new Signature(pssParam, keypair)来创建Signature对象，其中包含Hash算法，MGF算法和Salt的长度。

签名还可以使用先进行Hash运算（参见3.2），再用得到的Hash值使用signHash 方法进行签名，无需再调用update方法。

RSA签名的结果是大端字节序的，长度等于模长。

[纯JAVA] 使用net.netca.pki.encoding.asn1.pki.Signable接口的sign方法。有两个Signable接口的实现，net.netca.pki.encoding.asn1.pki.JCESigner是直接使用JCE来实现；net.netca.pki.encoding.asn1.pki. NetcaSigner是使用JNI来实现。

RSA签名的结果是大端字节序的，长度等于模长。

[简化封装]使用net.netca.pki.global.ISign接口。具体步骤如下：

1. 使用net.netca.pki.global.X509Certificate类的newSignObject方法创建ISign的实现对象。注意X509Certificate对象必须是RSA证书
2. 多次调用signUpdate方法，输入要进行签名的数据，直到输完所有的数据
3. 调用signFinal方法，进行签名。
4. 如果ISign的实现对象实现了Freeable接口，则使用Freeable的free方法释放

上面2-4步，可以使用X509Certificate类的sign方法来替代。当前实现对于RSA证书默认使用SHA256WithRSAEncrypt，如果需要使用其他签名算法，需要使用Pki类的init方法来设置。这种方式不支持RSA PSS的签名

RSA签名的结果是大端字节序的，长度等于模长。

## RSA验证签名

问题

如何使用RSA进行验证签名？

解决方案

RSA验证签名使用net.netca.pki.Signature类

使用RSA进行验证签名的步骤如下：

1. 调用new Signature(algo, publickey)创建Signature对象，用于验证签名
2. 多次调用update 方法，输入要进行签名的数据，直到输完所有的数据
3. 调用verify方法进行验证签名。
4. 调用free方法释放Signature对象

讨论

[JNI]通常的RSA签名是使用PKCS#1 V1.5方式填充的，通常实现的话，对同样的数据和私钥，签名值是一样的。如果使用RSA PSS的签名的话，由于带有随机数，每次签名的签名值是不一样的。RSA PSS的签名需要使用new Signature(pssParam, publickey)来创建Signature对象，其中包含Hash算法，MGF算法和Salt的长度。

验证签名还可以使用先进行Hash运算（参见3.2），再用得到的Hash值使用verifyHash方法进行验证签名，无需再调用update方法。

验证签名，要求RSA签名的结果是大端字节序的。

[纯JAVA] 使用net.netca.pki.encoding.asn1.pki.Verifible接口的verify方法。有两个Verifible接口的实现，net.netca.pki.encoding.asn1.pki.JCEVerifier是直接使用JCE来实现；net.netca.pki.encoding.asn1.pki.NetcaVerifier是使用JNI来实现。

验证签名，要求RSA签名的结果是大端字节序的。

[简化封装]使用net.netca.pki.global.IVerify接口。具体步骤如下：

1. 使用net.netca.pki.global.X509Certificate类的newVerifyObject方法创建IVerify的实现对象。注意X509Certificate对象必须是RSA证书
2. 多次调用verifyUpdate方法，输入要进行签名的数据，直到输完所有的数据
3. 调用verifyFinal方法，进行签名。
4. 如果IVerify的实现对象实现了Freeable接口，则使用Freeable的free方法释放

上面2-4步，可以使用X509Certificate类的verifySignature方法来替代。当前实现对于RSA证书默认使用SHA256WithRSAEncrypt，如果需要使用其他签名算法，需要使用Pki类的init方法来设置。这种方式不支持RSA PSS的签名

验证签名，要求RSA签名的结果是大端字节序的。

## SM2签名

问题

如何使用SM2进行签名？

解决方案

SM2签名使用net.netca.pki.Signature类

使用SM2进行签名的步骤如下：

1. 调用new Signature(Signature.SM3WITHSM2, keypair)创建Signature对象，用于签名
2. 多次调用update 方法，输入要进行签名的数据，直到输完所有的数据
3. 调用sign方法进行签名，得到签名值。
4. 调用free方法释放Signature对象

讨论

[JNI] 根据《GM/T 0009-2012 SM2密码使用规范》，SM2签名应该配合SM3来使用，而且ID使用1234567812345678。由于存在随机数，SM2签名每次的结果都会不同。

签名还可以使用先进行Hash运算（参见3.2），再用得到的Hash值使用signHash 方法进行签名，无需再调用update方法。注意：SM2签名需要的Hash值不是直接要签名数据的Hash值，而是（Z||要签名数据）的Hash值，因此需要先使用PublicKey 对象的ComputeSM2Z (Hash. SM3)方法计算Z值。

SM2签名的结果是《GM/T 0009-2012 SM2密码使用规范》中的SM2Signature的DER编码。

[纯JAVA] 使用net.netca.pki.encoding.asn1.pki.Signable接口的sign方法。有两个Signable接口的实现，net.netca.pki.encoding.asn1.pki.JCESigner是直接使用JCE来实现(需要支持SM2签名的JCE)；net.netca.pki.encoding.asn1.pki. NetcaSigner是使用JNI来实现。

SM2签名的结果是《GM/T 0009-2012 SM2密码使用规范》中的SM2Signature的DER编码。

[简化封装]使用net.netca.pki.global.ISign接口。具体步骤如下：

1. 使用net.netca.pki.global.X509Certificate类的newSignObject方法创建ISign的实现对象。注意X509Certificate对象必须是SM2证书
2. 多次调用signUpdate方法，输入要进行签名的数据，直到输完所有的数据
3. 调用signFinal方法，进行签名。
4. 如果ISign的实现对象实现了Freeable接口，则使用Freeable的free方法释放

上面2-4步，可以使用X509Certificate类的sign方法来替代。

SM2签名的结果是《GM/T 0009-2012 SM2密码使用规范》中的SM2Signature的DER编码。

## SM2验证签名

问题

如何使用SM2进行验证签名？

解决方案

SM2验证签名使用net.netca.pki.Signature类

使用SM2进行验证签名的步骤如下：

1. 调用new Signature(Signature.SM3WITHSM2, publickey)创建Signature对象，用于验证签名
2. 多次调用update 方法，输入要进行签名的数据，直到输完所有的数据
3. 调用verify方法进行验证签名。
4. 调用free方法释放Signature对象

讨论

[JNI] 根据《GM/T 0009-2012 SM2密码使用规范》，SM2签名应该配合SM3来使用，而且ID使用1234567812345678。直接使用NULL的参数就是使用这个默认ID。由于存在随机数，SM2签名每次的结果都会不同。

验证签名还可以使用先进行Hash运算（参见3.2），再用得到的Hash值使用verifyHash方法进行验证签名，无需再调用update方法。注意：SM2签名需要的Hash值不是直接要签名数据的Hash值，而是（Z||要签名数据）的Hash值，因此需要先使用PublicKey 对象的ComputeSM2Z (Hash. SM3)方法计算Z值。

验证签名，要求SM2签名的结果是《GM/T 0009-2012 SM2密码使用规范》中的SM2Signature的DER编码。

[纯JAVA] 使用net.netca.pki.encoding.asn1.pki.Verifible接口的verify方法。有两个Verifible接口的实现，net.netca.pki.encoding.asn1.pki.JCEVerifier是直接使用JCE来实现(无论JCE是否支持SM2，这个实现都支持SM2签名的验证)；net.netca.pki.encoding.asn1.pki.NetcaVerifier是使用JNI来实现。

验证签名，要求SM2签名的结果是《GM/T 0009-2012 SM2密码使用规范》中的SM2Signature的DER编码。

[简化封装]使用net.netca.pki.global.IVerify接口。具体步骤如下：

1. 使用net.netca.pki.global.X509Certificate类的newVerifyObject方法创建IVerify的实现对象。注意X509Certificate对象必须是SM2证书
2. 多次调用verifyUpdate方法，输入要进行签名的数据，直到输完所有的数据
3. 调用verifyFinal方法，进行签名。
4. 如果IVerify的实现对象实现了Freeable接口，则使用Freeable的free方法释放

上面2-4步，可以使用X509Certificate类的verifySignature方法来替代。

验证签名，要求SM2签名的结果是《GM/T 0009-2012 SM2密码使用规范》中的SM2Signature的DER编码。

## ECDSA签名

问题

如何使用ECDSA进行签名？

解决方案

ECDSA签名使用net.netca.pki.Signature类

使用ECDSA进行签名的步骤如下：

1. 调用new Signature(algo, keypair)创建Signature对象，用于签名
2. 多次调用update 方法，输入要进行签名的数据，直到输完所有的数据
3. 调用sign方法进行签名，得到签名值。
4. 调用free方法释放Signature对象

讨论

[JNI]由于存在随机数，ECDSA签名每次的结果都会不同。

签名还可以使用先进行Hash运算（参见3.2），再用得到的Hash值使用signHash 方法进行签名，无需再调用update方法。

ECDSA签名的结果是RFC3279中的Ecdsa-Sig-Value，的DER编码。

[纯JAVA] 使用net.netca.pki.encoding.asn1.pki.Signable接口的sign方法。有两个Signable接口的实现，net.netca.pki.encoding.asn1.pki.JCESigner是直接使用JCE来实现；net.netca.pki.encoding.asn1.pki. NetcaSigner是使用JNI来实现。

ECDSA签名的结果是RFC3279中的Ecdsa-Sig-Value，的DER编码。

## ECDSA验证签名

问题

如何使用ECDSA进行验证签名？

解决方案

验证ECDSA签名使用net.netca.pki.Signature类

验证ECDSA签名的步骤如下：

1. 调用new Signature(algo, publickey)创建Signature对象，用于验证签名
2. 多次调用update 方法，输入要进行签名的数据，直到输完所有的数据
3. 调用verify方法进行验证签名。
4. 调用free方法释放Signature对象

讨论

[JNI] 验证签名还可以使用先进行Hash运算（参见3.2），再用得到的Hash值使用verifyHash方法进行验证签名，无需再调用update方法。

验证签名，要求ECDSA签名的结果是RFC3279中的Ecdsa-Sig-Value，的DER编码。

[纯JAVA] 使用net.netca.pki.encoding.asn1.pki.Verifible接口的verify方法。有两个Verifible接口的实现，net.netca.pki.encoding.asn1.pki.JCEVerifier是直接使用JCE来实现；net.netca.pki.encoding.asn1.pki.NetcaVerifier是使用JNI来实现。

验证签名，要求ECDSA签名的结果是RFC3279中的Ecdsa-Sig-Value，的DER编码。

## ECDH密钥协商[JNI]

问题

如何使用ECDH进行密钥协商？

解决方案

使用net.netca.pki.KeyPair类的ecDH方法

讨论

ECDH密钥协商需要自己的密钥对对象和对方的公钥对象，而且它们的椭圆曲线要一样。

ECDH有一个变种Elliptic Curve Cryptography Cofactor Diffie-Hellman，这时h也参与运算，要进行这样的密钥协商只需要使用ecCDH方法即可。

ECDH密钥协商得到的共享数据长度为密钥对长度，通常都需要使用KDF方法转换为密钥来使用。

## 使用口令导出密钥

问题

如何使用口令导出密钥？

解决方案

使用net.netca.pki.Util类的pkcs5PBKDF2方法

讨论

使用口令导出密钥通常使用PKCS#5第二版的方式，需要一个Hash函数、盐值和迭代次数（通常至少大于1000）。这仅仅是导出密钥，如果加密还需要IV等数据，通常是使用随机数发生器来产生的。另外，PKCS#12有自己的导出密钥的方法但仅用于PFX中。

[JNI] net.netca.pki.Util类的pkcs5PBKDF2方法实现使用PKCS#5第二版的方法口令导出密钥

[纯JAVA] net.netca.pki.encoding.asn1.pki.IKDF接口的kdf方法用于导出密钥。net.netca.pki.encoding.asn1.pki.Pkcs5PBKDF2实现了IKDF接口，使用net.netca.pki.encoding.asn1.pki.hasher来计算Hash，实现使用PKCS#5第二版的方法口令导出密钥

# 证书

## 已知证书编码，获取证书对象

问题

如何根据证书编码，获取到证书对象？

解决方案

使用net.netca.pki.Certificate的构造函数

讨论

[JNI] net.netca.pki.Certificate的构造函数支持解析DER编码、BASE64编码的和带BEGIN和END头尾的简单PEM编码的X.509公钥证书，其他的格式不支持。微软IE的Internet选项->内容->证书里的导出证书导出的证书，可以支持DER编码和Base64编码，不支持PKCS#7证书、PKCS#12和Microsoft序列化证书存储。

另外，openssl的下面的PEM格式也不支持。例子如下：

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 11 (0xb)

Signature Algorithm: sha1WithRSAEncryption

Issuer: C=CN, CN=TEST ROOT

Validity

Not Before: Mar 29 09:31:46 2010 GMT

Not After : Mar 24 09:31:46 2030 GMT

Subject: C=CN, O=广州, CN=你好123, emailAddress=test@test.com

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:a2:53:ab:1c:bb:9e:92:78:62:45:06:0e:b1:43:

66:db:31:75:38:99:ee:52:83:06:29:29:67:cc:80:

1d:4a:e2:8f:0e:f5:f3:a2:28:9e:e9:d5:38:17:e5:

12:59:0a:0c:b7:12:0a:3a:7e:5d:89:6d:d0:3b:54:

ca:5c:04:9b:b0:52:4b:ba:2d:ff:1e:6e:3d:87:6b:

c7:19:21:15:5e:f1:46:7c:9d:0d:79:90:e3:1d:83:

e3:69:a6:55:2b:84:d0:f0:e5:ea:30:00:6d:c7:5a:

21:c5:d1:45:e9:cb:2d:cf:f2:7d:3b:c5:c9:ed:68:

54:46:f4:6a:76:58:13:60:fd

Exponent: 65537 (0x10001)

X509v3 extensions:

X509v3 Basic Constraints: critical

CA:FALSE

X509v3 Key Usage: critical

Digital Signature, Non Repudiation

X509v3 Subject Key Identifier:

66:A7:C8:D6:85:72:0E:8E:4E:28:8F:E0:C3:93:81:18:98:69:5D:8B

X509v3 Authority Key Identifier:

keyid:DA:E3:3A:C2:B8:9D:C7:44:DE:A7:A0:BE:AC:FD:12:EB:0E:1A:F4:AD

Signature Algorithm: sha1WithRSAEncryption

b0:4c:49:cf:48:6b:8d:5d:32:78:76:59:24:8a:46:ac:d9:ec:

de:b1:6c:a8:ac:eb:d2:f9:91:8b:77:bb:20:8d:2e:44:8d:18:

f4:51:c3:4a:ca:ce:10:1f:8e:bd:66:fb:e0:4f:27:85:ba:d8:

5d:e4:69:c7:94:f7:40:34:6b:b2:4a:c4:4c:8a:88:e2:da:c6:

ed:95:3b:d5:e2:d7:49:6b:eb:1a:6b:06:5a:77:f5:3c:3b:b3:

cd:a8:7f:31:21:25:16:be:c1:85:b9:49:20:58:fb:45:f2:61:

4e:29:69:8e:5f:db:4c:aa:43:a7:31:a8:1d:eb:68:d4:9c:99:

38:ba:14:d8:b2:c2:42:a8:90:45:33:59:5c:44:b5:17:35:1a:

74:09:4e:e1:86:20:4d:eb:a6:23:a7:4c:6b:3f:8f:bd:f3:d9:

e6:88:5a:ff:0d:7f:44:26:4a:15:8e:64:75:8d:b2:02:e5:ea:

1c:00:17:0b:de:a2:6d:75:f4:31:c4:48:b4:89:a1:46:d7:1c:

a3:fe:80:3b:43:ec:04:4f:9b:92:8f:74:a8:51:f3:dc:36:11:

d9:b6:aa:64:49:db:94:af:86:41:d6:32:71:4b:3d:7a:59:ab:

b1:00:9c:99:dd:ab:cb:1f:da:2d:7a:77:d5:8f:9d:8a:de:98:

76:0c:d3:8a

-----BEGIN CERTIFICATE-----

MIICyDCCAbCgAwIBAgIBCzANBgkqhkiG9w0BAQUFADAhMQswCQYDVQQGEwJDTjES

MBAGA1UEAxMJVEVTVCBST09UMB4XDTEwMDMyOTA5MzE0NloXDTMwMDMyNDA5MzE0

NlowUDELMAkGA1UEBhMCQ04xDzANBgNVBAoMBuW5v+W3njESMBAGA1UEAwwJ5L2g

5aW9MTIzMRwwGgYJKoZIhvcNAQkBFg10ZXN0QHRlc3QuY29tMIGfMA0GCSqGSIb3

DQEBAQUAA4GNADCBiQKBgQCiU6scu56SeGJFBg6xQ2bbMXU4me5SgwYpKWfMgB1K

4o8O9fOiKJ7p1TgX5RJZCgy3Ego6fl2JbdA7VMpcBJuwUku6Lf8ebj2Ha8cZIRVe

8UZ8nQ15kOMdg+NpplUrhNDw5eowAG3HWiHF0UXpyy3P8n07xcntaFRG9Gp2WBNg

/QIDAQABo2AwXjAMBgNVHRMBAf8EAjAAMA4GA1UdDwEB/wQEAwIGwDAdBgNVHQ4E

FgQUZqfI1oVyDo5OKI/gw5OBGJhpXYswHwYDVR0jBBgwFoAU2uM6wridx0Tep6C+

rP0S6w4a9K0wDQYJKoZIhvcNAQEFBQADggEBALBMSc9Ia41dMnh2WSSKRqzZ7N6x

bKis69L5kYt3uyCNLkSNGPRRw0rKzhAfjr1m++BPJ4W62F3kaceU90A0a7JKxEyK

iOLaxu2VO9Xi10lr6xprBlp39Tw7s82ofzEhJRa+wYW5SSBY+0XyYU4paY5f20yq

Q6cxqB3raNScmTi6FNiywkKokEUzWVxEtRc1GnQJTuGGIE3rpiOnTGs/j73z2eaI

Wv8Nf0QmShWOZHWNsgLl6hwAFwveom119DHESLSJoUbXHKP+gDtD7ARPm5KPdKhR

89w2Edm2qmRJ25SvhkHWMnFLPXpZq7EAnJndq8sf2i16d9WPnYremHYM04o=

-----END CERTIFICATE-----

上面这种格式是不支持的，只支持以下的PEM格式：

-----BEGIN CERTIFICATE-----

MIICyDCCAbCgAwIBAgIBCzANBgkqhkiG9w0BAQUFADAhMQswCQYDVQQGEwJDTjES

MBAGA1UEAxMJVEVTVCBST09UMB4XDTEwMDMyOTA5MzE0NloXDTMwMDMyNDA5MzE0

NlowUDELMAkGA1UEBhMCQ04xDzANBgNVBAoMBuW5v+W3njESMBAGA1UEAwwJ5L2g

5aW9MTIzMRwwGgYJKoZIhvcNAQkBFg10ZXN0QHRlc3QuY29tMIGfMA0GCSqGSIb3

DQEBAQUAA4GNADCBiQKBgQCiU6scu56SeGJFBg6xQ2bbMXU4me5SgwYpKWfMgB1K

4o8O9fOiKJ7p1TgX5RJZCgy3Ego6fl2JbdA7VMpcBJuwUku6Lf8ebj2Ha8cZIRVe

8UZ8nQ15kOMdg+NpplUrhNDw5eowAG3HWiHF0UXpyy3P8n07xcntaFRG9Gp2WBNg

/QIDAQABo2AwXjAMBgNVHRMBAf8EAjAAMA4GA1UdDwEB/wQEAwIGwDAdBgNVHQ4E

FgQUZqfI1oVyDo5OKI/gw5OBGJhpXYswHwYDVR0jBBgwFoAU2uM6wridx0Tep6C+

rP0S6w4a9K0wDQYJKoZIhvcNAQEFBQADggEBALBMSc9Ia41dMnh2WSSKRqzZ7N6x

bKis69L5kYt3uyCNLkSNGPRRw0rKzhAfjr1m++BPJ4W62F3kaceU90A0a7JKxEyK

iOLaxu2VO9Xi10lr6xprBlp39Tw7s82ofzEhJRa+wYW5SSBY+0XyYU4paY5f20yq

Q6cxqB3raNScmTi6FNiywkKokEUzWVxEtRc1GnQJTuGGIE3rpiOnTGs/j73z2eaI

Wv8Nf0QmShWOZHWNsgLl6hwAFwveom119DHESLSJoUbXHKP+gDtD7ARPm5KPdKhR

89w2Edm2qmRJ25SvhkHWMnFLPXpZq7EAnJndq8sf2i16d9WPnYremHYM04o=

-----END CERTIFICATE-----

[纯JAVA] net.netca.pki.encoding.asn1.pki.X509Certificate类表示X.509公钥证书。X509Certificate的构造函数也是只支持DER编码、BASE64编码的和带BEGIN和END头尾的简单PEM编码的X.509公钥证书，其他的格式不支持。

[简化封装] net.netca.pki.global.X509Certificate表示X.509公钥证书。可以根据证书编码，使用net.netca.pki.global.Pki类的decodeCert方法来创建。也只支持DER编码、BASE64编码的和带BEGIN和END头尾的简单PEM编码的X.509公钥证书，其他的格式不支持。

## 获取设备中的所有证书[JNI]

问题

如何获取当前连接的设备中的所有证书？

解决方案

1、使用new net.netca.pki.DeviceSet(Device. ANY\_DEVICE，Device.DEVICE\_FLAG\_CAN\_INSTALL\_CERT)获取当前连接有证书的加密设备组成的DeviceSet对象

2、使用DeviceSet对象的count()方法获取设备数

3、使用DeviceSet对象的get(i),逐个获取Device对象

4、逐个设备，调用getCerts()，获取设备的所有证书

5、使用free方法释放Device对象

6、使用free方法释放DeviceSet对象

讨论

从设备集可以获取所有的设备，从设备可以获取设备中所有的密钥对，从密钥对可以获取密钥对对应的所有证书。这个方法获得的证书是带有密钥对对象的。

[简化封装] net.netca.pki.global.Pki类的getAllCertificates方法也可以获取当前连接的设备的所有证书，但是仅仅对应JNI的实现有效。

## 选择签名证书

问题

如何选择签名证书？

解决方案

net.netca.pki.Certificate.select("Signature",expr)

讨论

[JNI]选择签名证书可以有两种方案：一个是直接从加密设备里读取，另外一个是从个人证书库里读取。直接从加密设备里读取的优点是通常可以直接读到加密设备里存在的所有证书。缺点是速度稍慢，在Windows下无法读取微软的个人证书库里的其他RSA的证书。从个人证书库里读取的优点是速度稍快，在Windows下可以读取微软的个人证书库里的其他RSA的证书。缺点是需要有其他工具把证书注册到个人证书库，而且可能出现证书残留现象。NETCA CRYPTO本身不会自动注册证书到证书库的，通常把证书注册到证书库是由KeyX来完成的。

net.netca.pki.Certificate.select也支持直接从加密设备里读取。只需要在type传入一个JSON字符串，如{ "UIFlag":"default", "InValidity":true, "Type":"signature" , "Method":"device", "Value":"any" }

如果存在多张证书，通常会弹出对话框让用户选择，在服务器端使用的话，这会存在问题的。

[简化封装] net.netca.pki.global.Pki类的getUserCert(Pki.USERCERT\_TYPE\_SIGN,null,pwd)可以获取用户的签名证书

## 选择加密证书[JNI]

问题

如何选择加密证书？

解决方案

net.netca.pki.Certificate.select("Encrypt",expr)

讨论

选择加密证书通常是从联系人证书库里获取的。这需要先配置好联系人证书库，可以使用KeyX来配置。如果加密的数据还需要自己以后能解开的话，这时还需要配置自己的证书来进行加密。

如需从个人证书库里获取加密证书，请使用net.netca.pki.Certificate.select("Decrypt",expr)。

[简化封装] net.netca.pki.global.Pki类的getUserCert(Pki.USERCERT\_TYPE\_ENCRYPT,null,pwd)可以获取用户的解密证书。没有简单的方式获取加密证书。

## 证书的微缩图

问题

如何获取证书的微缩图？

解决方案

使用net.netca.pki.Certificate类的computeThumbprint(hashAlgo)方法

讨论

[JNI]证书的微缩图就是证书的DER编码的Hash值，使用不同的Hash算法，得到的微缩图是不一样的。net.netca.pki.Certificate类的computeThumbprint可以直接算出来

[纯JAVA]没有直接的方式，只能先使用net.netca.pki.encoding.asn1.pki.X509Certificate对象的derEncode方法，然后进行Hash运算（参见3.2）。

[简化封装]使用net.netca.pki.global.X509Certificate类的computeThumbprint方法。

## 双证书的类型

问题

如何判断证书是签名证书还是加密证书？

解决方案

使用net.netca.pki.Certificate类的match方法

match (”CertType='Signature'”)返回true，表示签名证书

match (”CertType='Encrypt'”)返回true，表示加密证书

讨论

[JNI]对于用户的双证书来说，区分是加密证书还是签名证书是由密钥用法扩展来确定的。通常，密钥用法扩展的digitalSignature或者nonRepudiation位有一位置位就认为可以进行签名，keyEncipherment或者dataEncipherment位置位认为可以进行加密。对于通常的数字信封来说，通常使用keyEncipherment位或者keyAgreement位，因为都是先加密对称密钥的。如果直接使用非对称算法加密数据则使用dataEncipherment位。直接使用密钥用法扩展来判断的方法如下：

先使用getKeyUsage 方法获取密钥用法值ku。如果不存在密钥用法扩展，则ku返回-1，否则是根据密钥扩展的值逐位或的结果。ku&(Certificate.KEYUSAGE\_DIGITALSIGNATURE | Certificate.KEYUSAGE\_NONREPUDIATION)不为0表示可以签名。ku&( Certificate. KEYUSAGE\_KEYENCIPHERMENT | Certificate.KEYUSAGE\_DATAENCIPHERMENT) 不为0表示可以加密

[简化封装]使用net.netca.pki.global.X509Certificate类的getKeyUsage方法获取密钥用法的值，然后自己判断。ku&(X509Certificate.KEYUSAGE\_DIGITALSIGNATURE | X509Certificate.KEYUSAGE\_NONREPUDIATION)不为0表示可以签名。ku&( X509Certificate. KEYUSAGE\_KEYENCIPHERMENT | X509Certificate.KEYUSAGE\_DATAENCIPHERMENT) 不为0表示可以加密

## 证书的公钥[JNI]

问题

如何获取证书的公钥？

解决方案

使用net.netca.pki.Certificate类的getPublicKey方法

用来进行加密的公钥使用getPublicKey (Certificate.PURPOSE\_ENCRYPT)

用来进行验证签名的公钥使用getPublicKey (Certificate.PURPOSE\_SIGN)

用来进行密钥协商的公钥使用getPublicKey (Certificate.PURPOSE\_KEYAGREEMENT)

讨论

通常NETCA CRYPTO通常不能使用过期的证书来获取公钥，上面的两个方法使用过期证书的话，都会返回null。如果需要验证很久以前的签名，可以使用getPublicKey (Certificate.PURPOSE\_VERIFY\_OLD\_DATA)，但要考虑到证书过期后，如何验证该证书在签名的时间是否作废。这通常涉及到时间戳和预先保存以前的CRL或OCSP响应等作废信息。

注意：某些算法的证书可以把算法相关的参数保存到CA证书中，证书本身的公钥信息是不完整的，因此不能getPublicKey获取公钥，这时要先验证完证书路径才能获取公钥(使用net.netca.pki.CertVerify的getPublicKey方法)。不过， RSA和ECC（包括SM2）的证书都不会存在这种情况

## 证书的密钥对[JNI]

问题

如何获取证书对应的密钥对对象？

解决方案

使用net.netca.pki.Certificate类的getKeyPair(flag,purpose,pwd)方法

讨论

根据证书获取其对应的密钥对对象有两种方式：一是根据当前连接的加密设备，在加密设备里寻找和证书公钥匹配的密钥对，二是打开个人证书库寻找和目标证书匹配的证书，再根据证书里的相应密钥对属性来获取密钥对对象。参见4.2

flag决定着如何获取密钥对对象，如果Certificate.SEARCH\_KEYPAIR\_FLAG\_CURRENT\_USER置位则从当前用户的个人证书库中获取，Certificate.SEARCH\_KEYPAIR\_FLAG\_LOCAL\_MACHINE置位则从本机的个人证书库中获取，Certificate.SEARCH\_KEYPAIR\_FLAG\_DEVICE置位则从当前连接的加密设备中获取。如果有多个位置位则会依次搜索。无论这些位是否置位，首先检查的是证书对象里本身有没有密钥对属性或者密钥对对象。此外，如果Certificate.SEARCH\_KEYPAIR\_FLAG\_DEVICE置位，则flag的低位表示连接加密设备的标志。purpose表示要获取的密钥对的用途，是用于解密还是签名。如果用于签名的话，不能使用过期证书。如果用于解密或密钥协商，则可以使用过期证书。实现会根据证书的密钥用法来判断是否能进行那样的操作，如果不能，直接返回null。pwd如果不为null或者空串，则会用来校验用户密码，一般情况下，不建议使用。注意，flag可以取0，表示只从证书对象里获取。如果从证书库里获取是匹配证书，从设备里获取是匹配公钥。

获取证书的密钥对对象用于签名，通常可以使用：getKeyPair (Certificate.SEARCH\_KEYPAIR\_FLAG\_CURRENT\_USER, Certificate.PURPOSE\_SIGN ,null)或者getKeyPair (Certificate. SEARCH\_KEYPAIR\_FLAG\_DEVICE, Certificate.PURPOSE\_SIGN ,null)。

获取证书的密钥对对象用于解密，通常可以使用：getKeyPair (Certificate.SEARCH\_KEYPAIR\_FLAG\_CURRENT\_USER, Certificate.PURPOSE\_ENCRYPT ,null)或者getKeyPair (Certificate. SEARCH\_KEYPAIR\_FLAG\_DEVICE, Certificate.PURPOSE\_ENCRYPT ,null)。

## 获取证书的扩展

问题

如何获取指定证书扩展里的值？

解决方案

没有方法可以解析所有的扩展，可以考虑使用net.netca.pki.Certificate类的getExtensionValue方法，自己再根据扩展的定义来解码。

讨论

[JNI]证书扩展有很多种，很多的结构都比较复杂，没有统一的方式来解析，获取里面的值。有某些特定的扩展的值或者部分的值可以通过net.netca.pki.Certificate类的getIntegerAttribute、getAttribute和getMultipleValueAttribute方法来获取。参见《NETCA PKI API参考手册》

getExtensionValue方法根据扩展的OID获得扩展的编码，但是还需要根据具体的扩展的定义来解码，如果你有自己的ASN.1的解码库，可以自己进行解码。如果扩展的值是某些简单的字符串类型，可以使用getStringExtension 方法来获取。

[纯JAVA] net.netca.pki.encoding.asn1.pki.X509Certificate类可以在ASN.1层面处理绝大部分的扩展。getExtensions方法可以获取所有的扩展表示的Extensions对象。Extensions对象的get方法可以获取扩展Extension对象，Extension对象的getExtensionObject方法可以获取具体的扩展对象，然后可以从中解出具体的值

[简化封装]net.netca.pki.global.X509Certificate类的getStringExtension方法可以获取简单字符串编码类型的扩展的值。有些特定扩展值由对应的方法来获取。

## RSA证书

问题

如何判断证书是否RSA证书？

解决方案

使用net.netca.pki.Certificate类的match("Algorithm='RSA'")方法返回true表示是RSA证书。

讨论

[JNI]RSA证书指的是公钥为RSA算法的证书。最简单的判断方式是使用net.netca.pki.Certificate类的match方法。也可以使用getPublicKeyAlgorithm方法获取公钥算法，然后判断algo是否为KeyPair.RSA、0x11和0x21的其中一个值。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.X509Certificate类的isRSA方法

[简化封装]使用net.netca.pki.global.X509Certificate类的getPublicKeyType方法来获取公钥类型，如果为X509Certificate.RSA、X509Certificate. RSA\_PSS或者X509Certificate. RSA\_OAEP则是RSA证书

## SM2证书

问题

如何判断证书是否SM2证书？

解决方案

使用net.netca.pki.Certificate类的match("Algorithm='SM2'")方法返回true表示是SM2证书。

讨论

[JNI]SM2证书指的是公钥为SM2算法的证书。最简单的判断方式是使用net.netca.pki.Certificate类的match方法。也可以使用getPublicKeyAlgorithm方法获取公钥算法，然后判断是否为KeyPair.ECC，如果是，接着使用getIntegerAttribute (Certificate.ATTRIBUTE\_PUBKEY\_ECCCURVE)方法获取椭圆曲线号，最后判断椭圆曲线号是否为Device. ECC\_CURVE\_SM2，是则为SM2证书

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.X509Certificate类的isSM2方法

[简化封装]使用net.netca.pki.global.X509Certificate类的getPublicKeyType方法来获取公钥类型，如果为X509Certificate.SM2则是SM2证书

## ECC证书

问题

如何判断证书是否ECC证书？

解决方案

使用net.netca.pki.Certificate类的match("Algorithm='ECC'")方法返回true表示是ECC证书。

讨论

[JNI]ECC证书指的是公钥为ECC算法的证书。最简单的判断方式是使用net.netca.pki.Certificate类的match方法。也可以使用getPublicKeyAlgorithm方法获取公钥算法，然后判断algo是否为KeyPair.ECC、0x14或0x24之一。

如果需要剔除SM2证书，也就是判断是否为非SM2的ECC证书，使用match ( "Algorithm='ECC' && (! (Algorithm='SM2'))")。或者使用getPublicKeyAlgorithm判断出ECC证书之后，再使用getIntegerAttribute (Certificate.ATTRIBUTE\_PUBKEY\_ECCCURVE)方法获取椭圆曲线号，最后判断椭圆曲线号是否不是Device. ECC\_CURVE\_SM2。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.X509Certificate类的getSubjectPublicKeyInfo方法获取SubjectPublicKeyInfo对象，再根据公钥算法来判断

[简化封装]使用net.netca.pki.global.X509Certificate类的getPublicKeyType方法来获取公钥类型，如果为X509Certificate.SM2、X509Certificate.P224、X509Certificate.P256、X509Certificate.P384或者X509Certificate.P521则是ECC证书。

## 证书在有效期内

问题

如何判断证书是否在有效期内？

解决方案

使用net.netca.pki.Certificate类的isInValidity方法返回true表示证书在有效期内

讨论

[JNI] isInValidity方法是根据本机时间判断的。如果需要采用其他来源的时间来判断需要使用getValidityStart和getValidityEnd来分别获取有效期开始时间和结束时间，然后进行比较看看给定的时间是否在有效期开始时间和结束时间之间。证书的有效期是包含开始时间和结束时间这两个时间点的。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.X509Certificate类的isInValidity方法来判断

[简化封装] 使用net.netca.pki.global.X509Certificate类的isInValidity方法来判断

## 证书是否存在私钥[JNI]

问题

如何判断证书是否存在私钥？

解决方案

使用net.netca.pki.Certificate类的match (”HasPrivKey='True' && CheckPrivKey='True'”)返回true表示存在私钥。

讨论

使用证书编码构造得到的Certificate对象是没有私钥的。从加密设备里获取证书是带有私钥的。从其他人证书库、CA证书库和根证书库里获取的证书是没有私钥的。从个人证书库里获取的证书通常带有私钥属性，但是由于存在证书残留，所以不一定存在对应的私钥。

HasPrivKey='True'表示存在私钥属性或者私钥对象；CheckPrivKey='True'表示去连接加密设备判断是否存在私钥，这里不进行用户密码的验证，而且不检查证书和私钥是否匹配。对于必须验证用户密码才能使用的加密设备，并不适用。match的条件表达式是使用短路操作的。

hasPrivateKey方法仅仅判断是否存在私钥属性或者私钥对象，如果存在证书残留会误判。

## 证书对应的设备是否插入[JNI]

问题

如何判断证书对应的设备当前是插入的？

解决方案

1. 使用getKeyPair (0,purpose,pwd)获取证书对应的密钥对，如果成功转到2，不成功转到5
2. 使用KeyPair对象的getDevice方法从密钥对对象中获得设备对象，参见2.6
3. 使用Device对象的getType()方法和getSerialNumber ()方法分别获得设备的类型和序列号
4. 使用Device.findDevice (type,sn,flag)重新连接设备，能连上则是插入的，不能连上则是已经拔出
5. getKeyPair (Certificate.SEARCH\_KEYPAIR\_FLAG\_DEVICE,purpose,pwd)获取证书对应的密钥对，如果能获取到则是插入的，不能获取到则是拔出的

讨论

只能判断存在设备接口实现类型的密钥对的证书，不能判断CSP实现类型的密钥对的证书对应的设备是否插入。后者你可能需要调用一次私钥操作来判断

不能仅仅使用第5步来进行判断，因为证书对象中可能包含有密钥对对象，这时是不会搜索设备的。如果你能确定证书对象中不会包含有密钥对对象则可以使用第5步就行了

还有一种方式是自己手工获取设备集，然后获取设备，从设备里获取密钥对，然后逐个判断判断证书是否匹配。参见4.2

## 个人证书库的注册[JNI]

问题

如何把证书注册到个人证书库中？

解决方案

只有证书对象里保存着密钥对信息才应该注册到个人证书库中。具体的步骤如下:

1、使用new net.netca.pki.CertStore (CertStore.CURRENT\_USER, CertStore.MY)打开个人证书库。

1. 使用addCertificate方法，添加证书到证书库中
2. 使用close()，关闭证书库

讨论

个人证书库有两种，一种是NETCA的个人证书库，一种是Windows的个人证书库。NETCA的个人证书库会把Windows的个人证书库中RSA类型的CSP的证书挂接在其后面（如果该证书在NETCA的个人证书库不存在的话）。这样，通过NETCA的个人证书库就可以通过CSP来使用其他CA的证书了。NETCA的个人证书库是供NETCA CRYPTO使用的。Windows的个人证书库是供Windows系统的CryptoAPI使用的，微软自己的应用和其他第三方的应用都会用到。因此，把证书注册到个人证书库中需要同时注册NETCA的个人证书库和Windows的个人证书库。上述方法只能把证书注册到NETCA的个人证书库。

通常从设备里读到的证书、使用getKeyPair方法成功获取到的密钥对对象的证书、使用setKeyPair之后的证书都存在密钥对信息

## 使用其他CA的证书[JNI]

问题

如何使用其他CA的证书进行签名或者解密？

解决方案

如果其他CA的证书已经注册进Windows的个人证书库且是使用RSA类型的CSP，可以使用NETCA个人证书库来获取证书对应的密钥对进行签名和解密。

讨论

其他CA的证书通常没有实现设备接口的，因此只可能通过CSP来调用，这需要个人库里的注册信息。参见4.3、4.8

## 使用PFX

问题

如何使用PFX，进行签名或者解密？

解决方案

具体步骤如下：

1. 使用net.netca.pki.CertStore.loadPfx方法根据PFX方法构造出CertStore对象
2. 使用getCertificateCount方法得到证书数量。
3. 使用getCertificate (i) 方法逐个获取证书
4. 使用getKeyPair (0,purpose,null)获取密钥对对象，如果能获取到则是带有私钥的证书，可以进行签名或者解密操作，否则释放证书对象找下一个证书
5. 进行签名或者解密
6. 使用free()方法释放证书对象
7. 使用close()释放CertStore对象

讨论

PFX文件是根据PKCS#12定义的结构，可以用于保存证书、私钥和相关的证书路径。

[JNI]NETCA CRYPTO可以把PFX当作证书库来处理，使用相关的方法获取到密钥对对象就可以进行签名和解密了。

[纯JAVA] net.netca.pki.encoding.asn1.pki.pkcs12.PFX类表示PFX结构。要使用PFX里的证书进行签名或者解密步骤如下：

1. 使用PFX.decode解码得到PFX对象
2. 使用setHmacImplement方法设置HMAC实现
3. 使用setPkcs12KDF方法设置PKCS#12的KDF实现
4. 使用setPkcs5KDF方法设置PKCS#5的KDF实现
5. 使用setSymEncrypter方法设置对称解密实现
6. 使用decrypt方法输入口令进行解密
7. 使用getCertAndPrivateKeyIterator方法得到X509CertificateAndPrivateKey的迭代器
8. 从迭代器中获取对应的证书和私钥进行签名或者解密

另外，在JCE中是可以把PFX当作Keystore来处理的

## 获取证书链

问题

如何获取证书链？

解决方案

使用net.netca.pki.Certificate类的buildCertPath方法。

讨论

[JNI]使用buildCertPath方法，需要配置好安全环境。它根据安全环境里的根证书和CA证书来构造证书链的。得到的证书链中，第0个为根证书，第i个为第i+1个的颁发者证书，最后一个是为要构造证书链的证书。buildCertPath方法只能获取到一条证书链，也不保证这条证书链一定能验证成功。

此外，在使用CertVerify类相关的方法，验证证书成功后，可以使用getCertPath方法来获取证书链。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.X509CertificatePathBuilder类，具体的步骤如下：

1. new X509CertificatePathBuilder()构造X509CertificatePathBuilder对象
2. 使用addRootCert方法添加根证书
3. 可选的，使用addCACert方法添加CA证书
4. 使用build方法构造证书链

可能可以构造出多条证书链。得到的证书链中，第0个为根证书，第i个为第i+1个的颁发者证书，最后一个是为要构造证书链的证书。得到的证书链不保证能验证成功

[简化封装] 使用net.netca.pki.global.X509Certificate类的buildCertPath方法。得到的证书链中，第0个为根证书，第i个为第i+1个的颁发者证书，最后一个是为要构造证书链的证书。buildCertPath方法只能获取到一条证书链，也不保证这条证书链一定能验证成功。

## 验证证书

问题

如何验证证书？

解决方案

最简单的方式是使用net.netca.pki.Certificate类的verify方法。但是这种方式在特殊环境下存在一些问题，这时需要更复杂的方式来验证

讨论

[JNI]证书验证只能验证某一条证书路径，单独验证一张证书有时候是没意义的。通常一张证书只有一条证书路径，这时候可以使用verify (flag)来验证。verify方法需要依赖于安全环境的配置，它不能得到真正验证的证书路径，也不能处理过期证书的作废验证。如果证书已经作废，也就是返回Certificate.STATUS\_REVOKED或者Certificate.STATUS\_CA\_REVOKED可以使用getRevokeInfo方法获取作废信息。

更好的证书验证可以使用net.netca.pki.CertVerify相关的方法，具体步骤如下：

1. 使用new CertVerify()方法创建一个CertVerify对象。
2. 可选地，使用setOption(option)方法设置验证的选项。
3. 可选地，使用setVerifyTime(verifyTime)方法设置验证的时间，如果不设置，默认为当前时间
4. 可选地，使用setParam(param)方法来设置验证的参数。
5. 使用addRootCert(rootCert)方法来增加根证书。
6. 可选地，使用addCACert(caCert)方法来增加中级CA证书，如果没有中级CA证书不需要这一步
7. 根据设置的参数，可以自动根据证书里的信息来获取CRL或者OCSP来验证证书的作废状态的，如果证书里没有相关的信息或者在验证过期的证书，则需要使用addCRL(crl)或者addOCSP(ocspResp)
8. 使用verify(cert)方法来验证证书
9. 如果返回CertVerify.STATUS\_REVOKED或者CertVerify.STATUS\_CA\_REVOKED则表示证书已经作废，可以使用getRevokeInfo方法来或者作废信息
10. 如果能构造出证书链，使用getCertPath ()方法来获取证书链
11. 使用free()方法来释放CertVerify对象。

注意：默认情况下仅仅处理或者忽略下列关键扩展：Key Usage、Certificate Policies、Policy Mappings、Subject Alternative Name、Basic Constraints、Name Constraints、Policy Constraints、Inhibit anyPolicy。其他扩展（包括Extended Key Usage）如果是关键扩展，需要通过addIgnoreExtension方法来忽略该关键扩展，然后在验证成功后，再获取该扩展来进行相关的处理。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.X509CertificatePathValidator类。具体步骤如下：

1. new X509CertificatePathValidator()创建X509CertificatePathValidator对象
2. 可选地，setCheckTime方法设置验证时间。如果不设置则使用当前时间
3. 可选地，setOption方法设置验证选项
4. 可选地，setVerifier方法设置验证器
5. setVerifyLevel方法设置验证的级别
6. 可选地addCRL方法添加CRL，addOCSPResponse方法添加OCSP响应
7. 使用verify进行验证，不抛异常表示成功
8. 如果抛出CertStatusRevokedException异常，表示证书已经作废，可以通过getRevokeInfo方法作废信息
9. 如果抛出CertStatusUndeterminedException异常，表示证书的状态不能确定

注意：默认情况下仅仅处理或者忽略下列关键扩展：Key Usage、Certificate Policies、Policy Mappings、Subject Alternative Name、Basic Constraints、Policy Constraints、Inhibit anyPolicy。其他扩展（包括Extended Key Usage）如果是关键扩展，需要通过addExtensionProcessor方法来处理该关键扩展。

[简化封装] 使用net.netca.pki.global.X509Certificate类的verify方法。如果返回X509Certificate.STATUS\_REVOKED或者X509Certificate.STATUS\_CA\_REVOKED证书已经作废，可以使用getRevokeInfo方法获取作废信息。注意，JCE实现不支持验证证书作废状态

## 验证过期证书

问题

如何验证过期证书？

解决方案

验证过期证书涉及的问题很多，当前没有能验证所有情况的方法。只能处理一种最简单的情况。这种情况是所有相关的算法都没被破解。这时，只需要使用net.netca.pki.CertVerify类的相关的方法使用当时的时间验证。参见4.20。其中，第3步需要设置验证的时间，第7步需要拿到当时的OCSP和CRL进行设置。

讨论

[JNI]验证过期的证书是相当复杂的，其中最主要是涉及到算法可以被破解，因而，所有相关的证书都可能被伪造。这通常需要通过提前使用时间戳。还有就是通常CA发布的CRL和OCSP，只能验现在的证书的作废状态，不能验以前某个时间点的证书的作废状态，因此需要保留以前的CRL或OCSP响应。

[纯JAVA]和CertVerify类类似，也是X509CertificatePathValidator类第2步设置验证的时间，第6步需要拿到当时的OCSP和CRL进行设置。

## 限制获取指定算法的证书[JNI]

问题

如何限定只能获取指定算法的证书？比如：只能获取RSA的证书。

解决方案

使用net.netca.pki.Util.setAllowCertType(“RSA”)方法可以设置只能获取RSA证书

讨论

setAllowCertType方法可以限制从设备和证书库里获取的证书类型。如果参数为NULL或者“ANY”则为任意的证书，如果为“RSA”则为RSA证书，“SM2”则为SM2证书，“RSA:SM2”则为RSA证书和SM2证书都行。也可以在全局配置的AllowCertType项来实现，参见《NETCA PKI API参考手册》的相关说明。

也使用NetcaPKISelectCert方法的第二个参数条件表达式来达成相关的功能。比如net.netca.pki.Certificate.select (“Algorithm =’RSA’”,…)就是选择RSA证书。

# 证书请求

## 验证证书请求

问题

如何验证证书请求？

解决方案

使用net.netca.pki.CertReq类，首先newCertReq (data)解析证书请求得到CertReq对象，然后使用verify 方法来验证，最后使用free释放CertReq对象

讨论

[JNI]PKCS#10的证书请求是个自签的结构，用来证明产生者拥有公钥对应的私钥。验证证书请求就是使用证书请求里的公钥来验证其签名。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.CertificationRequest类。CertificationRequest.decode解析证书请求得到CertificationRequest对象，然后使用verify方法来验证

## 产生证书请求

问题

如何产生证书请求？

解决方案

使用net.netca.pki.CertReq类

具体步骤如下：

1. 使用new CertReq()创建CertReq对象
2. 使用setSubject (subject)直接设置主体（这个函数要求正序），或者逐项设置主体setC、setO、setOU、setCN等
3. 产生密钥对或者获取原有的密钥对对象
4. 使用sign 方法产生证书请求
5. 使用free释放CertReq对象

讨论

[JNI]通常对于USB Key等设备产生证书请求的时候，都是在设备中先产生新的密钥对的。对于加密机等设备其密钥对，通常是由厂家的工具产生的，产生证书请求是先从加密机里获取到密钥对的。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.CertificationRequestBuilder类。具体步骤如下：

1. new CertificationRequestBuilder()创建CertificationRequestBuilder类
2. setSubject方法设置主体信息
3. 可选的，setAttributes方法设置属性
4. setSubjectPublicKeyInfo方式设置公钥
5. 使用sign方法进行签名

[简化封装] 使用net.netca.pki.global. Device类的createCertificationRequest方法，主体名需要的是反序也就是LDAP序。

## 证书请求的主体采用UTF8String编码[JNI]

问题

产生证书请求的时候，如何使得主体项使用UTF8String编码？

解决方案

使用net.netca.pki.CertReq类的setStringAttribute (14, "{\"subject\":[\"utf8string\"]}")即可。

讨论

在没有进行配置的情况下，使用setSubject 方法设置主体的时候，如果主体项的值可以用PrintableString表示则使用PrintableString，否则使用UTF8String。对于使用NetcaPKICertReqSetStringAttribute逐项设置主体项，则使用UTF8String。如果需要使用其他类型的编码，则需要进行配置获取使用setStringAttribute (14, param)，后者会覆盖前者的配置。具体参见《NETCA PKI API参考手册》。注意：C、Email、DC这些项的编码是固定的，是不受影响的。

# 数字信封

## 构造数字信封

问题

如何构造数字信封？

解决方案

使用net.netca.pki.EnvelopedData类

构造BER编码的数字信封步骤如下：

1. 调用new EnvelopedData (true)方法，创建一个加密使用的EnvelopedData对象。
2. 调用setEncryptAlgorithm (encAlgo)方法，设置对称加密算法
3. 调用addCertificate(cert,flag)方法，添加各个接收者加密证书
4. 调用encryptInit()方法，开始进行加密，同时得到部分结果。
5. 多次调用encryptUpdate(clear)方法进行加密，得到加密的部分结果
6. 调用encryptFinal ()方法结束加密，得到最后部分的结果。4-6步得到的结果串起来就是整个的数字信封
7. 调用free()方法释放EnvelopedData对象

如果需要构造DER编码的数字信封，则把4-6步改为encrypt(clear)

讨论

[JNI]NETCA CRYPTO当前只支持使用密钥传输方式和密钥协商的数字信封。你需要加密给谁，就要获取到它的加密证书来调用addCertificate方法。数字信封可以一次性加密给多个人，如果你需要自己能够解开数字信封，则需要添加自己的加密证书。

数字信封使用两种方式来标识接收者：一是颁发者名和序列号，二是主体密钥标识。如果不换密钥更新加密证书的话，安装完加密证书，通常只能解开后一种方式标识的数字信封。因为不换密钥更新更新证书的后，主体密钥标识是保持不变的。前一种方式使用addCertificate (cert,0)来实现，后一种方式使用addCertificate (cert, 1)来实现。

当前实现对于ECDH的数字信封只支持AES算法。

[纯JAVA] net.netca.pki.encoding.asn1.pki.cms.EnvelopedDataBuilder类，具体步骤如下：

1. new EnvelopedDataBuilder()创建一个EnvelopedDataBuilder对象
2. setEncryptAlgorithm方式设置对称加密证书
3. addCert方法添加加密证书
4. setPublicKeyEncrypter方法设置公钥加密实现
5. setSymEncrypter方法设置对称加密实现
6. setSecureRandomGenerator方法设置随机数发生器
7. encypt进行加密，得到EnvelopedData对象
8. 使用EnvelopedData的encode(true)方法得到数字信封的编码

纯JAVA只支持使用密钥传输方式的数字信封

[简化封装]使用net.netca.pki.global.IEnvelopedDataEncrypt接口或者IEnvelopedDataMultiStepEncrypt接口

前者直接使用net.netca.pki.global.Pki类的envelopedDataEncrypt方法就行了。后者的具体步骤如下：

1. 使用Pki对象的getEnvelopedDataMultiStepEncryptObject方法得到IEnvelopedDataMultiStepEncrypt的实现
2. 使用addCert方法添加加密证书
3. 使用encryptInit方法开始加密，同时得到部分结果。
4. 多次调用encryptUpdate方法输入明文进行加密，同时得到部分结果，直到所有的明文输入完毕。
5. 调用encryptFinal方法结束加密，得到最后部分的结果。3-5的结果组合起来就是整个数字信封
6. 如果IEnvelopedDataMultiStepEncrypt的实现同时实现Freeable，调用Freeable的free方法释放

JCE的实现没有实现IEnvelopedDataMultiStepEncrypt接口

## 不带ContentInfo的数字信封

问题

如何构造不带ContentInfo的数字信封？

解决方案

使用net.netca.pki.EnvelopedData类的setNoContentInfo 方法即可。

讨论

[JNI]对外输出的数字信封，通常都是带ContentInfo的，也就是说，它实际上是个ContentInfo结构。如果在内部使用，通常是需要EnvelopedData的结构。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.cms.EnvelopedData的encode(false)方法即可

[简化封装]需要调用net.netca.pki.global.Pki类的init函数才行

## 构造SM1的数字信封

问题

如何构造SM1算法的数字信封？

解决方案

获取一个支持SM1的net.netca.pki.Devcie对象，然后使用net.netca.pki.EnvelopedData类的setDevice (device)即可。

讨论

[JNI]对于SM1、SSF33之类只能使用硬件实现的算法，需要有一个支持的加密设备才能构造数字信封。setDevice (device)会从device设备构造加解密算法对象来进行加密运算的。

产生SM1算法的数字信封，需要确保所有的接收者都有支持SM1算法的设备才行，否则就会有人解不开。

[纯JAVA] setSymEncrypter方法设置对称加密实现时，使用支持SM1加密的实现

## 符合国密标准的SM2数字信封

问题

如何构造符合国密标准GM/T 0010-2012的SM2的数字信封？

解决方案

构造符合国密标准GM/T 0010-2012的SM2的数字信封BER编码的数字信封步骤如下：

1. 调用new EnvelopedData (true)方法，创建一个加密使用的EnvelopedData对象。
2. 调用setIntegerAttribute（EnvelopedData.ATTRIBUTE\_USE\_SM2\_Q7,1）方法要求使用国密标准GM/T 0010-2012
3. 调用setEncryptAlgorithm (encAlgo)方法，设置对称加密算法
4. 调用addCertificate(cert,flag)方法，添加各个接收者加密证书，其中证书为SM2证书
5. 调用encryptInit()方法，开始进行加密，同时得到部分结果。
6. 多次调用encryptUpdate(clear)方法进行加密，得到加密的部分结果
7. 调用encryptFinal ()方法结束加密，得到最后部分的结果。4-6步得到的结果串起来就是整个的数字信封
8. 调用free()方法释放EnvelopedData对象

如果需要构造DER编码的数字信封，则把5-7步改为encrypt(clear)

讨论

通常情况下，构造得到的数字信封是IETF std70的数字信封，如果结合SM2来使用的话，SM2的相关格式，遵循GM/T 0009-2012《SM2密码算法使用规范》，要产生GM/T 0010-2012《SM2密码算法加密签名消息语法规范》的数字信封，需要设置setIntegerAttribute（EnvelopedData.ATTRIBUTE\_USE\_SM2\_Q7,1），并且addCertificate只能使用SM2证书。

GM/T 0010-2012和IETF std70的主要区别在于：GM/T 0010-2012的data和envelopedData使用了不同的OID，GM/T 0010-2012只支持使用颁发者名和序列号的方式来标识接收者，而IETF std70还可以使用主体密钥标识符来标识。GM/T 0010-2012还存在两个莫名奇妙的sharedInfo。此外，IETF std70有未保护的属性，可以接收者证书可以使用不同的算法。

解开符合国密标准GM/T 0010-2012的SM2的数字信封和IETF std70的数字信封方法是统一的

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.cms. EnvelopedDataBuilder类的setSM2Q7(true)方法

[简化封装]需要调用net.netca.pki.global.Pki类的init函数才行

## 解开数字信封

问题

如何解开数字信封？

解决方案

使用net.netca.pki.EnvelopedData类

解开数字信封的步骤如下：

1. 调用new EnvelopedData(false)方法，创建一个解密使用的EnvelopedData对象。
2. 调用decryptInit ()方法进行解密初始化
3. 多次调用decryptUpdate (cipher)方法进行解密，直到数字信封输入完毕为止。得到解密后的明文。
4. 调用decryptFinal ()方法结束解密
5. 调用free()方法释放数字信封对象

2-4步可以使用decrypt (cipher)来代替

讨论

[JNI]数字信封的解密可以有很多种方式：

1、默认情况下，不进行任何设置，解开数字信封会从个人证书库里寻找匹配的证书来进行解密的。

2、使用证书库里证书对应的私钥来进行解密。这时要使用addStore (store)方法。由于PFX可以作为证书库来使用(参见4.18)，因此也可以使用这种方式

3、直接指定解密的证书。如果证书本身带有密钥对对象，直接使用addDecryptCertificate方法就行了。否则可以先使用setDevice方法设置解密设备，再使用addDecryptCertificate方法添加解密证书，这时要求证书的私钥在解密设备中。参见6.6

4、直接设置解密的对称密钥。用setKey

过期的证书也是可以正常解开数字信封的

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.cms.EnvelopedData类，具体步骤如下：

1. new EnvelopedData(cipher)解码得到EnvelopedData对象
2. setCertAndPrivKey方法设置使用的解密证书
3. setSymEncrypter方法设置对称解密实现
4. decrypt方法进行解密

[简化封装] 使用net.netca.pki.global.IEnvelopedDataDecrypt接口或者IEnvelopedDataMultiStepDecrypt接口

前者直接使用net.netca.pki.global.Pki类的envelopedDataDecrypt方法就行了。后者的具体步骤如下：

1. 使用Pki对象的getEnvelopedDataMultiStepDecryptObject方法得到IEnvelopedDataMultiStepDecrypt的实现
2. 多次调用decryptUpdate方法输入数字信封数据进行解密，同时得到部分结果，直到所有的数字信封数据输入完毕。
3. 调用decryptFinal方法结束解密
4. 如果IEnvelopedDataMultiStepDecrypt的实现同时实现Freeable，调用Freeable的free方法释放

JCE的实现没有实现IEnvelopedDataMultiStepDecrypt接口

## 加密机解开数字信封

问题

如何使用加密机来解开数字信封？

解决方案

使用net.netca.pki.EnvelopedData类

解开数字信封的步骤如下：

1. 调用new EnvelopedData(false)方法，创建一个解密使用的EnvelopedData对象。
2. 获取加密机的设备对象，并验证其用户密码
3. setDevice方法设置第二步的加密机设备对象
4. addDecryptCertificate方法设置解密证书
5. 调用decryptInit ()方法进行解密初始化
6. 多次调用decryptUpdate (cipher)方法进行解密，直到数字信封输入完毕为止。得到解密后的明文。
7. 调用decryptFinal ()方法结束解密
8. 调用free()方法释放数字信封对象

5-7步可以使用decrypt (cipher)来代替

讨论

[JNI]通常加密机的证书不一定注册到证书库的，因此，要直接设置。上述的方法，还可以使用加密机中来进行对称算法的解密（如果是SM1和SSF33的话）。

[纯JAVA] setCertAndPrivKey方法设置使用加密机的实现就行了

## 获取数字信封的所有接收者信息[JNI]

问题

如何获取数字信封的所有接收者信息？

解决方案

获取数字信封的所有接收者信息的步骤如下：

使用net.netca.pki.EnvelopedData类

解开数字信封的步骤如下：

1. 调用new EnvelopedData(false)方法，创建一个解密使用的EnvelopedData对象。
2. dontDecrypt方法设置不要进行解密
3. 调用decryptInit ()方法进行解密初始化
4. 多次调用decryptUpdate (cipher)方法进行解密，直到数字信封输入完毕为止。
5. 每次调用decryptUpdate 后接着调用getRecipientInfosEncode方法，如果返回null，则还没有足够的数据，不为null则是所有接收者信息的编码，这时free()方法释放数字信封对象，然后返回

讨论

在某些情况下，需要获取数字信封的接收者信息的编码（上面的步骤），然后自己处理，找到匹配的接收者，把对应部分的接收者信息传给它进行处理得到对称密钥，最后才真正解密数字信封，参见6.5

# SignedData

## 带原文的签名

问题

如何产生带原文的签名？

解决方案

使用net.netca.pki.SignedData类

产生带原文的签名的步骤如下：

1. 调用new SignedData(true)创建一个签名的SignedData对象。
2. 调用setDetached(false)设置带签名的签名
3. 调用setSignCertificate (cert)设置签名证书
4. 调用setSignAlgorithm(0,signAlgo)设置签名算法
5. 可选地，使用addCertificate方法添加其他相关的证书
6. 调用signInit()方法初始化签名，得到部分签名的结果
7. 多次调用signUpdate (tbs)方法对原文进行签名，直到所有的原文输入完毕为止。得到部分签名的结果
8. 调用signFinal()方法结束签名，得到最后部分的结果。6-8步的结果串起来就是整个SignedData的编码
9. 调用free()释放SignedData对象

如果需要构造DER编码的SignedData，则把6-8步改为sign(tbs)

讨论

[JNI]SignedData作为一种常用的签名结构，可以带原文也可以不带。带原文的话，整个签名结构可能很大，容易验证，验证可以获取到原文。不带原文的话，无论读多大的数据签名，整个签名结构都很小，验证时，需要先获取到原文，才能验证。

SignedData很复杂的，还可以使用相关的函数来设置相关的属性

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.cms.SignedDataBuilder类，具体步骤如下：

1. new SignedDataBuilder()创建SignedDataBuilder对象
2. new Signer(cert,signer)创建Signer对象
3. 使用Signer对象的setSignatureAlgorithm设置签名算法
4. 可选的，在Signer对象添加签名属性，比如setSigningTime添加签名时间属性
5. 使用addSigner方法添加Signer对象
6. 使用setContent方式设置原文
7. 使用setDetached(false)设置带签名的签名
8. 使用setHasher方法设置Hash的实现
9. 使用sign方法进行签名得到SignedData对象。
10. 使用SignedData对象的encode（true）获得编码

[简化封装]使用net.netca.pki.global.ISignedDataSign接口或者net.netca.pki.global.ISignedDataMultiStepSign接口。前者直接使用net.netca.pki.global.Pki类的signedDataSign(false,cert,tbs)即可。后者的步骤如下：

1. 使用net.netca.pki.global.Pki类的getSignedDataMultiStepSignObject方法获得ISignedDataMultiStepSign的实现
2. 使用signInit方法初始化签名，得到部分签名的结果
3. 多次调用signUpdate方法对原文进行签名，直到所有的原文输入完毕为止。得到部分签名的结果
4. 使用signFinal方法结束签名，得到最后部分的结果。2-4步的结果串起来就是整个SignedData的编码
5. 如果同时实现了Freeable接口，使用Freeable接口的free方法来释放

当前JCE的实现没有实现ISignedDataMultiStepSign

## 不带原文的签名

问题

如何产生不带原文的签名？

解决方案

使用net.netca.pki.SignedData类

产生带原文的签名的步骤如下：

1. 调用new SignedData(true)创建一个签名的SignedData对象。
2. 调用setDetached(true)设置不带签名的签名
3. 调用setSignCertificate (cert)设置签名证书
4. 调用setSignAlgorithm(0,signAlgo)设置签名算法
5. 可选地，使用addCertificate方法添加其他相关的证书
6. 调用signInit()方法初始化签名
7. 多次调用signUpdate (tbs)方法对原文进行签名，直到所有的原文输入完毕为止。
8. 调用signFinal()方法结束签名，得到最终的结果。
9. 调用free()释放SignedData对象

如果可以一次得到原文，也可以把6-8步改为sign(tbs)

讨论

[JNI]不带原文的签名，总是得到DER编码的结果的。signInit和signUpdate产生的0长度的数据，signFinal得到的是整个SignedData的编码。这和带原文的签名是不同的。

验证不带原文的签名时，需要先获取到原文，才能验证。有时候，精确构造出原文是不容易的。使用不带原文的签名，必须考虑到如何精确构造出原文。

[纯JAVA]和带原文的签名类似，只是把setDetached(false)改为setDetached(true)

[简化封装] 使用net.netca.pki.global.ISignedDataDetachedSign接口。如果可以一次得到原文，直接使用net.netca.pki.global.Pki类的signedDataSign(true,cert,tbs)即可。不能一次性得到原文，签名的步骤如下：

1. 使用net.netca.pki.global.Pki类的getSignedDataDetachedSignObject方法获得ISignedDataMultiStepSign的实现
2. 多次调用detachedSignUpdate方法对原文进行签名，直到所有的原文输入完毕为止。
3. 使用detachedSignFinal方法结束签名，得到签名的结果。
4. 如果同时实现了Freeable接口，使用Freeable接口的free方法来释放

## 验证带原文的签名

问题

如何验证带原文的签名？

解决方案

使用net.netca.pki.SignedData类

验证带原文的签名的步骤如下：

1. 调用new SignedData (false)创建一个验证签名的SignedData对象。
2. 可选地，调用addCertificate(cert)方法添加签名证书。如果签名中带有签名证书，则不需要这一步。这里cert不一定就是真正的签名证书
3. 调用verifyInit ()方法进行验证的初始化
4. 多次调用verifyUpdate(sign)方法输入SignedData的编码进行验证，直到所有数据输入为止。得到原文tbs。把每次得到的tbs拼接起来就是完整的原文
5. 调用verifyFinal()方法结束验证签名
6. 检查完整的原文是否符合要求
7. 调用getSignCertificate (index)方法获得签名证书。并自己手工验证该签名证书。参见4.13。注意：可能有多个签名，每个签名会有一个签名证书
8. 调用free()释放SignedData对象

如果能一次获取到整个SignedData的编码，3-5步可以使用verify(sign)来代替

讨论

[JNI]验证的相关方法，对证书只做很少的验证（验证证书的有效期和密钥用法），因此得到签名证书之后，需要自己进行验证，如果需要同时验证证书参见7.28。另外，得到完整的原文后，一定要验证原文是否符合应用的要求。此外，对于不同的应用可能需要验证更多的信息（比如签名算法），这些都需要自己去做

注意：如果是不带任何签名属性的SM2签名的话，使用3-5来验证必然会失败，只有使用verify才会成功。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.cms.SignedData类，具体步骤如下：

1. new SignedData(sign)解析SignedData的编码，得到SignedData对象
2. 使用isDetached方法判断是否带原文
3. 使用getSignerInfoCount获取签名者个数
4. 逐个签名，使用verify(index,verifier,hasher)进行验证
5. 使用getEncapContentInfo().getTbs()获取原文
6. 检查原文是否符合要求
7. 使用getSignCert获取签名证书，并自己手工验证

[简化封装]使用net.netca.pki.global.ISignedDataVerify接口或者net.netca.pki.global. ISignedDataMultiStepVerify接口。前者直接使用net.netca.pki.global.Pki类的signedDataVerify方法即可。后者的步骤如下：

1. 使用net.netca.pki.global.Pki类的getSignedDataMultiStepVerifyObject方法获取ISignedDataMultiStepVerify的实现
2. 可选地，使用addCert方法添加签名证书
3. 多次调用verifyUpdate(sign)方法输入SignedData的编码进行验证，直到所有数据输入为止。得到原文tbs。把每次得到的tbs拼接起来就是完整的原文
4. 调用verifyFinal()方法结束验证签名
5. 检查完整的原文是否符合要求
6. 使用getSignCert方法获得真正的签名证书，并自己手工验证
7. 如果同时实现了Freeable接口，使用Freeable接口的free方法来释放

当前JCE的实现没有实现ISignedDataMultiStepVerify

## 验证不带原文的签名

问题

使用net.netca.pki.SignedData类

如何验证不带原文的签名？

解决方案

验证不带原文的签名的步骤如下：

1、调用new SignedData(false)创建一个验证签名的SignedData对象。

2、可选地，调用addCertificate (cert)方法添加签名证书。如果签名中带有签名证书，则不需要这一步。这里cert不一定就是真正的签名证书

3、调用detachedVerifyInit (sign)解析不带明文的签名。

4、多次调用detachedVerifyUpdate (tbs)方法输入原文，直到原文输完为止。

5、调用detachedVerifyFinal ()方法结束验证签名

6、调用getSignCertificate (index)方法获得签名证书。并自己手工验证该签名证书。参见4.13。注意：可能有多个签名，每个签名会有一个签名证书

7、调用free()释放SignedData对象

如果能一次获取到整个SignedData的编码，3-5步可以使用detachedVerify(tbs,sign)来代替

讨论

[JNI]验证不带原文的签名的关键在于构造出原文。验证的相关方法，对证书只做很少的验证（验证证书的有效期和密钥用法），因此得到签名证书之后，需要自己进行验证，如果需要同时验证证书参见7.28。此外，对于不同的应用可能需要验证更多的信息（比如签名算法），这些都需要自己去做

[纯JAVA] 使用net.netca.pki.encoding.asn1.pki.cms.SignedData类，具体步骤如下：

1. new SignedData(sign)解析SignedData的编码，得到SignedData对象
2. 使用isDetached方法判断是否带原文
3. 使用getSignerInfoCount获取签名者个数
4. 逐个签名，使用verify(index,tbs,verifier,hasher)进行验证
5. 使用getSignCert获取签名证书，并自己手工验证

[简化封装] 使用net.netca.pki.global.ISignedDataDetachedVerify接口，如果能一次获取原文，直接使用net.netca.pki.global.Pki类的signedDataVerify方法即可。多步验证的步骤如下：

1. 使用net.netca.pki.global.Pki类的getSignedDataDetachedVerifyObject方法获取ISignedDataDetachedVerify的实现
2. 可选地，使用addCert方法添加签名证书
3. 调用detachedVerifyInit方法进行解析SignedData的编码
4. 多次调用detachedVerifyUpdate方法输入原文进行验证，直到所有数据输入为止。
5. 调用detachedVerifyFinal ()方法结束验证签名
6. 使用getSignCert方法获得真正的签名证书，并自己手工验证
7. 如果同时实现了Freeable接口，使用Freeable接口的free方法来释放

## 验证不带证书的签名

问题

如何验证不带证书的SignedData签名？

解决方案

和普通验证一样，只是需要手工使用addCertificate (cert)方法添加签名证书。

讨论

[JNI]因为SignedData中没有带有证书，因此需要获取到签名证书才能进行验证，直接使用addCertificate (cert)添加是最简单的方式。但是在某些情况下，不一定会知道签名证书是哪张，这时可以使用NetcaPKIMsgSignedDataAddCert把所有可能的证书都加上。

[纯JAVA]使用verify(index,tbs,verifier,hasher,cert)替代verify(index,tbs,verifier,hasher)即可

[简化封装] 使用addCert方法添加签名证书

## 签名是否带原文

问题

如何判断SignedData是否带原文？

解决方案

使用net.netca.pki.SignedData类的isDetached方法，返回true表示不带原文，返回false表示带原文

讨论

[JNI] SignedData.isSign不需要输入整个SignedData的编码，其得到的结果如果为false，则肯定不是SignedData的编码或者输入数据过少无法判断。为true，则有可能是SignedData的编码。当然也有可能不是，因为这个方法仅仅处理部分的编码，没有处理全部的编码。SignedData.isDetachedSign同样不需要输入整个SignedData的编码，其得到的结果如果为false，则肯定不是不带原文的SignedData的编码。如果SignedData.isSign和SignedData.isDetachedSign都返回true，则可能是不带原文的SignedData。如果SignedData.isSign为true，且SignedData.isDetachedSign为false，则可能是带原文的SignedData。

如果需要准确判断数据是否是SignedData，如果是则判断是否带原文，可以使用下面的步骤：

1. new SignedData (false)创建一个验证签名的SignedData对象
2. 调用setVerifyLevel (SignedData.VERIFY\_LEVEL\_NO\_VERIFY)设置仅仅进行解析，而不进行验证
3. 使用verify (sign)进行解析。
4. 调用isDetached方法，返回true表示不带原文，返回false表示带原文
5. 调用free()释放SignedData对象

[纯JAVA] net.netca.pki.encoding.asn1.pki.cms.SignedData类的isDetached方法

[简化封装] net.netca.pki.global.Pki类的isDetachedSignedData方法

## 过期证书的签名

问题

如何使用过期证书的签名？

解决方案

不支持使用过期证书的签名。

讨论

过期的签名证书是不应该再使用的，因此不支持使用过期证书对应的私钥来进行签名。

## 证书过期后验证签名

问题

在证书过期后，如何验证签名？

解决方案

在验证之前使用setVerifyValidity (false)设置不验证证书的有效期

讨论

[JNI]长期的签名可能会在签名证书过期后，还需要验证的。验证SignedData的签名不难，难在如何验证过期证书，参见4.21

如果带有时间戳的签名，会先验证时间戳的签名，然后使用时间戳的时间来进行验证证书是否在有效期的。另外可以参见7.28

[纯JAVA]不做限制，仅仅验证签名本身，不对证书有效期和其他信息进行验证。所有证书相关的验证都需要自己手工处理

## 不带ContentInfo的SignedData

问题

如何构造不带ContentInfo的SignedData？

解决方案

使用setNoContentInfo ()即可。

讨论

[JNI]对外输出的SignedData，通常都是带ContentInfo的，也就是说，它实际上是个ContentInfo结构。如果在内部使用，通常是需要SignedData的结构。

[纯JAVA] net.netca.pki.encoding.asn1.pki.cms.SignedData对象的encode(false)方法

[简化封装]需要使用Pki的init方法

## 证书的标识

问题

SignedData签名如何选择证书的标识？

解决方案

SignedData可以使用颁发者的名称和证书的序列号来标识签名证书，也可以使用主体密钥标识符来标识签名证书。前者，可以使用setUseSubjectKeyId (index,false)；后者可以使用setUseSubjectKeyId (index,true)。

讨论

[JNI]默认的情况下，SignedData使用颁发者的名称和证书的序列号来标识签名证书，但是建议显式设置。为了提供密码学意义上的绑定，需要设置SigningCertificate或者SigningCertificateV2签名属性。这两个属性都可以提供证书（链）的微缩图，前者使用SHA-1算法，后者可以使用任意的算法。建议只有在维持兼容性的情况下，使用前者，其他情况下，使用后者，并使用一种安全性比SHA-1更高的算法。设置SigningCertificate签名属性，可以使用setUseSigningCertificate (index,true)设置SigningCertificateV2签名属性，可以使用setSigningCertificateV2HashAlgorithm (index,hashAlgo)

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.cms.Signer类的setSignerIdType来设置是使用颁发者的名称和证书的序列号还是使用主体密钥标识符。SigningCertificate和SigningCertificateV2属性要自己处理，以SigningCertificateV2为例，具体如下：

1. ESSCertIDv2 essCertId=new ESSCertIDv2(hashAlgo,cert,hasher,true)一个ESSCertIDv2对象
2. ArrayList<ESSCertIDv2> essCertIdList=new ArrayList<ESSCertIDv2>()一个ESSCertIDv2列表
3. essCertIdList.add(essCertId);
4. SigningCertificateV2 signingCert=new SigningCertificateV2(essCertIdList,null)创建SigningCertificateV2对象
5. Attribute attr=new Attribute(Attribute.SIGNING\_CERTIFICATE\_V2,signingCert.getASN1Object())创建SigningCertificateV2属性
6. signerObj.addSignedAttribute(attr);加到Signer对象中

[简化封装]需要使用Pki的init方法

## SignedData的证书

问题

SignedData签名应该包含哪些证书？

解决方案

SignedData可以包含签名证书也可以不包含。通常，应该包含除根证书外的证书链，以利于验证。可以使用setIncludeCertOption (SignedData.INCLUDE\_CERT\_OPTION\_CERTPATH)

讨论

[JNI]使用setIncludeCertOption，可以设置SignedData包含的证书。INCLUDE\_CERT\_OPTION\_NONE表示不包含签名证书，要验证这样的签名，必须自己寻找到签名证书。INCLUDE\_CERT\_OPTION\_SELF表示包含签名证书，要验证签名证书，必须自己想办法构造证书路径。INCLUDE\_CERT\_OPTION\_CERTPATH表示除根证书外的证书链，这时要使用安全环境来构造证书链，但构造不了证书链不会出错。验证证书的时候，可以从SignedData里的证书来构造证书链。INCLUDE\_CERT\_OPTION\_CERTPATHWITHROOT 和INCLUDE\_CERT\_OPTION\_CERTPATH类似，只是多包含根证书而已。

有时候，需要自己手工添加证书到SignedData里，这时可以使用addCertificate (cert)。这种情况，包括没有配置好安全环境，需要自己构造证书链；添加加密证书等。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.cms.SignedDataBuilder类的addX509PublicKeyCertificate方法手工添加

[简化封装] 需要使用Pki的init方法

## 使用加密机进行签名

问题

如何使用加密机产生SignedData签名？

解决方案

先获取加密机的设备对象，然后验证用户密码，接着在设置签名证书之前，使用setDevice(device)设置加密机的设备对象。

讨论

[JNI]设置了加密设备的对象之后，将在加密设备里寻找匹配的证书的私钥进行签名，而不是象默认情况那样，从个人证书库里寻找匹配的证书的私钥进行签名。

此外，可以根据证书先找到对应的密钥对对象，参见4.7，然后设置setSignCertificate (cert)，也可以实现使用加密机产生SignedData签名。原因是如果证书存在密钥对对象，总是优先使用的。

[纯JAVA]使用支持加密机的设备的Signable实现

## SM2签名

问题

SignedData如何实现SM2签名？

解决方案

产生SM2的签名和其他算法的签名没什么不同。只是如果为了使得可以正常验证，需要设置一个签名属性。

讨论

[JNI]带原文的SM2签名的SignedData，如果没有设置有签名属性的话，在一趟解码验证的情况下，将不能成功验证。也就是使用verifyInit、verifyUpdate和verifyFinal会失败。当然直接使用verify可以成功。原因是验证SM2的签名先需要公钥计算出Z，再用原文进行Hash,而SignedData的结构是先出现原文，然后才是证书的，因此不能验证。在使用签名属性后，SignedData不是直接对原文进行签名，而是对签名属性进行签名。因此，使用SM2签名的SignedData应该使用签名属性，比如：setUseSigningTime (index,true)设置使用签名时间签名属性，这个其实是SM2签名的默认设置。

不带原文的SM2签名的SignedData不管有没有签名属性，总是可以验证成功的。

[纯JAVA]SM2的签名总是可以验证的。

## 符合国密标准的SM2签名

问题

如何产生符合GM/T 0010-2012标准的SM2签名？

解决方案

需要进行以下的设置：

1、证书只能使用SM2的签名证书

2、签名算法只能使用SM3WITHSM2。setSignAlgorithm（index, Signature.SM3WITHSM2）

3、不能使用主体密钥标准符来标识签名者

4、设置使用GM/T 0010-2012标准的签名的OID和签名的签名算法OID，setSM2Q7（index）

其他的步骤和产生普通SignedData签名的一样

讨论

[JNI]通常情况下，进行SignedData签名得到的是符合IETF std70标准的，如果结合SM2来使用的话，SM2的相关格式，遵循GM/T 0009-2012《SM2密码算法使用规范》，签名算法OID使用SM3WithSM2的OID。要产生GM/T 0010-2012《SM2密码算法加密签名消息语法规范》的SignedData，需要设置安装上述步骤进行设置。

GM/T 0010-2012和IETF std70的主要区别在于：GM/T 0010-2012的data和signedData使用了不同的OID，GM/T 0010-2012只支持使用颁发者名和序列号的方式来标识接收者，而IETF std70还可以使用主体密钥标识符来标识。GM/T 0010-2012的签名算法要求使用SM2签名OID。GM/T 0010-2012是参考PKCS#7 V1.5版本的，原文的编码及处理在某些特定的情况下和IETF std70是不同的。此外，IETF std70可以支持更多的证书类型和作废信息类型，签名证书可以使用不同的算法。

验证符合国密标准GM/T 0010-2012的SM2的SignedData和IETF std70的SignedData方法是统一的。

[纯JAVA]使用SignedDataBuilder类的setSM2Q7方法

[简化封装] 需要使用Pki的init方法

## 不带签名属性

问题

如何产生不带签名属性的SignedData？

解决方案

和产生普通SignedData签名的一样，只是在设置签名算法之后，setUseSigningTime (hSignedData, index,false)设置不带签名时间属性，也不要调用任何会添加签名属性的相关方法

讨论

[JNI]通常情况下，SignedData签名所有的属性的默认值都是不添加。除了在SM2签名的情况下，会自动添加签名时间这个签名属性，主要是为了让多步验证的验证方法能正确验证带原文签名，参见7.13。因此，在SM2签名的情况下，需要手工设置不带签名时间属性才可以产生不带签名属性的SignedData。不过这时，可能会导致SignedData可能不能验证。对于其他签名算法的情况下，可以无需手工设置不带签名时间属性（默认就是没有设置），也不会造成不能验证的情况。

[纯JAVA]不会自动添加签名属性，因此，只要你不手工添加就不会带签名属性。

## 产生CAdES-BES

问题

如何产生CAdES-BES的SignedData？

解决方案

和产生普通SignedData签名的一样，设置setSigningCertificateV2HashAlgorithm (index, Hash.SHA256)即可

讨论

[JNI]CAdES-BES是旧的ETSI TS 101 733定义的，新的ETSI EN 319 122不再使用这样的名称，基本上就是新的CAdES-E-BES。

CAdES-BES需要有三个签名属性：Content-type、Message-digest和ESS signing-certificate或者ESS signing-certificate-v2中的一个。其中ESS signing-certificate是使用SHA-1算法的，除非为了兼容某些以前的应用，否则都应该使用ESS signing-certificate-v2设置安全的hash算法。NETCA\_CRYPTO如何添加签名属性的话，会自动添加Content-type和Message-digest属性，因此，只需要设置ESS signing-certificate-v2属性即可。这种方式不能精确控制ESS signing-certificate-v2属性中证书的数量。

还可以使用NetcaPKIMsgSignedDataSetSigningCertificateAttribute方法进行更精细的控制，这个方法的输入可以是单独的签名证书，也可以是整个证书路径。如果使用SHA-1，则是设置ESS signing-certificate属性，否则设置ESS signing-certificate-v2属性。注意：不要同时使用不同的方法多次设置ESS signing-certificate或者ESS signing-certificate-v2属性。

[纯JAVA]自己手工添加ESS signing-certificate-v2属性。参见7.10

[简化封装] 需要使用Pki的init方法

## 验证CAdES-BES

问题

如何验证CAdES-BES的SignedData？

解决方案

验证SignedData后，使用isUseSigningCertificate (index)和getSigningCertificateV2HashAlgorithm (index)来判断是否存在ESS signing-certificate属性和ESS signing-certificate-v2属性。如果有一个存在，则是CAdES-BES

讨论

[JNI]CAdES-BES是旧的ETSI TS 101 733定义的，新的ETSI EN 319 122不再使用这样的名称，基本上就是新的CAdES-E-BES。

如果SignedData带有ESS signing-certificate属性或者ESS signing-certificate-v2属性，在验证的时候，会对其进行验证。因此，验证后，只需要判断是否有ESS signing-certificate属性或者ESS signing-certificate-v2属性即可。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.cms.SignerInfo类的hasSigningCertificateAttribute方法和hasSigningCertificateV2Attribute方法。如果有一个存在，则是CAdES-BES

[简化封装] 需要使用Pki的init方法

## CAdES-EPES[JNI]

问题

如何产生CAdES-EPES的SignedData？

解决方案

CAdES-EPES为CAdES-BES+signature-policy-identifier属性。signature-policy-identifier属性使用setSignaturePolicyId (index,value)来设置。并且根据策略的要求设置，其他相关的属性。

讨论

CAdES-EPES是旧的ETSI TS 101 733定义的，新的ETSI EN 319 122不再使用这样的名称，基本上就是新的CAdES-E-EPES。

当前对CAdES-EPES的支持很不完善。策略值需要手工获取，策略要求的属性也需要手工设置。也没有自动的方式对策论进行验证，需要使用getSignaturePolicyId (index)获取策略值，并自己手工验证。而去还要根据策略的要求，自己手工验证其他相关的属性。

。

## SignedData的签名值

问题

如何获取SignedData的签名值？

解决方案

通常获取SignedData的签名值的意义不大。如果需要的话，可以使用getStringAttribute (22,index)来获取。signature是Hex编码的。

讨论

[JNI]有人希望可以从SignedData中获取到原文的签名值，但是这通常是做不到。原因是SignedData的签名分为两种：一种是对原文的签名，另外一种是对签名属性的签名。如果是前者，SignedData就是原文的签名值。但是除非你限定只能使用前者的方式，否则是没什么意义的。后者由于可以添加大量有用的签名属性，很多应用事实上是要求使用后者的方式的，另外，SM2的签名如果不带签名属性，可能会导致验证失败，参见7.11

带时间戳的签名需要SignedData的签名值，如果你需要手工构造或者验证的话，可以使用。但NETCA CRYPTO本身已经支持带时间戳的签名，参见7.20、7.21、7.22

[纯JAVA] 使用net.netca.pki.encoding.asn1.pki.cms.SignerInfo类的getSignature方法

## 产生带时间戳的签名

问题

如何产生带时间戳的签名？

解决方案

产生带时间戳的签名和普通的签名的步骤类似，只是把signFinal 方法改为signWithTimeStampFinal 方法。或者把sign 方法改为signWithTimeStamp方法即可。

讨论

[JNI]带时间戳的签名是签名后把签名值的Hash值发送给时间戳服务器得到时间戳Token，把时间戳Token作为一个不签名属性加到SignedData中。主要用于证明该签名在某个时间点已经存在。这种时间戳称为signature-time-stamp。

如果已经存在SignedData签名，可以在验证后使用attachSignatureTimeStamp来追加时间戳。

另外一种带时间戳的签名是对原文进行时间戳的，是签名属性，称为content-time-stamp。可以使用setContentTimeStamp来设置。

[纯JAVA]在net.netca.pki.encoding.asn1.pki.cms.Signer类中使用setSignatureTimeStampInfo方法。

也可以得到SignedData对象后，使用SignerInfo类的attachSignatureTimeStamp方法。

[简化封装]使用ISignedDataDetachedSign接口的attachSignatureTimeStamp方法，ISignedDataDetachedVerify接口的attachSignatureTimeStamp方法，ISignedDataSign接口的attachSignatureTimeStamp方法或者ISignedDataVerify接口的attachSignatureTimeStamp方法。

## 验证带时间戳的签名

问题

如何验证带时间戳的签名？

解决方案

验证带时间戳的签名的步骤如下：

1. 按照普通验证签名的方法来验证签名，但不释放SignedData对象。参见7.3、7.4、7.7
2. 使用getSignerCount ()获得签名个数
3. 对逐个签名，调用hasTimeStamp (index),判断是否存在时间戳签名。
4. 如果存在，调用verifyTimeStamp (index)来验证时间戳。这个方法不验证证书的有效期的
5. 调用getTimeStampTime(index)方法获取时间戳的时间
6. 调用getTimeStampTsaCert (index)方法获取时间戳证书。自己验证时间戳证书是否可信
7. 根据5得到的时间，验证签名证书是否有效
8. 重复3-7步，直到所有的时间戳签名验证完毕

9、调用free()释放SignedData对象

讨论

[JNI]验证带时间戳的签名是要在时间戳的时间验证签名是否有效。上面的方法是验证signature-time-stamp的。

如果需要验证content-time-stamp，使用NetcaPKIMsgSignedDataVerifyContentTimeStamp(hSignedData,index, tsaCert,&hTimeStamp)

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.cms.SignerInfo类，验证SignedData后，获取SignerInfo对象，进行以下验证：

1. hasSignatureTimeStamp方法判断是否有签名时间戳
2. 如果存在，使用verifySignatureTimeStamp方法验证时间戳
3. getSignatureTimeStampTime方法获取时间戳时间
4. getSignatureTimeStampCert方法获取时间戳证书，并自己手工验证

## 签名后追加时间戳

问题

签名后，如何追加时间戳？

解决方案

签名后追加时间戳的步骤如下：

1. 按照7.3和7.4进去签名验证，但是在调用验证之前，如果是带原文的签名，需要先调用keepTbs (true)设置保存原来的明文。不要释放SignedData对象。
2. 使用attachSignatureTimeStamp方法追加时间戳
3. 调用encode方法进行重新编码
4. 调用free方法释放SignedData对象

讨论

[JNI]带时间戳的签名是签名后把签名值的Hash值发送给时间戳服务器得到时间戳Token，把时间戳Token作为一个不签名属性加到SignedData中。主要用于证明该签名在某个时间点已经存在。

[纯JAVA] 使用SignerInfo类的attachSignatureTimeStamp方法

[简化封装]使用ISignedDataDetachedSign接口的attachSignatureTimeStamp方法，ISignedDataDetachedVerify接口的attachSignatureTimeStamp方法，ISignedDataSign接口的attachSignatureTimeStamp方法或者ISignedDataVerify接口的attachSignatureTimeStamp方法。

## PKCS#7的签名[JNI]

问题

如何产生PKCS#7的签名？

解决方案

通常产生的SignedData是CMS的SignedData，这在大部分情况下，是兼容PKCS#7的SignedData的，也就是说其实得到的数据也是PKCS#7的SignedData。其最大的区别在于，如果原文的类型不是data，则应该是个BER编码的结构的话，这种情况下，CMS的SigneData和PKCS#7的SignedData是不同的。产生这种情况下的PKCS#7的签名，需要设置setContentOid (contentOid)和setUsePKCS7 (true)。然后只能使用sign (tbs)进行签名，不能使用多步的方式。

讨论

CMS的SignedData是PKCS#7的SignedData的后续版本，大部分新的应用都是要求使用CMS的SignedData的。它们在大部分情况下是兼容的。不兼容的情况，按照上面的方法来处理即可。CMS的SignedData比PKCS#7的SignedData增加了更多的选项，严格来说，产生PKCS#7的SignedData不能使用这些新的东西的。比如：PKCS#7的SignedData定义了使用颁发者名和证书序列号来标识证书，没有定义使用主体密钥标识符的方式。

验证PKCS#7的SignedData和验证CMS的SignedData的方式是统一的。

## 多人签名[JNI]

问题

拿到一个签名后，如何再次签名？

解决方案

多人签名的步骤如下：

1、按照7.3和7.4进去签名验证，但是在调用验证之前，需要先调用keepTbs (true)设置保存原来的明文。不要释放SignedData对象。

2、使用getSignerCount()获得签名个数count

3、调用setSignCertificate (cert)设置签名证书

4、调用setSignAlgorithm (count,hashAlgo)设置签名算法

5、调用coSign ()进行签名。

6、调用free()释放SignedData对象

讨论

多人签名有几种方式：第一种是自己有两张证书，分别用这两张证书进行签名。第二种是甲进行签名了，把签名结果发给乙，乙再进行签名。第三种和第二种类似，但是乙不是对原文进行签名，而是对甲的签名值进行签名，这种方式叫做联署签名。

第一种方式的实现就是有多少张证书调用多少次setSignCertificate (cert)设置签名证书，调用setSignAlgorithm (index,hashAlgo)设置签名算法，以及设置其他的属性。其他的和单个签名是一样的。只不过是单个签名只用设置一次，多人签名需要设置多次。

第二种方式的实现如解决方案所示。

第三种方式的实现参见7.25

验证多人签名和其他签名的验证没有什么不同，只是它会有多个签名而已。

## 联署签名[JNI]

问题

如何实现联署签名？

解决方案

产生联署签名的步骤如下：

1. 按照7.3和7.4进去签名验证，但是在调用验证之前，如果是带原文的签名，需要先调用keepTbs (true)设置保存原来的明文。不要释放SignedData对象。
2. 使用countersign (index,signAlgo,cert,null)进行联署签名

3、调用free()释放SignedData对象

讨论

联署签名是对签名值的签名，因此需要2步中需要指定哪个签名。

## 验证联署签名[JNI]

问题

如何验证联署签名？

解决方案

验证联署签名的步骤如下：

1. 按照普通验证签名的方法来验证签名，但不释放SignedData对象
2. 使用getSignerCount()获得签名个数
3. 对逐个签名，调用getCountersignCount (index)方法,判断联署签名的个数
4. 对每个联署签名，调用verifyCountersign (index, countersignIndex)来验证联署签名。
5. 调用getCountersignCert (index,countersignIndex)获取联署签名证书。手工验证该证书
6. 重复3-5步，直到所有的联署签名都验证完毕
7. 调用free()释放SignedData对象

讨论

联署签名是对签名值的签名。

## P7b文件

问题

如何从P7b文件里获取证书？

解决方案

使用CertStore.loadP7c(data)把P7b文件的内容构造证书库，使用证书库相关方法来获取证书。

讨论

[JNI]P7b或者P7c的格式是一种退化形式的SignedData，也就是没有原文和签名的SignedData，只有证书，通常用于分发证书。NetcaPKICertLoadP7c不但可以处理这种退化形式的SignedData,完整形式的SignedData也可以处理。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.cms.SignedData类的构造函数new SignedData(data)解析P7b文件得到SignedData对象。然后使用getCertificates方法获取证书

## 验证SignedData的时候，同时验证证书

问题

如何在验证SignedData的签名的时候同时对证书进行验证？

解决方案

和普通验证一样，但是需要在验证前添加以下步骤

1. setVerifyLevel (SignedData.VERIFY\_LEVEL\_VERIFY\_CERTPATH\_REVOKE);如果不需要严格的验证可以使用VERIFY\_LEVEL\_VERIFY\_CERT或者VERIFY\_LEVEL\_VERIFY\_CERT\_REVOKE来代替VERIFY\_LEVEL\_VERIFY\_CERTPATH\_REVOKE
2. 使用addRootCert (rootCert)添加根证书
3. 可选的，使用addCertificate (cert)添加其他相关的证书
4. 如果需要验证证书的作废状态，可选的，可以使用addCRL (crl)添加CRL或者使用addOCSP (ocsp)添加OCSP响应

讨论

[JNI]默认验证仅仅验证签名本身，不验证证书。验证证书需要设置根证书，可以直接使用addRootCert方法，也可以使用setUseSystemCertStoreForVerify (true)。如果SignedData中已经包含有足够的证书和CRL等信息，无需进行3-4步的设置

如果验证证书失败会抛出异常，并且Util.getLastError()返回-36，这时可以通过getIntegerAttribute (43,index)获取验证的结果，如果为Util. SUCCESS, getIntegerAttribute (44,index)获取验证证书的状态。如果status为NETCA\_PKI\_CERTIFICATE\_STATUS\_REVOKED或者NETCA\_PKI\_CERTIFICATE\_STATUS\_CA\_REVOKED可以使用getIntegerAttribute (45,index)获取作废原因，getStringAttribute (46,index)获取作废时间。

如果是带时间戳的签名会同时使用相同的验证级别来对时间戳签名进行验证，如果验证成功，则使用时间戳的时间来验证。可以提前使用TimeStamp.addTsaCert设置时间戳证书

[纯JAVA]不支持同时验证证书，必须把证书获取处理，自己手工验证

## CAdES-T

问题

如何产生CAdES-T的SignedData？

解决方案

CAdES-BES+带时间戳的签名就是CAdES-T

讨论

CAdES-T是旧的ETSI TS 101 733定义的，新的ETSI EN 319 122不再使用这样的名称，基本上就是新的CAdES-E-T。

CAdES-BES+带时间戳的签名就是CAdES-T，注意是使用signature-time-stamp不签名属性，不是使用content-time-stamp签名属性。

参见7.16，7.17，7.20，7.21

## CAdES-C[JNI]

问题

如何产生CAdES-C的SignedData？

解决方案

先产生CAdES-T，验证CAdES-T，并保持原文，接着做以下步骤：

1. 获取整个证书链
2. 使用setCompleteCertificateRefs添加整个证书链的引用
3. 获取所有证书的CRL或者OCSP响应
4. 使用addCompleteRevocationRef逐个添加上一步的CRL和OCSP引用
5. 使用encode重新编码

讨论

CAdES-C是旧的ETSI TS 101 733定义的，新的ETSI EN 319 122不再使用这样的名称。

CAdES-C是CAdES-T加上complete-certificate-references和complete-revocation-references属性。也可以先构造CAdES-BES、验证并保持原文、然后使用NetcaPKIMsgSignedDataAttachSignatureTimeStamp、再使用上面的1-5步。注意：时间戳的时间应该保证在CRL和OCSP的有效期内。

验证CAdES-C，首先需要设置验证级别为VERIFY\_LEVEL\_VERIFY\_CERTPATH\_REVOKE，验证SignedData同时验证证书，验证时间戳，成功后，使用getIntegerAttribute (39,index)判断是否存在complete-certificate-references属性，使用verifyCompleteCertificateRefs (index)进行验证complete-certificate-references属性，使用getIntegerAttribute (51,index)判断是否存在complete-revocation-references属性。使用verifyCompleteRevocationRefs (index)进行验证complete-revocation-references属性

## CAdES-X Long[JNI]

问题

如何产生CAdES-X Long的SignedData？

解决方案

先产生CAdES-C，验证CAdES-C，并保持原文，接着做以下步骤：

1. 获取整个证书链
2. 使用setCertificateValues添加整个证书链的引用
3. 获取所有证书的CRL或者OCSP响应
4. 使用setRevocationValues添加上一步的CRL和OCSP引用
5. 使用encode重新编码

讨论

CAdES-X Long是旧的ETSI TS 101 733定义的，新的ETSI EN 319 122不再使用这样的名称。

CAdES-X Long是CAdES-C加上certificate-values和revocation-values属性。这里certificate-values属性要和complete-certificate-references属性匹配，revocation-values属性要和complete-revocation-references属性匹配。也就是1步获取的证书链要和7.30中1步的一样。3步获取的CRL和OCSP要和7.30中3步的一样。

验证CAdES-X LONG，首先按照CAdES-C的方式验证。成功后，使用getIntegerAttribute (52,index)判断是否存在certificate-values属性，使用verifyCertificateValues (index)进行验证certificate-values属性，使用getIntegerAttribute (53,index)判断是否存在revocation-values属性。使用verifyRevocationValues (index)进行验证revocation-values属性

## CAdES-X Type1[JNI]

问题

如何产生CAdES-X Type1的SignedData？

解决方案

先产生CAdES-C，验证CAdES-C，并保持原文，接着做以下步骤：

1. 使用addEscTimeStamp添加CAdES-C-time-stamp时间戳
2. 使用encode重新编码

讨论

CAdES-X Type1是旧的ETSI TS 101 733定义的，新的ETSI EN 319 122不再使用这样的名称。

CAdES-X Type1是CAdES-C加上CAdES-C-time-stamp属性。CAdES-C-time-stamp属性是对签名值、signature-time-stamp、complete-certificate-references和complete-revocation-references属性进行时间戳。CAdES-X Long Type1是CAdES-X Long+ CAdES-C-time-stamp属性。

验证CAdES-X Type1，首先按照CAdES-C的方式验证。成功后，使用使用getEscTimeStampCount (index)获取CAdES-C-time-stamp属性的时间戳个数，然后逐个调用verifyEscTimeStamp进行验证

## CAdES-X Type2[JNI]

问题

如何产生CAdES-X Type2的SignedData？

解决方案

先产生CAdES-C，验证CAdES-C，并保持原文，接着做以下步骤：

1. 使用addCertCRLTimeStamp添加time-stamped-certs-crls-references时间戳
2. 使用encode重新编码

讨论

CAdES-X Type2是旧的ETSI TS 101 733定义的，新的ETSI EN 319 122不再使用这样的名称。

CAdES-X Type2是CAdES-C加上time-stamped-certs-crls-references属性。time-stamped-certs-crls-references属性是对complete-certificate-references和complete-revocation-references属性进行时间戳。CAdES-X Long Type2是CAdES-X Long+ CAdES-C-time-stamp属性。

验证CAdES-X Type2，首先按照CAdES-C的方式验证。成功后，使用getCertCRLTimeStampCount (index)获取time-stamped-certs-crls-references属性的时间戳个数，然后逐个调用verifyCertCRLTimeStamp进行验证

## 使用ATSv2属性的CAdES-A[JNI]

问题

如何产生使用ATSv2属性的CAdES-A的SignedData？

解决方案

先产生CAdES-X Long，验证CAdES-Long，并保持原文，接着做以下步骤：

1. 使用addArchiveTimeStampV2添加archiveTimestampV2时间戳
2. 使用encode重新编码

讨论

使用ATSv2属性的CAdES-A是旧的ETSI TS 101 733定义的，新的ETSI EN 319 122不再使用这样的名称。

使用ATSv2属性的CAdES-A是CAdES-X Long加上archiveTimestampV2属性。archiveTimestampV2属性是对encapContentInfo、certificates、crls和所有的signerInfo项进行时间戳。使用ATSv2属性的CAdES-A也可以是CAdES-X Long Type1 or 2加上archiveTimestampV2属性。

验证使用ATSv2属性的CAdES-A，首先按照CAdES-X Long的方式验证。成功后，使用使用getArchiveTimeStampV2Count (index)获取archiveTimestampV2属性的时间戳个数，然后逐个调用verifyArchiveTimeStampV2进行验证

注意：ATSv2属性由于本身的问题，是很难和其他实现进行互操作，因此不建议使用。另外，验证ATSv2属性的CAdES-A需要全新的一个SignedData对象。

## 使用ATSv3属性的CAdES-A[JNI]

问题

如何产生使用ATSv3属性的CAdES-A的SignedData？

解决方案

先产生CAdES-X Long，验证CAdES-Long，并保持原文，接着做以下步骤：

1. 使用attachATSHashIndex(false,index,hashAlgo)添加ats-hash-index属性
2. 使用addArchiveTimeStampV3方法添加archiveTimestampV3时间戳
3. 使用encode重新编码

讨论

使用ATSv3属性的CAdES-A是旧的ETSI TS 101 733定义的，新的ETSI EN 319 122不再使用这样的名称，新的ATSv3属性和旧的也不相同。

使用ATSv3属性的CAdES-A可以是CAdES-BES等级及其以上的签名加上ats-hash-index属性和archiveTimestampV3属性。archiveTimestampV3属性是对encapContentInfo.eContentType、原文的Hash值、signerInfo的version, sid, digestAlgorithm, signedAttrs, signatureAlgorithm和signature以及ats-hash-index属性进行时间戳。ats-hash-index属性包含certificates域中每个证书的HASH值，crls域中每个作废信息的HASH值，每个不签名属性的Hash值。

验证使用ATSv3属性的CAdES-A，首先按照CAdES-X Long的方式验证。使用getArchiveTimeStampV3Count (index)获取archiveTimestampV3属性的时间戳个数，然后逐个调用verifyArchiveTimeStampV3进行验证，验证archiveTimestampV3时间戳成功后，使用verifyATSHashIndex (index)验证ats-hash-index属性中所有的Hash值都在SignedData中，此外，还需要使用getATSHashIndex (index)获取ats-hash-index属性值来验证证书路径里的证书和真正验证证书路径的作废信息是否匹配。

注意：ETSI EN 319 122标准的archiveTimestampV3属性已经有所改变，ats-hash-index属性变成了ats-hash-index-v3属性。ats-hash-index-v3属性和ats-hash-index属性的主要差别在于不是对不签名属性整个编码进行Hash，而是对不签名属性的签名类型和单个属性值进行Hash。创建这样的archiveTimestampV3属性和上面的方法类似，只是需要把attachATSHashIndex(false,index,hashAlgo)替换为attachATSHashIndex(true,index,hashAlgo)。验证方法也类似，只是在获取ats-hash-index属性值之前，要先用isATSHashIndexV3 (index)来判断是ats-hash-index属性还是ats-hash-index-v3属性，然后才能根据不同的属性进行手工验证不签名属性的Hash值

## 带加密证书的SignedData[JNI]

问题

如何产生带加密证书的SignedData？

解决方案

和产生普通SignedData的方法一样，只是需要添加setEncCertificate这个方法设置加密证书

讨论

在SignedData中添加加密证书是接收方获取加密证书的一种方式。

获取加密证书，需要使用getEncCertificate方法。

## 在SignedData中添加作废信息

问题

如何在SignedData中添加CRL和/或OCSP响应等作废信息呢？

解决方案

使用下面的这些方法的其中一个：

addCRL

addOCSP (SignedData. ADDOCSP\_IN\_REVOCATIONINFOCHOICE,ocsp)

setAdbeRevocationInfoArchival

setRevocationValues

讨论

[JNI]在SignedData中有很多地方都可以添加作废信息。不同的应用需要添加到不同的地方。

在crls中添加CRL，这是常见的应用都支持的情况，直接使用addCRL 来完成。

在crls中添加OCSP响应，这大部分的应用都不支持，可以使用addOCSP (SignedData. ADDOCSP\_IN\_REVOCATIONINFOCHOICE,ocsp)

在Adbe RevocationInfoArchival签名属性中，添加作废信息，这是PDF签名使用的方式，可以使用setAdbeRevocationInfoArchival

在RevocationValues不签名属性中，添加作废信息，这是CadES-X Long的要求，参见7.31。可以使用setRevocationValues

[纯JAVA]如果需要在crls中添加CRL或OCSP，使用net.netca.pki.encoding.asn1.pki.cms. SignedDataBuilder类的addCRL和addOCSP方法，或者net.netca.pki.encoding.asn1.pki.cms. SignedData的setCrls方法。

Adbe RevocationInfoArchival签名属性，不支持

RevocationValues不签名属性，先创建net.netca.pki.encoding.asn1.pki.cms.cades. RevocationValues对象，然后构造对应的属性，使用SignerInfo的addUnsignedAttrs添加进去。

## CAdES-B-B

问题

如何产生CAdES-B-B级别的SignedData签名？

解决方案

和普通的SignedData一样，但需要进行一些设置：

setIncludeCertOption (SignedData.INCLUDE\_CERT\_OPTION\_SELF)设置签名带签名证书

setUseSigningTime (index, true)设置使用签名时间属性

setSigningCertificateV2HashAlgorithm (index, hashAlgo)设置使用ESS signing-certificate-v2属性

讨论

[JNI]CAdES-B-B级别的SignedData签名有以下要求：

1. 必须带有签名证书，证书链上的其他证书，如果AIA属性可以获取或者验证者有办法获取可用不带。注意，如果需要带上其他证书，使用NetcaPKIMsgSignedDataAddCert手工来添加，不要使用setIncludeCertOption (SignedData.INCLUDE\_CERT\_OPTION\_CERTPATHWITHROOT)类似的方式，因为如果没能获取证书链是不会添加CA证书和根证书，而且该方法不会失败。
2. 必须带有签名时间属性
3. 必须带有ESS signing-certificate属性或者ESS signing-certificate-v2属性

[纯JAVA]

1. 设置SignedDataBuilder的setIncludeSignCert(true)包含签名证书，其他证书可以使用addX509PublicKeyCertificate方法。
2. 使用Signer的setSigningTime方法设置签名时间属性
3. 设置ESS signing-certificate-v2属性，参见7.10

## CAdES-B-T

问题

如何产生CAdES-B-T级别的SignedData签名？

解决方案

CAdES-B-T级别的SignedData签名基本上等于CAdES-B-B级别的SignedData签名+签名时间戳属性。因此，只需要在7.38的基础上，使用signWithTimeStamp代替sign。

讨论

[JNI]CAdES-B-T级别的SignedData签名有以下要求：

1. 必须带有签名证书，证书链(包括TSA证书链)上的其他证书，如果AIA属性可以获取或者验证者有办法获取可用不带。注意，如果需要带上其他证书，使用NetcaPKIMsgSignedDataAddCert手工来添加。
2. 必须带有签名时间属性
3. 必须带有ESS signing-certificate属性或者ESS signing-certificate-v2属性
4. 必须带有签名时间戳属性

参见7.29

[纯JAVA] 在7.38的基础上，使用Signer的setSignatureTimeStampInfo方法设置时间戳服务器信息。

如果要在已有的CAdES-B-B上，追加时间戳的话，使用SignerInfo的attachSignatureTimeStamp方法

## CAdES-B-LT

问题

如何产生CAdES-B-LT级别的SignedData签名？

解决方案

CAdES-B-LT级别的SignedData签名基本上等于CAdES-B-T级别的SignedData签名+在crls域添加包括完全作废信息的CRL或者OCSP。

验证完CAdES-B-B级别的SignedData签名后，使用addCRL(crl)添加CRL或者addOCSP (SignedData. ADDOCSP\_IN\_REVOCATIONINFOCHOICE,ocsp)添加OCSP响应

讨论

[JNI]CAdES-B-LT级别的SignedData签名有以下要求：

1. 必须带有签名证书，证书链(包括TSA证书链)上的其他证书，如果AIA属性可以获取或者验证者有办法获取可用不带。注意，如果需要带上其他证书，使用NetcaPKIMsgSignedDataAddCert手工来添加。
2. 必须带有签名时间属性
3. 必须带有ESS signing-certificate属性或者ESS signing-certificate-v2属性
4. 必须带有签名时间戳属性
5. crls域必须带有完整的CRL或者OCSP信息。
6. 不能带有certificate-values、complete-certificatereferences、revocation-values、complete-revocationreferences、attribute-certificatereferences、attribute-revocationreferences、CAdES-C-timestamp、time-stamped-certs-crlsreferences这些属性。

参见7.37

[纯JAVA]验证完CAdES-B-T级别后，使用setCrls方法设置CRL和OCSP

## CAdES-B-LTA

问题

如何产生CAdES-B-LTA级别的SignedData签名？

解决方案

CAdES-B-LTA级别的SignedData签名基本上等于CAdES-B-T级别的SignedData签名+archive-time-stamp-v3属性。

验证完CAdES-B-LT级别的SignedData签名后，使用attachATSHashIndex(true,index,hashAlgo)设置ats-hash-index-v3属性，然后使用addArchiveTimeStampV3来添加archive-time-stamp-v3时间戳

讨论

[JNI]CAdES-B-LT级别的SignedData签名有以下要求：

1. 必须带有签名证书，证书链(包括TSA证书链)上的其他证书，如果AIA属性可以获取或者验证者有办法获取可用不带。注意，如果需要带上其他证书，使用NetcaPKIMsgSignedDataAddCert手工来添加。
2. 必须带有签名时间属性
3. 必须带有ESS signing-certificate属性或者ESS signing-certificate-v2属性
4. 必须带有签名时间戳属性
5. crls域必须带有完整的CRL或者OCSP信息。
6. 不能带有certificate-values、complete-certificatereferences、revocation-values、complete-revocationreferences、attribute-certificatereferences、attribute-revocationreferences、CAdES-C-timestamp、time-stamped-certs-crlsreferences这些属性
7. 必须带有archive-time-stamp-v3时间戳，archive-time-stamp-v3时间戳必须带有ats-hash-index-v3属性，而不是旧的ats-hash-index属性

参见7.35

[纯JAVA] 验证完CAdES-B-LT级别的SignedData签名后，使用new ATSHashIndexV3(atsHashIndexHashAlgo,hasher, signedDataObj.getCertificates(),signedDataObj.getCrls(), signerInfo.getUnsignedAttrs())创建ATSHashIndexV3对象，然后使用signerInfo.attachArchiveTimeStampTokenV3追加archive-time-stamp-v3时间戳

## CAdES-E-A[JNI]

问题

如何产生CAdES-E-A级别的SignedData签名？

解决方案

CAdES-E-A级别的SignedData签名基本上等于CAdES-E-T级别的SignedData签名+所有的证书+所有的作废信息+archive-time-stamp-v3属性。

验证完CAdES-E-T级别（或者CAdES-T）的SignedData签名后，进行以下操作：

1. addCertificate添加证书链的其他证书
2. addCRL (crl)添加CRL
3. addOCSP (SignedData. ADDOCSP\_IN\_REVOCATIONINFOCHOICE,ocsp)添加OCSP响应
4. 使用attachATSHashIndex(true,index,hashAlgo)设置ats-hash-index-v3属性，
5. 使用addArchiveTimeStampV3来添加archive-time-stamp-v3时间戳

2-3根据需要可以只使用其中一步或者两步

讨论

CAdES-E-A级别的SignedData签名有以下要求：

1. 必须带有签名证书及证书链(包括TSA证书链)上的其他证书。
2. 必须带有ESS signing-certificate属性或者ESS signing-certificate-v2属性
3. 必须带有签名时间戳属性
4. 必须带有完整的CRL或者OCSP信息。
5. 必须带有archive-time-stamp-v3时间戳，archive-time-stamp-v3时间戳必须带有ats-hash-index-v3属性，而不是旧的ats-hash-index属性

# 时间戳

## 获取时间戳

问题

如何获取时间戳？

解决方案

使用net.netca.pki.TimeStamp类

获取时间戳的具体步骤如下：

1. 使用new TimeStamp ()方法构造TimeStamp对象
2. 使用setTsaURL (tsaUrl)方法设置时间戳服务器URL
3. 计算数据的Hash值，参见3.2
4. 使用setHashAlgorithm (algo)方法设置Hash算法
5. 使用setMessageImprint (hashValue)方法设置Hash值
6. 使用getResponse ()方法向时间戳服务器发送请求。检查返回的状态是否为TimeStamp.RESP\_STATUS\_GRANTED 或者TimeStamp.RESP\_STATUS\_GRANTEDWITHMODS，如果是则可以获取到时间戳，否则出错。
7. 得到时间戳之后，使用getTsaCertificate ()获取时间戳证书。得到证书后，验证其是否可信
8. 使用getPolicy ()方法得到时间戳策略
9. 使用getSerialNumber()方法得到时间戳序列号
10. 使用getTime()方法获得时间戳的时间值
11. 使用getToken ()方法获得时间戳Token。这个Token可以作为证据保存，以后进行验证。参见8.2
12. 使用free()方法释放TimeStamp对象

讨论

[JNI]当前只支持构造在HTTP协议之上的时间戳服务器，也就是URL开头为http://或者https://的。如果需要使用其他协议的话，需要自己手工获取，参见8.3。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.tsp.HttpGetTimeStamp类，具体步骤如下：

1. new HttpGetTimeStamp()创建HttpGetTimeStamp对象
2. setHashAlgorithm方法设置Hash算法
3. setData方法设置要进行hash的数据
4. getTimeStamp方法向时间戳服务器发送请求，获取时间戳时间
5. getTimeStampCert方法获取时间戳证书，得到证书后，验证其是否可信
6. getTimeStampToken方法获取时间戳Token

[简化封装]使用net.netca.pki.global.IGetTimeStamp接口。具体步骤如下：

1. 使用Pki类的getGetTimeStampObject方法获得IGetTimeStamp的实现
2. 使用getToken方法向时间戳服务器发送请求，获取时间戳Token
3. 使用getTsaCert方法获取时间戳证书。得到证书后，验证其是否可信
4. 使用getPolicy方法获取时间戳策略。
5. 使用getTime方法获取时间戳时间

注意：可能需要使用Pki类的init方法来设置时间戳服务器URL。

## 验证时间戳Token

问题

如何验证时间戳Token，得到时间？

解决方案

验证时间戳Token的步骤如下：

1. 使用new TimeStamp (token,cert) new TimeStamp (token)或者方法来验证时间戳Token的签名，并得到时间戳对象。
2. 使用getTsaCertificate ()获取时间戳证书，这个值可能和1中的证书不同。得到证书后，验证其是否可信
3. 使用getHashAlgorithm ()方法得到Hash算法
4. 使用该Hash算法，对数据进行Hash，参见3.2
5. 使用getMessageImprint ()方法得到Hash值
6. 比较4和5的Hash值是否一致
7. 使用getTime ()方法获得时间戳的时间值
8. 使用getPolicy ()方法得到时间戳策略
9. 使用getSerialNumber ()方法得到时间戳序列号
10. 使用free()方法释放时间戳对象

讨论

[JNI]时间戳Token通常是作为一种证据保存的，验证时间戳Token可以证明数据在某个时间点已经存在。new TimeStamp(token)方法仅仅验证时间戳证书符合某些要求，并不验证时间戳证书是否作废等。如果要进行更多的验证可以使用new TimeStamp(signedData,token)方法，它可以在验证之前设置SignedData对象的验证级别

[纯JAVA]

1. 使用new net.netca.pki.encoding.asn1.pki.cms.SignedData(token)解析时间戳Token
2. 使用net.netca.pki.encoding.asn1.pki.tsp.TimeStampResp.verifyTimeStamp方法验证时间戳
3. SignedData的getSignCert方法获取时间戳证书，并手工验证
4. 使用TimeStampResp 的getTstInfo方法获取TSTInfo对象
5. 使用TSTInfo的getMessageImprint方法获取MessageImprint对象
6. 从MessageImprint中获取Hash算法
7. 使用该Hash算法，对数据进行Hash，参见3.2
8. 从MessageImprint中获取Hash值
9. 比较7和8的Hash值是否一致
10. 使用TSTInfo的getTime获取时间戳时间

[简化封装]使用net.netca.pki.global.IVerifyTimeStamp接口。比如：Pki类的verifyTimeStamp(data,token)方法

## 手工获取时间戳[纯JAVA]

问题

如何获取其他协议的时间戳？

解决方案

先使用net.netca.pki.encoding.asn1.pki.tsp.TimeStampReqBuilder构造时间戳请求，然后把证书请求发送给时间戳服务器获得时间戳响应，使用net.netca.pki.encoding.asn1.pki.tsp.TimeStampResp处理响应，具体步骤如下：

1. 根据Hash算法计算数据的Hash值，参见3.2
2. 构造包含该Hash算法和Hash值的MessageImprint对象
3. 使用new TimeStampReqBuilder()方法构造TimeStampReqBuilder对象
4. 使用setMessageImprint方法设置MessageImprint对象
5. 使用setNonce 设置Nonce值
6. 使用build方法得到TimeStampReq对象
7. 使用TimeStampReq对象的derEncode方法获得时间戳请求
8. 使用其他的方式把时间戳请求发送到时间戳服务器来获取时间戳响应。
9. 得到时间戳响应后，使用 new TimeStampResp (resp)解析响应得到TimeStampResp对象
10. 使用getStatus().getStatus()获取响应的状态，如果为PKIStatusInfo.GRANTED RANTED或者PKIStatusInfo.GRANTEDWITHMODS，则得到的时间戳token是有效的
11. 使用getTimeStampTokenObject()方法得到SignedData对象
12. 使用TimeStampResp.verifyTimeStamp验证时间戳签名
13. 使用SignedData对象的getSignCert(0)获取签名证书，并验证其是否可信
14. 使用TimeStampResp对象的getTstInfo (hNewTimeStamp)方法TSTInfo对象
15. 从TSTInfo对象获取Hash算法和Hash值，并和1中比较是否相同
16. 使用TSTInfo对象的getTime方法获得时间戳的时间值

讨论

当前只支持自动获取构造在HTTP协议之上的时间戳服务器。其他协议的需要得到时间戳请求后，根据对应的协议构造合适的请求，发送给服务器，然后获取响应，这些步骤（也就是上面的8），需要自己来实现。

## 获取不带签名证书的时间戳

问题

如何获取不带签名证书的时间戳？

解决方案

和普通的获取时间戳一样，只是在请求时，增加setCertReq (false)要求响应不带时间戳证书。为了验证响应还需要使用setTsaCert设置时间戳证书。

讨论

[JNI]不带签名证书的时间戳可以减少时间戳响应的大小，但是需要知道时间戳证书才能验证请求。

还可以通过TimeStamp.addTsaCert来设置时间戳证书。

[纯JAVA]需要HttpGetTimeStamp类的setCertReq(false)设置响应不带时间戳证书，setTimeStampCert设置时间戳证书。

# S/MIME

## 简单的MIME[JNI]

问题

如何创建简单的MIME？

解决方案

使用net.netca.pki.Smime类，创建简单的MIME的具体步骤如下：

1. 使用new Smime (Smime.MIME\_TYPE\_BASIC)方法构造Smime对象
2. 使用addHeader方法来逐个添加MIME头
3. 使用setBody方法设置MIME体
4. 使用encode方法得到MIME的编码
5. 使用free()方法释放Smime对象

讨论

MIME通常是使用ASCII编码的，分为MIME头和MIME体两部分。MIME头是一组名值对。NETCA CRYPTO把值分号后面的部分作为参数。

## MIME头使用中文[JNI]

问题

如何在MIME头中使用中文？

解决方案

使用Smime.encodeMimeHeaderExts方法先进行编码再加进MIME头中

讨论

通常是使用Smime.encodeMimeHeaderExts把要编码的部分编码为ASCII码的。有时候仅仅是一个头项的某部分需要进行编码不是所有的部分都需要编码。编码后的结果格式为=?xxx?x?xxx?=。

Smime.decodeMimeHeaderExtsCharset、Smime.decodeMimeHeaderExtsEncode和Smime. decodeMimeHeaderExts可以对encodeMimeHeaderExts编码的结果进行解码。注意要先找到encodeMimeHeaderExts编码的部分。

## MIME体使用中文[JNI]

问题

如何在MIME体中使用中文？

解决方案

简单来说就是要对MIME体进行编码和在MIME头中设置相应的传输编码头。

具体来说有两种方式：QuotedPrintable编码和Base64编码。

QuotedPrintable编码的方式如下：

1. addHeader（pos, "Content-Transfer-Encoding: quoted-printable"）
2. 使用Smime.encodeQuotedPrintable进行QuotedPrintable编码
3. 把上一步的结果使用setBody方法设置MIME体

其他部分和简单MIME一样

Base64编码的方式如下：

1. addHeader（pos, "Content-Transfer-Encoding: base64"）
2. 使用Base64.encode方法或者其他方法进行Base64编码
3. 把上一步的结果使用setBody方法设置MIME体

其他部分和简单MIME一样

讨论

如果MIME体内容是简单的文本，通常还需要设置Content-Type头来指定其使用的字符集。

如果MIME体仅仅有少量的中文，其他大部分是ASCII码，则使用QuotedPrintable编码得到的编码较短。其他情况使用Base64编码更短。建议优先使用Base64编码

## 多部分的MIME[JNI]

问题

如何创建多部分的MIME？

解决方案

创建多部分的MIME的具体步骤如下：

1. 使用new Smime (Smime.MIME\_TYPE\_MULTIPART)方法构造Smime对象
2. 使用getBoundary获取边界值
3. 使用addHeader方法添加Content-Type头，其中要包括上一步的边界值
4. 使用addHeader方法来逐个添加其他的MIME头
5. 使用addSubEntity方法逐个添加包含的子MIME
6. 使用encode方法得到MIME的编码
7. 使用free方法释放MIME对象

讨论

多部分的MIME会包含多个子MIME，它们使用特定的边界字符串来分隔。多部分的MIME的Content-Type要包括正确的类型以及使用边界字符串。

子MIME可以是简单的MIME也可以是多部分的MIME

## 包含附件的MIME[JNI]

问题

如何创建包含附件的MIME？

解决方案

包含附件的MIME是多部分的MIME，创建方法参见9.4。附件的MIME可以使用Smime.newAttachment方法来创建，把它使用addSubEntity添加进去即可

讨论

附件其实是把文件的内容Base64编码放入MIME体中，并设置相应的MIME头而已，使用Smime.newAttachment是最简单的方法

## 获取MIME的内容[JNI]

问题

如何获取MIME的内容？

解决方案

获取MIME的内容的具体步骤如下：

1. 使用new Smime(mimeEncode)方法解析得到Smime对象
2. 使用getHeaderCount和getHeaderString方法来获取MIME头
3. 如果MIME头经有编码的部分需要解码，参见9.2
4. 使用getType方法获取MIME的类型
5. 如果是简单的基本类型，使用getBody获取MIME体，如果存在传输编码，则还要进行解码处理
6. 如果是多部分MIME使用getSubEntity来获取子MIME
7. 使用free方法释放Smime对象

讨论

MIME的信息获取，主要是要根据不同的MIME头和类型进行不同的处理。

## 创建签名的S/MIME[JNI]

问题

如何创建签名的S/MIME？

解决方案

创建签名的MIME的具体步骤如下：

1. 先创建要进行签名的MIME，这通常是个普通的MIME，参见9.1、9.2、9.3、9.4、9.5
2. 创建SignedData对象，并进行相关设置
3. 使用sign方法对第1步的MIME进行签名，得到一个新的签名的S/MIME
4. 使用free方法释放Smime对象

讨论

签名的S/MIME本身也是MIME，它是对MIME进行签名。签名的S/MIME有两种方式：一种是带原文的SignedData，也就是产生一个简单的MIME，其MIME体就是这个带原文的SignedData的Base64编码。另外一种方式是产生一个多部分的MIME，第1个子MIME就是要签名的MIME，第2个子MIME是个简单的MIME，其MIME体是不带原文的SignedData的Base64编码。这种方式对于旧式不支持S/MIME的客户端来说，可以直接看到原始的MIME，但是如果在传输过程中原始的MIME被改变的话，则无法验证签名

## 是否签名的S/MIME[JNI]

问题

如何判读是否签名的S/MIME？

解决方案

直接使用isSign()和isDetachedSign()其中一个返回true就是签名的S/MIME

讨论

isSign为真表示带原文的S/MIME，isDetachedSign为真表示多部分的S/MIME

## 验证签名的S/MIME[JNI]

问题

如何验证签名的S/MIME？

解决方案

使用verify方法即可

讨论

verify(signedData)方法可以在验证的时候设置要验证的级别和其他相关的参数。验证完签名后，需要自己获取原文的MIME来验证

## 产生加密的S/MIME[JNI]

问题

如何产生加密的S/MIME？

解决方案

1. 先创建要进行加密的MIME，这通常是个普通的MIME，也可以是签名的S/MIME参见9.1、9.2、9.3、9.4、9.5、9.7
2. 创建数字信封对象，并进行相关设置
3. 使用encrypt方法对第1步的MIME进行加密，得到一个新的加密的S/MIME
4. 使用free方法释放Smime对象

讨论

加密的S/MIME本身也是MIME，它是对明文的MIME进行数字信封，得到的是个简单的MIME，其MIME体是数字信封的Base64编码

## 是否加密的S/MIME[JNI]

问题

如何判断是否加密的S/MIME？

解决方案

使用isEncrypt方法即可

讨论

isEncrypt方法可以判断是否加密的S/MIME

## 解密加密的S/MIME[JNI]

问题

如何对加密的S/MIME进行解密？

解决方案

使用decrypt方法即可

讨论

decrypt(envelopedData)可以设置更多的解密相关参数

# JOSE

## 创建JWS签名

问题

如何创建JWS签名？

解决方案

使用net.netca.pki.JWS类，创建JWS签名的具体步骤如下：

1. new JWS (int type),创建一个JWS对象用于签名或者MAC。type为JWS.COMPACT\_SERIALIZATION，则产生紧凑格式，为JWS.JSON\_SERIALIZATION，则产生JSON格式
2. 使用setCertIdType方法设置签名证书的标识。
3. 使用setPayload方法设置有效载荷
4. 使用sign方法进行签名，得到JWS的编码
5. 使用free方法释放JWS对象

讨论

JWS签名可以分为紧凑格式和JSON格式两种。JWS签名包括数字签名和MAC。

[JNI] 使用net.netca.pki.JWS类表示JWS签名

[纯JAVA]使用net.netca.pki.encoding.json.jose.JWS表示JWS签名，net.netca.pki.encoding.json.jose.JWSBuilder类来产生JWS对象，创建JWS签名的具体步骤如下：

1、使用JWSBuilder.getInstance(type)来创建JWSBuilder对象。type为JWS.COMPACT\_SERIALIZATION，则产生紧凑格式，为JWS.JSON\_SERIALIZATION，则产生JSON格式

2、使用setHashImplement方法设置Hash的实现

3、使用setPayload方法设置有效载荷

4、net.netca.pki.encoding.json.jose.JWSSigner.getInstance()创建一个JWSSigner对象。

5、使用JWSSigner的setSignCert方法设置签名证书相关信息

6、可选地，使用JWSSigner的addProtectedHeader方法添加保护头

7、可选地，使用JWSSigner的addUnProtectedHeader方法添加不保护头

8、使用JWSBuilder的addSigner方法添加JWSSigner对象

9、使用JWSBuilder的sign方法进行数字签名或者MAC，得到JWS对象

10、使用JWS的encode方法得到JWS的编码

## 创建不带有效载荷的JWS

问题

如何创建不带有效载荷的JWS？

解决方案

只需要在创建JWS签名或者MAC中（参见[10.1](#_创建JWS签名)、[10.4](#_创建JWS的MAC)），使用setFlag (JWS.DETACHED\_FLAG)进行设置即可

讨论

通常的JWS都是带有有效载荷的。有时候为了省略空间，在有效载荷有办法从其他地方获取的情况下，可以使用不带有效载荷的JWS。

[JNI]使用JWS的setFlag (JWS.DETACHED\_FLAG)进行设置

[纯JAVA]使用JWSBuilder的setFlag(JWS.FLAG\_DETACHED)进行设置

## 创建不对有效载荷进行Base64Url编码的JWS

问题

如何创建有效载荷不进行Base64Url的JWS？

解决方案

只需要在创建JWS签名或者MAC中（参见[10.1](#_创建JWS签名)、[10.4](#_创建JWS的MAC)），使用setFlag (JWS. UNENCODED\_PAYLOAD\_FLAG)进行设置即可

讨论

通常的JWS的有效载荷是使用Base64Url编码的，但有时候想不进行编码。不对有效载荷进行Base64Url编码存在一些问题，不能使用一些特殊的字符，也就是不能处理任意二进制数据，因此，如非必须不要使用。

[JNI]使用JWS的setFlag (JWS.UNENCODED\_PAYLOAD\_FLAG)进行设置

[纯JAVA]使用JWSBuilder的setFlag(JWS.FLAG\_USE\_UNENCODED\_PAYLOAD)进行设置

## 创建JWS的MAC

问题

如何创建JWS的MAC？

解决方案

使用net.netca.pki.JWS类，创建JWS MAC的具体步骤如下：

1. new JWS (int type),创建一个JWS对象用于签名或者MAC。type为JWS.COMPACT\_SERIALIZATION，则产生紧凑格式，为JWS.JSON\_SERIALIZATION，则产生JSON格式
2. 使用setPayload方法设置有效载荷
3. 使用mac方法进行进行MAC运算，得到JWS的编码
4. 使用free方法释放JWS对象

讨论

JWS的MAC和签名的主要不同在于最后的签名值部分是MAC值而不是签名值。

通常有效载荷是使用Base64Url编码的，如果不需要有效载荷或者有效载荷不做编码，可以参见[10.2](#_创建不带有效载荷的JWS)、[10.3](#_创建不对有效载荷进行Base64Url编码的JWS)

[JNI] 使用net.netca.pki.JWS类表示JWS签名

[纯JAVA]使用net.netca.pki.encoding.json.jose.JWS表示JWS签名，net.netca.pki.encoding.json.jose.JWSBuilder类来产生JWS对象，创建JWS MAC的具体步骤如下：具体步骤如下：

1、使用JWSBuilder.getInstance(type)来创建JWSBuilder对象。type为JWS.COMPACT\_SERIALIZATION，则产生紧凑格式，为JWS.JSON\_SERIALIZATION，则产生JSON格式

2、使用setPayload方法设置有效载荷

3、net.netca.pki.encoding.json.jose.JWSSigner.getInstance()创建一个JWSSigner对象。

4、使用JWSSigner的setMacImplement方法设置MAC实现

5、使用JWSSigner的setMacKey方法设置MAC的算法和密钥

6、可选地，使用JWSSigner的addProtectedHeader方法添加保护头

7、可选地，使用JWSSigner的addUnProtectedHeader方法添加不保护头

8、使用JWSBuilder的addSigner方法添加JWSSigner对象

9、使用JWSBuilder的sign方法进行MAC，得到JWS对象

10、使用JWS的encode方法得到JWS的编码

## 验证JWS

问题

如何验证JWS？

解决方案

使用net.netca.pki.JWS类，验证JWS的具体步骤如下：

1. 使用new JWS (type, jwsEncode)解码得到JWS对象
2. 使用getSignCount() 方法获得签名的个数
3. 按照4-6步逐个验证签名
4. 使用isSignature (index)判断是签名还是MAC
5. 如果是签名则使用verifySignature (index,cert)进行验证签名值本身。这个函数不验证证书，还需要自己用其他的方式来验证证书（参见[4.20](#_验证证书)、[4.21](#_验证过期证书)）。
6. 如果是MAC则使用verifyMac (index,key)进行验证MAC
7. 使用getAllCriticalHeader (index)获取所以关键的头的名称。并自己获取关键头的值来处理。
8. 可选的，使用getProtectedHeader (index)来获取保护头
9. 可选的，使用getUnprotectedHeader (index)来获取不保护头
10. 使用getPayload ()来获取有效载荷，并自己进行验证
11. 使用free()释放JWS对象

讨论

验证JWS，需要处理所有的关键头，还需要验证有效载荷是否符合要求。

[JNI]验证JWS签名，需要验证JWS签名证书。如果JWS本身带有签名证书，可以使用verifySignature (index,( Certificate)null)验证签名值本身，然后使用getSignCert (index)获取签名证书，然后使用其他的方式来验证证书（参见[4.20](#_验证证书)、[4.21](#_验证过期证书)）。如果JWS签名不带证书，而是带公钥，当前的实现是不支持直接从JWS编码中获取公钥来验证。对于没有证书的JWS签名仅知道公钥，可以使用verifySignature (index,publicKey)来验证签名值。

如果是不带有效载荷的JWS，需要先使用setPayload (payLaod)设置有效载荷才能验证。

[纯JAVA] 使用net.netca.pki.encoding.json.jose.JWS来验证，具体步骤如下:

1. new JWS(type,encode)创建JWS对象
2. getSignerCount()获得签名个数
3. 逐个调用getSignerInfo（index）获取签名信息，得到JWSSignerInfo对象
4. 使用JWSSignerInfo的isSignature()判断是否签名
5. 如果是签名，使用JWS 的setHashImplement设置Hash实现，setVerifyImplement设置验证签名实现，然后使用verifySignature验证签名
6. 如果是MAC，则使用JWS 的setMacImplement设置MAC实现，然后使用verifyMac验证MAC
7. 使用getCritHeaderCount和getCritHeaderValue获取所有的关键头，并自己进行相应的处理
8. 使用getPayload获取有效载荷，并自己进行验证

如果是不带有效载荷的JWS，需要先使用setPayload (payLaod)设置有效载荷才能验证。当前并不支持使用公钥来验证JWS

## 多人JWS[JNI]

问题

如何在JWS带多个签名？

解决方案

先验证JWS（参见[10.5](#_验证JWS)），但不释放JWS对象

然后再使用同一JWS对象进行签名（参见[10.1](#_创建JWS签名)、[10.4](#_创建JWS的MAC)），但无需再设置有效载荷。

讨论

多个JWS签名仅仅支持JSON对象表示的JSON格式，不支持紧凑格式

## 创建JWE

问题

如何创建JWE？

解决方案

使用net.netca.pki.JWE类，创建JWE的步骤如下：

1. new JWE(type)函数创建一个JWE对象
2. 使用setContentEncAlgo (algo)设置明文加密算法
3. 可选地，使用setAdditionalProtectedHeader (header)设置其他的保护头
4. 可选地，使用setSharedUnprotectedHeader (header)设置共享的不保护头。紧凑格式的JWE不能有这一步
5. 可选地，使用setAAD (aad)设置AAD。紧凑格式的JWE不能有这一步
6. 逐个使用addCert (algo,cert,certIdType)或者addPwd(algo,pwd, saltLen,iterCount）添加接收者信息
7. 使用encrypt (clear)进行加密，得到JWE编码
8. 使用free()释放JWE对象

讨论

JSON Web Encryption(JWE)是JSON数字信封加密的一种标准。JWE包括五部分：头部、加密后的CEK、IV值、密文和完整性校验的Tag。JWE可以有一个或者多个接收者，每个接收者都有自己的头部（可选）和加密后的CEK，IV值、密文、完整性校验的Tag以及某些头部是多个接收者共享的。头部以JSON对象的方式来表示，包含保护头部、共享的不保护头部和接收者自己的头部三种，保护头部是受到完整性保护的，共享的不保护头部和接收者自己头部则不受完整性保护，共享的不保护头部是所有接收者共享的，接收者自己头部是每个接收者自己的。

JWE的明文可以是任意的数据。其加密过程如下：首先产生内容加密密钥（CEK，可能还包含MAC密钥）和IV，然后对明文进行带完整性的加密，其中保护头部、IV、和AAD（可选）是不加密但需要完整性保护的，得到密文和完整性校验的Tag。然后每个接收者对CEK进行加密得到加密后的CEK，接收者可以使用证书、对称密钥、口令等多种方式来进行加密。把上面两步的数据组合起来就得到完整的JWE。

JWE有两种表示：一种是以.分隔的紧凑格式，一种是JSON对象表示的JSON格式。紧凑格式仅仅能表示一个接收者的加密，不包含共享的不保护头部和接收者自己头部，也不包含AAD。

[JNI]使用net.netca.pki.JWE类

[纯JAVA] 使用net.netca.pki.encoding.json.jose.JWE表示JWE加密，net.netca.pki.encoding.json.jose.JWEBuilder类来产生JWE对象，创建JWE的具体步骤如下：

1. JWEBuilder.getInstance(type)创建JWEBuilder对象
2. setContentEncAlgo(algo)设置明文加密算法
3. 可选地，使用addProtectedHeader(key,value)添加其他的保护头
4. 可选地，使用addShareUnProtectedHeader (key,value)添加共享的不保护头。紧凑格式的JWE不能有这一步
5. 可选地，使用setAad(aad) 设置AAD。紧凑格式的JWE不能有这一步
6. 使用setSecureRandomGenerator设置随机数发生器
7. 使用setSymEncrypter设置对称加密实现
8. new JWERecipienter()一个接收者对象
9. 如果是使用公钥加密，使用JWEBuilder的setPublicKeyEncrypter设置公钥加密实现，setHashImplement设置Hash实现，然后使用JWERecipienter的setCert方法设置加密证书
10. 如果是使用口令加密，使用JWEBuilder的setKDFImplement设置KDF实现，使用setKeyWrapImplement设置KeyWrap实现，然后使用JWERecipienter的setPBKDF2Params方法设置口令导出参数，setKey设置口令。
11. 使用addRecipienter添加上面创建的接收者
12. 使用encrypt进行加密得到JWE对象
13. 使用JWE的encode方法得到JWE的编码

## 解密JWE

问题

如何解密JWE？

解决方案

使用net.netca.pki.JWE类，解密JWE的步骤如下：

1. 使用new JWE(type, jweEncode)解码JWE得到JWE对象
2. 使用getRecipientCount ()获得接收者个数
3. 如果是使用证书的接收者且有证书标识，可以使用setStore(store)函数来设置解密证书所在的证书库
4. 如果是使用证书的接收者且知道解密证书，可以使用setDecryptCert (index,cert)函数来设置解密证书
5. 如果是使用口令的接收者，可以使用setPwd (index,pwd)函数来设置解密需要的口令。
6. 使用decrypt()进行解密，得到明文
7. 使用getAllCriticalHeader (index)获取所以关键的头的名称。并自己获取关键头的值来处理。
8. 使用free()释放JWE对象

讨论

[JNI]JWE解密需要根据加密CEK的算法来确定怎么样解密，这个算法可以通过getCEKEncryptAlgo(index)来获取。

[纯JAVA] 使用net.netca.pki.encoding.json.jose.JWE表示JWE加密，具体步骤如下：

1. JWE.decode(type, jweEncode) 解码JWE得到JWE对象
2. 使用getRecipientCount()获得接收者个数
3. 使用setSymEncrypter设置对称解密实现
4. 如果是使用证书的接收者，则使用setHashImplement设置Hash实现，setX509CertificateAndPrivateKey设置解密使用的证书及其相关私钥。
5. 如果是使用口令的接收者，则使用setKDFImplement设置KDF实现，setKeyUnWrapImplement设置KeyUnWrap实现。然后使用getRecipientInfo(index).setDecryptKey(key)设置口令
6. 使用decrypt进行解密得到明文

## 创建签名的JWT

问题

如何创建签名的JWT？

解决方案

使用net.netca.pki.JWT类，创建签名的JWT步骤如下：

1. 使用new JWT(JWT.JWS\_TYPE)创建签名的JWT对象
2. 使用setClaim (name,value)设置声称
3. 可选的，使用setAdditionalProtectedHeader(header)设置其他保护头
4. 使用sign方法进行签名或者mac方法进行MAC运算,得到JWT的编码。
5. 使用free释放JWT对象

讨论

JSON Web Token (JWT)是一种URL安全的方式来表示声称（claims）。JWT可以使用紧凑的JWS的格式也可以使用紧凑的JWE的格式。对于JWS，有效载荷是JSON对象表示的声称或者JWT。对于JWE，明文是JSON对象表示的声称或者JWT。如果有效载荷或者明文是JWT，则是嵌套形式的JWT，这时，需要设置cty头的值为JWT。

创建签名的JWT，关键在于设置声称。

[JNI]使用net.netca.pki.JWT类

[纯JAVA] 使用net.netca.pki.encoding.json.jose.JWT表示JWT，net.netca.pki.encoding.json.jose.JWTBuilder类来产生JWT对象，创建JWT签名的具体步骤如下：

1. JWTBuilder.getInstance(JWT. TYPE\_JWS) 签名的JWTBuilder对象
2. 使用setClaimsSet设置声称
3. 如果是签名，使用setSignCert设置签名证书及其相关信息，然后使用sign进行签名得到JWT对象
4. 如果是MAC，使用setMacImplement设置MAC实现，setKey设置MAC的Key，然后使用mac进行MAC运算得到JWT对象
5. 使用JWT的encode方法得到JWT的编码

## 验证JWT签名

问题

如何创建验证JWT签名？

解决方案

使用net.netca.pki.JWT类，验证JWT签名步骤如下：

1. 使用new JWT(jwtEncode)函数解码得到JWT对象
2. 使用getType()获取JWT的类型，确保为签名(JWT.JWS\_TYPE)
3. 使用verifySignature (cert)验证签名值，并自己手工验证证书（参见[4.20](#_验证证书)、[4.21](#_验证过期证书)）或者verifyMac (key)验证MAC
4. 使用getClaimValue 或者getClaimStringValue获取声称的值。或者使用getIssuedAt等方法获取标准声称的值
5. 使用free释放JWT对象

讨论

[JNI]验证签名和JWS一样，仅仅验证签名值，不验证证书，证书需要自己去验证

如果是有效载荷是嵌套的JWT，需要用getNestedJWT获取子JWT，而不是直接获取声称的值

[纯JAVA] 使用net.netca.pki.encoding.json.jose.JWT来验证JWT的签名，步骤如下：

1. JWT.decode(jwtEncode)解码得到JWT对象
2. getType获取JWT的类型，确保为签名(JWT.TYPE\_JWS)
3. 如果是数字签名，使用setHashImplement设置Hash实现，setVerifyImplement设置验证实现，verifySignature进行签名验证。并自己手工验证证书（参见[4.20](#_验证证书)、[4.21](#_验证过期证书)）
4. 如果是MAC，使用setMacImplement设置MAC实现，verifyMac进行MAC验证
5. getClaimsSet获取声称

## 创建加密的JWT

问题

如何创建加密的JWT？

解决方案

使用net.netca.pki.JWT类，创建加密的JWT步骤如下：

1. 使用new JWT(JWT.JWE\_TYPE)创建JWT对象
2. 使用setClaim (name,value)设置声称
3. 可选的，使用setAdditionalProtectedHeader(header)设置其他保护头
4. 使用encryptByCert 方法或者encryptByPwd 方法进行加密, 得到JWT的编码。
5. 使用frer释放JWT对象

讨论

[JNI]使用net.netca.pki.JWT类

[纯JAVA] 使用net.netca.pki.encoding.json.jose.JWT表示JWT，net.netca.pki.encoding.json.jose.JWTBuilder类来产生JWT对象，创建加密的JWT的具体步骤如下：

1. JWTBuilder.getInstance(JWT. TYPE\_JWE) 签名的JWTBuilder对象
2. 使用setClaimsSet设置声称
3. 使用setSymEncrypter设置对称加密实现
4. 如果是证书加密，使用setPublicKeyEncrypter设置公钥加密实现，然后setEncryptCert设置加密证
5. 如果是口令加密，使用setKDFImplement设置KDF实现，setKeyWrapImplement设置KeyWrap实现，setPBKDF2Params设置导出口令的相关参数，然后使用setKey口令
6. 使用encrypt进行加密得到JWT对象
7. 使用JWT的encode方法得到JWT的编码

## 解密JWT

问题

如何解密JWT？

解决方案

使用net.netca.pki.JWT类，解密JWT步骤如下：

1. 使用new JWT(jwtEncode)解码得到JWT对象
2. getType()获取JWT的类型，确保为签名(JWT.JWE\_TYPE)
3. 使用decryptByCert (cert)或者decryptByPwd (key)进行解密
4. 使用getClaimValue 或者getClaimStringValue获取声称的值。或者使用getIssuedAt等方法获取标准声称的值
5. 使用free释放JWT对象

讨论

[JNI]如果是有效载荷是嵌套的JWT，需要用getNestedJWT获取子JWT，而不是直接获取声称的值

[纯JAVA] 使用net.netca.pki.encoding.json.jose.JWT来解密JWT，步骤如下：

1、JWT.decode(jwtEncode)解码得到JWT对象

2、getType获取JWT的类型，确保为签名(JWT.TYPE\_JWE)

3、使用setSymEncrypter设置对称解密实现

4、如果是证书加密，使用setHashImplement设置Hash实现，setX509CertificateAndPrivateKey设置解密证书及私钥

5、如果是口令加密，使用setKDFImplement设置KDF实现，setKeyUnWrapImplement设置keyunwrap实现，setDecryptKey设置解密Key

6、decrypt进行解密

7、getClaimsSet获取声称

## 嵌套的JWT

问题

如何创建嵌套的JWT？

解决方案

先创建子JWT(subJWT)，然后根据[10.9](#_创建签名的JWT)或者[10.11](#_创建加密的JWT)的方法创建JWT，只是把setClaim替换为setNestedJWT (subJWT)

讨论

嵌套的JWT只是有效载荷或明文是JWT的编码，而不是声称。嵌套的JWT需要有cty头，其值为JWT。

[JNI]使用setNestedJWT。cty头是自动添加的，无需手工设置

可以使用hasNestedJWT来判断是否嵌套JWT，如果是，使用getNestedJWT来获取内层的JWT

[纯JAVA]也是先创建子JWT(subJWT)，然后根据[10.9](#_创建签名的JWT)或者[10.11](#_创建加密的JWT)的方法创建JWT，但使用setNestedJWTObject来设置子JWT，而不是设置声称。cty头是自动添加的，无需手工设置

可以使用JWT的isNested判断是否嵌套JWT，如果是，使用getNestedJWTObject来获取内层的JWT

# SignedAndEnvelopedData

## 创建SignedAndEnvelopedData

问题

如何创建SignedAndEnvelopedData？

解决方案

使用net.netca.pki.SignedAndEnvelopedData类，创建SignedAndEnvelopedData的步骤如下：

1. new SignedAndEnvelopedData(true)函数创建一个SignedAndEnvelopedData对象
2. 使用setEncAlgorithm (encAlgo)设置对称加密算法。
3. 使用addEncCert (encCert)函数添加加密证书，如果有多个接收者，需要多次调用addEncCert方法
4. 使用NetcaPKIMsgSignedAndEnvelopedDataSetClearText(hSignedAndEnvelopedData,data,dataLen)函数设置明文
5. 可选地，使用addCert (otherCert)添加其他签名相关证书。
6. 使用signAndEncrypt (signAlgo, signCert)方法进行签名和加密得到SignedAndEnvelopedData的编码
7. 使用free数释放SignedAndEnvelopedData对象

讨论

SignedAndEnvelopedData是PKCS#7的一种数据结构，在后续的CMS中被删掉了。但国密的规范中还保留着SignedAndEnvelopedData。由于SignedAndEnvelopedData是个很古老的标准，因此，后续CMS引入的所有东西都在它身上不存在。

SignedAndEnvelopedData基本上是数字信封EnvelopedData和SignedData的结合。明文由数据信封加密，同时明文会进行签名，签名值最终也会经过加密。SignedAndEnvelopedData的SignerInfo不能有认证和不认证属性，因此是直接对明文的签名。

[JNI]当前仅仅支持一个签名。

如果所有的证书都是使用SM2证书，则产生的结果是符合GM/T 0010-2012的结构的。

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.cms.SignedAndEnvelopedData表示SignedAndEnvelopedData，net.netca.pki.encoding.asn1.pki.cms.SignedAndEnvelopedDataBuilder用来创建SignedAndEnvelopedData，具体步骤如下：

* 1. SignedAndEnvelopedDataBuilder.getInstance()创建SignedAndEnvelopedDataBuilder对象
  2. setEncryptAlgorithm设置加密算法
  3. addEncCert添加加密证书
  4. setSecureRandomGenerator设置随机数发生器
  5. setPublicKeyEncrypter设置公钥加密实现
  6. setSigner设置签名者
  7. 使用signAndEncrypt进行签名和加密得到SignedAndEnvelopedData对象
  8. 使用SignedAndEnvelopedData的encode(true)方法得到SignedAndEnvelopedData的编码

如果所有的证书都是使用SM2证书，并且使用setSM2Q7(true)则产生的结果是符合GM/T 0010-2012的结构的。

## 不带ContentInfo的SignedAndEnvelopedData

问题

如何构造不带ContentInfo的SignedAndEnvelopedData？

解决方案

使用setUseContentInfo (false)即可。

讨论

对外输出的SignedAndEnvelopedData，通常都是带ContentInfo的，也就是说，它实际上是个ContentInfo结构。如果在内部使用，通常是需要SignedAndEnvelopedData的结构。

[JNI]默认情况和setUseContentInfo (true)产生的是ContentInfo封装，setUseContentInfo (false)产生的是SignedAndEnvelopedData编码

[纯JAVA]最后一步, 使用SignedAndEnvelopedData的encode(false)即可。

## 如何验证SignedAndEnvelopedData

问题

如何验证SignedAndEnvelopedData？

解决方案

使用net.netca.pki.SignedAndEnvelopedData类，验证SignedAndEnvelopedData的步骤如下：

1. new SignedAndEnvelopedData (false)创建一个SignedAndEnvelopedData对象
2. 使用addStore (store)设置解密证书所在的证书库，如果不设置这一步则默认使用当前用户的个人证书库。
3. 如果SignedAndEnvelopedData不带签名证书，使用addCert (signCert)添加可能的签名证书
4. 使用decryptAndVerify进行解密并验证签名
5. 使用getSignerInfoCount ()获取签名个数，并使用getSignCert (index)逐个获取签名证书，自己手工验证签名证书
6. 使用free释放SignedAndEnvelopedData对象

讨论

由于SignedAndEnvelopedData对签名值进行加密，因此解密后才能验证。

[JNI]当前验证的时候会验证所有的签名，任何一个不通过都是验证失败。验证签名仅仅验证签名值，签名证书仅仅验证密钥用法和有效期，而且有效期可以通过verifyValidity参数来控制是否验证。因此，需要自己用其他的函数来验证签名证书

[纯JAVA] 使用net.netca.pki.encoding.asn1.pki.cms.SignedAndEnvelopedData表示SignedAndEnvelopedData，具体步骤如下：

1. new SignedAndEnvelopedData(data)解码得到SignedAndEnvelopedData对象
2. 使用setSymEncrypter设置对称解密实现
3. 使用setVerifyImplement设置验证实现
4. 使用setCertAndPrivKey设置解密证书及私钥
5. 使用decryptAndVerify进行验证
6. 使用getSignerInfoCount获取签名个数，并使用getSignCert (index)逐个获取签名证书，自己手工验证签名证书

# 电子印章和电子签章

## 创建电子印章

问题

如何创建电子印章？

解决方案

使用net.netca.pki.SEStamp，创建电子印章的步骤如下：

1. 使用new SEStamp (format)函数创建SEStamp对象, format为SEStamp.GMT\_0031表示要创建GM/T 0031格式的电子印章，为SEStamp.GBT\_38540则表示要创建GB/T 38540格式的电子印章。
2. 如果是GM/T 0031格式的电子印章，使用setVersion (version)设置版本号。GB/T 38540格式的电子印章版本号是固定的不需要进行设置
3. 使用setVid (vid)设置电子印章厂商ID
4. 使用setId(id)设置印章ID
5. 使用setType (type)设置电子印章的类型
6. 使用setName (name) 设置电子印章的名称。
7. 使用addCert (cert)逐个添加电子印章的签章人证书。如果是GB/T 38540的电子印章也可以不 添加证书，而是使用addCertDigest (hashAlgo,cert)添加签章人证书的杂凑值。同一电子印章只能同时是证书或者证书的杂凑值，不能证书和证书的杂凑值混用。
8. 使用setCreateDate (time)设置电子印章的制作时间
9. 使用setValidStart (time)设置电子印章的有效期开始时间
10. 使用setValidEnd (time)设置电子印章的有效期结束时间
11. 使用setPicture (type, data, width, heigth)设置电子印章的图片
12. 可选地，使用addExtension (oid, critical, value)添加电子印章的扩展
13. 使用sign (signAlgo ,signCert)签名得到电子印章的编码
14. 使用free ()释放stamp对象

讨论

当前支持GM/T 0031格式的电子印章和GB/T 38540格式的电子印章。这两种电子签章都是制章人对签章图片和签章人列表进行签名的一个结构。签章人列表在GM/T 0031只能使用签章人证书，在GB/T 38540可以使用签章人证书或者签章人证书杂凑值，但是同一印章仅能同时使用一种，不能有的签章人使用签章人证书，有的签章人使用签章人证书杂凑值。由于签章人证书杂凑值使用了一个自定义类型，而不是使用杂凑算法，因此不同厂家之间互操作可能会存在问题，慎用。

使用电子印章进行电子签章的时候，签章人必须在电子印章的签章人列表中。

[JNI]使用net.netca.pki.SEStamp

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.seseal.Stamp表示电子印章，使用net.netca.pki.encoding.asn1.pki.seseal.StampBuilder来创建电子印章。创建电子印章的步骤如下：

1. 使用StampBuilder.getInstance(type)创建一个StampBuilder对象
2. 使用如果是GM/T 0031格式的电子印章，使用setVersion (version)设置版本号。
3. 使用setVid (vid)设置电子印章厂商ID
4. 使用setId(id)设置印章ID
5. 使用setType (type)设置电子印章的类型
6. 使用setName (name) 设置电子印章的名称。
7. 使用addCert (cert)逐个添加电子印章的签章人证书。如果是GB/T 38540的电子印章也可以不 添加证书，而是使用addCertDigest (hashAlgo, hasher,cert)添加签章人证书的杂凑值。同一电子印章只能同时是证书或者证书的杂凑值，不能证书和证书的杂凑值混用。
8. 使用setCreateDate (time)设置电子印章的制作时间
9. 使用setValidStart (time)设置电子印章的有效期开始时间
10. 使用setValidEnd (time)设置电子印章的有效期结束时间
11. 使用setPicture (type, data, width, heigth)设置电子印章的图片
12. 可选地，使用addExtension (ext)添加电子印章的扩展
13. 使用sign (signCert, signer, signAlgo)签名得到Stamp对象
14. 使用Stamp对象的getASN1Object().encode()得到电子印章的编码

## 验证电子印章

问题

如何验证电子印章？

解决方案

使用net.netca.pki.SEStamp，验证电子印章的步骤如下：

1. 使用new SEStamp (data)解码得到电子印章对象
2. 使用getFormat ()获得电子印章的格式
3. 使用getVersion、getVid等方法获得电子印章的相关信息
4. 使用verify (time)验证电子印章来验证电子印章。
5. 使用getSignCert ()得到电子印章的签章人证书，然后自己调用其它函数手工验证
6. 使用free()释放电子印章对象

讨论

解码得到电子印章句柄后就可以直接获取电子印章相关信息了。

[JNI]getCertDigestType获得的类型，在GB/T 38540中说是自定义类型。如果验证签章人证书杂凑值就成为了一个问题

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.seseal.Stamp，验证电子印章的步骤如下：

1. Stamp.decode(data)解码得到电子印章对象
2. getStampType得到电子印章的格式
3. 使用getEstampInfo得到电子印章相关信息
4. verify(time, verifier)验证电子印章
5. 使用getCert得到电子印章的制章人证书，然后自己调用其它函数手工验证

## 产生电子签章

问题

如何产生电子签章？

解决方案

使用net.netca.pki.SESeal，产生电子签章的步骤如下：

1. 使用new SESeal (format)创建电子签章对象，format为SESeal.GMT\_0031表示要创建GM/T 0031格式的电子签章，为SESeal.GBT\_38540则表示要创建GB/T 38540格式的电子签章。
2. 如果是GM/T 0031格式的电子印章，使用setVersion (version)设置版本号。GB/T 38540格式的电子签章版本号和电子印章的是一样的，因此不需要进行设置
3. 使用setStamp (stamp)设置电子印章。注意，GB/T 38540的电子签章只能使用GB/T 38540的电子印章。GM/T 0031的电子签章只能使用GM/T 0031的电子印章
4. 使用setTime (time)函数设置签章时间
5. 计算原文的杂凑值
6. 使用setDataHash (hashValue)设置上一步得到的杂凑值
7. 使用setProperty (property)函数设置原文的属性信息
8. 如果是GB/T 38540的电子签章，可选地，使用addExtension (oid, critical, value)添加扩展
9. 使用sign (signAlgo, signCert)签名得到电子签章的编码。注意签名证书必须在电子印章中
10. 使用free()释放电子签章对象

讨论

GB/T 38540的电子签章只能使用GB/T 38540的电子印章。GM/T 0031的电子签章只能使用GM/T 0031的电子印章。

GB/T 38540的电子签章和GM/T 0031的电子签章相比，签名时间更明确，可以有扩展，还可以添加签名时间戳。GM/T 0031的电子签章的签名时间格式没有明确定义，因此互操作可能是问题

[JNI]使用net.netca.pki.SESeal

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.seseal.Seal表示电子签章，使用net.netca.pki.encoding.asn1.pki.seseal.SealBuilder创建电子签章，具体步骤如下：

1. SealBuilder. getInstance(type)创建SealBuilder对象
2. 如果是GM/T 0031格式的电子印章，使用setVersion (version)设置版本号。
3. 使用setStamp (stamp)设置电子印章。注意，GB/T 38540的电子签章只能使用GB/T 38540的电子印章。GM/T 0031的电子签章只能使用GM/T 0031的电子印章
4. 使用setTime (time)函数设置签章时间
5. 计算原文的杂凑值
6. 使用setPropertyInfo (property)函数设置原文的属性信息
7. 使用signHash(hashValue, signCert, signer,signAlgo, hasher)进行签名得到Seal对象，hashValue是第5步的结果
8. 使用Seal对象的getASN1Object().encode()得到电子印章的编码

## 验证电子签章

问题

如何验证电子签章？

解决方案

使用net.netca.pki.SESeal，验证电子签章的步骤如下：

1. 使用new SESeal（data）解码得到电子签章对象
2. 使用getFormat ()得到电子签章的格式
3. 使用getVersion、getVersion等方法获取电子签章的相关数据
4. 使用verify ( time)函数验证签名。如果有签名时间戳，可以自己手工获取时间戳中的时间作为验证时间参数。
5. 使用getSignCert ()得到签章人证书。然后自己调用其它函数手工验证
6. 使用getSignAlgorithm () 获取电子签章的签名算法
7. 使用SESeal.getHashAlgorithmFromSignAlgorithm (signAlgo)函数得到对应的杂凑算法
8. 使用上一步得到的杂凑算法，计算原文的杂凑值
9. 使用getDataHash ()得到电子签章中的原文杂凑值，并和上一步的比较，完全一样则验证通过
10. 使用free()释放电子签章对象

讨论

验证电子签章需要验证签章人证书在电子印章中，签章人证书是否有效，以及原文杂凑值是否正确。

[JNI]使用net.netca.pki.SESeal

[纯JAVA] 使用net.netca.pki.encoding.asn1.pki.seseal.Seal，验证电子签章的步骤如下：

1. Seal.decode(data)解码得到电子签章对象
2. getSealType得到电子签章的格式
3. 使用getVersion、getVersion等方法获取电子签章的相关数据
4. 使用verify(time,tbs,hasher, verifier)验证
5. 使用getCert获取签章证书，然后自己调用其它函数手工验证

## 追加电子签章签名时间戳

问题

如何为电子签章添加时间戳？

解决方案

使用net.netca.pki.SESeal的attachTimeStamp即可

讨论

只有GB/T 38540的电子签章才支持签名时间戳，而且仅支持一个签名时间戳。

[JNI]使用net.netca.pki.SESeal，验证电子印章后不释放，使用attachTimeStamp追加时间戳

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.seseal.Seal，验证电子印章后，使用setHttpImplement设置HTTP实现，setHashImplement设置Hash实现，setVerifyImplement设置验证实现，setSecureRandomGenerator设置随机数发生器，然后使用attachTimeStamp追加时间戳得到新的电子签名的编码。

## 验证带签名时间戳的电子签章

问题

如何验证带签名时间戳的电子签章？

解决方案

使用net.netca.pki.SESeal，验证电子签章签名时间戳的步骤如下：

1. 使用new SESeal（data）解码得到电子签章对象
2. 使用hasTimeStamp ()得判断是否带时间戳
3. 使用verifyTimeStamp验证时间戳
4. 使用getTimeStamp获取时间戳对象，
5. 在时间戳对象中获取时间戳证书，并自己调用其它函数手工验证
6. 在时间戳对象中获取时间
7. 按照[12.4](#_验证电子签章)的4-10步进行验证，其中第4步的验证时间传入上面第6步得到的时间戳时间

讨论

只有GB/T 38540的电子签章才支持签名时间戳。验证带签名时间戳的电子签章通常需要使用签名时间戳里的时间作为验证时间。

[JNI]使用net.netca.pki.SESeal

[纯JAVA]使用net.netca.pki.encoding.asn1.pki.seseal.Seal，验证电子签章签名时间戳的步骤如下：

1. Seal.decode(data)解码得到电子签章对象
2. hasTimeStamp判断是否有时间戳
3. setHashImplement设置Hash实现
4. setVerifyImplement设置验证实现
5. verifyTimeStamp验证时间戳
6. getTimeStampCert获取时间戳证书，并自己调用其它函数手工验证
7. getTimeStampTime获取时间戳时间
8. 使用verify(time,tbs,hasher, verifier)验证，time是上一步的时间戳时间
9. 使用getCert获取签章证书，然后自己调用其它函数手工验证

# 杂项

## 日志[JNI]

问题

如何打开NETCA CRYPTO的日志？

解决方案

使用NETCA\_TRACE环境变量，可以打开NETCA CRYPTO的日志。

讨论

NETCA\_TRACE环境变量用来控制NETCA CRYPTO的日志输出。当前，NETCA\_TRACE环境变量仅仅在NETCA CRYPTO库装载的时候才读入一次，因此在程序运行中改变NETCA\_TRACE环境变量不会有什么用。默认情况下，也就是不设置NETCA\_TRACE环境变量的情况下，日志是关闭的。

NETCA\_TRACE环境变量的取值如下表所示：

|  |  |
| --- | --- |
| NETCA\_TRACE环境变量的值 | 说明 |
| stdout | 日志输出在标准输出中 |
| stderr | 日志输出在标准错误输出中 |
| file:文件名 | 日志输出在“文件名”指定的文件名中，应该包含完整的路径。比如，file:c:\temp\test.log，表示日志输出在c:\temp\test.log文件中 |
| dll:文件名 | 日志输出由“文件名”的DLL文件实现的方法确定 |

当前，NETCA CRYPTO的日志输出不包含任何的日志等级，因此，输出结果可能是很大的。