Android NETCA PKI SDK接口说明

|  |  |
| --- | --- |
| 版 本 | 说 明 |
| V1.1.2 | * 修改PDF SM2签名。CNCA.DocVerify |

# 简介

在Android端上NETCA PKI中间件接口以aar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等功能。部分接口是基于网证通安全客户端（Android版）实现的，因此，使用接口时必须安装了网证通安全客户端。

## 体系架构

NETCA PKI SDK Android版的体系结构如图1.1所示。

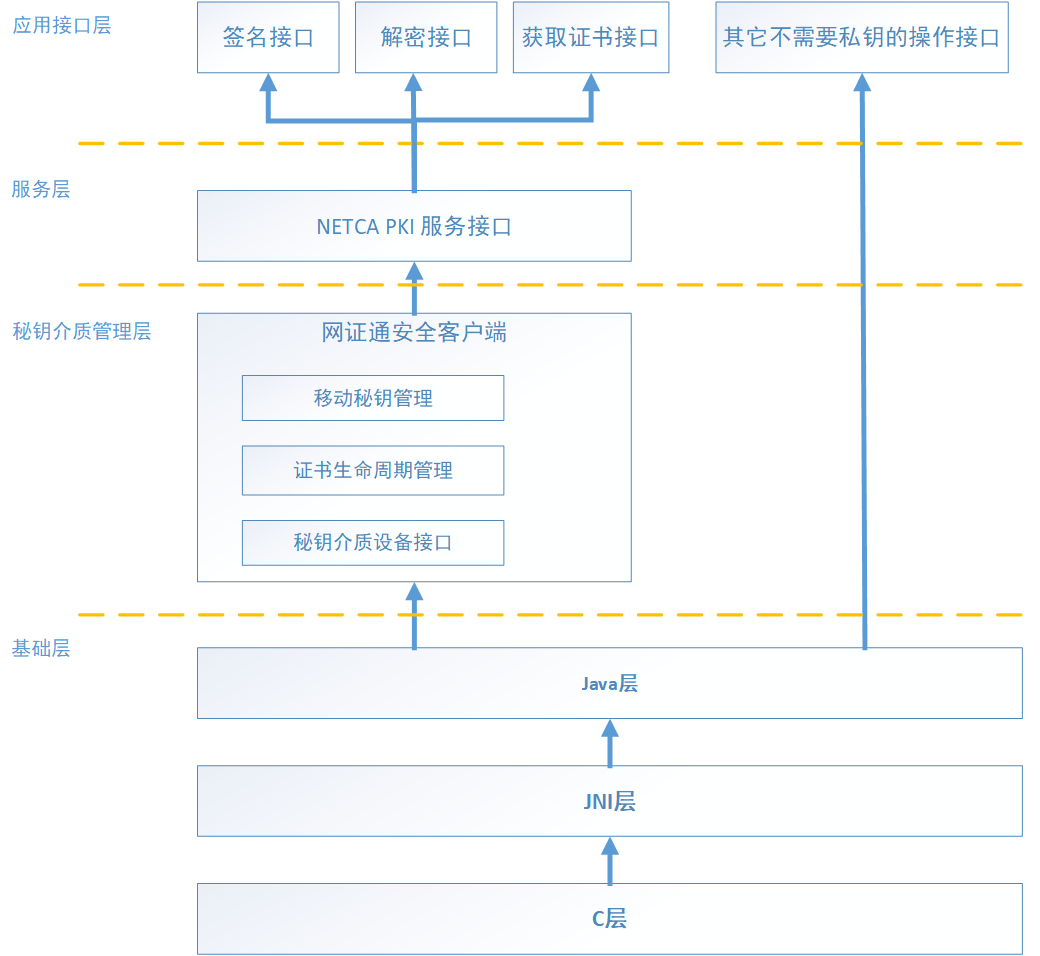


图1.1 NETCA PKI SDK体系结构图

NETCA PKI SDK Android版的体系分为四层，从上到下为应用接口层，服务层，密钥介质管理层和基础层。

应用接口层为封装后的应用接口，为第三方应用提供签名、解密、获取证书、Hash等各种应用调用，第三方应用可以直接引入jar包直接使用。

服务层为应用层需要私钥参与计算的部分接口和从已经接入的设备中选择证书提供支撑。

密钥介质管理层是NETCA PKI SDK的核心，其实现了对各种密钥设备（如蓝牙，OTG，文件型）的自动连接断开的管理，实现了证书生命周期的管理，移动密钥的管理，实现了服务接口。

基础层包括C代码层，JNI代码层和JAVA代码层，主要是为密钥介质管理层和应用接口层提供支持。其中，应用接口层中不需要密钥参与计算的接口都直接使用基础层的接口实现，例如Hash,时间戳等。

## 2、接口对象

NETCA PKI中间件包含了8个接口，每个接口由对应的类对象实现，类名称和其对应的类名全称见表1 NETCA PKI中间件接口列表。

|  |  |  |
| --- | --- | --- |
| 类名称 | 类名全称 | 备注 |
| Cert | net.netca.pki.crypto.keyx.Cert | 解析证书 |
| EnvelopedDataDecrypt | net.netca.pki.crypto.keyx.EnvelopedDataDecrypt | 数字信封解密 |
| EnvelopedDataEncrypt | net.netca.pki.crypto.keyx. EnvelopedDataEncrypt | 数字信封加密 |
| Hash | net.netca.pki.crypto.keyx.Hash | 计算HASH |
| SignedDataSign | net.netca.pki.crypto.keyx. SignedDataSign | SignedData生成 |
| SignedDataVerify | net.netca.pki.crypto.keyx. SignedDataVerify | SignedData验证 |
| Hex | net.netca.pki.crypto.keyx. Hex | Hex编码解密 |
| Signature | net.netca.pki.crypto.keyx. Signature | P1 签名 |

表1 NETCA PKI中间件接口列表

## 3、启动服务

需要在AndroidManifest.xml添加自定义权限

<uses-permission android:name="net.netca.pki.crypto.service.permission.aidl" />

### 1). 在onStart中启动

@Override

protected void onStart() {

super.onStart();

try {

GlobalSetting.getInstance().init(this, this);

}catch (Exception e){

e.printStackTrace();

}

}

### 2).在onStop中释放

@Override

protected void onStop() {

super.onStop();

GlobalSetting.getInstance().end();

}

# 常量说明

NETCA PKI中间件所使用到的常量被封装在类Hash , Cert, Signature, SignedData,

Hash类，包含中间件支持的HASH算法

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，

## Hash 常量

|  |  |  |  |
| --- | --- | --- | --- |
| 常量 | 类型 | 值 | 说明 |
| MD5 | 字符串 | MD5 | MD5哈希算法 |
| SHA1 | 字符串 | SHA1 | SHA1哈希算法 |
| SHA224 | 字符串 | SHA224 | SHA224哈希算法 |
| SHA256 | 字符串 | SHA256 | SHA256哈希算法 |
| SHA384 | 字符串 | SHA384 | SHA384哈希算法 |
| SHA512 | 字符串 | SHA512 | SHA512哈希算法 |
| SM3 | 字符串 | SM3 | SM3哈希算法 |
| SHA512\_224 | 字符串 | SHA512\_224 | SHA512\_224哈希算法 |
| SHA512\_256 | 字符串 | SHA512\_256 | SHA512\_256哈希算法 |
| SHA3\_224 | 字符串 | SHA3\_224 | SHA3\_224哈希算法 |
| SHA3\_256 | 字符串 | SHA3\_256 | SHA3\_256哈希算法 |
| SHA3\_384 | 字符串 | SHA3\_384 | SHA3\_384哈希算法 |
| SHA3\_512 | 字符串 | SHA3\_512 | SHA3\_512哈希算法 |

## Cert 常量

|  |  |  |  |
| --- | --- | --- | --- |
| 常量 | 类型 | 值 | 说明 |
| ATTRIBUTE\_CRL\_URL | 数字 | 56 | 证书CRL地址属性 |
| ATTRIBUTE\_CSP\_NAME | 数字 | 55 | 证书CSP名 属性 |
| ATTRIBUTE\_OCSP\_URL | 数字 | 58 | 证书OCSP地址属性 |
| CERT\_TYPE\_SIGNATURE | 字符串 | Signature | 选择签名证书 |
| CERT\_TYPE\_ENCRYPT | 字符串 | Encrypt | 选择加密证书 |

## Signature 常量

|  |  |  |  |
| --- | --- | --- | --- |
| 常量 | 类型 | 值 | 说明 |
| SHA1WITHRSA | 字符串 | SHA1WITHRSA | SHA1WITHRSA签名算法 |
| SHA224WITHRSA | 字符串 | SHA224WITHRSA | SHA224WITHRSA签名算法 |
| SHA256WITHRSA | 字符串 | SHA256WITHRSA | SHA256WITHRSA签名算法 |
| SHA384WITHRSA | 字符串 | SHA384WITHRSA | SHA384WITHRSA签名算法 |
| SHA512WITHRSA | 字符串 | SHA512WITHRSA | SHA512WITHRSA签名算法 |
| SM3WITHSM2 | 字符串 | SM3WITHSM2 | SM3WITHSM2签名算法 |
| MD5WITHRSA | 字符串 | MD5WITHRSA | MD5WITHRSA签名算法 |
| SHA1WITHSM2 | 字符串 | SHA1WITHSM2 | SHA1WITHSM2签名算法 |
| SHA224WITHSM2 | 字符串 | SHA224WITHSM2 | SHA224WITHSM2签名算法 |
| SHA256WITHSM2 | 字符串 | SHA256WITHSM2 | SHA256WITHSM2签名算法 |
| SHA384WITHSM2 | 字符串 | SHA384WITHSM2 | SHA384WITHSM2签名算法 |
| SHA512WITHSM2 | 字符串 | SHA512WITHSM2 | SHA512WITHSM2签名算法 |
| SM3WITHRSA | 字符串 | SM3WITHRSA | SM3WITHRSA签名算法 |

## SignedDataSign常量

|  |  |  |  |
| --- | --- | --- | --- |
| 常量 | 类型 | 值 | 说明 |
| INCLUDE\_CERT\_OPTION\_SELF | 数字 | 2 | 包含本身证书 |
| INCLUDE\_CERT\_OPTION\_NONE | 数字 | 1 | 不包含 |
| INCLUDE\_CERT\_OPTION\_CERTPATHWITHROOT | 数字 | 4 | 包含证书链+根证书 |
| INCLUDE\_CERT\_OPTION\_CERTPATH | 数字 | 3 | 包含证书链 |

## SignedDataVerify 常量

|  |  |  |  |
| --- | --- | --- | --- |
| 常量 | 类型 | 值 | 说明 |
| VERIFY\_LEVEL\_NO\_VERIFY | 数字 | 0 | 不验证 |
| VERIFY\_LEVEL\_VERIFY\_SIGNATURE\_ONLY | 数字 | 1 | 仅验证签名 |
| VERIFY\_LEVEL\_VERIFY\_CERT | 数字 | 2 | 验签名和证书 |

# 接口详细说明

## GlobalSetting

### 初始化

原型:

public void init **(**Context context, BindCallback callback**)** throws PkiException

说明：

初始化服务

### 释放

原型:

public void end**()**throws PkiException

说明：

释放服务对象

## Cert

### 根据条件获取多张证书编码

原型：

public static List**<**Cert**>** selectCerts**(**String certType**)** throws PkiException

说明：

\* 根据条件获取多张证书编码  
 \* type - 证书的类型。参见Cert中的证书类型常量  
 \* 证书列表

\* 失败抛出PkiException

### 获取一张证书编码

原型：

public static Cert selectSingleCert**(**String certType**)** throws PkiException

说明：

\* 获取一张证书编码(如果只有一张，直接返回，多张需要用户选择)  
 \* type - 证书的类型。参见Cert中的证书类型常量  
 \* 成功返回证书

\* 失败抛出PkiException

### 构造证书链

原型：

public List**<**Cert**>** buildCertPath**()**throws PkiException

说明：

\*构造证书链  
 \* 成功返回构造的证书链，排序是根证书第一个，签名证书最后一个

\* 失败抛出PkiException

### 计算证书的微缩图

原型：

public byte**[]** computeThumbprint**(**String hashAlgo**)** throws PkiException  
说明：

\*计算证书的微缩图  
 \* hashAlgo,hash算法，常见的hash算法包括，sha256,sha512等，具体参考Hash常量中的描述  
 \* 成功返回证书指定算法的微缩图

\* 失败抛出PkiException

### 计算Z值

原型：

public byte**[]** computeSM2Z**(**byte**[]** id**)** throws PkiException

说明：

\* 计算sm2证书Z值   
 \* id参数 可以为null

\* 成功返回Z值

\* 失败抛出PkiException

### 获取证书的DER编码

原型：

public byte**[]** derEncode**()**throws PkiException  
说明：  
 \*获取证书的DER编码  
 \*成功返回证书的DER编码

\* 失败抛出PkiException

### 获取证书的Base64编码

原型：

public String getB64Cert**()**throws PkiException

说明：  
 \* 获取证书的Base64编码  
 \* 成功返回证书的Base64编码

\* 失败抛出PkiException

### 获取Attribute属性

原型：

public String getAttribute**(**int id**)** throws PkiException

说明：  
 \* 获取attribute属性

\* id 参考Cert常量表   
 \* 证书属性

\* 失败抛出PkiException

### 非对称加密

原型：

public byte**[]** encrypt**(**byte**[]** data**)** throws PkiException

说明：  
 \* 非对称加密  
 \* data 加密明文数据

\* 成功返回加密密文

\* 失败抛出PkiException

### 非对称解密

原型：

public byte**[]** decrypt**(**byte**[]** data**,** String pin**)** throws PkiException

说明：  
 \* 非对称解密

\* data 密文数据

\* pin码  
 \* 成功返回解密数据

\* 失败抛出PkiException

### 获取CRL地址

原型：

public String getCRLURL**()**throws PkiException

说明：  
 \* 获取CRL地址

\* 成功返回CRL地址

\* 失败抛出PkiException

### 获取OCSP地址

原型：

public String getOCSPURL**()**throws PkiException

说明：  
 \* 获取OCSP地址

\* 成功返回OCSP地址

\* 失败抛出PkiException

### 获取Oid的扩展值

原型：

public byte**[]** getExtensionValue**(**String oid**)** throws PkiException

说明：  
 \*获取Oid的扩展值

\* oid 值

\* 成功返回oid数据

\* 失败抛出PkiException

### 获取证书的颁发者名

原型：

public String getIssuer**()** throws PkiException

说明：   
 \*获取证书的颁发者名  
 \* 成功返回证书的颁发者

\* 失败抛出PkiException

### 获取颁发者的C

原型：

public String getIssuerC**()** throws PkiException

说明：

\* 获取颁发者的C   
 \* 成功返回证书的颁发者的C

\* 失败抛出PkiException

### 获取颁发者的CN

原型：

public String getIssuerCN**()** throws PkiException

说明：  
 \*获取颁发者的CN   
 \* 成功返回证书的颁发者的CN

\* 失败抛出PkiException

### 获取颁发者的L

原型：

public String getIssuerL**()** throws PkiException

说明：   
 \*获取颁发者的L   
 \* 成功返回证书的颁发者的L

\* 失败抛出PkiException

### 获取颁发者的O

原型：

public String getIssuerO**()** throws PkiException

说明：   
 \*获取颁发者的O   
 \* 成功返回证书的颁发者的O

\* 失败抛出PkiException

### 获取颁发者的Email项

原型：

public String getIssuerEmail**()** throws PkiException

说明：   
 \*获取颁发者的Email   
 \* 成功返回证书的颁发者的Email

\* 失败抛出PkiException

### 获取颁发者的OU

原型：

public String getIssuerOU**()** throws PkiException

说明：   
 \*获取颁发者的OU   
 \* 成功返回证书的颁发者的OU

\* 失败抛出PkiException

### 获取颁发者的ST

原型：

public String getIssuerST**()** throws PkiException

说明：   
 \*获取颁发者的ST   
 \* 成功返回证书的颁发者的ST

\* 失败抛出PkiException

### 获取证书的密钥用法

原型：

public int getKeyUsage**()** throws PkiException

说明：   
 \*获取证书的密钥用法，没有该扩展则返回-1

\* 失败抛出PkiException

### 获取证书的序列号

原型：

public String getSerialNumber**()** throws PkiException  
说明：  
 \*获取证书的序列号  
 \* 成功返回证书的颁发者的序列号

\* 失败抛出PkiException

### 获取字符串类型的扩展值

原型：

public String getStringExtension**(**String extOid**)** throws PkiException  
说明：  
 \*获取字符串类型的扩展值  
 \* extOid，扩展id.  
 \* 成功返回证书的颁发者的扩展

\* 失败抛出PkiException

### 获取证书的主体名

原型：

public String getSubject**()** throws PkiException

说明：  
 \*获取证书的主体名  
 \* 成功返回证书的主体名

\* 失败抛出PkiException

### 获取主体的C

原型：

public String getSubjectC**()** throws PkiException

说明：  
 \*获取主体的C   
 \*成功返回证书的主体的C

\* 失败抛出PkiException

### 获取主体的CN

原型：

public String getSubjectCN**()** throws PkiException

说明：  
 \*获取主体的CN   
 \* 成功返回证书的主体的CN

\* 失败抛出PkiException

### 获取主体的Email

原型：

public String getSubjectEmail**()** throws PkiException

说明：  
 \*获取主体的Email   
 \* 成功返回证书的主体的Email

\* 失败抛出PkiException

### 获取主体的L

原型：

public String getSubjectL**()** throws PkiException

说明：  
 \*获取主体的L   
 \* 成功返回证书的主体的L

\* 失败抛出PkiException

### 获取主体的O

原型：

public String getSubjectO**()** throws PkiException

说明：  
 \*获取主体的O   
 \* 成功返回证书的主体的O

\* 失败抛出PkiException

### 获取主体的OU

原型：

public String getSubjectOU**()** throws PkiException

说明：  
 \*获取主体的OU   
 \* 成功返回证书的主体的OU

\* 失败抛出PkiException

### 获取主体的ST

原型：

public String getSubjectST**()** throws PkiException

说明：  
 \*获取主体的ST   
 \* 成功返回证书的主体的ST

\* 失败抛出PkiException

### 获取证书的有效期结束时间

原型：

public Date getValidityEnd**()** throws PkiException

说明：  
 \*获取证书的有效期结束时间  
 \* 成功返回证书的有效期结束时间

\* 失败抛出PkiException

### 获取证书的有效期开始时间

原型：

public Date getValidityStart**()** throws PkiException

说明：  
 \*获取证书的有效期开始时间  
 \* 成功返回证书的有效期开始时间

\* 失败抛出PkiException

### 判断证书是否为sm2

原型：

public boolean isSm2**()** throws PkiException

说明：  
 \* 判断证书是否为sm2   
 \* 是SM2证书返回true, RSA返回false

\* 失败抛出PkiException

### 判断证书是否包含oid的扩展

原型：

public boolean hasExtension**(**String oid**)** throws PkiException

说明：  
 \*判断证书是否包含oid的扩展  
 \* 有返回true

\* 失败抛出PkiException

## Hex

### Hex解码

原型：

public static byte**[]** decode**(**String data**)** throws PkiException

说明：  
 \* data 需要解密的hex数据  
 \*成功返回byte[]数据

\* 失败抛出PkiException

### Hex编码

原型：

public static String encode**(**byte**[]** data**)** throws PkiException

说明：

\* data 需要编码的数据  
 \*成功返回Hex编码数据

\* 失败抛出PkiException

## EnvelopedDataDecrypt

### 一步解密数字信封

原型：

public byte**[]** envelopedDataDecrypt**(**byte**[]** data**,** String pin**)** 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。

\* data，数字信封编码

\* pin ,设备密码

\* 成功返回原文

\* 失败抛出PkiException

### 多步解密数字信封初始化

原型：

public void decryptInit**(**String pin**)** 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

\* pin,解密密码

\* 失败抛出PkiException

### 多步解密数字信封update

原型：

public void decryptUpdate**(**byte**[]** data**,** int offset**,** int len**)** throws PkiException

说明：

\* 可多次调用

\* data，数字信封编码

\* offset,data的偏移量

\* len,用于解密的数据长度

\* 分步解密的,输入数字信封编码，进行解密。

\* 失败抛出PkiException

### 多步解密数字信封结束

原型：

public DataPlain decryptFinal**()** throws PkiException

说明：

\*结束数字信息解密

\* 成功返回解密数据流

\* 失败抛出PkiException

### 获取解密数字信封的加密证书

原型：

public Cert getDecryptCert**()** throws PkiException

说明：

\*解密成功后，获取解密证书

\* 成功返回所使用的解密证书

\* 失败抛出PkiException

### 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

## EnvelopedDataEncrypt

### 一步完成数字信封加密

原型：

public byte**[]** envelopedDataEncrypt**(**List**<**Cert**>** certs**,** byte**[]** data**)** throws PkiException

说明：

\*一步完成 数字信封加密

\* certs,加密使用的加密证书

\* data,原文

\* 成功返回加密密文

\* 失败抛出PkiException

### 多步完成数字信封加密添加证书

原型：

public void addCert**(**Cert cert**)** throws PkiException

说明：

\*添加加密证书，可多次添加不同的加密证书。

\* cert,加密证书

\* 失败抛出PkiException

### 多步完成数字信封加密初始化

原型：

public byte**[]** encryptInit**()** throws PkiException

说明：

\* 开始多步数字信封加密

\* 成功返回数字信封的开始部分

\* 失败抛出PkiException

### 多步完成数字信封加密update

原型：

public byte**[]** encryptUpdate**(**byte**[]** data**,** int offset**,** int len**)** 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

\* data,原文

\* offset,偏移位置

\* len,用于加密的长度

\* 成功返回加密部分的数字信封

\* 失败抛出PkiException

### 多步完成数字信封加密结束

原型：

public byte**[]** encryptFinal**()** throws PkiException

说明：

\*结束数字信封加密

\* 成功返回最后部分的数据

\* 失败抛出PkiException

## Hash

### 一步完成计算HASH

原型：

public static byte**[]** computeHash**(**byte**[]** data**,** String hashAlgo**)** throws PkiException

说明：

\*计算HASH,一次性的计算，

\* data，数据原文

\* hashAlgo,hash算法,参考Hash 常量部分的说明

\* 成功返回hash的结果

\* 失败抛出PkiException

### 设置HASH算法

原型：

public void init**(**String hashAlgo**)** throws PkiException

说明：

\*设置HASH算法，多步计算HASH值，一般用于较大的文件，分步读取计算

\* hashAlgo,hash算法,参考Hash 常量部分的说明

\* 失败抛出PkiException

### 多步计算HASH update

原型：

public void update**(**byte**[]** data**,** int offset**,** int len**)** throws PkiException

说明：

\*可多次调用，多步计算HASH值，一般用于较大的文件，分步读取计算

\* data，原文数据

\* offset,数组偏移量

\* len,用于计算的数据长度

\* 失败抛出PkiException

### 多步计算结束

原型：

public byte**[]** doFinal**()** throws PkiException

说明：

\*结束Hash运算，得到Hash值

## Signature

### 设置签名算法

原型：

public void init**(**String signatureAlgo**,** Cert cert**,** String pin**)** throws PkiException

说明：

设置签名算法

signAlgo,签名算法，具体参考Signature常量

cert,证书

pin, 签名时所用的密码；验证时不需要，可以设为null

失败抛出PkiException

### 对Hash值进行签名

原型：

public byte**[]** signHash**(**byte**[]** data**)** throws PkiException

说明：

对Hash值进行签名。

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

成功返回签名结果

失败抛出PkiException

### 输入数据进行Hash运算

原型：

public void signUpdate**(**byte**[]** data**,** int offset**,** int len**)** throws PkiException

说明：

输入数据进行Hash运算

data,数据原文

offset，位移

len,进行计算hash的数据长度

失败抛出PkiException

### 对update后的数据进行签名

原型：

public byte**[]** sign**()** throws PkiException

说明：

对update后的数据进行签名

成功返回签名结果

失败抛出PkiException

### 输入数据进行Hash运算

原型：

public void verifyUpdate**(**byte**[]** data**,** int offset**,** int len**)** throws PkiException

说明：

输入数据进行Hash运算

data,数据原文

offset，位移

len,进行计算hash的数据长度

失败抛出PkiException

### 验证签名

原型：

public boolean verify**(**byte**[]** siganture**)** throws PkiException

说明：

验证签名

signature,签名值

成功返回true，失败返回false

出错抛出PkiException

### 验证签名

原型：

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

说明：

验证签名

hash - Hash值

signature - 签名值

成功返回true，失败返回false

注意：SM3WithSM2的签名输入hash值不是原文的Hash值

失败抛出PkiException

## SignedDataSign

### 添加证书的CRL

原型：

public void addCRL**(**byte**[]** data**)** 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

crl,用户自行下载的CRL数据

失败抛出PkiException

### 添加adbe属性证书的CRL

原型：

public void addAdbeRevocationInfoArchivalCRL**(**byte**[]** data**)** throws PkiException

说明：

添加adbe签名属性的crl数据

失败抛出PkiException

### 添加证书的OCSP

原型：

public void addOCSP**(**byte**[]** data**)** 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

失败抛出PkiException

### 添加adbe属性证书的OCSP

原型：

public void addAdbeRevocationInfoArchivalOCSP**(**byte**[]** data**)** throws PkiException

说明：

OCSP URL从证书OCSP分布点中获取，然后请求对应的OCSP。

失败抛出PkiException

### 设置包括证书的标志

原型：

public void setIncludeCertOptions**(**int options**)** throws PkiException

说明：

\* 设置包括证书的标志

\* value,包含证书的标识,具体参考SignedDataSign常量里的说明

失败抛出PkiException

### 一步完成签名

原型：

public byte**[]** signedDataSign**(**Cert cert**,** byte**[]** source**,** String signatureAlgo**,** boolean detach**,** int includeOptions**,** String pin**,** String tsaURL**)** throws PkiException

说明：

\* 一步完成签名

\* cert,证书

\* source 待签名数据

\* signatureAlgo 签名算法

\* detach 不带原文

\* includeOptions SignedDataSign 常量。

\* pin 设备pin码

\* tsaURL 时间戳服务器 （可为空）

\* 失败抛出PkiException

### 多步完成签名初始化

原型：

public void signInit**(**Cert cert**,** boolean detach**,** String signAlgo**,** String pin**)** throws PkiException

说明：

\*多步完成签名，使用与签名原文较大时

\* cert 证书

\* detach 不带原文

\* signAlgo 签名算法

\* pin 设备pin码

\* 失败抛出PkiException

### 多步完成签名的update

原型：

public void signUpdate**(**byte**[]** data**,** int offset**,** int len**)** throws PkiException

说明：

\*输入原文进行Hash,可以多次调用

\* data,签名原文

\* offset,偏移量

\* length,用于计算的数据长度

\* 失败抛出PkiException

### 多步完成签名的结束

原型：

public DataPlain signFinal**()** throws PkiException

说明：

\*成功返回签名数据流

\* 失败抛出PkiException

## SignedDataVerify

### 设置验证级别

原型：

public void setVerifyLevel**(**int verifyLevel**)** 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,只验证签名和证书，并且验证签名证书是否作废，但是不验证CA证书是否作废

\*level,验证级别，参考SignedDataVerify常量

\* 失败抛出PkiException

### 一步完成带原文验证

原型：

public byte**[]** signedDataVerify**(**byte**[]** signeddata**)** throws PkiException

说明：

\*验证SignedData签名,一步完成验证

\* signeddata，SignedData编码

\* 验证成功返回原文.

\* 失败抛出PkiException

### 一步完成不带原文验证

原型：

public boolean signedDataDetachVerify**(**byte**[]** data**,** byte**[]** sign**)** throws PkiException

说明：

\* 验证不带原文SignedData签名,一步完成验证

\* data,签名原文

\* sign，SignedData编码

\* 验证成功返回true，失败返回false.

\* 错误抛出PkiException

### 一步完成验证

原型：

public byte[] signedDataVerifyAll**(**byte**[]** data**,** byte**[]** sign**)** throws PkiException

说明：

\* 自动判断signeddata是否带原文，验证SignedData签名,一步完成验证

\* data,签名原文 带原文是可以为空。

\* sign，SignedData编码

\* 验证成功返回true，失败返回false.

\* 错误抛出PkiException

### 判断是否为不带原文的SignedData

原型：

public boolean isDetachedSignedData**(**byte**[]** signData**)** 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

\* signData,SignedData编码

\*是，返回true

\* 错误抛出PkiException

### 判断是否为SignedData

原型：

public boolean isSignedData**(**byte**[]** signData**)** 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

\* signData,SignedData编码

\*是，返回true

\* 失败抛出PkiException

### 不带原文的多步验证初始化

原型：

public void detachedVerifyInit**(**byte**[]** signedData**)** throws PkiException

说明：

\* 不带原文的多步验证初始化

\* signedData，数字信封的DER编码

\* 失败抛出PkiException

### 不带原文的多步验证update

原型：

public void detachedVerifyUpdate**(**byte**[]** data**,** int offset**,** int len**)** throws PkiException

说明：

\* 不带原文的多步验证update

\* data，数字信封的DER编码，

\* offset,移位长度

\* len,SignedData编码中用于验证的数据长度

\* 失败抛出PkiException

### 不带原文的多步验证结束

原型：

public void detachedVerifyFinal**()** throws PkiException

说明：

\* 不带原文的多步验证结束

\* 失败抛出PkiException

### 带原文的多步验证初始化

原型：

public void verifyInit**()** throws PkiException

说明：

\*开始多步验证带原文的签名

\*成功返回true

\* 失败抛出PkiException

### 带原文的多步验证update

原型：

public byte**[]** verifyUpdate**(**byte**[]** signature**,** int offset**,** int len**)** throws PkiException

说明：

\* 多步验证带签名原文的签名,可以多次调用

\* signature,SignedData编码

\* offset，偏移量

\* len,SignedData编码中用于验证的数据长度

\* 成功返回签名原文部分数据

\* 失败抛出PkiException

### 带原文的多步验证结束

原型：

public boolean verifyFinal**()** throws PkiException

说明：

\*结束带原文的签名验证

\*成功返回true

\*失败抛出PkiException

### 验证后获取签名证书

原型：

public List<Cert> getSignCert**()** throws PkiException

说明：

\* 获取签名证书

\* 返回签名的证书编码

\* 失败抛出PkiException

### 释放资源

原型：

public void free**()**

说明：

\*释放资源

# 调用例程

详细的例程见NetcaAndroidSDKTestTool工程中的例子，下面简单的描述工程中的各个类的功能。

# 常见问题

1. 问：调用接口是应用没有弹出，证书选择窗口，或者PIN码输入窗口。

答：网证通安全客户端需要系统悬浮窗权限才能弹出窗口（部分小米手机默认禁止系统悬浮窗权限）需要手动开启

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，解密等耗时操作最好在线程中操作