NETCA PKI中间件SDK接口说明

# 文件修改记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **修改者** | **主要修改内容说明** |
| 20200512 | V1.1.10 | 张东桂 | 增加产生p10，安装签名加密证书，获取证书列表，获取KeypairName列表，获取PIN码重试次数等。  增加设备创建，设置文件软设备的路径，获取文件软设备的路径 接口等。  增加口令管理系统接口。  增加日志启用关闭接口。  增加解析证书信息。  增加根据PEM编码获取证书对象。  增加根据DER编码获取证书对象。 |
| 20200613 | V1.1.10 | 张东桂 | 增加插入云设备接口。  增加移除云设备接口。  修改PDF签名增加 CNCA.DocVerify |

# 简介

在Android端上NETCA PKI中间件接口以aar包的形式提供相应的功能，主要功能包括解析数字证书，数字信封加解密，SignedData生成和验证，计算HASH，PDF签名，签章读写等功能。

## 接口和类

NETCA PKI中间件包含了17个接口和1个类，每个接口实现相应的功能，接口名称和其对应的完整包名称见表1 NETCA PKI中间件接口列表。

|  |  |  |
| --- | --- | --- |
| 接口名称 | 完整名称 | 备注 |
| NetcaCryptoFactory | net.netca.pki.crypto.android.interfaces. NetcaCryptoFactory | 创建NetcaCryptoInterface实例 |
| NetcaCryptoInterface | net.netca.pki.crypto.android.interfaces. NetcaCryptoInterface | 提供创建其它接口实例 |
| SignatureInterface | net.netca.pki.crypto.android.interfaces. SignatureInterface | P1 签名 |
| CertInterface | net.netca.pki.crypto.android.interfaces.CertInterface | 数字证书接口 |
| DeviceInterface | net.netca.pki.crypto.android.interfaces. DeviceInterface | 设备接口，可读写数据，读写签章 |
| DeviceSetInterface | net.netca.pki.crypto.android.interfaces. DeviceSetInterface | 枚举设备接口 |
| EnvelopedDataDecryptInterface | net.netca.pki.crypto.android.interfaces. EnvelopedDataDecryptInterface | 数字信封解密接口 |
| EnvelopedDataEncryptInterface | net.netca.pki.crypto.android.interfaces. EnvelopedDataEncryptInterface | 数字信封加密接口 |
| PdfSignInterface | net.netca.pki.crypto.android.interfaces. PdfSignInterface | PDF签名验证接口 |
| SignedDataSignInterface | net.netca.pki.crypto.android.interfaces. SignedDataSignInterface | SignedData签名接口 |
| SignedDataVerifyInterface | net.netca.pki.crypto.android.interfaces. SignedDataVerifyInterface | SignedData验证接口 |
| BluetoothInterface | net.netca.pki.crypto.android.interfaces.BluetoothInterface | 操作蓝牙类型的密钥接口 |
| QrcodeSignInterface | net.netca.pki.crypto.android.interfaces. QrcodeSignInterface | 二维码扫码登录接口 |
| TimeStampInterface | net.netca.pki.crypto.android.interfaces.TimeStampInterface | 时间戳接口 |
| PinManageSystemInterface | net.netca.pki.crypto.android.interfaces.PinManageSystemInterface | 口令管理系统接口 |
| HashInterface | net.netca.pki.crypto.android.interfaces.HashInterface | Hash接口 |
| CipherInterface | net.netca.pki.crypto.android.interfaces.CipherInterface | 对称加解密接口 |
| MobileKeyInterface | net.netca.pki.crypto.android.interfaces.MobileKeyInterface | 移动电子密钥接口 |

表1 NETCA PKI中间件接口列表

## 权限相关

使用SDK 需要用到的权限**注意** 最后三个权限在android 6.0 以上需要手动申请。

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.BLUETOOTH"/>

<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN"/>

<uses-feature android:name="android.hardware.usb.host" />

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"></uses-permission>

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"></uses-permission>

<uses-permission android:name="android.permission.SYSTEM\_ALERT\_WINDOW" />

## 依赖

### 外部依赖

compile 'com.alibaba:fastjson:1.1.59.android'

compile 'com.squareup.okhttp3:okhttp:3.5.0'

compile group: 'org.bouncycastle', name: 'bcprov-jdk15on', version: '1.55'

compile group: 'org.bouncycastle', name: 'bcpkix-jdk15on', version: '1.55'

### PDF 签名部分

pdfbox\_android

### core

netca\_crypto

### 海泰otg

haitai\_otg

### 飞天OTG

feitian\_otg

### 飞天蓝牙

feitian\_ble

### 文鼎创蓝牙

eskey\_ble

### 龙脉蓝牙, tf, otg

longmai

### 移动Key

mkey\_sdk\_device

## 集成

1. 在build.gradle 加入外部依赖

implementation 'com.alibaba:fastjson:1.1.59.android'

implementation 'com.squareup.okhttp3:okhttp:3.5.0'

implementation group: 'org.bouncycastle', name: 'bcprov-jdk15on', version: '1.55'

implementation group: 'org.bouncycastle', name: 'bcpkix-jdk15on', version: '1.55'

2. 复制 pdfbox\_android 库，setting.gradle 加入 include ':pdfbox\_android'

3. 复制netca\_crypto 库，setting.gradle 加入 include ':netca\_crypto'

4. 在build.gradle 加入

implementation project(':pdfbox\_android')

implementation project(':netca\_crypto')

5. 根据需要复制加入相关key 的库(步奏与2,3,4 相同)

implementation project(':haitai\_otg')

implementation project(':feitian\_otg')

implementation project(':feitian\_ble')

implementation project(':longmai')

implementation project(': mkey\_sdk\_device')

6. 如需添加自助申请证书，则需要 netca\_view

implementation 'com.dlazaro66.qrcodereaderview:qrcodereaderview:2.0.3'

implementation project(':netca\_view)

## 调用例程

调用流程

1. 初始化，只需要初始化一次
2. 刷新证书，至少需要调用一次，当找不到设备时可再次调用
3. 最后释放，程序结束时调用，只需要释放一次

### 初始化

netcaCryptoInterface = NetcaCryptoFactory.createNetcaCrypto(NetcaCryptoFactory.TYPE\_JAR);

netcaCryptoInterface.init(this);

### 刷新证书

netcaCryptoInterface.refresh();

### 释放

netcaCryptoInterface.end();

### 应用内弹框\*重要\*

@Override

protected void onResume() {

super.onResume();

//需增加这段代码

netcaCryptoInterface.updateActivity(this);

}

@Override

protected void onPause() {

super.onPause();

//需增加这段代码

netcaCryptoInterface.updateActivity(null);

}

//或在调用签名，解密，选择单张证书接口前调用

netcaCryptoInterface.updateActivity(this);

### 添加移动Key 设备

### 服务器参数

//注意修改服务器ID和服务器地址。设备一旦初始化，ID与服务器地址不会被修改。

//67代表移动Key设备

netcaCryptoInterface.addExtraParam("67","{\"id\":\"200951922012717056\",\"u\":\"http://192.168.20.100:7091\"}");

### 自助服务申请证书

#### 添加依赖

implementation 'com.dlazaro66.qrcodereaderview:qrcodereaderview:2.0.3'

implementation project(':netca\_view)

#### 添加权限，注册Activity

\*涉及到二维码扫码需要相机权限以及网络变化权限,添加申请证书所用的webviewactivity.以及扫码所用的qrcodeactivity\*

<uses-permission android:name="android.permission.CAMERA"></uses-permission>

<uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE"/>

<activity android:name="net.netca.pki.netcaview.activity.QrCodeActivity"

android:screenOrientation="portrait"></activity>

<activity android:name="net.netca.pki.netcaview.activity.WebViewActivity"

android:screenOrientation="portrait"></activity>

#### 跳转调用方法

Intent it = new Intent();

//设置自助服务地址，正式地址使用https://bpms.cnca.net/um/index.html

it.putExtra("self\_service", "https://bpms.cnca.net/um/index.html"); startActivity(it);

### 签名/登录

//随机数由服务器参数，用户对该随机数进行签名，后台验证签名，拿出签名证书，找到证书绑定的用户，使用该用户登录。

String data = "random data";

//进行P7带原文签名，需在\*线程\*中执行

Certificate signCert = netcaCryptoInterface.getCertInterface().getSingleSignCert();

SignedDataSignInterface signedDataSignInterface = netcaCryptoInterface.getSignedDataSignInterface();

if(netcaCryptoInterface.getCertInterface().isSm2(signCert)) {

signedDataSignInterface.setSignAlgorithm(0, NetcaPKIConst.Signature.SM3WITHSM2);

}else{

signedDataSignInterface.setSignAlgorithm(0, NetcaPKIConst.Signature.SHA256WITHRSA);

}

signedDataSignInterface.setSignCertificate(signCert);

signedDataSignInterface.setDetached(false);

signedDataSignInterface.setIncludeCertOption(NetcaPKIConst.SignedData.INCLUDE\_CERT\_OPTION\_CERTPATHWITHROOT);

byte[] signedData = signedDataSignInterface.sign(mPassword, data.getBytes("utf-8"),0, data.getBytes("utf-8").length);

//该签名值需传给后台进行验证

String signedDataBase64 = Base64.encodeToString(signedData, Base64.NO\_WRAP);

### P1签名/验证

//签名

//需在\*线程\*中执行

String data = "签名数据";

Certificate signCert = <签名证书>;

String password = <用户口令>;

SignatureInterface signatureInterface = mCryptoInterface.getSignatureInterface();

if (mCryptoInterface.getCertInterface().isSm2(signCert)) {

signatureInterface.init(NetcaPKIConst.Signature.SM3WITHSM2, signCert, password);

} else {

signatureInterface.init(NetcaPKIConst.Signature.SHA256WITHRSA, signCert, password);

}

signatureInterface.signUpdate(data.getBytes("utf-8"), 0, data.getBytes("utf-8").length);

byte[] signature = signatureInterface.sign();

//验证

SignatureInterface signatureInterface = mCryptoInterface.getSignatureInterface();

if (mCryptoInterface.getCertInterface().isSm2(signCert)) {

signatureInterface.init(NetcaPKIConst.Signature.SM3WITHSM2, signCert, null);

} else {

signatureInterface.init(NetcaPKIConst.Signature.SHA256WITHRSA, signCert, null);

}

signatureInterface.verifyUpdate(data.getBytes("utf-8"), 0, data.getBytes("utf-8").length);

boolean verify = signatureInterface.verify(signature);

String verifyResult = verify ? "验证通过" : "验证不通过";

### P7签名/验证

//签名

//需在\*线程\*中执行

String data = "签名数据";

Certificate signCert = <签名证书>;

String password = <用户口令>;

boolean isDetach = false;//是否带不原文

SignedDataSignInterface signedDataSignInterface = mCryptoInterface.getSignedDataSignInterface();

if (mCryptoInterface.getCertInterface().isSm2(signCert)) {

signedDataSignInterface.setSignAlgorithm(0, NetcaPKIConst.Signature.SM3WITHSM2);

} else {

signedDataSignInterface.setSignAlgorithm(0, NetcaPKIConst.Signature.SHA256WITHRSA);

}

signedDataSignInterface.setSignCertificate(signCert);

signedDataSignInterface.setDetached(isDetach);

signedDataSignInterface.setIncludeCertOption(NetcaPKIConst.SignedData.INCLUDE\_CERT\_OPTION\_SELF);

byte[] signedData = signedDataSignInterface.sign(password, data.getBytes("utf-8"), 0, data.getBytes("utf-8").length);

//验证

SignedDataVerifyInterface signedDataVerifyInterface = mCryptoInterface.getSignedDataVerifyInterface();

signedDataVerifyInterface.setVerifyLevel(NetcaPKIConst.SignedData.VERIFY\_LEVEL\_VERIFY\_SIGNATURE\_ONLY);

boolean result = signedDataVerifyInterface.signedDataVerify(data.getBytes("utf-8"), signedData);

if (result) {

List<Certificate> certificates = signedDataVerifyInterface.getSignCerts();

Certificate cert = certificates.get(0);

displayResult("验证通过,原文 " + data + " 签名证书:" + cert);

} else {

displayResult("验证不通过");

}

### 数字信封加密/解密

//数字信封加密

Certificate encCert = <加密证书>;

EnvelopedDataEncryptInterface envelopedDataEncryptInterface = mCryptoInterface.getEnvelopedDataEncryptInterface();

envelopedData = envelopedDataEncryptInterface.envelopedDataEncrypt(Arrays.asList(encCert), data.getBytes("utf-8"));

String password = <用户口令>;

//数字信封解密

EnvelopedDataDecryptInterface envelopedDataDecryptInterface = mCryptoInterface.getEnvelopedDataDecryptInterface();

envelopedDataDecryptInterface.decrypt(password, envelopedData, 0, envelopedData.length);

### 公钥加密/解密

//使用证书公钥加密

mPkEncData = mCertInterface.publicKeyEncrypt(mEncCert, data);

//使用设备私钥解密

mSelectDevice = mDeviceSetInterface.getDeviceByCert(mEncCert);

if (mSelectDevice == null) {

toastMsg("未加密证书找到相关设备");

return;

}

boolean isSm2 = mCertInterface.isSm2(mEncCert);

int algo = isSm2 ? NetcaPKIConst.KeyPair.SM2\_ENC : NetcaPKIConst.KeyPair.RSA\_PKCS1\_V1\_5\_ENC;

String kp = mSelectDevice.getKeyPairName(mEncCert);

byte[] data = mSelectDevice.privateKeyDecrypt(pin, kp, algo, mPkEncData);

### 签章写入/读取

//签章写入

DeviceInterface deviceInterface = <选择的设备对象>;

String sealName = "张三";//签章名

byte[] imageData = <签章文件字节>;

String password = <用户口令>;

int index = 0;//签章位置。默认是0

deviceInterface.writeSeal(password, index, sealName, imageData);

//签章读取

byte[] data = deviceInterface.readSeal(index);

//签章名读取

String name = deviceInterface.getSealName(index);

### 蓝牙设备管理

使用流程：需要调用系统的蓝牙接口去发现设备，判断是否支持，连接设备，断开设备

#### 是否支持蓝牙设备

netcaCryptoInterface.getBluetoothInterface().isSupport(mac, name)

#### 连接蓝牙设备

netcaCryptoInterface.getBluetoothInterface().connect (mac, name)

#### 断开蓝牙设备

netcaCryptoInterface.getBluetoothInterface().disconnect(mac)

### 二维码扫码登录流程

使用流程：需要调用接口， 初始化url，判断是否需要授权码，获取内容，签名并发送

#### 扫码登录

QrcodeSignInterface qrcodeSignInterface = netcaCryptoInterface.getQrcodeSignInterface();

qrcodeSignInterface.init(url);

if (qrcodeSignInterface.isNeedAuthCode()) {

qrcodeSignInterface.requestSignContent(authCode);

} else {

qrcodeSignInterface.requestSignContent(null);

}

qrcodeSignInterface.signContent(signCert, password);

### 时间戳接口流程

#### 请求时间戳

timestamp\_token = netcaCryptoInterface.getTimeStampInterface().requestTimeStampToken(ts\_url, origin.getBytes("utf-8"), algo);

#### 验证时间戳

TimeStampInterface tsi = netcaCryptoInterface.getTimeStampInterface();

tsi.verifyInit(origin.getBytes("utf-8"), timestamp\_token);

tsaCert = tsi.getTsaCert();

Date d = tsi.getTime();

String policy = tsi.getPolicy();

String sn = tsi.getSerialNumber();

### 设备初始化，证书申请安装流程

使用流程：

1. 设置指定设备类型的参数。
2. 刷新设备列表后，判断指定设备类型是否存在，不存在就调用创建设备，已存在指定设备，选择该设备。
3. 使用指定设备产生P10。
4. 使用P10到NETCA生成签名证书。
5. 将获取到的证书，调用安装接口，安装到设备中。

#### 设置指定设备类型的参数

//设置指定设备类型的参数

//67代表移动Key设备,52文件型设备不需要设置

netcaCryptoInterface.addExtraParam("67","{\"id\":\"200951922012717056\",\"u\":\"http://192.168.20.100:7091\"}");

#### 指定设备类型是否存在

//67代表移动Key设备，52代表文件型设备

DeviceSetInterface deviceSetInterface = mCryptoInterface.getDeviceSetInterface();

boolean hasDeviceType = deviceSetInterface.hasDeviceType( NetcaPKIConst.Device.TYPE\_MKEY\_SDK\_DEVICE);

#### 获取指定类型的设备

//67代表移动Key设备，52代表文件型设备

DeviceInterface softwareDevice;

DeviceSetInterface deviceSetInterface = mCryptoInterface.getDeviceSetInterface();

List<DeviceInterface>deviceList = deviceSetInterface.getDeviceList();

for (DeviceInterface device : deviceList) {

if (device.getDeviceType() == NetcaPKIConst.Device.TYPE\_MKEY\_SDK\_DEVICE) {

softwareDevice = device;

}

}

#### 创建指定类型的设备

创建成功后需要刷新。

继续使用 <获取指定类型的设备> 获取指定设备。

//67代表移动Key设备，

DeviceInterface softwareDevice;

DeviceSetInterface deviceSetInterface = mCryptoInterface.getDeviceSetInterface();

deviceSetInterface.createDevice(NetcaPKIConst.Device.TYPE\_MKEY\_SDK\_DEVICE, pin);

deviceSetInterface.refresh();

#### 使用指定设备产生P10

产生p10的pin码，如果未修改pin码，必须与创建设备的一致。

//产生的P10需要到NETCA 生成证书。然后调用安装证书接口。

String p10 = softwareDevice.genP10(pin, dn, 256, NetcaPKIConst.Signature.SM3WITHSM2);

#### 使用指定设备安装签名加密证书

公钥私钥必须对应才能安装成功。 即产生p10的设备，与p10申请的证书必须安装到同一个设备上。

Certificate signCert = new Certificate(Base64.decode(signCertCode, Base64.NO\_WRAP));

Certificate encCert = new Certificate(Base64.decode(encCertCode, Base64.NO\_WRAP));

String encKeyPairB64 = NetcaPKIConst.Device.NETCA\_PKI\_ENCKEYPAIR\_TYPE\_GBT\_35276;

softwareDevice.installCert(pin,encCert,signCert,platform,encKeyPairB64) ;

#### 读取设备的所有证书

List<Certificate> certList = softwareDevice.getCerts();

# 常量说明

NETCA PKI中间件所使用到的常量被封装在类NetcaPKIConst中，其包含嵌套类Hash,Error, Cert, Signature, SignedData,Device

Hash类，包含中间件支持的HASH算法。

Error类，包含中间件常见的错误码和描述。

Cert类，包含选择证书时所使用的证书类型，证书类别，证书状态码，验证证书状态标志。

Signature类，包含签名算法。

SignedData类，包含生成SignedData签名时包含证书的选项，SignedData验签级别选项。

Device 类，包含安装加密密钥对的平台。

## NetcaPKIConst.Hash

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| SHA1 | SHA1 hash算法 |
| SHA224 | SHA224 hash算法 |
| SHA256 | SHA256 hash算法 |
| SHA512 | SHA512 hash算法 |
| SM3 | SM3 hash 算法 |

## NetcaPKIConst.Error

### 错误码信息

|  |  |  |
| --- | --- | --- |
| 常量属性 | 值 | 意义 |
| NETCA\_PKI\_SUCCESS | 1 | 成功 |
| NETCA\_PKI\_FAIL | 0 | 失败 |
| NETCA\_PKI\_INVALIDARG | -1 | 无效的参数 |
| NETCA\_PKI\_INVALIDHANDLE | -2 | 无效的句柄 |
| NETCA\_PKI\_OUTOFMEMORY | -3 | 内存分配失败 |
| NETCA\_PKI\_MOREDATA | -4 | 输出缓冲区太小 |
| NETCA\_PKI\_PWD\_ERROR | -5 | 密码错误 |
| NETCA\_PKI\_EXISIT | -6 | 已经存在 |
| NETCA\_PKI\_NOTEXISIT | -7 | 不存在 |
| NETCA\_PKI\_ACCESSDENIED | -8 | 拒绝访问 |
| NETCA\_PKI\_UNSUPPORTED | -9 | 不支持 |
| NETCA\_PKI\_NOTIMPLEMENT | -10 | 未实现 |
| NETCA\_PKI\_TIMEOUT | -11 | 超时 |
| NETCA\_PKI\_BULID\_CERTPATH\_FAIL | -12 | 构造证书链失败 |
| NETCA\_PKI\_VERIFY\_CERTPATH\_FAIL | -13 | 验证证书链失败 |
| NETCA\_PKI\_NOT\_IN\_VALIDITY | -14 | 不在有效期内 |
| NETCA\_PKI\_VERIFY\_SIGNEDDATA\_CERT\_FAIL | -36 | 验证SignedData的证书失败 |
| NETCA\_PKI\_ENVELOPEDDATA\_BAD\_DATA | -37 | EnvelopedData格式错误 |
| NETCA\_PKI\_ENVELOPEDDATA\_DECRYPT\_KEY\_FAIL | -38 | EnvelopedData解密对称密钥失败 |
| NETCA\_PKI\_ENVELOPEDDATA\_DECRYPT\_DATA\_FAIL | -39 | EnvelopedData解密数据失败 |
| NETCA\_PKI\_ENVELOPEDDATA\_NO\_MATCH\_CERT | -40 | EnvelopedData没有匹配的证书用于解密对称密钥 |
| NETCA\_PKI\_SIGNEDDATA\_BAD\_DATA | -41 | SignedData格式错误 |
| NETCA\_PKI\_SIGNEDDATA\_VERIFY\_CONTENT\_TYPE\_FAIL | -42 | 验证SignedData的Content-Type失败 |
| NETCA\_PKI\_SIGNEDDATA\_VERIFY\_HASH\_FAIL | -43 | 验证SignedData的Hash失败 |
| NETCA\_PKI\_SIGNEDDATA\_VERIFY\_SIGNATURE\_FAIL | -44 | 验证SignedData的签名失败 |
| NETCA\_PKI\_SIGNEDDATA\_NO\_MATCH\_CERT | -45 | 没有签名证书用于验证SignedData的签名 |

## NetcaPKIConst.Cert

### 证书类型

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| CERT\_TYPE\_ENCRYPT | 加密证书类型 |
| CERT\_TYPE\_SIGN | 签名证书类型 |

### 证书状态

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| STATUS\_CA\_REVOKED | CA证书已作废 |
| STATUS\_REVOKED | 证书已作废 |
| STATUS\_UNDETERMINED | 证书状态不能确定 |
| STATUS\_UNREVOKED | 证书未作废 |

### 证书密钥用法

同时包含多个密钥用法，逐位或运算。

即 keyUsage & KEYUSAGE\_DIGITALSIGNATURE != 0.说明该证书可用于“数字签名”。

一般加密证书有 KEYUSAGE\_KEYENCIPHERMENT 和 KEYUSAGE\_DATAENCIPHERMENT。

签名证书至少有 KEYUSAGE\_DIGITALSIGNATURE 。

|  |  |
| --- | --- |
| 常量 | 说明 |
| KEYUSAGE\_DIGITALSIGNATURE | 数字签名 |
| KEYUSAGE\_NONREPUDIATION | 不可否认 |
| KEYUSAGE\_KEYENCIPHERMENT | 加密密钥 |
| KEYUSAGE\_DATAENCIPHERMENT | 加密数据 |
| KEYUSAGE\_KEYAGREEMENT | 密钥协商 |
| KEYUSAGE\_KEYCERTSIGN | 签证书 |
| KEYUSAGE\_CRLSIGN | 签CRL |
| KEYUSAGE\_ENCIPHERONLY | 密钥协商时，只加密 |
| KEYUSAGE\_DECIPHERONLY | 密钥协商时，只解密 |

## NetcaPKIConst.Signature

### 签名算法

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| SHA1WITHRSA | SHA1WITHRSA 签名算法 |
| SHA256WITHRSA | SHA256WITHRSA签名算法 |
| SHA512WITHRSA | SHA512WITHRSA 签名算法 |
| SM3WITHSM2 | SM3WITHSM2 签名算法 |

## NetcaPKIConst.SignedData

### 签名包含的证书选项

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| INCLUDE\_CERT\_OPTION\_CERTPATH | 包含整个证书链，但不包含根证书 |
| INCLUDE\_CERT\_OPTION\_CERTPATHWITHROOT | 包含整个证书链，且包含根证书 |
| INCLUDE\_CERT\_OPTION\_NONE | 不包含任何证书 |
| INCLUDE\_CERT\_OPTION\_SELF | 仅包含签名证书（一般使用该参数） |

### 验签级别选项

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| VERIFY\_LEVEL\_NO\_VERIFY | 不做验证 |
| VERIFY\_LEVEL\_VERIFY\_CERT | 验证签名和证书，但是不验证证书是否作废 |
| VERIFY\_LEVEL\_VERIFY\_SIGNATURE\_ONLY | 仅验证签名不验证证书有效期。（签名数据长期有效，使用该参数） |

## NetcaPKIConst.Device

### 加密密钥对平台

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| NETCA\_PKI\_ENCKEYPAIR\_TYPE\_06 | 06系统 |
| NETCA\_PKI\_ENCKEYPAIR\_TYPE\_PUHUA | 普华系统 |
| NETCA\_PKI\_ENCKEYPAIR\_TYPE\_GDKM | GDKM |
| NETCA\_PKI\_ENCKEYPAIR\_TYPE\_GBT\_35276 | GBT\_35276 |
| NETCA\_PKI\_ENCKEYPAIR\_TYPE\_ALL | 全平台，逐个尝试 |

### 设备类型

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| TYPE\_SOFT\_DEVICE | 52文件软设备 |
| TYPE\_MKEY\_SDK\_DEVICE | 67 分散密钥设备 |

## NetcaPKIConst.KeyPair

### 密钥对算法

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| RSA | RSA算法 |
| DSA | DSA算法 |
| DH | DH算法 |
| ECC | ECC算法 |

## NetcaPKIConst.Cipher

### Padding 类型

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| PADDING\_NONE | 无Padding |
| PADDING\_PKCS5 | PKCS5 Padding |

### 对称加解密算法

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| AES\_CBC | AES CBC |
| AES\_ECB | AEC ECB |
| DES\_ECB | DES ECB |
| DES\_CBC | DES CBC |

## NetcaPKIConst. MobileKey

### 口令UI类型

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| TYPE\_PIN\_UI\_NONE | 默认类型 |
| TYPE\_PIN\_UI\_6\_NUMBER | 六位字符类型 |

### 口令缓存类型

|  |  |
| --- | --- |
| 常量属性 | 意义 |
| TYPE\_PIN\_CACHE\_TYPE\_NOT\_NEED | 不需要缓存 |
| TYPE\_PIN\_CACHE\_TYPE\_NEED | 需要缓存 |
| TYPE\_PIN\_CACHE\_TYPE\_CONTROL\_BY\_USER | 由用户决定 |

# 接口详细说明

## NetcaCryptoFactory

### 根据类型创建NetcaCrypto接口对象

原型：

public static NetcaCryptoInterface createNetcaCrypto(int type)

说明：

根据类型创建NetcaCrypto接口对象

参数:

type NetcaCryptoFactory.TYPE\_JAR

返回:

NetcaCrypto接口对象

### 获取NetcaCrypto接口对象

原型：

public static NetcaCryptoInterface getInstance()

说明：

根据类型创建NetcaCrypto接口对象

参数:

无

返回:

NetcaCrypto接口对象

## NetcaCryptoInterface 接口

### 初始化

原型：

void init(Context c) throws PkiException;

说明：

初始化

参数:

c 上下文

返回:

无

### 释放

原型：

void end() throws PkiException;

说明：

资源释放

参数:无

返回:无

### 刷新

原型：

void refresh() throws PkiException;

说明：

刷新证书

参数:无

返回:无

### 添加额外参数

原型：

void addExtraParam(String key,String value);

说明：

添加额外的参数

参数:

key 设备类型字符串。例如移动Key “67”

value 设备需要的额外参数。不同设备定义不同。

返回:无

### 移除额外参数

原型：

void removeExtraParam(String key);

说明：

移除额外的参数。

参数:

key 设备类型字符串。例如移动Key “67”

返回:无

### 创建设备

原型：

void createDevice(int type,String password) throws PkiException;

说明：

创建指定类型的设备

参数:

type 设备类型。例如移动Key 67。

password 设备PIN码。用户输入的PIN码。

返回:无

### 更新Activity

原型：

void updateActivity (Activity activity)

说明：

更新当前activity, 如需要弹出选证书和Pin码框但没有悬浮窗权限时，用于获取当前activity对象。

参数:

activity 当前activity

返回:无

### 获取蓝牙接口实例

原型：

BluetoothInterface getBluetoothInterface() throws PkiException;

说明：

获取蓝牙接口实例。

参数:无

返回:

蓝牙接口实例。

### 获取证书接口实例

原型：

CertInterface getCertInterface() throws PkiException;

说明：

获取证书接口实例

参数:无

返回:

证书接口实例。

### 获取数字信封加密实例

原型：

EnvelopedDataEncryptInterface getEnvelopedDataEncryptInterface() throws PkiException;

说明：

获取数字信封加密实例。

参数:无

返回:

数字信封加密实例。

### 获取数字信封解密实例

原型：

EnvelopedDataDecryptInterface getEnvelopedDataDecryptInterface() throws PkiException;

说明：

获取数字信封解密实例。

参数:无

返回:

数字信封解密实例。

### 获取SignedData签名实例

原型：

SignedDataSignInterface getSignedDataSignInterface() throws PkiException;

说明：

获取SignedData签名实例。

参数:无

返回:

SignedData签名实例。

### 获取SignedData验签接口实例

原型：

SignedDataVerifyInterface getSignedDataVerifyInterface() throws PkiException

说明：

获取SignedData验签实例。

参数:无

返回:

SignedData验签实例。

### 获取设备列表接口实例

原型：

DeviceSetInterface getDeviceSetInterface() throws PkiException;

说明：

获取设备列表接口实例

参数:无

返回:

设备列表接口实例。

### 获取p1签名接口实例

原型：

SignatureInterface getSignatureInterface() throws PkiException;

说明：

获取p1签名实例

参数:无

返回:

P1签名接口实例。

### 获取pdf签名接口实例

原型：

PdfSignInterface getPdfSignInterface() throws PkiException;

说明：

获取pdf签名实例

参数:无

返回:

PDF签名接口实例。

### 获取二维码签名接口实例

原型：

QrcodeSignInterface getQrcodeSignInterface() throws PkiException;

说明：

获取二维码签名接口

参数:无

返回:

二维码签名接口实例。

### 获取时间戳接口实例

原型：

TimeStampInterface getTimeStampInterface() throws PkiException;

说明：

获取时间戳接口

参数:无

返回:

时间戳接口实例。

### 设置口令管理系统服务器地址

原型：

void setPinManageSystemAddr(String addr);

说明：

设置口令管理系统服务器地址

参数:

addr 服务器地址信息。

返回:

证书接口。

### 获取口令管理系统接口实例

原型：

PinManageSystemInterface getPinManageSystemInterface() throws PkiException;

说明：

获取口令管理系统接口。

参数:无

返回:

口令管理系统实例。

### 日志启用关闭

原型：

void setLogEnable(boolean enable);

说明：

日志启用关闭。

参数:

enable 是否启用。

返回:无

### 获取Hash 接口实例

原型：

HashInterface getHashInterface() throws PkiException;

说明：

获取Hash对象接口。

参数:无

返回:

获取Hash实例。

### 获取对称加解密接口实例

原型：

CipherInterface getCipherInterface() throws PkiException;

说明：

V1.1.10开始。

获取对称加解密接口

参数:无

返回:

对称加解密接口实例。

## SignatureInterface接口

### 设置签名算法

原型：

void init(int signAlgo, Certificate cert,String pwd) throws PkiException;

说明：

设置签名算法

参数:

signAlgo,签名算法，具体参考NetcaPKIConst.Signature

cert,证书对象

pwd,签名时所用的PIN码，设为null时，会弹出对话框供用户输入；验证时不需要，可以设为null

返回:无

### 对Hash值进行签名

原型：

byte[] signHash(byte[] hash) throws PkiException;;

说明：

对Hash值进行签名。

参数:

hash,需要签名的hash值

注意：SM3WithSM2的签名输入hash值不是原文的Hash值

返回:

签名结果

### 输入数据进行Hash运算

原型：

void signUpdate(byte[] data, int offset, int length) throws PkiException;

说明：

输入数据进行Hash运算

data 数据原文

offset偏移量

length 进行计算hash的数据长度

返回:无

### 对update后的数据进行签名

原型：

byte[] sign()throws PkiException;

说明：

对update后的数据进行签名

参数:无

返回:

签名结果

### 输入数据进行Hash运算

原型：

void verifyUpdate(byte[] data, int offset, int length) throws PkiException;

说明：

输入数据进行Hash运算

参数:

data,数据原文

offset，位移

length,进行计算hash的数据长度

返回:无

### 验证签名

原型：

boolean verify(byte[] signature) throws PkiException;

说明：

验证签名

参数:

signature,签名值

返回:

成功返回true，失败返回false

### 验证签名

原型：

boolean verifyHash(byte[] hash, byte[] signature) throws PkiException;

说明：

验证签名

参数:

hash Hash值 注意：SM3WithSM2的签名输入hash值不是原文的Hash值

signature 签名值

返回:

成功返回true，失败返回false

### 计算SM2签名需要的Z值

原型：

byte[] computeSM2Z(byte[] id) throws PkiException;

说明：

计算SM2签名需要的Z值（注意，需要在init调用后，才可使用）

参数:

id, 用户传入的id

返回:无

## CertInterface

### 获取有效期内的签名证书列表

原型：

List<Certificate> getSignCertList()throws PkiException;

说明：

获取有效期内的签名证书列表。

参数:无

返回:

有效期内的签名证书列表。

### 选择有效期内的单个签名证书

原型：

Certificate getSingleSignCert()throws PkiException;

说明：

选择有效期内的单个签名证书，需要调用NetcaCryptoInterface.updateActivity。

参数:无

返回:

成功返回证书对象

用户没有设备或者取消选择证书 抛出异常

### 获取有效期内的加密证书列表

原型：

List<Certificate> getEncCertList()throws PkiException;

说明：

获取有效期内的加密证书列表

参数:无

返回:

有效期内的加密证书列表。

### 选择有效期内的单个加密证书

原型：

Certificate getSingleEncCert()throws PkiException;  
说明：

选择有效期内的单个加密证书。需要调用NetcaCryptoInterface.updateActivity。

参数:无

返回:

成功返回证书对象

用户没有设备或者取消选择证书 抛出异常

### 通过ocsp验证证书

原型：

byte[] verifyCertByOcsp(Certificate cert, String url) throws PkiException  
说明：

通过ocsp验证证书

参数:

cert，证书对象

url 可以为空，如果为空则默认使用证书扩展中的地址(如果证书不带url地址，可从外部传入)

返回:

ocsp 成功返回ocsp数据

### 通过crl验证证书

原型：

byte[] verifyCertByCrl(Certificate cert, String url) throws PkiException

说明：

通过crl验证证书

参数:

cert 证书对象

url可以为空，如果为空则默认使用证书扩展中的地址(如果证书不带url地址，可从外部传入)

返回:

crl 成功返回crl 数据

### 判断证书是否为sm2证书

原型：

boolean isSm2(Certificate cert) throws PkiException

说明：

判断证书是否为sm2证书

参数:

cert 证书对象

返回:

是sm2证书返回true, rsa证书返回false

### 使用证书公钥加密

原型：

byte[] publicKeyEncrypt(Certificate cert, byte[] data) throws PkiException;

说明：

V1.1.10开始。

使用证书公钥加密。

参数:

cert 证书对象。

data 需要加密的数据。

返回：

加密数据。

### 解析证书信息

原型：

NetcaCertInfo parseCertInfo(Certificate cert);

说明：

V1.1.10开始。

解析证书信息。

参数:

cert 证书对象。

返回:

解析后的对象。

### 根据PEM编码获取证书对象

原型：

Certificate getCertFromString(String pem) throws PkiException

说明：

V1.1.10开始。

从PEM编码获取证书对象。

参数:

pem 证书PEM编码

返回:

证书对象。

### 根据DER编码获取证书对象

原型：

Certificate getCertFromDer(byte[] data) throws PkiException

说明：

V1.1.10开始。

从DER编码获取证书对象。

参数:

data DER编码数据

返回:

证书对象。

## DeviceInterface接口

### 获取设备唯一信息

原型：

String getDeviceId()throws PkiException

说明：

获取设备唯一信息。设备会以类型+“#”分隔符+设备SN。例如软设备ID 52#abcdef

参数:无

返回：

设备ID。设备类型+“#”+设备SN。

### 获取证书列表

原型：

List<Certificate> getCerts() throws PkiException;

说明：

获取证书列表。V1.1.10开始

参数:无

返回：

证书列表，包括过期证书。

### 获取设备类型名

原型：

String getDeviceTypeName() throws PkiException;

说明：

获取设备类型名

V1.1.10开始

参数:无

返回：

代表某种设备类型的名字。

### 获取设备类型

原型：

int getDeviceType() throws PkiException;

说明：

获取设备类型 V1.1.10开始

参数:无

返回:

代表某种设备类型的编号。是设备ID“#”号前的数字。

### 产生p10

原型：

String genP10(String pin, String dn, int keyBits, int algo) throws PkiException;

说明：

V1.1.10开始

产生P10

参数：

pin 设备pin码

dn 证书DN 如 CN=张三,C=CN，信息从小到大。

keyBits 密钥位数。保留字段，未使用。默认可填入 RSA 2048， SM2 256。

algo 算法 NetcaPKIConst.Signature.SM3WITHSM2 NetcaPKIConst.Signature.SHA256WITHRSA

返回：

p10 的base64编码

### 获取KeypairName 列表

原型：

List<String> getKeyPairList() throws PkiException;

说明：

V1.1.10开始。

获取keypairName 列表

参数:无

返回:

当前存在的KeypairName列表

### 获取密钥对下的证书列表

原型：

List<Certificate> getCerts(String kpName) throws PkiException;

说明：

V1.1.10开始。

获取密钥对下的证书列表，包括过期证书。

参数:

kpName 密钥对名。

返回:

当前存在的证书列表

### 获取证书的密钥对名

原型：

String getKeyPairName(Certificate cert) throws PkiException;

说明：

V1.1.10开始。

获取证书的密钥对名

参数:

cert 证书对象

返回:

密钥对名

### 使用私钥解密

原型：

byte[] privateKeyDecrypt(String pin, String kp, int algo, byte[] cipher) throws PkiException;

说明：

V1.1.10开始。

使用私钥解密

参数:

pin pin码。

kp 密钥对名。

algo 算法 RSA NetcaPKIConst.KeyPair.RSA\_PKCS1\_V1\_5\_ENC SM2 NetcaPKIConst.KeyPair.SM2\_ENC

cipher 密文数据。

返回:

解密的明文数据。

### 安装签名加密证书

原型：

void installCert(String pin, Certificate encCert, Certificate signCert, int platform, String encKeyPairB64) throws PkiException;

说明：

V1.1.10开始。

安装根据平台安装签名加密证书

参数:

pin pin码。

encCert 加密证书，加密证书没有传null。

signCert 签名证书。

platform

NetcaPKIConst.Device.NETCA\_PKI\_ENCKEYPAIR\_TYPE\_06 或

NetcaPKIConst.Device.NETCA\_PKI\_ENCKEYPAIR\_TYPE\_PUHUA 或

NetcaPKIConst.Device.NETCA\_PKI\_ENCKEYPAIR\_TYPE\_GDKM

encKeyPairB64 加密的加密密钥对数据，加密密钥对没有传null。

返回:无

### 获取PIN码重试次数错误提示

原型：

String getPinLeftErrorMsg() throws PkiException;

说明：

获取PIN码重试次数错误提示。在pin码错误时调用。

不支持，提示PIN码错误。

V1.1.10开始。

参数:无

返回:

获取PIN码重试次数错误提示

### 验证PIN码

原型：

boolean verifyPin(String pwd) throws PkiException

说明：

验证pin码

参数:

pwd PIN码

返回

PIN码正确返回true.错误返回false

### 修改PIN码

原型：

boolean changePin(String oldPwd, String newPwd) throws PkiException

说明：

V1.1.10开始。

修改pin码

参数:

oldPwd 旧PIN码

newPwd 新PIN码

返回:

修改成功返回true.失败返回false。

### 获取PIN码重试次数

原型：

int getPwdRetryNumber() throws PkiException;

说明：

V1.1.10开始。

获取PIN码重试次数 -1表示设备不支持，0表示锁定，>0表示次数

参数：无

返回:

PIN码重试次数 -1表示设备不支持，0表示锁定，>0表示次数

### 创建持久数据

原型：

void createData(int id, String mode, int length) throws PkiException

说明：

创建持久数据

参数：

id 数据ID

mode 访问权限

length 长度

返回：无

### 读持久数据

原型：

int readData(int id, int offset, byte[] buf, int start, int length) throws PkiException;

说明：

从dataOffset开始读持久数据

参数:

id 数据ID

offset 持久数据离开始点的位移长度

buf 读取的数据缓存

start 开始写入的缓存位置

length 希望读出的长度

返回:

成功返回读出的数据长度

### 获取持久数据的长度

原型：

int getDataLength(int id) throws PkiException

说明：

获取持久数据的长度

参数:

id 数据ID

返回:

成功返回数据长度

### 删除持久数据

原型：

void deleteData(String pin, int id) throws PkiException;

说明：

删除持久数据

参数：

pin 设备pin码

id 数据ID

返回：无

### 写持久数据

原型：

void writeData(String pin, int id, int dataOffset, byte[] buf) throws PkiException;

说明：

从dataOffset开始写持久数据

参数：

pin 设备PIN码

id 数据ID

dataOffset 持久数据位移，从该点开始写数据

buf 要写入的数据

返回：

无

### 获取签章的数量

原型：

int getSealCount()throws PkiException

说明：

获取签章的数量

参数：无

返回：

签章的数量

### 获取签章名字

原型：

String getSealName(int index) throws PkiException

说明：

根据index 获取名字

参数：

index 签章的索引。

返回：

签章名。

### 获取印章

原型：

byte[] readSeal(int index) throws PkiException

说明：

获取印章

参数：

index 签章index

返回：

图片byte数组

### 写章

原型：

void writeSeal(String pin, int index, String name, byte[] buf) throws PkiException;

说明：

写章接口

参数：

pin 设备PIN码

index 签章的位置

name 签名的名字

buf 图片文件byte数组

返回：无

### 删除章

原型：

void deleteSeal(String pin, int index) throws PkiException;

说明：

删除章

参数：

pin 设备pin码

index 签章的位置

返回：无

### 获取设备SN

原型：

String getDeviceSN() throws PkiException;

说明：

获取设备序列号

参数：无

返回：

设备序列号

## DeviceSetInterface接口

### 获取设备列表

原型：

List<DeviceInterface> getDeviceList()throws PkiException

说明：

获取设备 列表

参数：无

返回：

设备列表

### 根据证书获取设备

原型：

DeviceInterface getDeviceByCert(Certificate cert) throws PkiException;

说明：

V1.1.10开始。

根据证书获取设备。

参数：

cert 证书对象

返回：

设备对象。未找到返回null

### 是否存在该类型的设备

原型：

boolean hasDeviceType(int type) throws PkiException;

说明：

V1.1.10开始。

参数：

type 设备类型。分散密钥使用NetcaPKIConst.Device.TYPE\_MKEY\_SDK\_DEVICE

返回:

是否存在该类型的设备。

### 是否存在该类型的设备创建接口

原型：

boolean hasCreateDeviceSDK(int type) throws PkiException;

说明：

V1.1.10开始。

是否包含创建接口

参数：

type 设备类型。分散密钥使用

NetcaPKIConst.Device.TYPE\_MKEY\_SDK\_DEVICE

返回：

是否存在该类型的设备创建接口。

### 根据id获取设备

原型：

DeviceInterface getDeviceById(String id) throws PkiException;

说明：

根据id获取设备

V1.1.10开始。

参数：

id设备ID

返回：

设备对象，不存在返回null。

### 刷新设备列表

原型：

void refresh() throws PkiException;

说明：

刷新设备列表，通常为创建完设备后，需要刷新获取新创建的设备对象。

V1.1.10开始。

参数：无

返回：无

### 创建指定类型的设备

原型：

void createDevice(int type, String password) throws PkiException;

说明：

创建指定类型的设备。设备已存在就不需要创建。大部分设备需要提前设置额外的外部参数。需要调用NetcaCryptoInterface.addExtraParam 添加参数。

V1.1.10开始。

参数：

type 设备类型

password 设备PIN码

返回：无

### 设置文件软设备的路径

原型：

void setSoftwareDevicePath(String filePath) throws PkiException;

说明：

设置52类型文件软设备的路径，

init函数调用后，会重置为默认路径。

需要在init调用后再次调用，刷新设备才会使用新设置的地址。

V1.1.10开始。

参数：

filePath 软设备文件路径，必须以.key后缀结尾

返回：

无

### 获取文件软设备的路径

原型：

String getSoftwareDevicePath() throws PkiException;

说明：

52类型文件软设备的路径。

V1.1.10开始。

参数：无

返回：

52类型文件软设备的路径

### 产生随机数

原型：

byte[] generateRandom(int len) throws PkiException;

说明：

获取随机数

V1.1.10开始。

参数：

len 随机数长度。

返回：

随机数据。

### 插入云设备

原型：

String mountCloudDevice(Certificate signCert, String pin, String jsonParam) throws PkiException;

说明：

插入云设备

参数:

signCert 选择机构员工 的 签名证书

pin 密码

jsonParam 云设备服务器信息。json 字符串。

返回:

jsonResult json字符串。

jsonParam 参数说明

|  |  |  |
| --- | --- | --- |
| **参数名** | **JSON类型** | **说明** |
| cloudkeyURL | String | 第三方的服务地址 |
| deviceType | int | 设备类型 |

jsonResult 返回值说明

|  |  |  |
| --- | --- | --- |
| **参数名** | **JSON类型** | **说明** |
| deviceSN | String | 设备序列号 |
| deviceType | int | 设备类型 |
| certs | 数组 | 证书数组对象，每个元素是certobject对象 |

certobject对象参数说明

|  |  |  |
| --- | --- | --- |
| **参数名** | **JSON类型** | **说明** |
| signCertText | 字符串 | 签名证书的base64不换行编码，如果不存在为空。 |
| encCertText | 字符串 | 加密证书的base64不换行编码，如果不存在为空。 |

### 移除云设备

原型：

void unMountCloudDevice(String jsonParam) throws PkiException;

说明：

移除云设备

参数:

jsonParam json字符串。

返回值:

无

jsonParam 参数说明

|  |  |  |
| --- | --- | --- |
| **参数名** | **JSON类型** | **说明** |
| deviceType | int | 设备类型 |
| deviceSN | string | 设备序列号 |
| cert | string | 设备对应证书， |

## EnvelopedDataDecryptInterface接口

### 一步解密数字信封

原型：

byte[] decrypt(String pwd,in byte[] data, int offset, int length) throws PkiException;

说明：

一步解密，用于数据数字信封编码不是很大的情况

输入数字信封编码，进行解密。pwd为空需要调用。updateActivity

参数：

pwd,设备PIN码

data，数字信封编码

offset,data的偏移量

length,用于解密的数据长度

返回：

成功返回原文，失败返回空数组

### 多步解密数字信封初始化

原型：

void decryptInit(String pwd) throws PkiException;

说明：

多步解密 ，一般用于数字信封编码结果数据很大的情况

分步解密的初始化

参数：

pwd,解密PIN码

返回：无

### 多步解密数字信封update

原型：

byte[] decryptUpdate(byte[] data, int offset, int length) throws PkiException;

说明：

多步解密数字信封，可多次调用。分步解密的,输入数字信封编码，进行解密。

参数：

data，数字信封编码

offset,data的偏移量

length,用于解密的数据长度

返回：

成功返回解密得到部分明文

### 多步解密数字信封结束

原型：

void decryptFinal()throws PkiException;

说明：

结束数字信封解密

参数：无

返回：无

### 获取解密数字信封的加密证书

原型：

Certificate getDecryptCert()throws PkiException;

说明：

解密成功后，获取解密证书

参数：无

返回：

成功返回所使用的解密证书,失败返回null。

### 释放外部资源

原型：

void free();

说明：

释放外部资源,其它调用流程完毕后再调用。

参数：无

返回：无

## EnvelopedDataEncryptInterface

### 一步完成数字信封加密

原型：

byte[] envelopedDataEncrypt(List< Certificate > certs, byte[] data) throws PkiException;

说明：

一步完成 数字信封加密

参数：

certs,加密使用的加密证书

data,原文

返回：

成功返回加密密文

### 一步完成数字信封加密

原型：

byte[] envelopedDataEncrypt2( List< Certificate > certs, byte[] data, int offset, int length) throws PkiException;

说明：

一步完成 数字信封加密

参数：

certs,加密使用的加密证书

data,原文

offset,偏移位置

length,用于加密的数据长度

返回：

成功返回加密密文

### 多步完成数字信封加密添加证书

原型：

boolean addCert(String cert) throws PkiException;

说明：

添加加密证书，可多次添加不同的加密证书。

参数：

cert,加密证书

返回：

成功返回true，失败返回false

备注：多步实现数字信封加密，加密的密文为encryptInit，encryptUpdate，encryptFinal三者结果的拼接

### 多步完成数字信封加密初始化

原型：

byte[] encryptInit()throws PkiException;

说明：

开始多步数字信封加密

参数：无

返回：

成功返回数字信封的开始部分，失败抛出异常。

### 多步完成数字信封加密update

原型：

byte[] encryptUpdate(byte[] data, int offset, int length) throws PkiException;

说明：

多步加密，输入明文，进行数字信封加密，可以多次调用

参数：

data,原文

offset,偏移位置

length,用于加密的长度

返回：

成功返回加密部分的数字信封

### 多步完成数字信封加密结束

原型：

byte[] encryptFinal()throws PkiException;

说明：

结束数字信封加密

参数：无

返回：

成功返回最后部分的数数据.

## PdfSignInterface接口

### PDF签名

原型：

void signPDFFile(String pdfFilePath, byte[] byteSeal, String signedPdfFilePath, Certificate certCode, String password, List<byte[]> crlData, List<byte[]> ocspData, String TSAUrl, int pageIndex, String filedName, int left, int top, int width, int height) throws PkiException

说明：

对PDF使用签章图片在指定的域进行签名

参数：

pdfFilePath 需要签名的pdf路径

byteSeal 签章的图片文件的byte数据。（不能过大）

signedPdfFilePath 保存签名后pdf的路径

certCode 证书编码

password 设备PIN码

crlData 验证的crl数据（可以为空)

ocspData 验证的ocsp数据（可以为空）

TSAUrl 时间戳服务器地址（可以为空）

pageIndex pdf 页码，从 1 开始

filedName pdf 签名域，（可以为空，新建）

left pdf 签章的left点（PDF 坐标系为左下叫为原点）

top pdf 签章的 top点 （PDF 坐标系为左下叫为原点）

width pdf 签章的宽度

height pdf 签章的高度

返回：无

### PDF签名(不可见)

原型：

void signPDFFile(String pdfFilePath, String signedPdfFilePath, Certificate certCode, String password, List<byte[]> crlData, List<byte[]> ocspData, String TSAUrl, String filedName) throws PkiException

说明：

对PDF在指定的域进行签名。签名区域不可见。

参数：

pdfFilePath 需要签名的pdf路径

signedPdfFilePath 保存签名后pdf的路径

certCode 证书编码

password 设备PIN码

crlData 验证的crl数据（可以为空)

ocspData 验证的ocsp数据（可以为空）

TSAUrl 时间戳服务器地址（可以为空）

filedName pdf 签名域，（可以为空，新建）

返回：

无

### PDF验证

原型：

boolean verifySignedPdfFile(String signedPdfFilePath, int verifyLevel) throws PkiException

说明：

对签名PDF进行验证，仅验证PDF签名值的有效性。

参数：

signedPdfFilePath 需要验证的PDF 路径

verifyLevel NetcaPKIConst.SignedData.VERIFY\_LEVEL\_VERIFY\_SIGNATURE\_ONLY

返回：

验证通过返回true。

### PDF验证

原型：

boolean verifySignedPdfFile(String signedPdfFilePath, int verifyLevel, PdfVerifyCallback callback) throws PkiException;

说明：

对签名PDF进行验证，仅验证PDF签名值的有效性。签名有效时，回调中返回签名的域名，以及签名的证书。

参数：

signedPdfFilePath 需要验证的PDF 路径

verifyLevel 验证级别

NetcaPKIConst.SignedData.VERIFY\_LEVEL\_VERIFY\_SIGNATURE\_ONLY

callback 验证签名域与证书的回调。

返回：

验证通过返回true

## SignedDataSignInterface接口

### 添加证书的CRL

原型：

void addCRL(byte [] crl) throws PkiException;

说明：

添加证书的CRL,可以逐个添加签名证书，CA证书的CRL

参数：

crl,用户自行下载的CRL数据

返回：

无

### 添加Adobe 签名属性的CRL

原型：

void addAdbeRevocationInfoArchivalCRL (byte[] crl) throws PkiException;

说明：

添加Adobe 签名属性CRL

参数：

crl, 证书的CRL

返回：

无

### 添加证书的OCSP

原型：

void addOCSP(byte[] ocsp) throws PkiException;

说明：

用户个人请求的OCSP 添加OCSP，可以逐个添加签名证书的，CA证书的OCSP验证结果

参数：

ocsp ocsp数据

返回：无

### 添加Adobe 签名属性的OCSP

原型：

void addAdbeRevocationInfoArchivalOCSP (byte[] oscp) throws PkiException;

说明：

添加Adobe 签名属性OCSP

参数：

ocsp数据

返回：无

### 设置是否带签名原文

原型：

boolean setDetached(boolean value) throws PkiException;

说明：

设置是否带签名原文

参数：

value,true表示不带原文，false表示带原文

返回：

成功返回true

### 设置签名证书

原型：

boolean setSignCertificate(Certificate cert) throws PkiException;

说明：

设置签名证书

参数：

cert，签名证书

返回：

成功返回true

### 设置签名算法

原型：

boolean setSignAlgorithm(int index, int algo) throws PkiException;

说明：

设置签名算法，应该先设置签名证书，后设置签名算法

参数：

index，签名索引值，从0算起

algo,签名算法，参考NetcaPKIConst.Signature

返回：

成功返回true

### 设置包括证书的标志

原型：

boolean setIncludeCertOption(int value) throws PkiException;

说明：

设置包括证书的标志

参数：

value,包含证书的标识,具体参考NetcaPKIConst.SignedData里的说明

返回：

成功返回true

### 一步完成签名

原型：

byte[] sign(String pwd, byte[] tbs, int offset, int length) throws PkiException;

说明：

一步完成签名，使用与签名原文比较小时，应先调用前面的4个接口后才调用本接口

参数：

pwd, PIN码

tbs,签名原文

offset,偏移量

length,用于签名的数据长度

返回：

成功返回签名数据

### 多步完成签名初始化

原型：

byte[] signInit(String pwd) throws PkiException;

说明：

多步完成签名，使用与签名原文较大时，应先调用前面的接口3.10.5至3.10.8接口后才调用本接口。多步完成的签名结果为 signInit, 多次调用（1..n）signUpdate，signFinal三者最后的返回值的拼接。

参数：

pwd PIN码 。PIN码为空时，需调用updateActivity 用于弹框。

返回：

成功返回签名的开始部分

### 多步完成签名的update

原型：

byte[] signUpdate( byte[] tbs, int offset, int length) throws PkiException;

说明：

输入原文进行Hash,可以多次调用。多步完成的签名结果为 signInit, 多次调用（1..n）signUpdate，signFinal三者最后的返回值的拼接

参数：

tbs,签名原文

offset,偏移量

length,用于计算的数据长度

返回：

成功返回产生的部分签名

### 多步完成签名的结束

原型：

byte[] signFinal()throws PkiException;

说明：

成功返回签名的最后部分

多步完成的签名结果为 signInit, 多次调用（1..n）signUpdate，signFinal三者最后的返回值的拼接

参数：无

返回：

签名数据的最后部分

### 多步完成带时间戳的签名的结束

原型：

byte[]addTimeStamp (String tsaServer) throws PkiException;

说明：

多步完成签名，或者单步签名后调用该接口，结束返回带时间戳的签名

参数：

tsaName,时间戳地址

返回：

签名数据

## SignedDataVerifyInterface接口

### 设置验证级别

原型：

void setVerifyLevel(int level) throws PkiException;

说明：

设置验证级别，所有验证接口前应该调用此接口，做验证级别的设置，

如果不设置则使用默认的验证级别,只验证签名。

参数：

level,验证级别，参考

NetcaPKIConst.SignedData.VERIFY\_LEVEL\_VERIFY\_SIGNATURE\_ONLY

返回：无

### 一步完成验证

原型：

boolean signedDataVerify( byte[] data, byte[] sign) throws PkiException;

说明：

验证SignedData签名,一步完成验证

参数：

data,签名原文

sign，SignedData编码

返回：

验证成功返回true，失败返回false.

### 一步完成验证

原型：

boolean signedDataVerify2(in byte[] data, int offset, int length, in byte[] sign) throws PkiException;

说明：

验证SignedData签名,一步完成验证

参数：

data,签名原文

offset,偏移量

length,data中用于验证的数据长度

sign，SignedData编码

signOffset,SignedData编码偏移量

signLength,SignedData编码中用于验证的数据长度

返回：

验证成功返回true，失败返回false.

### 判断是否为不带原文的SignedData

原型：

boolean isDetachedSignedData( byte[] sign) throws PkiException;

说明：

判断是否是不带原文的SignedData格式的数据，只需要输入前面的一小部分数据，不需要整个SignedData

参数：

sign,SignedData编码

返回：

是，返回true

### 判断是否为SignedData

原型：

boolean isSignedData(in byte[] sign) throws PkiException;

说明：

判断是否是SignedData格式的数据，只需要输入前面的一小部分数据，不需要整个SignedData

参数：

sign,SignedData编码

返回：

是，返回true

### 不带原文的多步验证初始化

原型：

boolean detachedVerifyInit( byte[] signedData, int offset, int length) throws PkiException;

说明：

不带原文的多步验证初始化

参数：

signedData，数字信封的DER编码，

offset,移位长度

length,SignedData编码中用于验证的数据长度

返回：

成功返回true

### 不带原文的多步验证update

原型：

boolean detachedVerifyUpdate( byte[] data, int offset, int length) throws PkiException;

说明：

不带原文的多步验证update

参数：

signedData SignedData签名数据。

offset,移位长度

length,SignedData编码中用于验证的数据长度

返回：

成功返回true

### 不带原文的多步验证结束

原型：

boolean detachedVerifyFinal()throws PkiException;

说明：

不带原文的多步验证结束

参数：无

返回：

成功返回true

### 带原文的多步验证初始化

原型：

boolean verifyInit()throws PkiException;

说明：

开始多步验证带原文的签名

参数：无

返回：

成功返回true

### 带原文的多步验证update

原型：

byte[] verifyUpdate(in byte[] signature, int offset, int length) throws PkiException;

说明：

多步验证带签名原文的签名,可以多次调用

参数：

signature,SignedData编码

offset，偏移量

length,SignedData编码中用于验证的数据长度

返回：

成功返回签名原文

### 带原文的多步验证结束

原型：

boolean verifyFinal()throws PkiException;

说明：

结束带原文的签名验证

参数：

返回：

成功返回true

### 验证后获取签名证书

原型：

List< Certificate> getSignCerts()throws PkiException;

说明：

获取签名证书

参数：无

返回：

返回签名的证书编码

### 释放资源

原型：

void free();

说明：

释放资源

参数：无

返回：无

## BluetoothInterface 接口

### 连接蓝牙设备

原型：

void connect(String mac, String name) throws PkiException;

说明：

连接蓝牙， 连接失败，或者设备不支持抛出异常

参数：

mac 蓝牙的mac地址

name 蓝牙的name

返回：无

### 断开蓝牙

原型：

void disconnect(String mac) throws PkiException;

说明：

参数：

mac 蓝牙的mac地址

返回：无

### 是否支持蓝牙设备

原型：

boolean isSupport (String mac, String name) throws PkiException;

说明：

判断是否支持蓝牙

参数：

mac 蓝牙的mac地址

name 蓝牙的name

返回

支持返回true。

### 获取已连接的蓝牙设备

原型：

List<BluetoothDeviceItem> getConnectDeviceList();

说明：

已连接蓝牙设备列表

参数：无

返回：

返回已连接蓝牙设备列表

## QrcodeSignInterface接口

### 初始化url

原型：

void init(String url) throws PkiException;

说明：

初始化url，url格式不正常抛出异常

参数：

url 扫码的url

返回：无

### 判断是否需要授权码

原型：

boolean isNeedAuthCode() throws PkiException;

说明：

判断是否需要授权码

参数：无

返回：

需要授权码返回true

### 请求签名数据

原型：

String requestSignContent(String authCode) throws PkiException;

说明：

请求签名数据

参数：

authCode 需要授权码只输入，否则输入空

返回：

签名的内容

### 签名并发送数据

原型：

String signContent(Certificate cert, String pin) throws PkiException;

说明：

签名并发送数据。

参数：

cert 证书对象

pin 设备pin码

返回：

成功跳转的url(可能为空，SDK版可忽视)

## TimeStampInterface接口

### 请求timestapme 服务器token

原型：

byte[] requestTimeStampToken(String tsaUrl, byte[]data, String hashAlgo)throws PkiException

说明：

请求timestapme 服务器token

参数：

tsaUrl 服务器地址

data 原文数据

hashAlgo hash 算法，

支持NetcaPKIConst.Hash.SHA1, NetcaPKIConst.Hash.SHA256

返回：

成功 token

### 初始化验证时间戳

原型：

void verifyInit(byte[]data, byte[]token) throws PkiException;

说明：

初始化验证

参数：

data 原文

token 时间戳数据

返回：

无

### 获取tsa证书

原型：

Certificate getTsaCert() throws PkiException

说明：

验证成功后获取tsa证书

参数：无

返回：

tsa证书

### 获取时间

原型：

Date getTime() throws PkiException;

说明：

验证成功获取时间

参数：无

返回：

时间

### 获取策略

原型：

String getPolicy() throws PkiException;

说明：

验证成功获取策略

参数：无

返回：

返回策略。

### 获取序列号

原型：

String getSerialNumber() throws PkiException;

说明：

获取序列号

参数：无

返回：

返回序列号

## PinManageSystemInterface接口

### 设置设备对象

原型：

void setDevice(DeviceInterface device);

说明：

设置设备对象。

V1.1.10开始。

参数：

device 设备对象。必须是67设备类型的设备。

返回：无

### 注册管理员密钥

原型：

void registerSoPin(String phone, String userPin) throws PkiException;

说明：

注册管理员密钥。

需要先设置口令管理系统服务器地址，V1.1.10开始。

参数：

phone 手机号，可选

userPin 用户口令，用于获取管理员加密口令

返回：无

### 请求获取管理员密钥

原型：

String requestGetSoPin() throws PkiException;

说明：

请求获取管理员密钥。

需要先设置口令管理系统服务器地址。V1.1.10开始。

参数：无

返回：

返回需要签名数据

### 重置用户口令

原型：

void unlockUserPin(String signature, String newUserPin) throws PkiException;

说明：

重置用户口令。

需要先设置口令管理系统服务器地址。V1.1.10开始

参数：

signature 对 requestGetSoPIN 返回数据的utf8进行签名的值，base64编码。

newPin 输入新的PIN码。

返回：无

## HashInterface接口

### 一步完成Hash

原型：

byte[] hash(int hashAlgo, byte[] data) throws PkiException;

说明：

异步完成Hash ，V1.1.10开始

参数：

hashAlgo Hash算法 参考NetcaPKIConst.Hash.SHA256

data 数据

返回：

返回Hash数据

### 设置Hash算法

原型：

void hashInit(int hashAlgo) throws PkiException;

说明：

设置Hash算法

参数：

hashAlgo 参考NetcaPKIConst.Hash.SHA256

返回：无

### 多步 update Hash

原型：

void hashUpdate(byte[] data) throws PkiException;

说明：

多步 updateHash

参数：

data 待Hash数据

返回：无

### 完成Hash计算

原型：

byte[] hashFinal() throws PkiException;

说明：

完成Hash计算。

参数：无

返回：

返回Hash 数据。

## CipherInterface接口

### 设置padding

原型：

void setPadding(int padding) throws PkiException;

说明：

设置padding

参数：

padding 参考NetcaPKIConst.Cipher#PADDING\_PKCS5

返回：

无

### 设置对称加解密的Key

原型：

void setKey(byte[] key) throws PkiException;

说明：

设置对称加解密的Key 。V1.1.10开始

参数：

key 对称加解密的Key

返回：无

### 设置对称加解密的IV

原型：

void setIV(byte[] iv) throws PkiException;

说明：

设置对称加解密的IV。

参数：

iv 设置的IV值。

返回：无

### 初始化对称加解密

原型：

void init(boolean isEnc, int algo) throws PkiException;

说明：

初始化对称加解密，需要先设置key值，padding或者iv。

参数：

isEnc 是否为加密

algo 加密算法参考 NetcaPKIConst.Cipher#AES\_CBC

返回：无

### 一步完成对称加解密

原型：

byte[] cipher(byte[] data) throws PkiException;

说明：

一步完成对称加解密

参数：

data 对称加解密数据。

返回：

返回加密或解密的数据

### 多步对称加解密

原型：

byte[] update(byte[] data) throws PkiException;

说明：

多步对称加解密，V1.1.10开始

参数：

data 对称加解密数据。

返回：

返回加密或解密的数据

### 完成对称加解密

原型：

byte[] doFinal() throws PkiException;

说明：

结束对称加解密

参数：无

返回：

返回加密或解密的数据

## MobileKeyInterface接口

### 设置口令弹出UI框类型

原型：

void setPinUIType(int pinUIType);

说明：

设置口令弹出UI框类型。V1.1.10开始

参数：

pinUIType 口令弹出框类型。

返回：无

### 是否需要密码缓存

原型：

void setNeedPinCache(int pinCacheType);

说明：

\*是否需要密码缓存。

\* @param pinCacheType口令弹出框类型。

\* V1.1.10开始。

### 显示移动电子密钥界面

原型：

void connectToPC(android.support.v7.app.AppCompatActivity activity, String qrcode, String parameter, MobileKeyCallback callback);

说明：

连接并显示移动电子密钥界面。退出界面时或关闭屏幕时，会断开连接。V1.1.10开始

参数：

activity 当前Activity

qrcode 扫码的二维码

parameter 保留参数。

callback 回调函数。

返回：无

# 对象

## NetcaCertInfo 对象

通过证书对象获取解析后的证书信息。

|  |  |
| --- | --- |
| 方法 | 意义 |
| getKeyUsage | 密钥用法。参考常量说明 证书密钥用法。 |
| getPublicKeyBits | 证书的公钥长度。 |
| getPublicKeyAlgorithm | 证书公钥算法。参考常量说明 密钥对算法 |
| getSerialNumber | 证书的序列号 |
| getSubject | 证书的主体 |
| getSubjectC | 主体的C |
| getSubjectCN | 主体的CN,如果没有或者出错返回null。 |
| getSubjectEmail | 主体的Email,如果没有或者出错返回null。 |
| getSubjectL | 主体的,如果没有或者出错返回null。L |
| getSubjectO | 主体的O,如果没有或者出错返回null。 |
| getSubjectOU | 主体的OU,如果没有或者出错返回null。 |
| getSubjectST | 主体的ST,如果没有或者出错返回null。 |
| getValidityEnd | 证书的有效期结束时间 |
| getValidityStart | 证书的有效期开始时间 |
| getIssuer | 颁发者 |
| getIssuerC | 颁发者的C，如果没有或者出错返回null。 |
| getIssuerCN | 颁发者的CN，如果没有或者出错返回null。 |
| getIssuerEmail | 颁发者的Email，如果没有或者出错返回null。 |
| getIssuerL | 颁发者的L，如果没有或者出错返回null。 |
| getIssuerO | 颁发者的O，如果没有或者出错返回null。 |
| getIssuerOU | 颁发者的OU，如果没有或者出错返回null。 |
| getIssuerST | 颁发者的ST，如果没有或者出错返回null。 |
| getDer | 证书对象的der编码。 |
| getSha1FingerPrint | 证书SHA1姆印。 |
| getSha256FingerPrint | 证书SHA256 姆印。 |
| getSignAlgorithm | 证书签名算法。 |
| isSignCert | 是否为签名证书。 |
| isEncCert | 是否为加密证书。 |

# 回调

## MobileKeyCallback

### 连接成功

原型：

void onConnectedPC();

说明：

连接成功回调

### 连接失败

原型：

void onFailToConnectPC(int code, String msg);

说明：

连接失败

参数：

code 错误码。

msg 错误信息。

返回：

无

### 连接断开

原型：

void onDisconnectPC(int code, String msg);

说明：

连接断开

参数：

code 错误码。

msg 错误信息。

返回：无

## PdfVerifyCallback

### PDF签名值验证成功回调证书

原型：

void verifyCert(String signatureName, Certificate certificate);

说明：

PDF签名值验证成功回调证书

参数：

signatureName 签名域名

certificate 签名证书。

返回：无

# 调用例程

详细的例程见SDKDemo工程中的例子。

# 常见问题

1. 问：蓝牙搜索不出设备。

答：手机蓝牙设备必须支持BLE，系统必须在Android 4.4以上，并开启蓝牙。部分系统需要地理位置权限。

1. 问: 签名，哈希，解密时间过久卡住界面了。

答：签名，哈希，解密等耗时操作在线程中操作 。

1. 问: PIN码框和证书选择框没有弹出来就崩溃了。

答，需要增加以下代码

@Override

protected void onResume() {

super.onResume();

//需增加这段代码

netcaCryptoInterface.updateActivity(this);

}

@Override

protected void onPause() {

super.onPause();

//需增加这段代码

netcaCryptoInterface.updateActivity(null);

}

1. 问：文件软设备读取不到。

答：需要有读写存储权限，不能修改so文件。

不修改签名值配置。

android{

packagingOptions {

doNotStrip '\*/armeabi/.\*so'

doNotStrip '\*/armeabi-v7a/.\*so'

doNotStrip '\*/x86/.\*so'

doNotStrip '\*/arm64-v8a/.\*so'

}

}

1. 问：混淆配置。

-dontwarn net.netca.pki.\*\*

-keep class net.netca.pki.\*\*{\*;}