**文件修改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **SDK版本** | **修改者** | **主要修改内容说明** |
| 20181228 | V1.0 | 黄志明 | 1、首个版本 |
| 20190218 | V1.0.1 | 黄志明 | 1. 接入海泰蓝牙设备 |
| 20200108 | V1.0.6 | 黄志明 | 1. 修复iOS13的bug |
| 20200414 | V1.1.0 | 黄志明 | 1. 由.a改为.framework 2. 接入SDK版移动Key 3. 签章读写 4. 增加设备配置信息设置接口 |
| 20200512 | V1.1.5 | 黄志明 | 1. 增加创建设备接口 2. 增加产生证书请求接口 3. 增加安装证书接口 4. 增加口令管理系统相关接口 |
| 20200605 | V1.1.5 | 黄志明 | 1. 增加移动电子密钥接口 2. 增加PIN缓存接口 3. 增加PIN弹窗样式接口 4. 增加云设备接口 |
| 20201021 | V1.1.6 | 黄志明 | 1. 增加云设备接口（分步） 2. 增加设置机构证书印章类型接口 3. 增加删除证书、删除密钥对接口 4. 增加NetcaKeyPair 5. 数字信封解密改成不需要设置解密证书 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

NETCA PKI中间件SDK接口说明

# 1 引言

## 概述

本文档主要介绍了NETCA PKI中间件SDK（iOS版）的集成步骤、接口调用方式和常见问题答疑等；本文适用于业务方集成中间件SDK的iOS客户端开发人员以及项目对接中其他相关人员等。

## 开发平台及编程语言

开发平台：Xcode9.2

编程语言：Objective-C

## 运行环境

SDK支持iOS9.0及以上，只提供真机版本

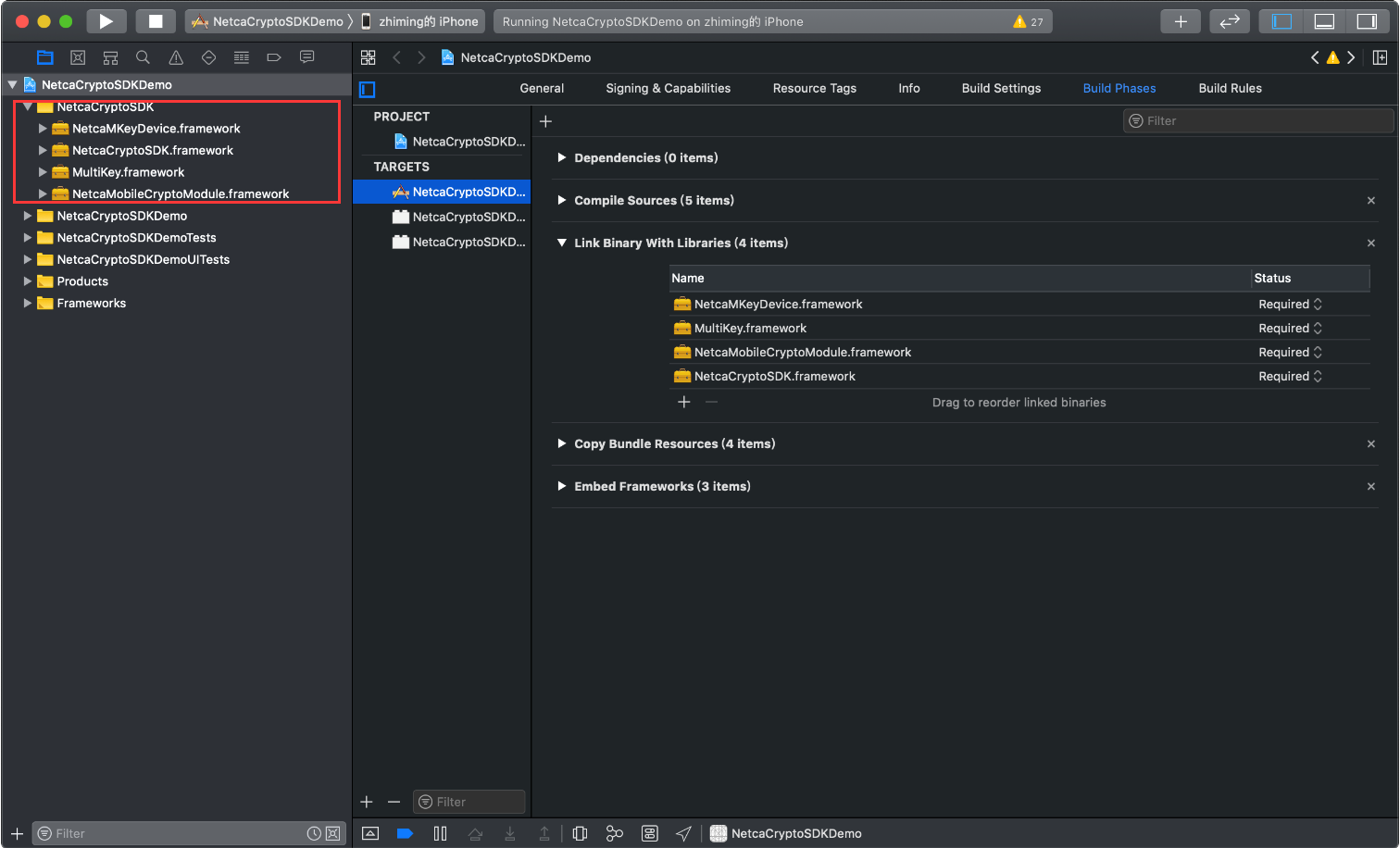
## 主要功能

1. 证书自助服务
2. 读取设备、数字证书
3. 数字签名验签
4. SignedData生成和验证
5. 非对称加解密
6. 数字信封加解密
7. 二维码签名
8. PDF签名
9. 读写签章
10. 产生证书请求、安装证书
11. 重置口令

# 集成SDK

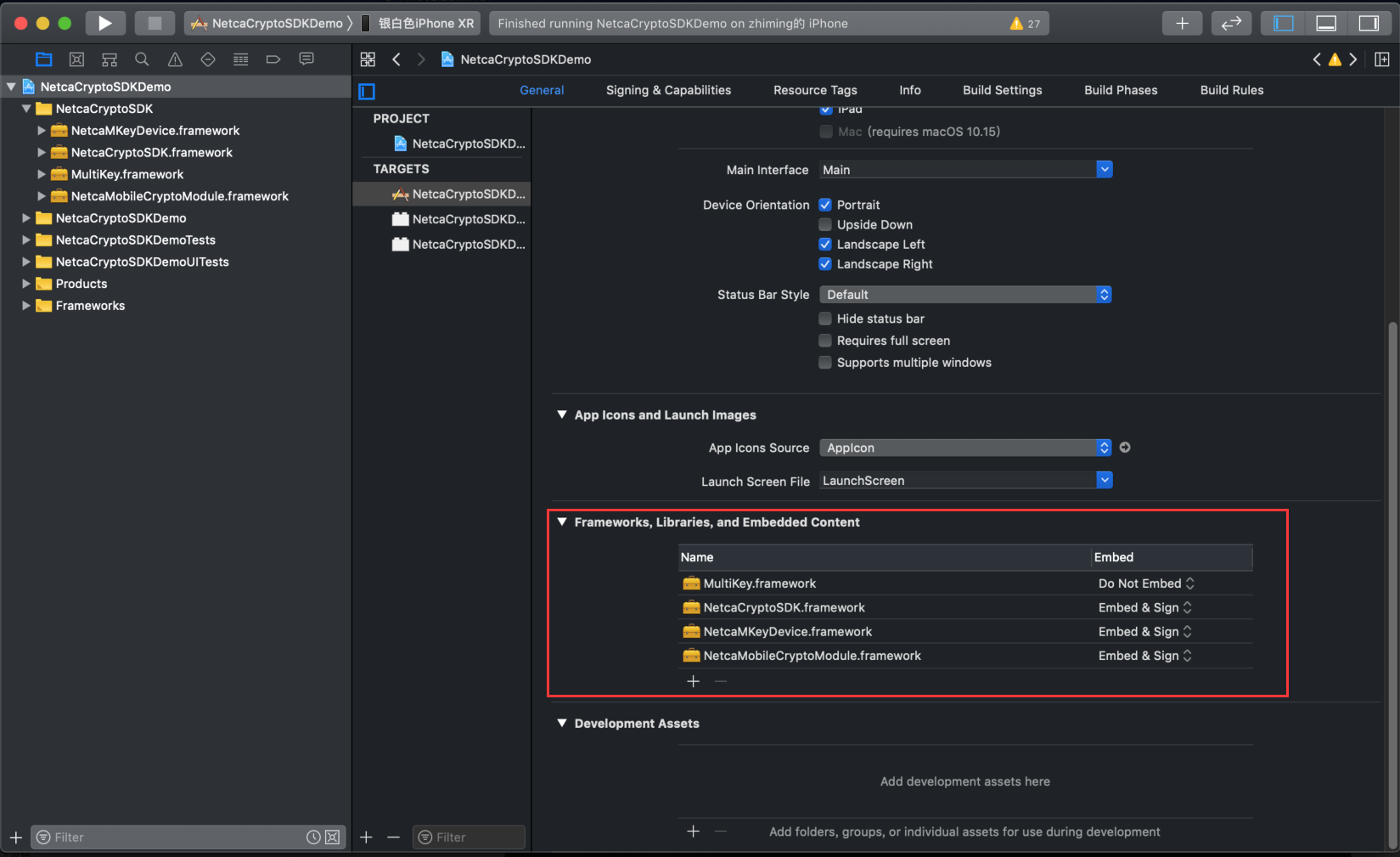
## 导入SDK

包括几个framework，如下图：

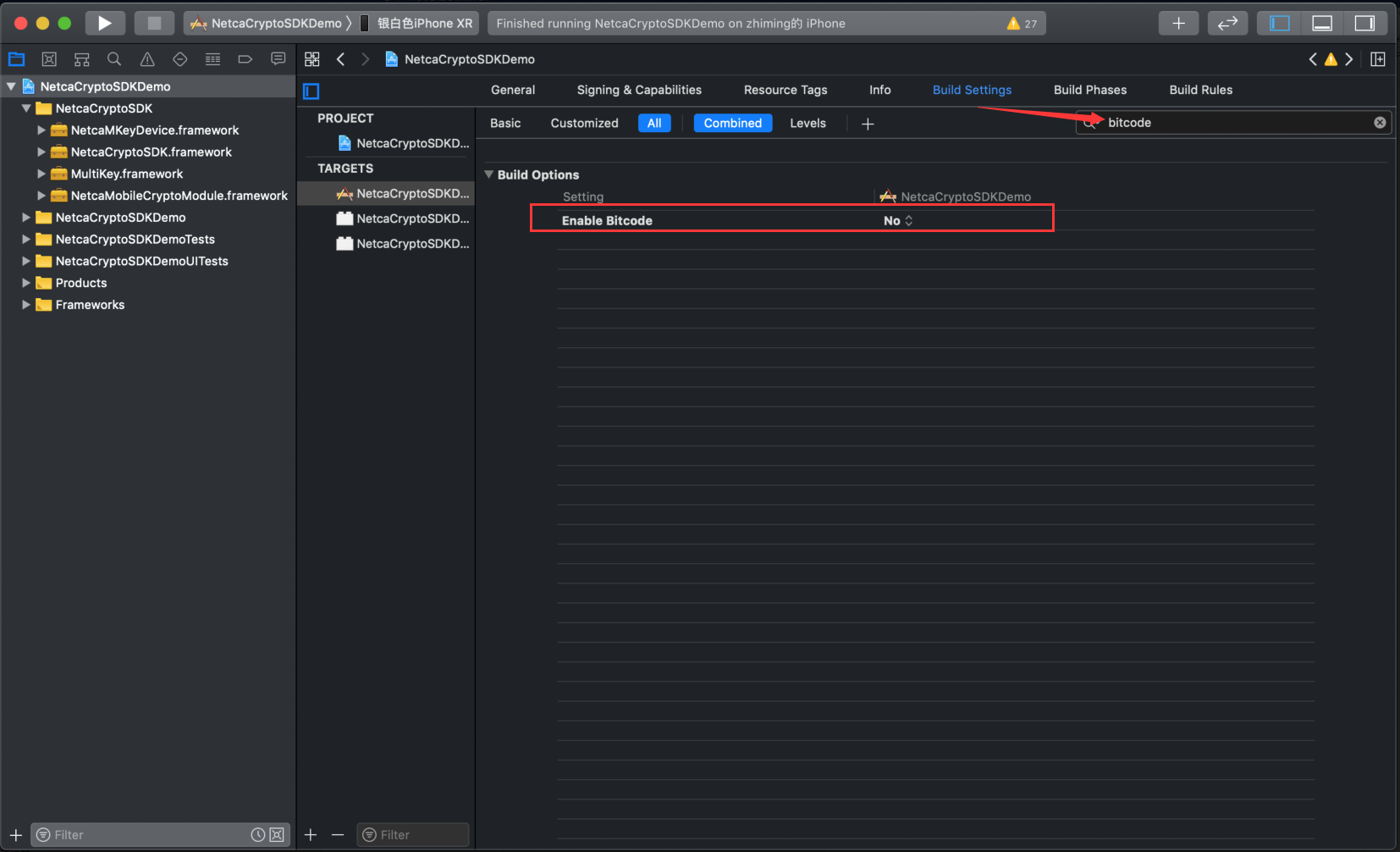


## 工程配置

1. 设置Embed Framework，如下图：

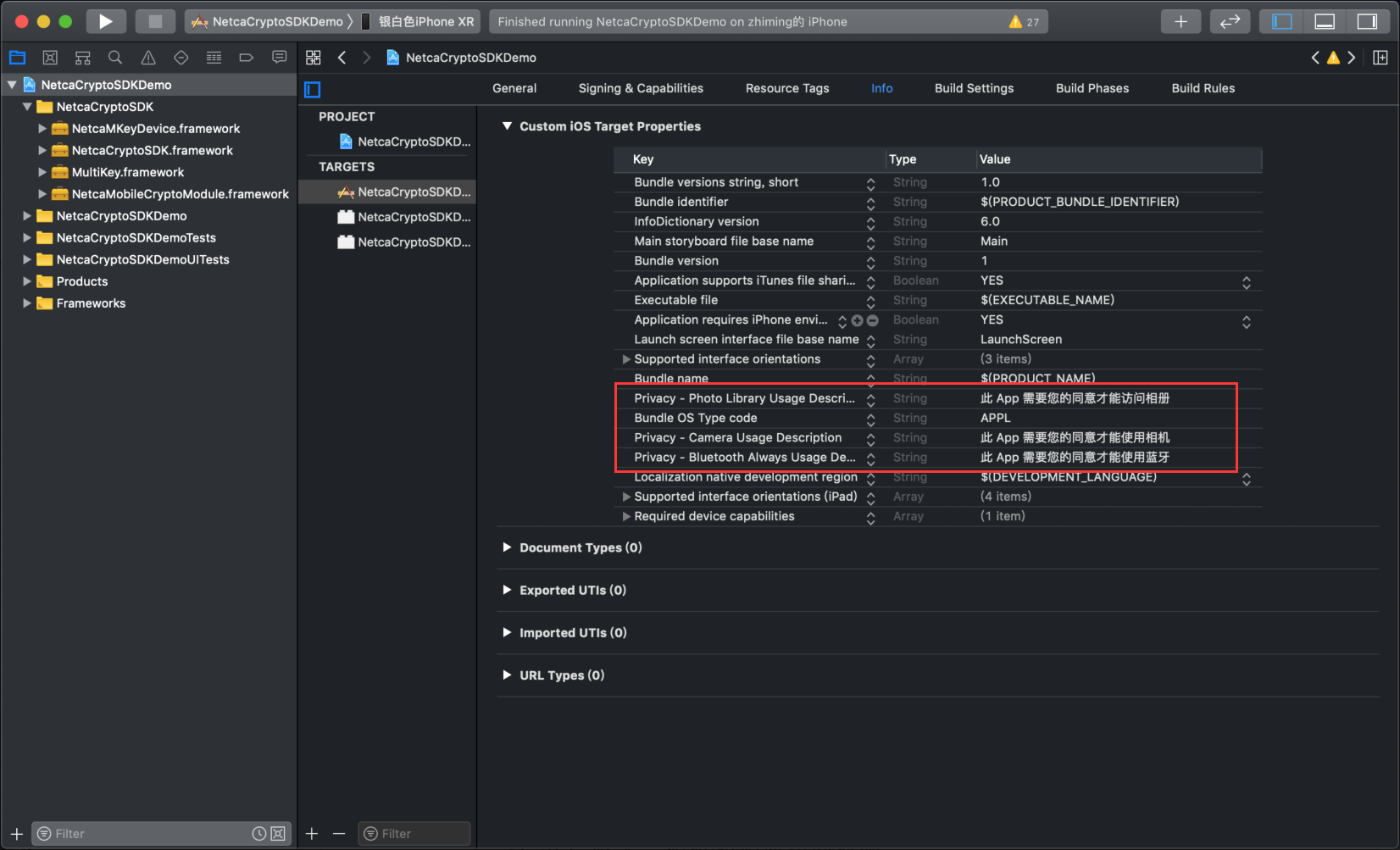


1. Enable Bitcode设置为No：



1. 开启蓝牙、相机和相册访问权限：

（注：若不使用相关功能，可以不需要。比如，使用了蓝牙设备才需要蓝牙权限）



# 对象说明

## NetcaDevice

### 属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 说明 |
| label | NSString | 设备的标签 |
| sn | NSString | 设备的序列号 |
| type | int | 设备的类型 |
| keypairs | NSArray | 数组，每一项是一个对象，表示一个密钥对 |

### 读取设备中的证书

原型：- (nullable NSArray<NetcaCertificate \*> \*)getCerts;

参数：无

返回：返回设备中的所有证书

### 验证设备PIN码

原型：- (BOOL)verifyPwdOfType:(nullable NSNumber \*)type pwd:(NSString \*)pwd pRetryNum:(nullable int \*)pRetryNum;

参数：

type: 类型

pwd: PIN码

pRetryNum: 重试次数

返回：成功返回true，失败返回false

### 修改设备PIN码

原型：- (BOOL)changePwdOfType:(nullable NSNumber \*)type oldpwd:(NSString \*)oldPwd newPwd:(NSString \*)newPwd pRetryNum:(nullable int \*)pRetryNum;

参数：

type: 类型

oldpwd: 旧的PIN码

newpwd: 新的PIN码

pRetryNum: 重试次数

返回：成功返回true，失败返回false

### 获取设备PIN码重试次数

原型：- (int)pwdRetryNumberOfType:(nullable NSNumber \*)type;

参数：

type: 类型

oldpwd: 旧的PIN码

返回：重试次数

### 解锁设备PIN码

原型：- (BOOL)unlockDeviceOfType:(nullable NSNumber \*)type sopwd:(NSString \*)sopwd newPwd:(NSString \*)newPwd pRetryNum:(nullable int \*)pRetryNum;

参数：

type: 类型

sopwd: 管理员PIN码

newpwd: 新的PIN码

pRetryNum: 重试次数

返回：成功返回true，失败返回false

### 创建持久化数据

原型：- (BOOL)createPersistentData:(int)dataId mode:(NSString \*)mode length:(int)length;

参数：

dataId:

mode: 管理员PIN码

length: 新的PIN码

返回：成功返回true，失败返回false

### 删除持久化数据

原型：- (BOOL)deletePersistentData:(int)dataId;

参数：

dataId:

返回：成功返回true，失败返回false

### 获取持久化数据长度

原型：- (int)getPersistentDataLength:(int)dataId;

参数：

dataId:

返回：成功返回true，失败返回false

### 写入持久化数据

原型：- (BOOL)writePersistentData:(int)dataId withOffset:(int)offset dataHex:(NSString \*)dataHex;

参数：

dataId:

offset: 管理员PIN码

dataHex: 新的PIN码

返回：成功返回true，失败返回false

### 读取持久化数据

原型：- (nullable NSData \*)readPersistentData:(int)dataId withOffset:(int)offset length:(int)length;

参数：

dataId:

offset: 管理员PIN码

length: 新的PIN码

返回：成功返回true，失败返回false

### 产生证书请求

原型：- (NSString \*)genP10:(NSString \*)pwd

dn:(NSString \*)dn

algo:(NetcaSignAlgorithm)algo

keyBits:(int)keyBits

keyType:(NetcaKeyPairType)keyType;

参数：

pwd: PIN码

dn: 证书DN

algo: 算法，参见常量定义文件NetcaAlgorithm.h

keyBits: 密钥位数

keyType: 密钥对类型，参见常量定义文件NetcaType.h

返回：成功证书请求，失败返回空

### 安装证书

原型：- (BOOL)installCert: (NSString \*)pwd

platform:(int)platform

signCert:(NSString \*)signCert

encCert:(NSString \*)encCert

encKeyPair:(NSString \*)encKeyPair;

参数：

pwd: PIN码

platform: 平台

signCert: 签名证书的base64编码

encCert: 签名证书的base64编码

encKeyPair: 加密的加密密钥对的base64编码

返回：成功返回true

### 判断设备是否支持签章读写

原型：- (BOOL)isSupportSeal;

参数：无

返回：支持返回true

### 获取设备中的签章个数

原型：- (int)getSealCount;

参数：无

返回：签章个数

### 读章

原型：- (nullable NSData \*)readSeal:(int)count;

参数：

count: 第几个，从0开始

返回：签章图片数据

### 写章

原型：- (BOOL)writeSeal:(NSString \*)sealName picture:(NSData \*)picture;

参数：

sealName: 签章名称

picture: 签章图片数据

返回：成功返回true

说明：调用前需要先验证PIN码

### 根据名称删除签章

原型：- (BOOL)deleteSealByName:(NSString \*)sealName;

参数：

sealName: 签章名称

返回：成功返回true

说明：调用前需要先验证PIN码

### 删除全部签章

原型：- (BOOL)deleteAllSeal;

参数：无

返回：成功返回true

说明：调用前需要先验证PIN码

### 删除密钥对

原型：- (BOOL)deleteKeypair:(NSString \*)pin label:(NSString \*)label;

参数：

pin: 用户PIN

label: 密钥对label

返回：成功返回true

### 删除证书

原型：- (BOOL)deleteCert:(NSString \*)pin cert:(NetcaCertificate \*)cert;

参数：

pin: 用户PIN

cert: 证书

返回：成功返回true

### 获取证书对应的密钥对label

原型：- (nullable NSString \*)getKeyPairLabelBy:(NetcaCertificate \*)target;

参数：

cert: 证书

返回：成功返回密钥对label

## NetcaCertificate

### 属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 说明 |
| derEncode | NSData | 证书DER编码 |
| version | NSInteger | 证书版本号 |
| signAlgo | NSInteger | 签名算法 |
| pubKeyAlgo | NSInteger | 公钥算法 |
| pubKeyBits | NSInteger | 公钥的位数 |
| keyusage | NSInteger | 密钥用法 |
| sn | NSString | 序列号 |
| valid | BOOL | 是否在有效期内 |
| issuer | NSString | 颁发者 |
| issuerC | NSString | 颁发者的C值 |
| issuerO | NSString | 颁发者的O值 |
| issuerOU | NSString | 颁发者的OU值 |
| issuerCN | NSString | 颁发者的CN值 |
| issuerST | NSString | 颁发者的ST值 |
| issuerL | NSString | 颁发者的L值 |
| issuerEmail | NSString | 颁发者的Email值 |
| subject | NSString | 主体 |
| subjectDisplayName | NSString | 主体的显示名 |
| subjectC | NSString | 主体的C值 |
| subjectST | NSString | 主体的ST值 |
| subjectO | NSString | 主体的O值 |
| subjectL | NSString | 主体的L值 |
| subjectOU | NSString | 主体的OU值 |
| subjectCN | NSString | 主体的CN值 |
| subjectEmail | NSString | 主体的Email值 |
| validityStart | NSString | 证书的有效期开始时间 |
| validityEnd | NSString | 证书的有效期结束时间 |
| thumbprintMD5 | NSString | 证书的微缩图，MD5加密 |
| thumbprintSHA1 | NSString | 证书的微缩图，SHA1加密 |
| thumbprintSHA256 | NSString | 证书的微缩图，SHA256加密 |
| thumbprintSHA512 | NSString | 证书的微缩图，SHA512加密 |
| crlURLs | NSArray | 证书CRL发布点扩展里的URL |
| ocspURLs | NSArray | 证书AIA扩展里的OCSP URL |

### 初始化

原型：- (instancetype)initWithDer:(NSData \*)der;

参数：

der: 证书的DER编码

返回：NetcaCertificate实例

### 获取证书对应的证书链

原型：- (nullable NSArray<NetcaCertificate \*> \*)buildCertPath;

参数：无

返回：证书链

### 判断是否是SM2证书

原型：- (bool)isSM2;

参数：无

返回：是返回true，否返回false

### 判断是否是签名证书

原型：- (bool)isSignCert;

参数：无

返回：是返回true，否返回false

### 判断是否是加密证书

原型：- (bool) isEncCert;

参数：无

返回：是返回true，否返回false

## NetcaBluetoothDevice

### 属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 说明 |
| name | NSString | 蓝牙设备名称 |
| mac | NSString | 蓝牙设备MAC地址 |

## NetcaBluetoothManager

说明：这是封装好的蓝牙设备的管理类，调用方通过实现NetcaBluetoothManagerDelegate的代理方法，即可获取蓝牙的各种事件回调（详情见3.5）。

### 属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 说明 |
| delegate | id | 委托者 |

### 开始扫描

原型：- (void)startScan;

参数：无

返回：无

### 结束扫描

原型：- (void) stopScan;

参数：无

返回：无

### 连接蓝牙设备

原型：- (void)connectDevice:(NetcaBluetoothDevice \*)device;

参数：

device: 蓝牙设备

返回：无

### 断开蓝牙设备

原型：- (void) disconnectDevice:(NetcaBluetoothDevice \*)device;

参数：

device: 蓝牙设备

返回：无

## NetcaBluetoothManagerDelegate

### 蓝牙状态更新的回调

原型：- (void)bluetoothManagerDidUpdateState:(CBCentralManager \*)central;

参数：

central: 蓝牙设备

返回：无

说明：必选

### 发现蓝牙设备的回调

原型：- (void)bluetoothManagerDidDiscoverDevice:(NetcaBluetoothDevice \*)device;

参数：

device: 蓝牙设备

返回：无

说明：可选

### 连接蓝牙设备成功的回调

原型：- (void) bluetoothManagerDidConnectDevice:(NetcaBluetoothDevice \*)device;

参数：

device: 蓝牙设备

返回：无

说明：可选

### 连接蓝牙设备失败的回调

原型：- (void) bluetoothManagerDidFailToConnectDevice:(NetcaBluetoothDevice \*)device error:(nullable NSError \*)error;

参数：

device: 蓝牙设备

error: 错误

返回：无

说明：可选

### 蓝牙设备断开的回调

原型：- (void) bluetoothManagerDidDisconnectDevice:(NetcaBluetoothDevice \*)device error:(nullable NSError \*)error;

参数：

device: 蓝牙设备

error: 错误

返回：无

说明：可选

## NetcaQRCodeScannerController

说明：这是封装好的二维码扫描控制器，调用方push该控制器即可进入二维码扫描页面, 通过实现NetcaQRCodeScannerDelegate的代理方法，即可获取到二维码内容（详情见3.7）。

## NetcaQRCodeScannerDelegate

### 扫描后的回调

原型：-(void)getQrCodeContent:(NSString \*)content;

参数：

content: 二维码内容

返回：无

说明：必选

## NetcaMobileKeyDelegate

### 连接成功的回调

原型：- (void)netcaMobileKeyDidConnectPC;

参数：无

返回：无

说明：必选

### 连接失败的回调

原型：- (void)netcaMobileKeyDidFailToConnectPC:(nonnullNSError \*)error;

参数：

error: 错误

返回：无

说明：可选

### 连接失败的回调

原型：- (void)netcaMobileKeyDidDisconnectPC:(nullable NSError \*)error;

参数：

error: 错误

返回：无

说明：可选

## NetcaSelfServiceViewController

说明：这是封装好的自助服务（证书新申请）控制器，调用方push该控制器即可进入证书新申请页面，实现证书的在线申请和安装。

### 属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 说明 |
| qrcode | NSString | 具体的业务二维码 |
| netca\_ss\_networkurl | NSString | 自助服务地址，若不传，则使用生产环境的地址 |

## NetcaPDFArea

### 属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 说明 |
| name | NSString | 签名域名称 |
| pageIndex | int | 页码 |
| left | CGFloat | 签名域左上角的x轴坐标 |
| top | CGFloat | 签名域左上角的y轴坐标 |
| width | CGFloat | 签名域的宽 |
| height | CGFloat | 签名域的高 |

## NetcaKeyPair

### 属性

|  |  |  |
| --- | --- | --- |
| 属性 | 类型 | 说明 |
| label | NSString | 密钥对的标签 |
| type | int | 密钥对的类型 |
| algo | int | 密钥对的算法 |
| curve | int | 密钥对的ECC曲线号 |

# 接口说明

## NetcaCryptoSDK

### 单例

原型：- (instancetype)sharedInstance;

参数：无

返回：NetcaCryptoInterface实例。

### 获取错误码

原型：+ (long long)getLastErrorCode;

参数：无

返回：返回最后的错误码

### 获取设备接口实例

原型：- (NetcaDeviceInterface \*)getDeviceInterface;

参数：无

返回：返回设备接口实例

### 获取证书接口实例

原型：- (NetcaCertInterface \*)getCertInterface;

参数：无

返回：返回证书接口实例

## NetcaDeviceInterface

### 获取设备列表

原型：- (NSArray<NetcaDevice \*> \*)getDeviceList;

参数：无

返回：返回设备列表

### 根据设备类型获取设备

原型：- (nullable NSArray<NetcaDevice \*> \*)findDeviceByType:(int)deviceType;

参数：

deviceType: 设备类型

返回：对应设备或nil

说明：可能有多个，比如软证书

### 设置设备的配置信息

原型：- (NetcaError)setDeviceConfig:(NetcaDeviceType)type jsonStr:(NSString \*)jsonStr;

参数：

type: 设备类型，参见常量定义文件NetcaType.h

jsonStr: 配置信息的json串

返回：错误码

说明：用于设置移动key的服务器地址等。

### 创建指定类型的设备

原型：- (NetcaError)createDevice:(NetcaDeviceType)type pwd:(NSString \*)pwd;

参数：

type: 设备类型，参见常量定义文件NetcaType.h

pwd: PIN码

返回：错误码

说明：部分设备需要先调用setDeviceConfig()去设置一些必要的信息，如移动Key。

### 设置口令管理系统地址

原型：- (NetcaError)setPinManagerSystemAddr:(NSString \*\_Nonnull)addr;

参数：

addr: 服务器地址，如"http://127.0.0.1"

返回：错误码

### 产生随机数

原型：- (NSData \*)generateRandom:(int)len;

参数：

len: 要获取的随机数的长度

返回：成功返回随机数

### 插入云设备

原型：- (void)mountCloudDevice:(NetcaCertificate \*\_Nonnull)signCert

pin:(NSString \*\_Nonnull)pin

jsonParam:(NSString \*\_Nonnull)jsonParam

completion:(void (^\_Nonnull)(NSError \*\_Nullable error, NSString \*\_ Nullable jsonResult))completion;

参数：

signCert: 员工签名证书

pin: 用户密码

jsonParam: json字符串参数，包含云设备类型、云设备服务器信息等

completion: 结果回调，成功返回设备类型、序列号和机构证书

返回：无

### 移除云设备

原型：- (void)unMountCloudDevice:(NSString \*\_Nullable)jsonParam

completion:(void (^\_Nonnull)(NSError \*\_Nullable error))completion;

参数：

jsonParam: json字符串参数，包含云设备类型、云设备服务器信息等

completion: 结果回调

返回：无

### 插入云设备（未登录）

原型：- (void)mountCloudDevice:(NetcaCertificate \*\_Nonnull)signCert

jsonParam:(NSString \*\_Nonnull)jsonParam

completion:(void (^\_Nonnull)(NSError \*\_Nullable error, NSString \*\_ Nullable jsonResult))completion;

参数：

signCert: 员工签名证书

jsonParam: json字符串参数，包含云设备类型、云设备服务器信息等

completion: 结果回调，成功返回设备类型、序列号和机构证书

返回：无

### 登录云设备

原型：- (void) loginCloudDevice:(NetcaCertificate \*\_Nonnull)signCert

pin:(NSString \*\_Nonnull)pin

completion:(void (^\_Nonnull)(NSError \*\_Nullable error))completion;

参数：

signCert: 员工签名证书

pin: 用户密码

completion: 结果回调

返回：无

### 设置云设备使用的印章类型

原型：- (NetcaError)setCloudDeviceSealType:(NetcaCloudDeviceSealTyp)type;

参数：

type: 印章类型

返回：错误码

## NetcaCertInterface

### 获取有效期内的签名证书列表

原型：- (NSArray<NetcaCertificate \*> \*)getSignCertList;

参数：无

返回：返回签名证书列表

### 获取有效期内的加密证书列表

原型：- (NSArray<NetcaCertificate \*> \*)getEncCertList;

参数：无

返回：返回加密证书列表

### 通过OCSP验证证书

原型：- (BOOL)verifyCertByOcsp:(NetcaCertificate \*)cert ocspData:(nonnull NSData \*)ocspData;

参数：

cert: 证书对象

ocspData: ocspData数据

返回：证书未吊销返回成功，其他返回失败

### 通过CRL验证证书

原型：- (BOOL)verifyCertByCrl:(NetcaCertificate \*)cert crlData:(nonnull NSData \*)crlData;

参数：

cert: 证书对象

crlData: crl数据

返回：证书未吊销返回成功，其他返回失败

## NetcaHashInterface

### 一步完成Hash运算

原型：- (nullable NSData \*)hash:(int)hashAlgo data:(nonnull NSData \*)data;

参数：

hashAlgo: 哈希算法

data: 要哈希的数据

返回：哈希值

### 多步计算Hash的初始化

原型：- (BOOL)hashInit:(int)hashAlgo;

参数：

hashAlgo: 哈希算法

返回：成功返回true

### 多次输入原文进行Hash运算

原型：- (BOOL)hashUpdate:(nonnull NSData \*)data;

参数：

hashAlgo: 哈希算法

data: 要哈希的数据

返回：成功返回true

### 结束Hash运算

原型：- (nullable NSData \*)hashFinal;

参数：无

返回：哈希值

## NetcaSignatureInterface

### 设置签名证书

原型：- (void)setSignCert:(Certificate \*)cert;

参数：

cert: 签名证书

返回：无

### 设置签名算法

原型：- (void)setSignAlgo:(int)algo;

参数：

algo: 签名算法

返回：无

### 一步完成签名

原型：- (NSData \*)sign:(NSString \*)pwd data:(NSData \*)data;

参数：

pwd: 密码

data：签名原文

返回：签名值

说明：用于签名原文比较小的情况

### 一步完成hash签名

原型：- (NSData \*)signHash:(NSString \*)pwd hash:(NSData \*)hash;

参数：

pwd: 密码

data：需要签名的hash值

返回：签名值

说明：用于对hash进行签名

### 多步完成签名初始化

原型：- (void)signInit:(NSString \*)pwd;

参数：

pwd: 密码

返回：无

说明：多步签名初始化

### 多步完成签名update

原型：- (void)signUpdate:(NSData \*)data;

参数：

data: 签名原文

返回：无

说明：多次输入原文进行签名

### 多步完成签名结束

原型：- (NSData \*)signFinal;

参数：无

返回：签名值

说明：结束多步签名

### 计算SM2签名需要的Z值

原型：- (nullable NSData \*)computeSM2Z:(NSData \*)ID;

参数：

ID: 用户传入的id

返回：Z值

## NetcaSignatureVerifyInterface

### 设置签名证书

原型：- (bool)setSignCertificate:( NetcaCertificate \*)cert;

参数：

cert: 签名证书

返回：成功返回true

### 设置签名算法

原型：- (bool)setSignAlgorithm:(int)algo;

参数：

algo: 签名算法

返回：成功返回true

### 一步验证签名

原型：- (bool)verify:(NSData \*)data signature:(NSData \*)signature;

参数：

data: 签名原文

signature: 签名值

返回：验证成功返回true，失败返回false

### 一步验证hash签名

原型：- (bool)verifyHash:(NSData \*)hash signature:(NSData \*)signature;

参数：

hash: 要签名的hash

signature: 签名值

返回：验证成功返回true，失败返回false

### 多步验证签名初始化

原型：- (bool)verifyInit;

参数：无

返回：成功返回true

说明：初始化多步签名验证

### 多步验证签名update

原型：- (bool)verifyUpdate:(NSData \*)data;

参数：

data: 签名原文

返回：成功返回true

说明：多次输入签名原文进行hash运算

### 多步验证签名结束

原型：- (bool)verifyFinal:(NSData \*)signature;

参数：

signature: 签名值

返回：成功返回true

说明：结束多步签名验证

## NetcaSignedDataSignInterface

### 设置是否带原文

原型：- (bool)setDetached:(bool)value;

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：- (void)setSignCert:(Certificate \*)cert;

参数：

cert: 签名证书

返回：无

### 设置签名算法

原型：- (void)setSignAlgo:(int)index algo:(int)algo;

参数：

Index：签名索引值，从0开始

algo: 签名算法

返回：无

### 添加CRL

原型：- (BOOL)addCrl:(NSData \*)crl;

参数：

crl: crl数据

返回：成功返回true

### 添加OCSP

原型：- (BOOL)addOCSP:(NSData \*)ocsp cert:(nullable NetcaCertificate \*)cert;

参数：

ocsp：ocsp数据

cert: ocsp证书，可以为nil

返回：成功返回true

### 一步完成SignedData签名

原型：- (NSData \*)sign:(NSString \*)pwd data:(NSData \*)data;

参数：

pwd: 密码

data: 签名原文

返回：成功返回签名值

说明：用于签名原文比较小的情况

### 一步完成SignedData签名

原型：- (nullable NSData \*)sign:(NSString \*)pwd data:(NSData \*)data tsaUrl:(nullable NSString \*)tsaurl;

参数：

pwd: 密码

data: 签名原文

tsaurl: 时间戳地址

返回：成功返回签名值

说明：用于签名原文比较小，并且带时间戳的情况

### 多步完成SignedData签名初始化

原型：- (NSData \*)signInit:(NSString \*)pwd;

参数：

pwd: 密码

返回：成功返回部分签名值

说明：多步签名的初始化

说明：多步签名的签名结果为signInit, 多次(1..n)signUpdate, signFinal三者返回值的拼接

### 多步完成SignedData签名update

原型：- (NSData \*)signUpdate:(NSData \*)data;

参数：

data: 签名原文

返回：成功返回部分签名值

说明：多次输入原文进行签名

### 多步完成SignedData签名结束

原型：- (NSData \*)signFinal;

参数：无

返回：成功返回部分签名值

说明：结束多步签名

### 多步完成SignedData签名结束，带时间戳

原型：- (nullable NSData \*)signFinalWithTsaUrl:(NSString \*)tsaurl;

参数：

tsaurl: 时间戳地址

返回：成功返回部分签名值

说明：结束多步签名

## NetcaSignedDataVerifyInterface

### 一步验证带原文的SignedData签名

原型：- (bool)verify:(NSData \*)signature;

参数：

signature: 签名值

返回：验证成功返回true，失败返回false

说明：一步验证带原文的SignedData签名

### 一步验证不带原文的SignedData签名

原型：- (bool)verify:(NSData \*)data signature:(NSData \*)signature;

参数：

data: 签名原文

signature: 签名值

返回：验证成功返回true，失败返回false

说明：一步验证不带原文的SignedData签名

### 获取签名证书

原型：- (NetcaCertificate \*)getSignCert;

参数：无

返回：返回签名证书

### 多步验证带原文的SignedData签名初始化

原型：- (bool)verifyInit;

参数：无

返回：成功返回true

说明：初始化带原文的签名验证

### 多步验证带原文的SignedData签名update

原型：- (NSData \*)verifyUpdate:(NSData \*)signature;

参数：

signature: 签名值

返回：返回签名原文

说明：多次输入SignedData的编码值进行带原文的签名验证

### 多步验证带原文的SignedData签名结束

原型：- (bool)verifyFinal;

参数：无

返回：成功返回true

说明：结束带原文的签名验证

### 多步验证不带原文的SignedData签名初始化

原型：- (bool)detachedVerifyInit:(NSData \*)signature;

参数：

signature: 签名值

返回：成功返回true

说明：初始化不带原文的签名验证

### 多步验证不带原文的SignedData签名update

原型：- (bool)detachedVerifyUpdate:(NSData \*)data;

参数：

data: 签名原文

返回：成功返回true

说明：多次输入不带原文的签名对应的原文

### 多步验证不带原文的SignedData签名结束

原型：- (bool)detachedVerifyFinal;

参数：无

返回：成功返回true

说明：结束不带原文的签名验证

## NetcaCipherInterface

### 设置IV值

原型：- (void)setIv:(NSData \*)iv;

参数：

iv: iv值

返回：无

### 设置key

原型：- (void)setKey:(NSData \*)key;

参数：

key: key值

返回：无

### 设置algo

原型：- (void)setAlgo:(int)algo;

参数：

algo: 对称加解密算法

返回：无

### 设置aad

原型：- (void)setAad:(NSString \*)aad;

参数：

algo: 附加的认证数据

返回：无

### 设置padding

原型：- (void)setPadding:(int)padding;

参数：

algo:

返回：无

### 多步对称加解密初始化

原型：(BOOL)cipherInit:(BOOL)isEncrypt;

参数：

isEncrypt: true表示加密，false表示解密

返回：成功返回true

### 多步对称加解密update

原型：- (nullable NSData \*)cipherUpdate:(nonnull NSData \*)data;

参数：

data: 数据

返回：加密返回密文，解密返回原文

### 多步对称加解密结束

原型：- (nullable NSData \*)cipherFinal;

参数：无

返回：加密返回密文，解密返回原文

## NetcaEncryptInterface

### 公钥加密

原型：- (NSData \*)encrypt:(NSData \*)data usingCert:(NetcaCertificate \*)cert;

参数：

data: 原文

cert: 加密证书

返回：成功返回加密密文

## NetcaDecryptInterface

### 设置加密证书

原型：- (void)setEncCert:(Certificate \*)cert;

参数：

cert: 加密证书

返回：无

### 私钥解密

原型：- (nullable NSData \*)privateKeyDecrypt:(nonnull NSData \*)cipherData

pwd:(nonnull NSString \*)pwd

usingDecryptAlgo:(int)algo

isDecryptKey:(bool)isDecryptKey;

参数：

cipherData: 密文

pwd: PIN码

algo: 加密算法

isDecryptKey: 是否用于解密数字信封的对称密钥

cert: 加密证书

返回：明文，如果是大文件数字信封解密，则结果为对称密钥

## NetcaEnvelopedDataEncryptInterface

### 一步完成数字信封加密

原型：- (NSData \*)encrypt:(NSData \*)data certs:(NSArray<NetcaCertificate \*> \*)certs;

参数：

data: 原文

certs: 加密证书列表

返回：成功返回加密密文

### 多步完成数字信封加密初始化

原型：- (NSData \*)encryptInit :(NSArray<NetcaCertificate \*> \*)certs;;

参数：

certs: 加密证书列表

返回：成功返回部分密文

说明：开始构造数字信封

注意：多步实现数字信封加密，加密结果为encryptInit, 多次(1..n)encryptUpdate, encryptFinal三者返回值的拼接

### 多步完成数字信封加密update

原型：- (NSData \*)encryptUpdate:(NSData \*)data;

参数：

data: 原文

返回：成功返回部分原文

说明：多次输入原文，进行加密

### 多步完成数字信封加密结束

原型：- (NSData \*)encryptFinal;

参数：无

返回：成功返回部分密文

说明：结束构造数字信封

## NetcaEnvelopedDataDecryptInterface

### 一步解密数字信封

原型：- (NSData \*)decrypt:(NSString \*)pwd data:(NSData \*)data;

参数：

pwd: 密码

data: 数字信封编码

返回：成功返回密文

### 获取解密证书

原型：- (NetcaCertificate \*)getDecryptCert;

参数：无

返回：返回解密证书

### 多步解密数字信封初始化

原型：- (bool)decryptInit:(NSString \*)pwd;

参数：

pwd: 密码

返回：成功返回true

说明：开始多步解开数字信封

### 多步解密数字信封update

原型：- (NSData \*)decryptUpdate:(NSData \*)data;

参数：

data: 数字信封编码

返回：成功返回明文

说明：多次输入数字信封编码，解开数字信封

### 多步解密数字信封结束

原型：- (bool)decryptFinal;

参数：无

返回：成功返回true

说明：结束多步解开数字信封

## NetcaRevokeInfoUtil

### 根据证书获取颁发者证书的OCSP地址

原型：+ (nullable NSString \*)getIssuerOCSPUrl:(NetcaCertificate \*)cert;

参数：

cert: 证书

返回：颁发者证书的OCSP地址

### 根据证书获取颁发者证书的CRL地址

原型：+ (nullable NSString \*) getRootCRLUrl:(NetcaCertificate \*)cert;

参数：

cert: 证书

返回：根证书的CRL地址

## NetcaPDFSignInterface

### PDF签名

原型：+ (bool)signPDFFile:(nonnull NSString \*)pdfFilePath

imagePath:(nullable NSString \*)imagePath

signedPdfFilePath:(nonnull NSString \*)signedPdfFilePath

cert:(nonnull NetcaCertificate \*)cert

pin:(nonnull NSString \*)pin

crlData:(nullable NSData \*)crlData

ocspData:(nullable NSData \*)ocspData

tsaurl:(nullable NSString \*)tsaurl

area:(nonnull NetcaPDFArea \*)area;

参数：

pdfFilePath: 需要签名的pdf文档路径

imagePath: 签章图片的路径

signedPdfFilePath: 保存签名后的pdf文档路径

cert: 签名证书

pin: 电子密钥PIN码，若为NULL，则在WPKI中输入

crlData: 验证的crl数据（可为NULL）

ocspData: 验证的ocsp数据（可为NULL）

tsaurl: 时间戳服务器地址（可为NULL）

area: 签名域信息

返回：成功返回true

## NetcaQrCodeSignInterface

### 初始化

原型：- (void)setQrCodeContent:(NSString \*)qrCodeContent;

参数：

qrCodeContent: 二维码内容

返回：哈希值

### 判断是否需要验证码

原型：- (bool)isNeedAuthCode;

参数：无

返回：true需要，false不需要

### 获取签名内容。根据二维码内容获取要签名的数据

原型：- (NSString \*)querySignData:(NSString \*)authCode;

参数：

authCode: 验证码

返回：成功返回签名内容

### 二维码签名。签名并提交签名值，返回后台验证结果

原型：- (bool)qrcodeSign:(NetcaCertificate \*)cert pin:(NSString \*)pin;

参数：

cert: 签名证书

pin: PIN码

返回：成功返回true

## NetcaPinManageSystemInterface

### 设置要重置口令的设备

原型：- (void)setDevice:(NetcaDevice \*)device;

参数：

device: 设备

返回：无

### 注册管理员密钥

原型：- (void)registerSoPin:(NSString \*)phoneNum

userPin:(NSString \*)userPin

completion:(void (^)(NSError \*error))completion;

参数：

phoneNum: 手机号

userPin: 用户口令

completion: 结果回调

### 请求获取管理员密钥

原型：- (void)requestGetSoPin:(void (^)(NSError \*error, NSString \*dataToSign))completion;

参数：

completion: 结果回调。若成功，则返回的dataToSign是要签名的内容，第三方应用对该签名内容进行签名后，再调用重置用户口令接口。

### 重置用户口令

原型：- (void)unlockUserPin:(NSString \*)newUserPin

signature:(NSString \*)signature

completion:(void (^)(NSError \*error))completion;

参数：

newUserPin: 新的用户口令

signature: 签名值的base64编码

completion: 结果回调

## NetcaMobileKeyInterface

通过实现NetcaMobileKeyDelegate的代理方法，可以获取到连接成功、失败、断开的回调。（详情见3.8）。

### 设置是否需要密码缓存

原型：- (void)setNeedPINCache:(NetcaPINCacheType)type;

参数：

type: 样式，参考 NetcaPINCacheType

返回：无

### 设置密码输入框的样式

原型：- (void)setPINUIType:(NetcaPINUIType)type;

参数：

type: 样式，参考 NetcaPINUIType

返回：无

### 连接PC

原型：- (NetcaError)connectToPC:(NSString \*\_Nonnull)qrcode

sourceVC:(UIViewController \*\_Nullable)sourceVC

displayMode:(NetcaVCDisplayMode)mode

parameter:(NSString \*\_Nullable)parameter;

参数：

qrcode: PC端的二维码内容

sourceVC: 调起 UI 界面的 ViewController

mode: 调起UI的模式，参考 NetcaVCDisplayMode

parameter: 保留参数

返回：错误码

# 调用例程

以移动Key为例，简述下实现各功能的的接口调用流程。

（注：以下只是主要的代码片段，详细的请查照DEMO。）

## 创建设备

//1、创建NetcaCryptoInterface实例

NetcaCryptoSDK \*crypto = [NetcaCryptoSDK sharedInstance];

//2、配置设备信息

//配置信息是一个json串，包括服务器地址和appid。这是测试环境的地址，正式使用需要替换

NSString \*configInfo = @"{\"u\":\"http://120.25.164.100:7091\",\"id\":\"263277445690425344\"}";

NetcaError rv = [deviceInterface setDeviceConfig:NetcaDeviceTypeMKeyDevice jsonStr:configInfo];

if (rv != NetcaErrorCodeSuccess) {

NSLog(@"设置设备信息失败");

}

//3、创建设备

NetcaDeviceInterface \*deviceInterface = [\_crypto getDeviceInterface];

NetcaError rv = [deviceInterface createDevice:NetcaDeviceTypeMKeyDevice pwd:PIN];

if (rv != NetcaErrorCodeSuccess) {

NSLog(@"创建设备失败");

}

## 读取设备

NetcaDeviceInterface \*deviceInterface = [\_crypto getDeviceInterface];

NSArray<NetcaDevice \*> \*deviceList = [deviceInterface findDeviceByType:NetcaDeviceTypeMKeyDevice];

if (deviceList.count <= 0) {

NSLog(@"设备个数为0");

return;

}

NetcaDevice \*device = deviceList[0];

## 产生密钥对

//1、先获取设备device，参考5.2

//2、产生证书请求

// dn: 证书的各项信息

// algo: NetcaSignAlgorithmSM3WithSM2

// keyBits: 256

// keyType: NetcaKeyPairTypeSignature

NSString \*dn = @"CN=移动key测试,OU=部门,O=移动key测试,S=Guangdong,C=CN";

NSString \*p10 = [device genP10:PIN dn:dn algo:NetcaSignAlgorithmSM3WithSM2 keyBits:256 keyType:NetcaKeyPairTypeSignature];

if (!p10) {

NetcaError rv = [NetcaCryptoSDK getLastErrorCode];

NSLog(@"产生p10失败, 错误码:%lld, 错误信息:%@", rv, errMsgOfCode(rv));

}

//产生p10后，由调用者自行将p10发到CA后台进行制证，得到签名证书、加密证书和加密密钥对

## 安装证书

//1、先获取设备device，参考5.2

//2、安装证书

NSString \*signCert = @"";// 签名证书的base64编码

NSString \*encCert = @"";// 加密证书的base64编码

NSString \*encKeyPair = @"";// 加密密钥对的base64编码

BOOL rv = [device installCert:PIN platform:NetcaEncKeyPairTypeGBT\_35276 signCert:signCert encCert:encCert encKeyPair:encKeyPair];

if (!rv) {

NetcaError rv = [NetcaCryptoSDK getLastErrorCode];

NSLog(@"安装证书失败, 错误码:%lld, 错误信息:%@", rv, errMsgOfCode(rv));

}

## 读取证书

//1、先获取设备device，参考5.2

//2、读取设备中的证书

NSArray<NetcaCertificate \*> \*certList = [device getCerts];  
    if (certList.count <= 0) {  
        NSLog(@"证书个数为0");  
        return;  
    }  
    NetcaCertificate \*cert = certList[0];  
  
    // 判断是签名证书还是加密证书  
    if ([cert isSignCert]) {  
        NSLog(@"签名证书");  
    };  
    // 判断证书是否在有效期内  
    if (![cert isValid]) {  
        NSLog(@"证书已过期");  
    };  
    // 判断是RSA证书还是SM2证书  
    if ([cert isSM2]) {  
        NSLog(@"SM2证书");  
    };

## 非对称加解密

//1、先读取在有效期内的加密证书encCert，参考5.5

//2、非对称加密

NSData \*data = [@"hello world" dataUsingEncoding:NSUTF8StringEncoding];

NetcaEncryptInterface \*encryptInterface = [NetcaEncryptInterface new];

NSData \*cipher = [encryptInterface encrypt:data usingCert:encCert];

if (cipher) {

NSLog(@"非对称加密成功，密文的: %@", cipher);

}else {

NSLog(@"非对称加密失败");

}

//3、非对称解密

NetcaDecryptInterface \*decryptInterface = [[NetcaDecryptInterface alloc]init];

decryptInterface.encCert = encCert;

int algo = NetcaEncryptAlgorithm\_RSA\_PKCS1\_V1\_5\_ENC;

if (decryptInterface.encCert.isSM2) {

algo = NetcaEncryptAlgorithm\_SM2\_ENC;

}

NSData \*plain = [decryptInterface privateKeyDecrypt:cipher pwd:PIN usingDecryptAlgo:algo isDecryptKey:false];

if (plain) {

NSLog(@"非对称解密结果: %@", plain);

}else {

NSLog(@"非对称解密失败");

}

## P1签名、验证

//1、先读取在有效期内的签名证书signCert，参考5.5

//2、签名

int algo = signCert.isSM2 ? NetcaSignAlgorithmSM3WithSM2 : NetcaSignAlgorithmSHA256WithRSA;

NetcaSignatureInterface \*signatureInterface = [[NetcaSignatureInterface alloc]init];

signatureInterface.signCert = signCert;

signatureInterface.signAlgo = algo;

NSData \*data = [@"hello world" dataUsingEncoding:NSUTF8StringEncoding];

NSData \*p1Signature = [signatureInterface sign:PIN data:data];

NSString \*p1SignatureBase64String = [p1Signature base64EncodedStringWithOptions:0];

NSLog(@"p1签名成功，签名值的base64编码: %@", p1SignatureBase64String);

//3、验证

NetcaSignatureVerifyInterface \*verifyInterface = [[NetcaSignatureVerifyInterface alloc]init];

[verifyInterface setSignCertificate:signCert];

[verifyInterface setSignAlgorithm:algo];

BOOL ret = [verifyInterface verify:data signature:p1Signature];

if (!ret) {

NSLog(@"验签失败");

return;

}

## P7签名、验证

//1、先读取在有效期内的签名证书signCert，参考5.5

//2、签名

int algo = signCert.isSM2 ? NetcaSignAlgorithmSM3WithSM2 : NetcaSignAlgorithmSHA256WithRSA;

NSData \*data = [@"hello world" dataUsingEncoding:NSUTF8StringEncoding];

NetcaSignedDataSignInterface \*signedDataInterface = [[NetcaSignedDataSignInterface alloc]init];

signedDataInterface.signCert = signCert;

signedDataInterface.signAlgo = algo;

signedDataInterface.detach = true;//不带原文

NSData \*p7Signature = [signedDataInterface sign:PIN data:data];

NSString \*p7SignatureBase64String = [p7Signature base64EncodedStringWithOptions:0];

NSLog(@"p7签名成功，签名值的base64编码: %@", p7SignatureBase64String);

//3、验证

NetcaSignedDataVerifyInterface \*verifyInterface = [[NetcaSignedDataVerifyInterface alloc]init];

BOOL ret = [verifyInterface verify:data signature:p7Signature];

if (!ret) {

NSLog(@"验签失败");

return;

}

## 重置口令

重置口令的前提是已经注册管理员密钥到口令管理系统，操作步骤如下：

1. 设置口令管理系统的服务器地址；
2. 注册管理员密钥。

//1、设置口令管理系统的服务器地址

NetcaDeviceInterface \*deviceInterface = [\_crypto getDeviceInterface];

//配置信息是一个json串，包括服务器地址和appid。这是测试环境的地址，正式使用需要替换

NSString \*serverUrl = @"http://192.168.20.142:7093/mobilekey/sopin";

NetcaError rv = [deviceInterface setPinManagerSystemAddr:serverUrl];

if (rv != NetcaErrorCodeSuccess) {

NSLog(@"设置口令管理系统服务器地址失败");

}

//2、获取设备device，参考5.4

//3、注册管理员密钥

NetcaPinManageSystemInterface \*pinManageSystemInterface = [[NetcaPinManageSystemInterface alloc]init];

pinManageSystemInterface.device = device;

[pinManageSystemInterface registerSoPin:@"" userPin:PIN completion:^(NSError \* \_Nullable error) {

if (error) {

NSLog(@"注册管理员密钥失败，错误码:%ld, 错误信息:%@", (long)error.code, error.localizedDescription);

} else {

NSLog(@"注册管理员密钥成功");

}

}];

注册管理员密钥成功后，下次口令忘记或锁死，即可进行口令的重置。

重置步骤如下：

1. 请求获取管理员密钥，得到需要签名的内容dataToSign；
2. 由调用者自行对dataToSign进行签名，得到签名值；
3. 调用重置口令接口。

//1、请求获取管理员密钥

NetcaPinManageSystemInterface \*pinManageSystemInterface = [[NetcaPinManageSystemInterface alloc]init];

pinManageSystemInterface.device = device;

[pinManageSystemInterface requestGetSoPin:^(NSError \* \_Nullable error, NSString \* \_Nullable dataToSign) {

if (error) {

NSLog(@"请求获取管理员密钥失败，错误码:%ld, 错误信息:%@", (long)error.code, error.localizedDescription);

} else {

NSLog(@"请求获取管理员密钥成功，要签名的内容:%@", dataToSign);

}

}];

// 接下来，调用者自行对dataToSign进行签名，得到签名值后，进行下一步

//2、重置口令

NSString \*newUserPin = @"123456"; //新的用户密码

NSString \*signature = @""; //签名值

[pinManageSystemInterface unlockUserPin:newUserPin signature:signature completion:^(NSError \* \_Nullable error) {

if (error) {

NSLog(@"重置密码失败，错误码:%ld, 错误信息:%@", (long)error.code, error.localizedDescription);

} else {

NSLog(@"重置密码成功");

}

}];

## 写章

//1、先获取设备device，参考5.2

//2、判断设备是否支持读写章

BOOL ret = [device isSupportSeal];

if (!ret) {

NSLog(@"该设备不支持读写章");

return;

}

//3、验证用户PIN码

int retryNum;

ret = [device verifyPwdOfType:@1 pwd:PIN pRetryNum:&retryNum];

if (!ret) {

NSLog(@"用户PIN码验证不通过");

return;

}

//4、写章

NSString \*sealName = @"用户签章";//签章名称

NSString \*sealImagePath = [[NSBundle mainBundle]pathForResource:@"seal" ofType:@"png"];

NSData \*sealImageData = [NSData dataWithContentsOfFile:sealImagePath];//签章图片数据

ret = [device writeSeal:sealName picture:sealImageData];

if (!ret) {

NSLog(@"写章失败");

return;

}

NSLog(@"写章成功");

## 读章

//1、先获取设备device，参考5.2

//2、判断设备是否支持读写章

BOOL ret = [device isSupportSeal];

if (!ret) {

NSLog(@"该设备不支持读写章");

return;

}

//3、 获取签章个数

int count = [device getSealCount];

if (count <= 0) {

NSLog(@"该设备中没有签章");

return;

}

//4、 读章

NSData \*imageData = [device readSeal:0];

if (!imageData) {

NSLog(@"读章失败");

return;

}

NSLog(@"读章成功，签章图片数据:%@", imageData);

}

# 常量

参见头文件NetcaType.h、NetcaAlgorithm.h。

# 错误码

参见头文件NetcaError.h、NetcaMobileCryptoModuleConst.h。