****

**信安移动安全中间件**

**接口文档**

**(Android)**

**北京信安世纪科技股份有限公司**

版权声明

本文档受版权保护，未经北京信安世纪科技股份有限公司许可，任何人不得以任何理由和形式使用、复制、传播和编辑本文档，除非是在版权法的许可范围内。

本手册所涉及的案例均是当前情况，北京信安世纪科技股份有限公司有权利随时更改，恕不提前通知。对于本手册内容，包括但不限于隐含的商业性能和特定用途适应性说明，北京信安世纪科技股份有限公司不承担任何责任。

北京信安世纪科技股份有限公司对于因本手册的提供造成的设备性能、使用和按键错误而导致的相关事故或因之引起的损害不承担任何责任。

商标声明

本手册中所涉及的产品名称仅作识别之用。手册中涉及的其他公司的注册商标或版权属各商标注册人所有，恕不逐一列明。

合格声明

北京信安世纪科技股份有限公司自主声明移动统一认证安全管理平台系列产品符合FCC规定第15条。

关于信安世纪

北京信安世纪科技股份有限公司是中国领先的应用安全产品和解决方案供应商，主要为金融、能源、电信等行业和政府机构提供应用安全的产品、解决方案和服务，致力于实现互联网络和企业内部网络的通讯安全、交易安全和网络资源安全保护。

联系信安世纪

**官方网站**: https://www.infosec.com.cn/

**电子信箱**: [support@infosec.com.cn](mailto:support@infosec.com.cn)

**地址**: 北京市西城区宣武门外大街甲一号环球财讯中心C座4层403

**邮编**: 100052

**电话**: 86-10-68025518

86-10-68025528

86-10-68025519（传真）

电话联系时间：周一至周五早9点至晚6点

目录

[版权声明 I](#_Toc101346948)

[商标声明 I](#_Toc101346949)

[合格声明 I](#_Toc101346950)

[关于信安世纪 II](#_Toc101346951)

[联系信安世纪 II](#_Toc101346952)

[第1章 引言 1](#_Toc101346953)

[1.1 编写目的 1](#_Toc101346954)

[1.2 适用对象 1](#_Toc101346955)

[1.3 术语 1](#_Toc101346956)

[第2章 接口目标 2](#_Toc101346957)

[2.1 需求概述 2](#_Toc101346958)

[第3章 接口说明 3](#_Toc101346959)

[3.1 基础功能接口 3](#_Toc101346960)

[3.1.1 调用类 3](#_Toc101346961)

[3.1.2 SDK初始化 3](#_Toc101346962)

[3.1.3 SDK初始化(网络接管模式) 4](#_Toc101346963)

[3.1.4 SDK网络超时设置 5](#_Toc101346964)

[3.1.5 获取设备标记 6](#_Toc101346965)

[3.1.6 获取SDK版本 6](#_Toc101346966)

[3.1.7 获取签名模式 7](#_Toc101346967)

[3.1.8 获取协同签名算法 8](#_Toc101346968)

[3.1.9 获取是否下载双证 8](#_Toc101346969)

[3.1.10 获取是否支持自主注册 9](#_Toc101346970)

[3.1.11 获取证书功能是否可用 9](#_Toc101346971)

[3.1.12 获取令牌功能是否可用 10](#_Toc101346972)

[3.1.13 获取PDF功能是否可用 11](#_Toc101346973)

[3.1.14 获取签名是否可用 11](#_Toc101346974)

[3.1.15 获取集成应用是否强制更新 12](#_Toc101346975)

[3.1.16 获取PDF印章大小限制 12](#_Toc101346976)

[3.1.17 获取MAuth服务器产生的二维码的有效期 13](#_Toc101346977)

[3.1.18 获取MAuth服务器产生的注册码的有效期 14](#_Toc101346978)

[3.2 动态令牌接口 14](#_Toc101346979)

[3.2.1 调用类 14](#_Toc101346980)

[3.2.2 种子下载 15](#_Toc101346981)

[3.2.3 种子更新 15](#_Toc101346982)

[3.2.4 获取令牌状态 16](#_Toc101346983)

[3.2.5 生成令牌口令 17](#_Toc101346984)

[3.2.6 修改令牌种子PIN码 18](#_Toc101346985)

[3.2.7 验证令牌种子PIN码 19](#_Toc101346986)

[3.2.8 检查本地令牌种子是否存在 20](#_Toc101346987)

[3.3 证书管理接口 20](#_Toc101346988)

[3.3.1 调用类 20](#_Toc101346989)

[3.3.2 下载证书 21](#_Toc101346990)

[3.3.3 更新证书 22](#_Toc101346991)

[3.3.4 获取证书 22](#_Toc101346992)

[3.3.5 获取证书公钥 23](#_Toc101346993)

[3.3.6 删除一张证书 24](#_Toc101346994)

[3.3.7 修改证书PIN 25](#_Toc101346995)

[3.3.8 验证PIN 26](#_Toc101346996)

[3.3.9 获取证书状态 26](#_Toc101346997)

[3.3.10 检查证书是否存在 27](#_Toc101346998)

[3.3.11 获取证书信息 28](#_Toc101346999)

[3.3.12 使用证书进行加密 29](#_Toc101347000)

[3.3.13 使用证书进行解密 30](#_Toc101347001)

[3.3.14 制作数字信封 31](#_Toc101347002)

[3.3.15 使用证书打开数字信封 32](#_Toc101347003)

[3.3.16 获取服务器加密证书 33](#_Toc101347004)

[3.3.17 制作带签名的数字信封 33](#_Toc101347005)

[3.3.18 制作带签名的数字信封(对接NetSign) 35](#_Toc101347006)

[3.4 签名接口 36](#_Toc101347007)

[3.4.1 调用类 36](#_Toc101347008)

[3.4.2 签名 36](#_Toc101347009)

[3.4.3 对Hash数据签名 37](#_Toc101347010)

[3.4.4 Raw签名验证（在线） 38](#_Toc101347011)

[3.4.5 Attach签名验证（在线） 39](#_Toc101347012)

[3.4.6 Detach签名验证（在线） 40](#_Toc101347013)

[3.4.7 Raw签名验证（离线） 40](#_Toc101347014)

[3.4.8 Attach签名验证（离线） 41](#_Toc101347015)

[3.4.9 Detach签名验证（离线） 42](#_Toc101347016)

[3.5 SM3算法接口 43](#_Toc101347017)

[3.5.1 构造方法 43](#_Toc101347018)

[3.5.2 SM3摘要计算 43](#_Toc101347019)

[3.6 SM4算法接口 44](#_Toc101347020)

[3.6.1 构造方法 44](#_Toc101347021)

[3.6.2 生成SM4秘钥 45](#_Toc101347022)

[3.6.3 SM4加密 45](#_Toc101347023)

[3.6.4 SM4解密 46](#_Toc101347024)

[3.7 HmacSM3算法接口 47](#_Toc101347025)

[3.7.1 构造方法 47](#_Toc101347026)

[3.7.2 HmacSM3计算 48](#_Toc101347027)

[3.8 PDF签章接口 49](#_Toc101347028)

[3.8.1 调用类 49](#_Toc101347029)

[3.8.2 从二维码中获取PDF关联随机数 49](#_Toc101347030)

[3.8.3 印章图片注册 50](#_Toc101347031)

[3.8.4 印章图片更新 51](#_Toc101347032)

[3.8.5 印章图片下载 52](#_Toc101347033)

[3.8.6 根据随机数获取PDF的下载地址 53](#_Toc101347034)

[3.8.7 执行PDF签章 54](#_Toc101347035)

[3.9 PC登录接口 56](#_Toc101347036)

[3.9.1 调用类 56](#_Toc101347037)

[3.9.2 绑定PC设备 56](#_Toc101347038)

[3.9.3 获取PushID 57](#_Toc101347039)

[3.9.4 获取已经绑定的PC列表 58](#_Toc101347040)

[3.9.5 获取登陆日志 58](#_Toc101347041)

[3.10 二维码登陆 59](#_Toc101347042)

[3.10.1 调用类 59](#_Toc101347043)

[3.10.2 二维码数字签名登录 60](#_Toc101347044)

[3.10.3 二维码动态令牌登录 61](#_Toc101347045)

[3.11 对称秘钥管理 61](#_Toc101347046)

[3.11.1 调用类 62](#_Toc101347047)

[3.11.2 生成新的对称秘钥 62](#_Toc101347048)

[3.11.3 获取DEKID 63](#_Toc101347049)

[3.11.4 加密数据 63](#_Toc101347050)

[3.11.5 解密数据 65](#_Toc101347051)

[3.11.6 对数据做Hmac 66](#_Toc101347052)

[3.12 集成应用管理 67](#_Toc101347053)

[3.12.1 调用类 67](#_Toc101347054)

[3.12.2 APP版本检查 67](#_Toc101347055)

[3.13 下载管理接口 68](#_Toc101347056)

[3.13.1 调用类 68](#_Toc101347057)

[3.13.2 执行下载 68](#_Toc101347058)

[3.14 HTTPS接口 69](#_Toc101347059)

[3.14.1 构造方法 69](#_Toc101347060)

[3.14.2 设置信任证书 70](#_Toc101347061)

[3.14.3 设置客户端证书(非SDK内部证书) 71](#_Toc101347062)

[3.14.4 设置客户端证书(SDK内部证书) 72](#_Toc101347063)

[3.15 SSLSocket接口 73](#_Toc101347064)

[3.15.1 构造方法 73](#_Toc101347065)

[3.15.2 设置信任证书 74](#_Toc101347066)

[3.15.3 设置客户端证书(非SDK内部证书) 74](#_Toc101347067)

[3.15.4 设置客户端证书(SDK内部证书) 75](#_Toc101347068)

[3.15.5 建立连接 76](#_Toc101347069)

[3.16 用户管理接口 77](#_Toc101347070)

[3.16.1 调用类 77](#_Toc101347071)

[3.16.2 用户签约 77](#_Toc101347072)

[3.16.3 用户创建 78](#_Toc101347073)

[3.16.4 重新获取注册码 79](#_Toc101347074)

[3.16.5 获取用户状态 80](#_Toc101347075)

[3.16.6 删除用户 82](#_Toc101347076)

[3.16.7 判断用户是否已经锁定 83](#_Toc101347077)

[3.16.8 判断用户是否是证书用户 84](#_Toc101347078)

[3.16.9 判断用户是否是令牌用户 85](#_Toc101347079)

[3.16.10 获取用户的签约码 86](#_Toc101347080)

[3.16.11 获取用户的邮箱地址 87](#_Toc101347081)

[3.16.12 获取用户的手机号 88](#_Toc101347082)

[3.16.13 获取用户的姓名 89](#_Toc101347083)

[第4章 基础回调接口 90](#_Toc101347084)

[4.1 异步方法回调接口Result和ResultListener 90](#_Toc101347085)

[4.1.1 异步方法定义 90](#_Toc101347086)

[4.1.2 异步方法实现 90](#_Toc101347087)

[4.1.3 异步方法返回值获取 90](#_Toc101347088)

[4.2 网络接口NetworkInterface和NetworkCallback 91](#_Toc101347089)

[4.2.1 网络接管模式 91](#_Toc101347090)

[4.2.2 NetworkCallback 91](#_Toc101347091)

[4.2.3 NetworkDownloadCallback 91](#_Toc101347092)

[第5章 SDK集成说明 91](#_Toc101347093)

[5.1 引入Java依赖库 91](#_Toc101347094)

[5.2 引入JNI动态链接库 92](#_Toc101347095)

[5.3 添加配置信息 92](#_Toc101347096)

[第6章 SDK功能使用说明 92](#_Toc101347097)

[6.1 基础功能 92](#_Toc101347098)

[6.1.1 SDK初始化 92](#_Toc101347099)

[6.1.2 用户签约 93](#_Toc101347100)

[6.1.3 用户创建 94](#_Toc101347101)

[6.1.4 用户锁定和解锁 94](#_Toc101347102)

[6.1.5 用户状态获取 95](#_Toc101347103)

[6.1.6 用户其他信息获取 95](#_Toc101347104)

[6.1.7 证书的申请和更新 96](#_Toc101347105)

[6.1.8 获取证书状态 96](#_Toc101347106)

[6.1.9 证书信息操作 96](#_Toc101347107)

[6.1.10 证书功能操作 96](#_Toc101347108)

[6.1.11 令牌种子的申请和更新 97](#_Toc101347109)

[6.1.12 获取令牌状态 97](#_Toc101347110)

[6.1.13 令牌信息操作 97](#_Toc101347111)

[6.1.14 生成动态口令 97](#_Toc101347112)

[6.2 业务场景功能 98](#_Toc101347113)

[6.2.1 数字签名认证 98](#_Toc101347114)

[6.2.2 动态令牌认证 100](#_Toc101347115)

[6.2.3 二维码Web登录 102](#_Toc101347116)

[6.2.4 OS扫码登录 104](#_Toc101347117)

[6.2.5 PDF签章 107](#_Toc101347118)

[6.2.6 对称数据加解密 110](#_Toc101347119)

[6.2.7 HTTPS功能 112](#_Toc101347120)

[6.2.8 SSLSocket功能 113](#_Toc101347121)

[附录A 错误码 115](#_Toc101347122)

# 引言

## 编写目的

本文描述了信安移动安全中间件的接口列表。为使用信安移动安全中间件的开发人员提供基本的接口介绍，使用场景，注意事项等内容。

## 适用对象

应用系统业务人员、技术开发人员、测试人员。

## 术语

* SM2：一种椭圆曲线公钥密码算法，其加解密采用不同的密钥。210位的SM2密钥强度与2048位的RSA强度相当。该算法于2010年12月在国家密码管理局网站正式对外公布。
* SM3：对于给定的长度为k(k<264)的消息，SM3密码杂凑算法经过填充、迭代压缩和选裁，生成杂凑值。SM3对于任意长度的报文的输入，经预处理过的消息分组长度为512比特，最终产生一个32字节的哈希值。
* NetPass：NetPass 动态密码认证系统是信安世纪公司研发的、具有自主知识产权的、低成本、易推广、易使用的新型身份认证系统，其安全强度高于静态口令，以种子驱动技术为手段，采用了“一次一密，挑战响应”的技术，为用户提供身份认证安全保障。

# 接口目标

## 需求概述

信安移动安全中间件是信安移动管理平台（以下简称MAuth）在移动端的功能延伸，安全中间件以软件开发工具(Software Development Kit)的方式提供，移动软件在开发过程中集成了SDK，就可以通过利用MAuth实现以下的功能：

* 初始化功能：完成MAuth服务端相关通信参数设置(IP、端口和通信方式配置等)，SDK和配置信息获取；
* 用户签约：使用授权码完成与MAuth服务器签约认证，只有签约成功的用户才能使用移动安全中间件的相关功能；
* 令牌管理：完成动态令牌的种子密钥下载、存储保护、更新、动态口令的生成；
* 证书管理：完成P10生成、证书下载、证书更新、获取证书信息
* 证书密钥保护：证书PIN码验证，修改
* 密码算法功能：基于数字证书的签名(裸签、Attached、Detached)、验签、加密、解密
* 安全通讯功能：SSL和HTTPS通信
* 客户端支持：Android和iOS。

# 接口说明

安全中间件所提供的主要客户端接口有以下几种，主要根据功能模块进行划分。

## 基础功能接口

基础功能主要包括SDK初始化，关键参数配置，以及SDK相关信息获取等。

### 调用类

IMSSdk

**构造方法:** 无

**成员变量:** 无

### SDK初始化

#### 功能描述

初始化信安世纪移动安全中间件，所有使用信安世纪安全中间件的应用必须尽早进行初始化操作，在SDK尚未成功初始化以前，SDK不能保证所有功能模块的正常工作

#### 前置条件

已经在应用中成功集成了SDK

#### 方法原型

public static boolean initialization(Context context, String host， String base64HttpsTrustCert，String appID，String appSecret，Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| context | Context | 上下文句柄 | 是 |
| host | String | 移动认证平台服务器地址 | 是 |
| base64HttpsTrustCert | String | https信任证书(Base64格式) | 否 |
| appID | String | 应用ID | 是 |
| appSecret | String | 应用Secret | 是 |
| resultListener | ResultListener | 初始化结果回调接口 | 是 |

#### 返回值

初始化成功与否都会通过回调接口进行回调

### SDK初始化(网络接管模式)

#### 功能描述

初始化信安世纪移动安全中间件，所有使用信安世纪安全中间件的应用必须尽早进行初始化操作，在SDK尚未成功初始化以前，SDK不能保证所有功能模块的正常工作

#### 前置条件

已经在应用中成功集成了SDK

#### 方法原型

public static boolean initialization(Context context, NetworkInterface networkInterface，String appID，String appSecret，Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| context | Context | 上下文句柄 | 是 |
| networkInterface | NetworkInterface | 网络通信接口实例 | 是 |
| appID | String | 应用ID | 是 |
| appSecret | String | 应用Secret | 是 |
| resultListener | ResultListener | 初始化结果回调接口 | 是 |

#### 返回值

初始化成功与否都会通过回调接口进行回调

### SDK网络超时设置

#### 功能描述

设置SDK和MAuth服务器之间的网络连接超时时间,单位为毫秒

#### 前置条件

已经在应用中成功集成了SDK,使用网络接管模式进行初始化时该设置无效

#### 方法原型

public static void setNetworkTimeout(int millSeconds)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| millSeconds | int | 超时时间(毫秒) | 是 |

#### 返回值

无

### 获取设备标记

#### 功能描述

获取与SDK绑定的设备标记信息

#### 前置条件

已经在应用中成功集成了SDK

#### 方法原型

public static String deviceIdentifier()

#### 请求参数

无

#### 返回值

字符串类型的设备标记

### 获取SDK版本

#### 功能描述

获取当前SDK的版本信息

#### 前置条件

已经在应用中成功集成了SDK

#### 方法原型

public static String version()

#### 请求参数

无

#### 返回值

字符串类型的SDK版本

### 获取签名模式

#### 功能描述

获取签名模式配置信息

#### 前置条件

已经成功进行了初始化，否则返回以前的配置信息

#### 方法原型

public static boolean isCollaborative()

#### 请求参数

无

#### 返回值

true表示使用协同签名模式，false表示非协同签名模式

### 获取协同签名算法

#### 功能描述

获取协同签名所使用的算法

#### 前置条件

已经成功进行了初始化，否则返回以前的配置信息

#### 方法原型

public static boolean isInfosecCollaborative()

#### 请求参数

无

#### 返回值

true表示使用信安世纪算法，false表示使用中科院算法

### 获取是否下载双证

#### 功能描述

获取所下载的证书是否是双证

#### 前置条件

已经成功进行了初始化，否则返回以前的配置信息

#### 方法原型

public static boolean isPairedCert()

#### 请求参数

无

#### 返回值

true表示使用双证，false表示使用单证

### 获取是否支持自主注册

#### 功能描述

获取是有允许在客户端自主注册

#### 前置条件

已经成功进行了初始化，否则返回以前的配置信息

#### 方法原型

public static boolean clientRegistEnable()

#### 请求参数

无

#### 返回值

true表示支持自主注册，false表示不支持自主注册

### 获取证书功能是否可用

#### 功能描述

获取证书是否可用的配置信息

#### 前置条件

已经成功进行了初始化，否则返回以前的配置信息

#### 方法原型

public static boolean isCertAvailable()

#### 请求参数

无

#### 返回值

true表示证书功能可用，false表示证书功能不可用

### 获取令牌功能是否可用

#### 功能描述

获取令牌是否可用的配置信息

#### 前置条件

已经成功进行了初始化，否则返回以前的配置信息

#### 方法原型

public static boolean isTokenAvailable()

#### 请求参数

无

#### 返回值

true表示令牌功能可用，false表示令牌功能不可用

### 获取PDF功能是否可用

#### 功能描述

获取PDF签章是否可用的配置信息

#### 前置条件

已经成功进行了初始化，否则返回以前的配置信息

#### 方法原型

public static boolean isPDFAvailable()

#### 请求参数

无

#### 返回值

true表示PDF签章功能可用，false表示PDF签章功能不可用

### 获取签名是否可用

#### 功能描述

获取数字签名是否可用的配置信息

#### 前置条件

已经成功进行了初始化，否则返回以前的配置信息

#### 方法原型

public static boolean isSignAvailable()

#### 请求参数

无

#### 返回值

true表示数字签名功能可用，false表示数字签名功能不可用

### 获取集成应用是否强制更新

#### 功能描述

获取集成了当前SDK的应用程序是否应该强制更新的配置信息

#### 前置条件

已经成功进行了初始化，否则返回以前的配置信息

#### 方法原型

public static boolean isAppForceUpdate()

#### 请求参数

无

#### 返回值

true表示需要强制更新，false表示不需要强制更新

### 获取PDF印章大小限制

#### 功能描述

获取PDF印章数据大小限制的配置信息，单位为KB

#### 前置条件

已经成功进行了初始化，否则返回以前的配置信息

#### 方法原型

public static int getSealerImageSize()

#### 请求参数

无

#### 返回值

Int类型的印章大小限制

### 获取MAuth服务器产生的二维码的有效期

#### 功能描述

获取在MAuth服务器上产生的二维码的有效期的配置信息，时间单位为毫秒，自二维码在MAuth服务器上产生的时间开始计算

#### 前置条件

已经成功进行了初始化，否则返回以前的配置信息

#### 方法原型

public static int getQRCodeExpireMillSeconds()

#### 请求参数

无

#### 返回值

Int类型的有效期数据

### 获取MAuth服务器产生的注册码的有效期

#### 功能描述

获取在MAuth服务器上产生的注册码的有效期的配置信息，时间单位为毫秒，自注册码在MAuth服务器上产生的时间开始计算

#### 前置条件

已经成功进行了初始化，否则返回以前的配置信息

#### 方法原型

public static int getSignCodeExpireMillSeconds()

#### 请求参数

无

#### 返回值

Int类型的有效期数据

## 动态令牌接口

动态口令相关接口主要用于存储种子密钥，计算动态口令等操作。

### 调用类

TokenManager类

**构造方法:** public TokenManager(Context context)

**成员变量:** 无

### 种子下载

#### 功能描述

连接MAuth服务器进行令牌种子下载

#### 前置条件

已经成功初始化，并且用户签约成功

#### 方法原型

public void requestSeed(String userName, String PIN, Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| PIN | String | 为种子指定的PIN码 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 种子更新

#### 功能描述

连接MAuth进行令牌种子更新

#### 前置条件

已经成功下载了种子

#### 方法原型

public void updateSeed(final String userName,String PIN,Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| PIN | String | 为种子指定的PIN码 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 获取令牌状态

#### 功能描述

连接MAuth服务器获取令牌状态

#### 前置条件

已经成功集成SDK并且初始化

#### 方法原型

public void getStatusOnLine(final String username, Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调，通过Result对象的getResultDesc方法获取状态描述

### 生成令牌口令

#### 功能描述

生成令牌口令

#### 前置条件

已经成功下载种子

#### 方法原型

public void getOTP(String userName,String PIN, String algorithm, long time, String challenge, int digit, int step, Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| PIN | String | 种子PIN口令 | 是 |
| algorithm | String | 生成口令的算法   1. “GUOMI”:国密算法 2. “OATH”:国际算法 | 是 |
| time | long | 要参与运算的时间值 | 是 |
| challenge | String | 要参与运算的挑战值 | 否 |
| digit | int | 运算位数 | 是 |
| step | int | 运算的时间间隔 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调，通过Result对象的getResultDesc方法获取口令值

### 修改令牌种子PIN码

#### 功能描述

修改令牌种子的PIN

#### 前置条件

已经成功下载种子

#### 方法原型

public void changePIN(String userName,String oldPIN,String newPIN, Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| oldPIN | String | 旧PIN码 | 是 |
| newPIN | String | 新的PIN码 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 验证令牌种子PIN码

#### 功能描述

验证令牌种子的PIN

#### 前置条件

已经成功下载种子

#### 方法原型

public void verifyPIN(String userName, String PIN, Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| PIN | String | 要验证的PIN码 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 检查本地令牌种子是否存在

#### 功能描述

检查本地是否已经存在已经下载的令牌种子

#### 前置条件

已经成功进行初始化

#### 方法原型

public boolean isSeedExist(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

true表示已经存在，false表示不存在

## 证书管理接口

证书管理接口主要涉及到证书的申请，更新，证书查看以及利用证书进行的加解密等操作。

### 调用类

CertManager类

**构造方法:** public CertManager(Context context)

**成员变量:** 无

### 下载证书

#### 功能描述

连接MAuth进行证书申请并下载

#### 前置条件

已经成功集成了SDK并初始化

#### 方法原型

public void requestCert(final String userName, final String PIN, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| PIN | String | 为证书指定的PIN码 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 更新证书

#### 功能描述

连接MAuth进行证书更新

#### 前置条件

已经成功下载了证书

#### 方法原型

public void updateCert(final String userName, final String PIN, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| PIN | String | 证书的PIN码 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 获取证书

#### 功能描述

获取指定别名的证书

#### 前置条件

已经成功集成SDK并初始化

#### 方法原型

public String getCert(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 要获取证书的用户名 | 是 |

#### 返回值

获取成功返回Base64格式的证书,获取失败返回null

### 获取证书公钥

#### 功能描述

获取指定用户的证书公钥

#### 前置条件

已经成功下载证书

#### 方法原型

public String getCertPubKey(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 要获取公钥的用户名 | 是 |

#### 返回值

获取成功返回公钥16进制的字符串,获取失败返回null

### 删除一张证书

#### 功能描述

删除指定用户名的证书

#### 前置条件

已经成功下载证书

#### 方法原型

public boolean deleteCert(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 要删除证书的用户名 | 是 |

#### 返回值

删除成功返回true，否则返回false

### 修改证书PIN

#### 功能描述

修改指定用户证书的PIN码

#### 前置条件

已经成功下载了证书

#### 方法原型

public void changePIN(final String userName, final String oldPIN, final String newPIN, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 要修改PIN码的用户名 | 是 |
| oldPIN | String | 原来的PIN | 是 |
| newPIN | String | 新的PIN | 是 |
| resultListener | ResultListener | 操作结果回调 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 验证PIN

#### 功能描述

检查给定的PIN码是否符合指定用户名的证书

#### 前置条件

已经成功下载证书

#### 方法原型

public void verifyPIN(String userName, String PIN, Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 要检查证书的用户名 | 是 |
| PIN | String | 要测试的证书PIN | 是 |
| resultListener | ResultListener | 操作结果回调 | 是 |

#### 返回值

通过resultListener进行回调，PIN码错误返回Result.INCORRECT\_CERTPIN并在resultID中返回剩余尝试次数。

### 获取证书状态

#### 功能描述

连接MAuth服务器获取证书状态

#### 前置条件

已经成功集成SDK并初始化成功

#### 方法原型

public void getStatusOnLine(String userName, Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调，通过Result对象的getResultDesc方法获取状态描述

### 检查证书是否存在

#### 功能描述

检查指定用户的证书是否存在

#### 前置条件

已经成功集成SDK并初始化成功

#### 方法原型

public boolean isCertExist(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 要检查证书的用户名 | 是 |

#### 返回值

证书存在返回true,证书不存在返回false

### 获取证书信息

#### 功能描述

获取指定用户的证书对应的字段

#### 前置条件

已经成功下载证书

#### 方法原型

public String getCertField(String userName, int field)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| field | int | 要获取的证书字段   1. 证书DN 2. 证书SN 3. 证书颁发者 4. 保留字段 5. 密钥用法 6. 密钥长度 7. 证书有效期开始 8. 证书有效期结束 9. 证书签名算法 10. 证书版本 | 是 |

#### 返回值

根据传入的字段，返回对应的值

### 使用证书进行加密

#### 功能描述

使用指定的证书证书内容，对一个原文数据进行加密

#### 前置条件

已经成功下载了或者已经有了外部证书

#### 方法原型

public byte[] encryptByCert(String certStr, byte[] plain)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| certStr | String | 证书内容，Base64格式 | 是 |
| plain | byte[] | 要加密的数据 | 是 |

#### 返回值

Byte[]类型的被加密后的数据

### 使用证书进行解密

#### 功能描述

对使用该证书加密后的数据进行解密

#### 前置条件

已经成功下载双证证书

#### 方法原型

public void decryptByCert(String userName, String PIN, boolean useSignCert, String encryptData, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| PIN | String | 证书的PIN | 是 |
| useSignCert | boolean | 是否使用签名证书解密 | 是 |
| encryptData | String | 被加密的数据，Base64格式 | 是 |
| resultListener | ResultListener | 回调监听器 | 是 |

#### 返回值

通过回调监听器返回解密后的数据,Base64格式

### 制作数字信封

#### 功能描述

使用指定的证书制作数字信封

#### 前置条件

已经成功进行了初始化

#### 方法原型

public String makeEnvelop(@NonNull String base64Cert, @NonNull String symmetricAlgorithm, @NonNull byte[] plain)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| base64Cert | String | 用于制作信封的证书 | 是 |
| symmetricAlgorithm | String | 信封要使用的对称算法("des"/"des3"/"aes-256-ecb"/"aes-256-cbc"/"rc2"/"rc4"/"sms4-cbc"),如果对接NetSign，则只支持"des3"/"rc4"/"sms4-cbc"算法 | 是 |
| plain | Byte[] | 要加密到信封的原文数据 | 是 |

#### 返回值

返回信封数据,Base64格式

### 使用证书打开数字信封

#### 功能描述

使用用户名和PIN打开制作好的数字信封，必须是本地证书做的信封

#### 前置条件

已经成功下载证书

#### 方法原型

public void openEnvelop(String userName, String PIN, boolean useSignCert，String envelopData, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| PIN | String | 证书的PIN | 是 |
| useSignCert | boolean | 是否使用签名证书打开信封 | 是 |
| envelopData | String | 信封数据 | 是 |
| resultListener | ResultListener | 回调监听器 | 是 |

#### 返回值

通过回调接口返回信封原文数据,Base64格式

### 获取服务器加密证书

#### 功能描述

获取服务器加密证书,用于进行制作带签名的数字信封

#### 前置条件

已经成功进行了初始化

#### 方法原型

public void getCertForEnvelop(final String certDN, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| certDN | String | 证书DN | 是 |
| resultListener | ResultListener | 回调监听器 | 是 |

#### 返回值

通过回调接口返回证书数据,Base64格式

### 制作带签名的数字信封

#### 功能描述

使用指定的外部证书和已经下载好的签名证书制作一个带有签名数据的数字信封

#### 前置条件

已经成功进行了初始化并下载了证书

#### 方法原型

public void makeSignedEnvelop(@NonNull String userName, @NonNull String PIN, @NonNull String encCert, byte[] plain, @NonNull String symmetricAlgorithm, @NonNull Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| PIN | String | 证书的PIN码 | 是 |
| encCert | String | 用于制作信封的加密证书 | 是 |
| symmetricAlgorithm | String | 信封要使用的对称算法("des"/"des3"/"aes-256-ecb"/"aes-256-cbc"/"rc2"/"rc4"/"sms4-cbc"),如果对接NetSign，则只支持"des3"/"rc4"/"sms4-cbc"算法 | 是 |
| plain | Byte[] | 要加密到信封的原文数据 | 是 |
| resultListener | ResultListener | 回调监听器 | 是 |

#### 返回值

通过回调接口返回信封数据,Base64格式

### 制作带签名的数字信封(对接NetSign)

#### 功能描述

使用指定的外部证书和已经下载好的签名证书制作一个带有签名数据的数字信封，制作的信封如果需要对接NetSign，需要修改NetSign服务器的config目录的algorithm.xml文件中的SM4算法OID，在其后面加上.2即可

#### 前置条件

已经成功进行了初始化并下载了证书

#### 方法原型

public void makeSignedEnvelopForNetSign(@NonNull String userName, @NonNull String PIN, @NonNull String encCert, byte[] plain, @NonNull String symmetricAlgorithm, @NonNull Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| PIN | String | 证书的PIN码 | 是 |
| encCert | String | 用于制作信封的加密证书 | 是 |
| symmetricAlgorithm | String | 信封要使用的对称算法"des3"/"rc4"/"sms4-cbc" | 是 |
| plain | Byte[] | 要加密到信封的原文数据 | 是 |
| resultListener | ResultListener | 回调监听器 | 是 |

#### 返回值

通过回调接口返回信封数据,Base64格式

## 签名接口

签名接口主要包括裸签，Attach签名以及Detach签名，以及不同签名结果的验证

### 调用类

SignManager类

**构造方法:** public SignManager(Context context)

**成员变量:** 无

### 签名

#### 功能描述

使用数字证书进行签名

#### 前置条件

已经成功下载了证书

#### 方法原型

public void sign(final String userName, final String PIN, final byte[] plain, final int type, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| plain | Byte[] | 签名原文 | 是 |
| type | int | 签名类型  0:Raw签名   1. Attach签名 2. Detach签名 | 是 |
| PIN | String | 证书PIN | 是 |
| resultListener | ResultListener | 回调操作接口 | 是 |

#### 返回值

签名成功返回Base64格式签名信息，其余均表示签名失败，通过resultListener进行回调

### 对Hash数据签名

#### 功能描述

使用数字证书对已经做过Hash操作的数据进行签名

#### 前置条件

已经成功下载了证书

#### 方法原型

public void signHash(final String userName, final String PIN, final byte[] plain, final int type, inal String hashAlg, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| plain | Byte[] | 已经做过Hash的要签名的数据 | 是 |
| type | int | 签名类型  0:Raw签名  1:Attach签名  2:Detach签名 | 是 |
| PIN | String | 证书PIN | 是 |
| hashAlg | String | 做Hash操作时用的算法 |  |
| resultListener | ResultListener | 回调操作接口 | 是 |

#### 返回值

签名成功返回Base64格式签名信息，其余均表示签名失败，通过resultListener进行回调

### Raw签名验证（在线）

#### 功能描述

对Raw签名数据进行验证

#### 前置条件

已经成功初始化

#### 方法原型

public void rawVerifyOnLine(String signData, String plainText, String base64Cert, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| signData | String | Raw签名数据 | 是 |
| plainText | String | 签名时的原文数据 | 是 |
| base64Cert | String | 签名时使用的证书(base64格式) | 是 |
| resultListener | ResultListener | 回调操作接口 | 是 |

#### 返回值

通过resultListener进行回调

### Attach签名验证（在线）

#### 功能描述

对Attach签名数据进行验证

#### 前置条件

已经成功初始化

#### 方法原型

public void attachVerifyOnLine(String signData, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| signData | String | Raw签名数据 | 是 |
| resultListener | ResultListener | 回调操作接口 | 是 |

#### 返回值

通过resultListener进行回调

### Detach签名验证（在线）

#### 功能描述

对Detach签名数据进行验证

#### 前置条件

已经成功初始化

#### 方法原型

public void detachVerifyOnLine(String signData, String plainText, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| signData | String | Raw签名数据 | 是 |
| plainText | String | 签名时的原文数据 | 是 |
| resultListener | ResultListener | 回调操作接口 | 是 |

#### 返回值

通过resultListener进行回调

### Raw签名验证（离线）

#### 功能描述

对Raw签名数据进行验证

#### 前置条件

已经成功初始化

#### 方法原型

public boolean rawVerifyLocal( String signData, byte[] plain, String base64Cert)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| signData | String | Raw签名数据 | 是 |
| plain | Byte[] | 签名时的原文数据 | 是 |
| base64Cert | String | 签名时使用的证书(base64格式) | 是 |

#### 返回值

true表示验证通过，false表示验签失败

### Attach签名验证（离线）

#### 功能描述

对Attach签名数据进行验证

#### 前置条件

已经成功初始化

#### 方法原型

public boolean attachVerifyLocal(String signData)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| signData | String | Raw签名数据 | 是 |

#### 返回值

true表示验证通过，false表示验签失败

### Detach签名验证（离线）

#### 功能描述

对Detach签名数据进行验证

#### 前置条件

已经成功初始化

#### 方法原型

public boolean detachVerifyLocal(String signData, byte[] plain)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| signData | String | Raw签名数据 | 是 |
| plain | Byte[] | 签名时的原文数据 | 是 |

#### 返回值

true表示验证通过，false表示验签失败

## SM3算法接口

该接口提供了基于SM3国密摘要算法的操作。

### 构造方法

#### 功能描述

创建一个SM3对象

#### 前置条件

已经成功集成SDK

#### 请求示例

public SM3()

#### 请求参数

无

#### 返回值

SM3对象

### SM3摘要计算

#### 功能描述

针对特定的数据进行SM3摘要计算

#### 前置条件

已经成功集成SDK并初始化

#### 请求示例

public String digestString(byte[] plainBytes)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| plainBytes | byte[] | 原文数据 | 是 |

#### 返回值

Base64格式的摘要数据

## SM4算法接口

该接口提供了基于SM4国密对称秘钥算法的秘钥生成，加解密等操作。

### 构造方法

#### 功能描述

创建一个SM4对象

#### 前置条件

已经成功集成SDK

#### 请求示例

public SM4()

#### 请求参数

无

#### 返回值

SM4对象

### 生成SM4秘钥

#### 功能描述

为SM4算法生成一个用来加密的秘钥

#### 前置条件

已经成功集成SDK并初始化

#### 请求示例

public byte[] generateSecretKey(byte[] seed)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| seed | byte[] | 种子参数 | 是 |

#### 返回值

秘钥数据

### SM4加密

#### 功能描述

使用SM4秘钥对特定数据进行加密操作

#### 前置条件

已经成功集成SDK并初始化

#### 请求示例

public String encrypt(byte[] secretKey, byte[] iv, byte[] plain, int mode)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| secretKey | byte[] | 种子参数 | 是 |
| iv | byte[] | 向量 | Mode为1时不能为空 |
| plain | byte[] | 原文数据 | 是 |
| mode | int | 是否是CBC模式 | 1时为CBC模式，否则为ECB模式 |

#### 返回值

Base64格式被加密的数据

### SM4解密

#### 功能描述

使用SM4秘钥对加密数据进行解密

#### 前置条件

已经成功集成SDK并初始化

#### 请求示例

public byte[] decrypt(byte[] secretKey, byte[] iv, byte[] encryptBytes, int mode)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| secretKey | byte[] | 种子参数 | 是 |
| iv | byte[] | 向量 | Mode为1时不能为空 |
| encryptBytes | byte[] | 密文数据 | 是 |
| mode | int | 是否是CBC模式 | 1时为CBC模式，否则为ECB模式 |

#### 返回值

原文数据

## HmacSM3算法接口

该接口提供了基于SM3国密MAC算法的操作。

### 构造方法

#### 功能描述

创建一个HmacSM3对象

#### 前置条件

已经成功集成SDK

#### 请求示例

public HmacSM3(byte[] key)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| key | byte[] | 秘钥数据 | 是 |

#### 返回值

HmacSM3对象

### HmacSM3计算

#### 功能描述

针对特定的数据进行HmacSM3摘要计算

#### 前置条件

已经成功集成SDK并初始化

#### 请求示例

public byte[] doFinal(byte[] plain)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| plain | byte[] | 原文数据 | 是 |

#### 返回值

byte[] 格式的MAC数据

## PDF签章接口

PDF签章主要有以下功能

### 调用类

PDFManager类

**构造方法:** public PDFManager(Context context)

**成员变量:** 无

### 从二维码中获取PDF关联随机数

#### 功能描述

从来自于MAuth产生的二维码中获取一个和PDF关联的随机数

#### 前置条件

已经下载证书

#### 方法原型

public String randomFromQR(String qrData)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| qrData | String | Raw签名数据 | 是 |

#### 返回值

一个String类型和和PDF绑定的随机数

### 印章图片注册

#### 功能描述

注册一个印章图片到服务器

#### 前置条件

已经成功下载了证书

#### 方法原型

public void registeSealer(final String userName, String PIN, Bitmap sealer, final int sealerType, final String companyName, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| PIN | String | 用户证书的PIN | 是 |
| sealer | File | 要注册签章图片文件,由于要进行Base64编码,所以实际可上传内容要比后台配置小1/4左右 | 是 |
| sealerType | int | 签章图片的类型:  pdf外接签章：  4表示个人章 ，123表示单位章  pdf内置签章：  1 单位公章  2 合同专用章  3财务专用章  4个人章 | 是 |
| companyName | String | 机构名称 | 否 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 印章图片更新

#### 功能描述

更新服务器上已经注册过的印章图片

#### 前置条件

已经成功下载了证书,并且已经注册过印章图片

#### 方法原型

public void updateSealer(final String userName, String PIN, File sealer, final int sealerType, final String companyName, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| PIN | String | 用户证书的PIN | 是 |
| sealer | File | 要更新的签章图片文件,,由于要进行Base64编码,所以实际可上传内容要比后台配置小1/4左右 | 是 |
| sealerType | int | 签章图片的类型:  pdf外接签章：  4表示个人章 ，123表示单位章  pdf内置签章：  1 单位公章  2 合同专用章  3财务专用章  4个人章 | 是 |
| companyName | String | 机构名称 | 否 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 印章图片下载

#### 功能描述

从服务器下载已经注册过的印章图片

#### 前置条件

已经成功下载证书

#### 方法原型

public void downloadSealer(final String userName, int sealerType, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| sealerType | int | 签章图片的类型:  pdf外接签章：  4表示个人章 ，123表示单位章  pdf内置签章：  1 单位公章  2 合同专用章  3财务专用章  4个人章 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调，成功后在resultDesc字段返回图片的数据

### 根据随机数获取PDF的下载地址

#### 功能描述

根据和PDF关联的随机数，获取和这个随机数关联的PDF文件的下载地址

#### 前置条件

已经成功上传了PDF文件

#### 请求示例

public void downloadPDF(final String userName, String random, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| random | String | 和PDF关联的随机数 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调，成功后在resultDesc字段返回PDF的URL

### 执行PDF签章

#### 功能描述

对和随机数关联的PDF文件进行签章

#### 前置条件

已经成功下载了证书

#### 请求示例

public void signPDF(final String userName, final String PIN, int sealerType, final String random, int pageNumber, Rect sealerRegion, String signReason, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| PIN | String | 证书PIN | 是 |
| sealerType | int | 签章图片的类型:  pdf外接签章：  4表示个人章 ，123表示单位章  pdf内置签章：  1 单位公章  2 合同专用章  3财务专用章  4个人章 | 是 |
| random | String | PDF关联随机数 | 是 |
| pageNumber | int | 要签章的PDF页码 | 是 |
| sealerRegion | Rect | 签名图片摆放的位置区域 | 是 |
| signReason | String | 签名原因或者描述 | 是 |
| resultListener | Result.ResultListener | 签章结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调，签章成功在resultDesc字段返回已签章文件的下载URL

## PC登录接口

PC登陆功能主要有以下接口

### 调用类

OSLoginManager类

**构造方法:** public OSLoginManager(Context context)

**成员变量:** 无

### 绑定PC设备

#### 功能描述

根据指定的参数，绑定一台PC设备

#### 前置条件

已经下载证书

#### 方法原型

public void bind(String userName, String pcName, String pcIdentifier, String bindCode, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| pcName | String | PC设备名 | 是 |
| pcIdentifier | String | PC设备标记 | 是 |
| bindCode | String | 绑定码 | 是 |
| resultListener | Result.ResultListener | 回调接口 | 是 |

#### 返回值

通过回调接口返回绑定结果

### 获取PushID

#### 功能描述

设置或者获取PushID

#### 前置条件

已经下载证书并绑定了设备

#### 方法原型

public void getPushID(String userName, String pushID, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| pushID | String | 要设置的PushID | 是 |
| resultListener | Result.ResultListener | 回调接口 | 是 |

#### 返回值

通过回调接口返回

### 获取已经绑定的PC列表

#### 功能描述

获取指定用户已经绑定过的PC设备列表

#### 前置条件

已经下载证书

#### 方法原型

public void getPCList(String userName, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| pushID | String | PushID，不为空表示设置ID，为空表示获取ID | 否 |
| resultListener | Result.ResultListener | 回调接口 | 是 |

#### 返回值

通过回调接口返回

### 获取登陆日志

#### 功能描述

获取当前用户登录的日志记录

#### 前置条件

已经下载证书，已经注册过设备

#### 方法原型

public void getLoginLog(String userName, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| pcIMEI | String | PC的IMEI | 是 |
| resultListener | Result.ResultListener | 回调接口 | 是 |

#### 返回值

通过回调接口返回

## 二维码登陆

Web登陆功能主要有以下接口

### 调用类

QRManager类

**构造方法:** public QRManager(Context context)

**成员变量:** 无

### 二维码数字签名登录

#### 功能描述

根据MAuth生成的二维码数据，进行Web页面认证登录或者PC扫码登录

#### 前置条件

已经下载证书

#### 方法原型

public void qrLoginByCert(final String userName, String PIN, String qrData, final int type, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| PIN | String | 证书PIN码 | 是 |
| qrData | String | 二维码数据 | 是 |
| type | int | 签名类型  0:raw签名  1:attach签名  2:detach签名 | 是 |
| resultListener | Result.ResultListener | 回调接口 | 是 |

#### 返回值

通过回调接口返回登录结果

### 二维码动态令牌登录

#### 功能描述

根据MAuth生成的二维码数据，进行Web页面认证登录或者PC扫码登录

#### 前置条件

已经下载令牌种子

#### 方法原型

public void qrLoginByToken(final String userName, String PIN, String qrData, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| PIN | String | 令牌PIN码 | 是 |
| qrData | String | 二维码数据 | 是 |
| resultListener | Result.ResultListener | 回调接口 | 是 |

#### 返回值

通过回调接口返回登录结果

## 对称秘钥管理

SM4秘钥下载和加解密，HMAC等功能接口

### 调用类

SymmetricKeyManager类

**构造方法:** public SymmetricKeyManager(Context context)

**成员变量:** 无

### 生成新的对称秘钥

#### 功能描述

在MAuth服务器上生成一个新的对称秘钥

#### 前置条件

已经下载证书

#### 方法原型

public void genSymmKey(String userName, String clientID, ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| clientID | String | 客户端ID | 是 |
| resultListener | Result.ResultListener | 回调接口 | 是 |

#### 返回值

通过回调接口返回生成结果

### 获取DEKID

#### 功能描述

获取上一次在服务器上生成的DEKID

#### 前置条件

已经生成了协同对称秘钥DEK

#### 方法原型

public String getDEKID(@NonNull String userName, @NonNull String clientID)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| clientID | String | 客户端ID | 是 |

#### 返回值

String类型的DEKID

### 加密数据

#### 功能描述

使用对称秘钥加密业务数据

#### 前置条件

已经生成了DEK

#### 方法原型

public void encryptData(String userName, String PIN, String clientID, @NonNull String dekID, int mode, int padding, byte[] plainBytes, Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| PIN | String | 证书的PIN码 | 是 |
| clientID | String | 客户端ID | 是 |
| dekID | String | 秘钥ID | 是 |
| mode | int | 加密模式(1-ECB, 2-CBC) | 是 |
| padding | int | 补位方式(0-NoPadding, 1-PKCS5Padding)，NoPadding时原文数据长度必须为16整数倍 | 是 |
| plainBytes | byte[] | 原文数据 | 是 |
| resultListener | Result.ResultListener | 回调接口 | 是 |

#### 返回值

通过回调接口返回加密结果

### 解密数据

#### 功能描述

使用对称秘钥解密加密密文

#### 前置条件

已经生成了DEK

#### 方法原型

public void decryptData(String userName, String PIN, String clientID, String dekID， int mode, int padding, byte[] cipherBytes, Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| PIN | String | 证书的PIN码 | 是 |
| clientID | String | 客户端ID | 是 |
| dekID | String | 秘钥ID | 是 |
| mode | int | 加密模式(1-ECB, 2-CBC) | 是 |
| padding | int | 补位方式(0-NoPadding, 1-PKCS5Padding)，NoPadding时原文数据长度必须为16整数倍 | 是 |
| cipherBytes | byte[] | 密文数据 | 是 |
| resultListener | Result.ResultListener | 回调接口 | 是 |

#### 返回值

通过回调接口返回解密结果

### 对数据做Hmac

#### 功能描述

使用对称秘钥对业务数据做Hmac摘要

#### 前置条件

已经生成了DEK

#### 方法原型

public void hmacData(String userName, String PIN, String clientID, String dekID， String hashAlg, byte[] plainBytes, Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| PIN | String | 证书的PIN码 | 是 |
| clientID | String | 客户端ID | 是 |
| dekID | String | 秘钥ID | 是 |
| hashAlg | String | 摘要算法("SM3","MD5", "SHA1", "SHA256", "SHA512") | 是 |
| plainBytes | byte[] | 原文数据 | 是 |
| resultListener | Result.ResultListener | 回调接口 | 是 |

#### 返回值

通过回调接口返回Mac结果

## 集成应用管理

该功能主要是为集成应用提供版本检查

### 调用类

AppManager类

**构造方法:** public AppManager(Context context)

**成员变量:** 无

### APP版本检查

#### 功能描述

根据当前集成应用的版本号，提供MAuth服务器上是否有新版本的信息

#### 前置条件

已经成功集成SDK

#### 方法原型

public void checkIntegratorVersion(final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| resultListener | Result.ResultListener | 回调接口 | 是 |

#### 返回值

通过回调接口返回，如果有新版本，会在ResultDesc中返回下载URL

## 下载管理接口

提供从MAuth服务器下载文件的接口

### 调用类

DownloadManager类

**构造方法:** public DownloadManager()

**成员变量:** 无

### 执行下载

#### 功能描述

从MAuth服务器下载一个文件

#### 前置条件

已经成功集成SDK

#### 方法原型

public void download(String url, String destFileName, File destDirectory, NetworkInterface.NetworkDownloadCallback downloadCallback)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| url | String | 要下载的文件地址 | 是 |
| destFileName | String | 目标文件名称 | 是 |
| destDirectory | String | 目标文件存储的目录 | 是 |
| downloadCallback | NetworkDownloadCallback | 回调接口 | 是 |

#### 返回值

通过NetworkDownloadCallback 回调接口返回下载的结果

## HTTPS接口

HTTPS接口支持单双向RSA算法和国密的HTTPS连接，通过InfosecHttpsURLConnection类实现，该类是HttpURLConnection的子类，故可以直接使用HttpURLConnection的API，InfosecHttpsURLConnection中额外扩展的方法如下。

### 构造方法

#### 功能描述

创建一个InfosecHttpsURLConnection对象

#### 前置条件

已经成功下载证书

#### 请求示例

InfosecHttpsURLConnection(@NonNull URL url)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| url | URL | 要连接的URL | 是 |

#### 返回值

构造方法无返回值

### 设置信任证书

#### 功能描述

为该InfosecHttpsURLConnection对象设置用于信任对端服务器证书的信任证书

#### 前置条件

已经成功集成SDK

#### 请求示例

public void setTrustCert(@NonNull String trustCert)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| trustCert | String | 设置信任证书，支持文件路径 | 是 |

#### 返回值

无

### 设置客户端证书(非SDK内部证书)

#### 功能描述

为该InfosecHttpsURLConnection对象设置客户端签名证书和加密证书，并设置私钥的保护密码，可以支持文件路径格式

#### 前置条件

已经成功集成了SDK

#### 请求示例

public void setClientCert(@NonNull String signCert, @NonNull String signKey, String encCert, String encKey, String keyPass)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| signCert | String | 签名公钥证书 | 是 |
| signKey | String | 签名私钥 | 是 |
| encCert | String | 加密公钥证书(RSA算法没有可为null) | 否 |
| encKey | String | 加密私钥(RSA算法没有可为null) | 否 |
| keyPass | String | 私钥保护口令 | 否 |

#### 返回值

无

### 设置客户端证书(SDK内部证书)

#### 功能描述

为该InfosecHttpsURLConnection对象设置一个SDK内部已经下载的客户端证书

#### 前置条件

SDK已经成功下载了证书,并且是非协同签名模式

#### 请求示例

public void setClientCert(@NonNull String certAlias, @NonNull String certPIN)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| certAlias | String | 证书别名 | 是 |
| certPIN | String | 证书PIN | 是 |

#### 返回值

无

## SSLSocket接口

SSLSocket接口支持单双向RSA算法和国密的Socket连接，通过InfosecSocket类实现，该类是Socket的子类，故可以直接使用Socket的API，InfosecSocket中额外扩展的方法如下。

### 构造方法

#### 功能描述

可以直接创建一个InfosecSocket对象

#### 前置条件

已经成功下载了证书

#### 请求示例

public InfosecSocket(@NonNull String ip, int port)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| ip | String | 要连接的主机IP | 是 |
| port | int | 要连接的主机端口 | 是 |

#### 返回值

构造方法无返回值

### 设置信任证书

#### 功能描述

为该InfosecHttpsURLConnection对象设置用于信任对端服务器证书的信任证书

#### 前置条件

已经成功集成SDK

#### 请求示例

public void setTrustCert(@NonNull String trustCert)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| trustCert | String | 设置信任证书，支持文件路径 | 是 |

#### 返回值

无

### 设置客户端证书(非SDK内部证书)

#### 功能描述

为该InfosecHttpsURLConnection对象设置客户端签名证书和加密证书，并设置私钥的保护密码，可以支持文件路径格式

#### 前置条件

已经成功集成了SDK

#### 请求示例

public void setClientCert(@NonNull String signCert, @NonNull String signKey, String encCert, String encKey, String keyPass)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| signCert | String | 签名公钥证书 | 是 |
| signKey | String | 签名私钥 | 是 |
| encCert | String | 加密公钥证书(RSA算法没有可为null) | 否 |
| encKey | String | 加密私钥(RSA算法没有可为null) | 否 |
| keyPass | String | 私钥保护口令 | 否 |

#### 返回值

无

### 设置客户端证书(SDK内部证书)

#### 功能描述

为该InfosecHttpsURLConnection对象设置一个SDK内部已经下载的客户端证书

#### 前置条件

已经成功下载了证书

#### 请求示例

public void setClientCert(@NonNull String certAlias, @NonNull String certPIN)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| certAlias | String | 证书别名 | 是 |
| certPIN | String | 证书PIN | 是 |

#### 返回值

无

### 建立连接

#### 功能描述

建立一个到对端主机的连接通道

#### 前置条件

已经成功集成SDK

#### 请求示例

public boolean connect()

#### 请求参数

无

#### 返回值

连接成功返回true，连接失败返回false

## 用户管理接口

用户管理接口主要包含用户创建，签约，锁定解锁和用户信息获取等操作

### 调用类

UserManager类

**构造方法:** public UserManager(Context context)

**成员变量:** 无

### 用户签约

#### 功能描述

使用用户名和签约码连接MAuth服务器进行签约认证

#### 前置条件

已经成功集成SDK

#### 请求示例

public static void signUp(String userName,String signCode,Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| signCode | String | 签约码 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 用户创建

#### 功能描述

在MAuth服务器上创建一个新用户

#### 前置条件

已经成功集成SDK

#### 请求示例

public void newUser(String userName, String realName, String mobileNo, String emailAddr, int regCodeFlag, int userType, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 是 |
| realName | String | 用户姓名 | 是 |
| mobileNo | String | 手机号码 | 否 |
| emailAddr | String | 邮箱地址 | 否 |
| regCodeFlag | int | 注册码发送方式  1:发送到邮箱  2:发送到手机  3:不发送  其他:直接返回,格式为(uid:regCode) | 是 |
| userType | int | 用户类型  0:证书用户  1:令牌用户  2:证书&令牌用户 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 重新获取注册码

#### 功能描述

获取MAuth服务器上已经存在用户的新的注册码,用于进行用户注册

#### 前置条件

已经成功集成SDK

#### 请求示例

public void getSignCode(@NonNull String userName, @NonNull String uid, int regCodeFlag, @NonNull final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| userName | String | 用户名 | 否 |
| uid | String | 用户创建ID,由用户注册接口返回 | 否 |
| regCodeFlag | int | 注册码发送方式  1:发送到邮箱  2:发送到手机  其他:不发送也不返回注册码 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 获取用户状态

#### 功能描述

获取当前用户在MAuth服务器上的状态信息

#### 前置条件

已经成功集成SDK

#### 请求示例

public void getStatusOnLine(final String userName, final Result.ResultListener resultListener)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |
| resultListener | Result.ResultListener | 结果回调接口 | 是 |

#### 返回值

通过Result.ResultListener接口进行回调

### 删除用户

#### 功能描述

删除用户在本地的数据信息，包括令牌和证书信息

#### 前置条件

已经成功集成SDK，并且成功签约用户

#### 请求示例

public boolean deleteUser(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |

#### 返回值

删除成功返回true，否则返回false

### 判断用户是否已经锁定

#### 功能描述

判断用户是否已经被锁定

#### 前置条件

已经成功集成SDK，并且成功签约用户

#### 请求示例

public boolean isLocked(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |

#### 返回值

已经锁定返回true，否则返回false

### 判断用户是否是证书用户

#### 功能描述

判断用户是否是证书用户

#### 前置条件

已经成功集成SDK，并且成功签约用户

#### 请求示例

public boolean isCertUser(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |

#### 返回值

是证书用户返回true，否则返回false

### 判断用户是否是令牌用户

#### 功能描述

判断用户是否是令牌用户

#### 前置条件

已经成功集成SDK，并且成功签约用户

#### 请求示例

public boolean isTokenUser(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |

#### 返回值

是令牌用户返回true，否则返回false

### 获取用户的签约码

#### 功能描述

获取用户签约时的签约码

#### 前置条件

已经成功集成SDK，并且成功签约用户

#### 请求示例

public String getSignCode(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |

#### 返回值

String 类型的签约码，没有获取到返回null

### 获取用户的邮箱地址

#### 功能描述

获取用户签约时的邮箱地址,如果有

#### 前置条件

已经成功集成SDK，并且成功签约用户

#### 请求示例

public String getEmail(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |

#### 返回值

String 类型的邮箱地址，没有获取到返回null

### 获取用户的手机号

#### 功能描述

获取用户的手机号

#### 前置条件

已经成功集成SDK，并且成功签约用户

#### 请求示例

public String getMobileNo(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |

#### 返回值

String 类型的手机号，没有获取到返回null

### 获取用户的姓名

#### 功能描述

获取用户的姓名

#### 前置条件

已经成功集成SDK，并且成功签约用户

#### 请求示例

public String getRealName(String userName)

#### 请求参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数** | **类型** | **说明** | **非空要求** |
| username | String | 用户名 | 是 |

#### 返回值

String 类型的姓名，没有获取到返回null

# 基础回调接口

## 异步方法回调接口Result和ResultListener

### 异步方法定义

所有需要传入Result.ResultListener接口才能获取到的返回值的方法称为异步方法，之所以定义为异步方法，是因为这些方法需要进行联网或者是其他耗时操作，在Android应用开发规范中，这些任务需要执行在工作线程，所以必须需要有一个异步回调来接收任务执行的返回值。

### 异步方法实现

集成应用在调用异步方法时需要传入一个ResultListener的子类，并实现其中的handleResult方法接收方法的执行结果。

### 异步方法返回值获取

Result对象中有两个成员变量:resultID和resultDesc，调用程序可以使用getter方法获取其对应的值。其中resultID对应附录1中的常量字段，resultDesc存储的是对resultID常量的语言化描述，一般情况下，resultID和resultDesc是一一对应的，但是在某些需要返回给调用程序具体结果的接口中(例如获取令牌状态，证书状态，签名等接口)，resultDesc中存储的是要返回的具体参数，更具体的返回内容将会在下面章节说明

## 网络接口NetworkInterface和NetworkCallback

### 网络接管模式

在SDK中，所有与MAuth连接的网络通信操作，都被抽象为NetworkInterface接口，通过定义这个抽象接口，集成应用可以通过调用[SDK初始化(网络接管模式)](#_SDK初始化(网络接管模式)_1)接口自定义与MAuth通信的过程处理类，SDK会把所有的请求数据以及要请求的URL地址进行包装，由集成应用指定的NetworkInterface实现类负责网络请求和回复，这种方式，称为网络接管。网络接管特别适合集成应用有自己的独立安全通信渠道的场景。

### NetworkCallback

该回调定义在NetworkInterface类中，NetworkInterface的实现类在实现抽象方法时，也必须传递一个NetworkCallback的实现类的实例，该接口负责将网络请求的返回数据回调到该请求的调用方法中。

### NetworkDownloadCallback

该回调接口继承自NetworkCallback，用在NetworkInterface的下载方法中，如果集成应用实现了下载的方法，就需要传入这个接口的实例，来接收下载的进度以及下载结果。

# SDK集成说明

本章节说明了开发商集成应用(以下简称集成应用)对信安移动安全中间件SDK的集成步骤。

## 引入Java依赖库

将产品目录中LIB(S)目录下的IMSSDK\_LIBS开头的压缩包解压，将其中的aar文件引入到项目中，aar依赖包的具体引入方法属于Android开发的技术技能，在这里不再赘述，对此不熟悉的开发人员可以参考[这篇文章](http://jingyan.baidu.com/article/2a13832890d08f074a134ff0.html)或者是[Android官方](https://developer.android.com/studio/projects/android-library.html)中的《Add your library as a dependency》这一段。

## 引入JNI动态链接库

将产品目录中的jniLibs目录下的文件及文件夹拷贝到你的集成项目中的jniLibs目录下。

安全中间件SDK产品中提供了适用于多种CPU API架构的动态链接库文件，集成应用可以根据自己项目的实际需求，选择性的引入对应架构的动态链接库文件，其他不需要引入的文件可以不用拷贝进项目目录，这样既可以满足开发需求，又不至于过于扩大集成应用的安装包体积。例如，如果项目仅支持基础arm架构的设备，则可引入armeabi目录即可，如果需要支持x86架构的设备，则引入x86架构文件，其他架构文件依次类推。

## 添加配置信息

将以下权限添加到 AndroidManifest.xml 文件中:

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

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

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

<uses-permission android:name="android.permission.READ\_PHONE\_STATE"/>

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

<uses-permission android:name="android.permission.GET\_TASKS"/>

# SDK功能使用说明

## 基础功能

### SDK初始化

SDK初始化是SDK所有其他接口能正常使用的基础，在每次应用启动时,都必须尽早的调用该接口进行初始化操作，并且保证SDK初始化成功，这样才能保证可以正常使用IMSSDK中的所有功能。

SDK如果初始化失败，集成应用不应该使用SDK中的其他功能接口，此时集成应用应该退出，或者稍后再进行初始化。

我们建议在集成应用的第一个 Activity 或者Application类中或者是应用启动的第一个类中， 进行SDK初始化调用，在整个应用生命周期中只需要成功初始化一次即可，不需要重复调用该接口。[SDK初始化](#_SDK初始化)的参数列表如下:

IMSSdk.initialization(context，host，base64HttpsTrustCert，appID，appSecret，resultListener);

* Context：集成应用的应用Context句柄，建议传入ApplicationContext以保证SDK能尽可能的和集成应用的生命周期保持一致
* base64HttpsTrustCert：MAuth服务器的通信连接如果使用了HTTPS，则需要设置MAuth服务器站点信任证书，该参数要求是Base64格式的字符串。MAuth是否启用了HTTPS以及该参数的来源请咨询项目经理，我司的技术支持(SE)工程师，或者MAuth服务器端开发人员
* Host：MAuth服务器的通信参数，格式为“IP:PORT”，该参数的来源请咨询项目经理，我司的技术支持(SE)工程师，或者MAuth服务器端开发人员。
* appID，appSecret：MAuth分配给当前应用的应用ID和应用密钥，集成应用应该妥善保管该ID和密钥，防止泄露。该参数的来源请咨询项目经理，我司的技术支持(SE)工程师，或者MAuth服务器端开发人员。
* resultListener：初始化回调监听器，如果初始化成功，resultID为OPERATION\_SUCCEED，初始化失败则为其他值，后面所有的resultListener参数的回调结果都是如此，所以不会再标明出来详细说明，如果回调结果有特殊的个例，会标明出来进行解释。初始化失败，集成应用不应该使用SDK的任何功能接口，

### 用户签约

用户是集成应用通过SDK和MAuth服务器进行互动的基本单位，所有需要使用SDK功能的用户均需要在MAuth服务器上签约，只有经过签约的用户，才能使用SDK中的功能。

用户签约必须要有MAuth服务器签发的用户名和签约码，在整个用户的生命周期中，一台设备上的用户只需要成功签约一次即可。[用户签约](#_用户签约)的参数列表如下：

UserManager.signUp(userName，signCode，resultListener());

* userName：在MAuth服务器上生成的用户名，用来对用户进行标记。
* signCode：在MAuth服务器上生成的和用户名对应的签约码，用来对本次签约操作进行认证，签约码仅可以使用一次，签约成功即失效。

为了保证签约的安全性和合法性，用户名和签约码一般不会直接通过MAuth平台直接通过代码接口返回，通常会通过邮箱，短信，二维码或者其他线上线下方式传递到用户，或者由使用该系统的企业管理员统一分发，具体的生成和分发方案，请咨询项目经理，我司的技术支持(SE)工程师，或者MAuth服务器端开发人员。

### 用户创建

MAuth服务器上的用户来源有2中，自主注册和邀请注册：

* 邀请注册：指的是MAuth服务器上的用户是预先通过其他方式添加，导入的并且在用户签约之前已经存在了，并且已经预先生成了签约码。这种方式适用于已经有帐号系统并且需要复用的集成商。
* 自主注册：指的是截止到用户注册之前，MAuth服务器上没有该用户的信息。需要调用当前SDK的“用户创建”接口先进行用户添加，并获得签约码。这种方式适用于预先没有帐号系统，或者不需要复用帐号系统的集成商。

该用户创建接口仅仅适用于自主注册的情况，这两种注册情况如何选择和确定，请咨询项目经理，我司的技术支持(SE)工程师，或者MAuth服务器端开发人员。

[用户创建](#_用户创建)的参数列表如下：

UserManager.newUser(userName, realName, mobile, emailAddr, sendSMS, resultListener);

* userName：要注册的用户名
* realName，mobile，emailAddr：用户的姓名，手机号码，邮箱地址
* sendSMS：签约码是否通过短信发送，如果传true，mobile参数不能传空，并且签约码会发送到该手机号中。如果传false，emailAddr不能传空，并且签约码会发送到该邮箱地址中。

### 用户锁定和解锁

SDK的令牌和证书功能中，都需要使用PIN码对用户身份进行认证，每个模块的PIN码均有10次错误尝试次数，如果令牌或者证书的PIN码连续错误输入次数达到10次，就会被锁定，锁定后的用户无法再使用需要PIN码验证的接口。

集成应用可以调用[判断用户是否已经锁定](#_判断用户是否已经锁定)接口获取当前用户的锁定状态。

用户一旦被锁定，如果需要继续使用，需要调用[用户解锁](#_用户解锁)接口进行解锁，该接口的参数列表如下：

UserManager.unLockUser(userName，unLockCode，resultListener());

* userName：用户名
* unLockCode：解锁码，用户被锁定时在MAuth服务器生成，解锁码一般不会直接通过MAuth平台直接通过代码接口返回，通常会通过邮箱，短信，二维码或者其他线上线下方式传递到用户，或者由使用该系统的企业管理员统一分发，具体的生成和分发方案，请咨询项目经理，我司的技术支持(SE)工程师，或者MAuth服务器端开发人员。

### 用户状态获取

[用户状态获取](#_获取用户状态)接口用于获取当前用户在MAuth服务器上的状态，参数列表如下：

UserManager.getStatusOnLine(userName, resultListener)

* userName：用户名
* resultListener：如果返回成功，则在resultDesc字段返回一个JSONArray，对应的字段描述如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **参数类型** | **参数描述** | **备注** |
| userstate | String | 用户状态 | 1未注册；  2已注册； |
| certstate | String | 证书状态 | 1-未申请  3-有效  4-冻结  6-已作废 |
| islock | String | 用户锁定状态 | 0：未锁定  1：锁定 |
| netpassstate | String | 令牌状态 | 0：未申请；  1：冻结；  2：有效；  3：已作废； |
| certNotAfter | String | 证书有效期截止时间 | 证书已下载时返回 |
| netpassNotAfter | String | 令牌有效期截止时间 | 令牌种子已下载时返回 |

### 用户其他信息获取

包括：

1. [删除用户](#_删除用户)
2. [获取签约码](#_获取用户的签约码)
3. [获取手机号](#_获取用户的手机号)
4. [获取姓名](#_获取用户的姓名)

这些接口比较简单，集成应用开发者查看接口列表即可。

### 证书的申请和更新

数字证书是SDK中各种功能的基础，在进行签名，二维码Web登陆，PDF签章和OS扫码登陆之前，必须要先有一个数字证书。

可以使用[证书下载](#_下载证书)接口进行数字证书的下载，使用[证书更新](#_更新证书)接口进行数字证书更新。这两个接口的参数和返回值比较简单，不再赘述。

### 获取证书状态

该接口用于获取当前用户的数字证书在MAuth服务器上的状态，[获取证书状态](#_获取证书状态)接口的参数列表如下：

CertManager.getStatusOnLine(userName，resultListener);

该接口通过resultListener返回用户的证书状态，如果获取状态成功，其resultID和resultDesc字段可能返回以下的状态之一：

* Result.STATUS\_NOT\_ACQUIRED：证书未申请
* Result.STATUS\_EFFECTIVE：证书有效
* Result.STATUS\_LOCKED：证书被冻结
* Result.STATUS\_REVOKED：证书被作废

### 证书信息操作

这些接口包括：

1. [获取证书](#_获取证书)
2. [获取证书公钥](#_获取证书公钥)
3. [删除证书](#_删除一张证书)
4. [修改证书PIN](#_修改证书PIN)
5. [验证证书PIN](#_验证PIN)
6. [检查证书是否存在](#_检查证书是否存在)
7. [获取证书信息](#_获取证书信息)

这些接口比较简单，开发者自行查看接口列表即可。

### 证书功能操作

这些接口包括：

1. 使用证书进行数据的[加密](#_使用证书进行加密)和[解密](#_使用证书进行解密)
2. 使用证书[制作](#_使用证书制作数字信封)和[打开](#_使用证书打开数字信封)数字信封

需要注意的是，这些功能接口需要下载双证证书才可以调用成功，是否是双证证书可以在MAuth服务器的配置中查看，这里不再赘述。

### 令牌种子的申请和更新

动态令牌功能使用之前，必须要有种子，可以使用[种子下载](#_种子下载)和[种子更新](#_种子更新)接口进行种子操作。这两个接口的参数和返回值比较简单，不再赘述。

### 获取令牌状态

[获取令牌状态](#_获取令牌状态)的接口和获取证书状态的接口是类似的，参数列表如下：

TokenManager.getStatusOnLine(userName，resultListener);

该接口通过resultListener返回用户的令牌状态，如果获取状态成功，其resultID和resultDesc字段可能返回以下的状态之一：

* Result.STATUS\_NOT\_ACQUIRED：种子未申请
* Result.STATUS\_EFFECTIVE：令牌有效
* Result.STATUS\_LOCKED：令牌被冻结
* Result.STATUS\_REVOKED：令牌被作废

### 令牌信息操作

这些接口包括：

1. [修改令牌PIN](#_修改令牌种子PIN码)
2. [验证令牌PIN](#_验证令牌种子PIN码)
3. [检查本地种子是否存在](#_检查本地令牌种子是否存在)

这些接口比较简单，开发者可自行查看接口列表。

### 生成动态口令

如果令牌种子已经下载成功，则可以使用该接口进行动态口令的生成，生成动态口令的参数列表如下：

TokenManager.getOTP(userName，PIN，algorithm，time，challenge，digit，step);

* PIN：令牌的种子PIN码，用来进行身份验证，
* Algorithm：动态口令计算算法，可选”OATH“或者”GUOMI“
* Time：动态口令计算的时间因子，”OATH”算法使用毫秒值时间，”GUOMI“算法使用秒值
* Challenge：动态口令的挑战因子,如果要生成挑战口令,则需要传入具体的挑战值,不生成挑战口令可以传nulll
* Digit： 口令位数，可选8位或者6位
* step ：计算时间步长，单位为秒，可选60秒或者30秒

## 业务场景功能

### 数字签名认证

本章节描述了使用SDK中的数字证书进行数字签名，从而利用数字签名技术进行用户身份鉴别，数据防篡改，交易认证等业务场景。

#### 前置条件

已经完成了证书下载。

#### 参与成员

* 集成应用-APP
* 安全中间件-SDK，SDK本身被集成在集成应用APP中，这里为了使角色和流程清晰，单独列为一个成员
* 集成应用的业务服务器-Business
* MAuth服务器-MAuth

#### 流程图

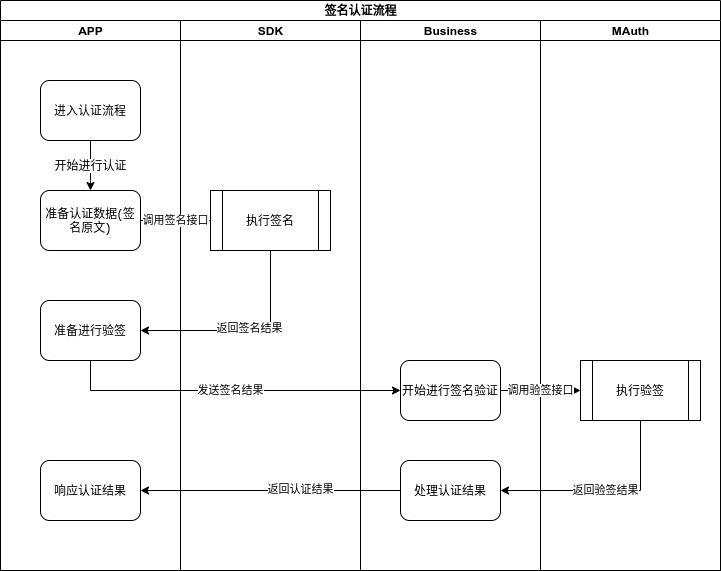


图6.2.1.3签名认证流程图

#### 流程说明

1. APP在进入认证流程的环节时，准备签名原文等数据，然后调用SDK中的[签名](#_签名)接口。
2. 签名接口在签名成功时，通过resultDesc字段，返回签名值，APP拿到该签名值之后，准备验签数据，通过调用Business或者直接调用MAuth进行验签，获得验签结果。
3. APP或者Business得到验签结果之后，处理本次的认证信息。
4. 在整个交互流程中，SDK仅参与了签名过程，后续的验签认证流程由APP，Business与MAuth完成。验签过程中的MAuth接口调用以及其他详细操作，请查看MAuth的文档或者咨询项目经理，我司的技术支持(SE)工程师，或者MAuth服务器端开发人员。

### 动态令牌认证

本章节描述了在业务场景中使用SDK中的动态令牌，结合MAuth服务器对用户或者交易进行认证的详细流程。

#### 前置条件

已经成功下载了令牌种子。

#### 参与成员

* 集成应用-APP
* 安全中间件-SDK，SDK本身被集成在集成应用APP中，这里为了使角色和流程清晰，单独列为一个成员
* 集成应用的业务服务器-Business
* MAuth服务器-MAuth

#### 流程图

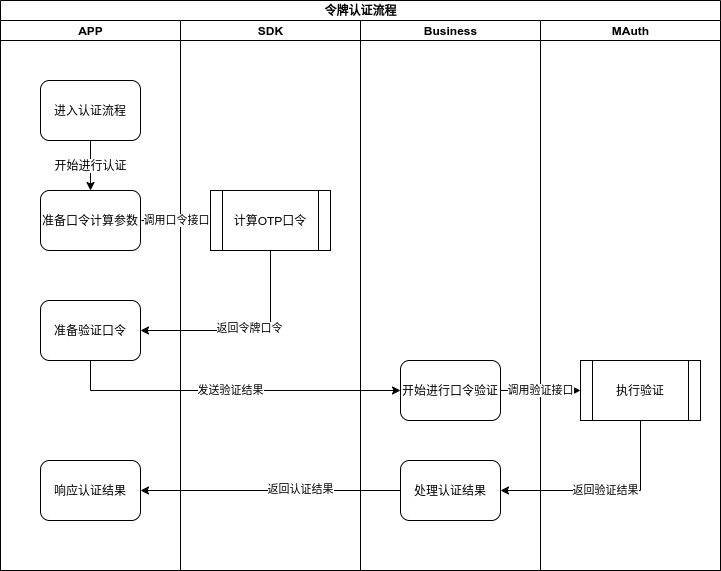


图6.2.2.3令牌认证流程图

#### 流程说明

1. APP在进入认证流程的环节时，准备要参与令牌计算的参数，然后调用SDK中的[生成令牌口令](#_生成令牌口令)接口。
2. 生成令牌口令接口计算成功时，返回一个String类型的动态口令，APP拿到该动态口令之后，加上用户信息和其他验证数据，通过调用Business或者直接调用MAuth进行口令的验证，获得验证结果。
3. APP或者Business得到验证结果之后，处理本次的认证信息。
4. 在整个交互流程中，SDK仅参与了口令计算过程，后续的口令验证流程由APP，Business与MAuth完成。验证过程中的MAuth接口调用以及其他详细操作，请查看MAuth的文档或者咨询项目经理，我司的技术支持(SE)工程师，或者MAuth服务器端开发人员。

### 二维码Web登录

该章节介绍了使用SDK配合MAuth服务器，利用数字签名认证技术实现二维码扫描登录Web服务器的场景和具体流程

#### 前置条件

已经成功下载了证书

#### 参与成员

* 集成应用-APP
* 安全中间件-SDK，SDK本身被集成在集成应用APP中，这里为了使角色和流程清晰，单独列为一个成员
* 与集成应用对接的Web服务器-Web
* MAuth服务器-MAuth

#### 流程图

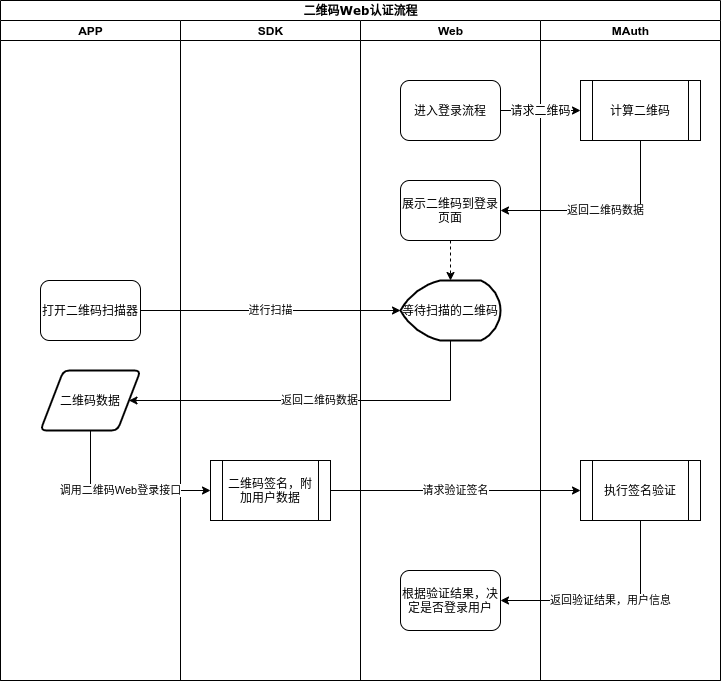


图6.2.3.3 二维码Web认证登录流程

#### 流程说明

1. Web服务在需要用户登录的时候，首先调用MAuth的获取二维码接口，该接口返回二维码数据。
2. Web服务将该二维码展示在登录页面等待移动端用户扫描。
3. 用户打开集成应用，扫描二维码得到二维码数据，并将该数据传递到SDK的[二维码Web登录](#_Web认证登录)接口。
4. SDK中该接口会将二维码数据进行签名，并附带用户数据，请求MAuth对签名进行验证，MAuth服务器验证通过，返回验证结果和用户信息到Web服务器
5. Web服务器根据验证结果和用户信息，决定是否的在Web服务器上登录该用户。过程中的MAuth接口调用以及其他详细操作，请查看MAuth的文档或者咨询项目经理，我司的技术支持(SE)工程师，或者MAuth服务器端开发人员。

### OS扫码登录

本章节描述了利用SDK配合MAuth服务器进行PC扫码登录操作系统的业务场景。

#### 前置条件

已经成功下载了证书

#### 参与成员

* 集成应用-APP
* 安全中间件-SDK，SDK本身被集成在集成应用APP中，这里为了使角色和流程清晰，单独列为一个成员
* Windows安全登录助手客户端-Client
* 安装了Client的PC设备-PC
* MAuth服务器-MAuth

#### 流程图

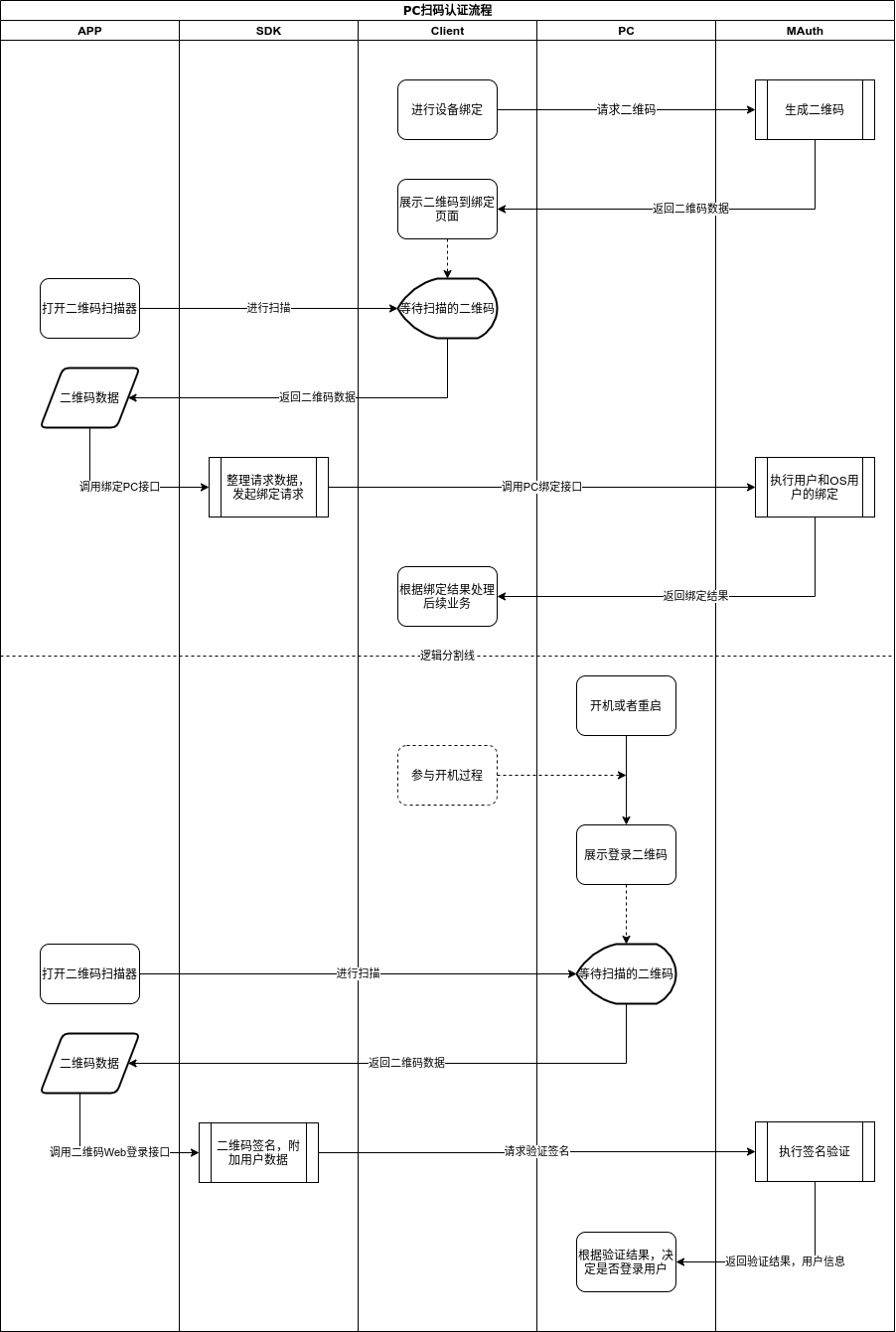


图6.2.4.3OS扫码认证登录流程

#### 流程说明

1. 如果Client上已经绑定过当前集成应用上的用户，则可跳过用户绑定步骤，从第5步直接开始执行扫码登录
2. Client开始进行用户绑定时，请求MAuth服务器生成二维码，Client展示该二维码，供集成应用扫描。
3. 集成应用扫描二维码，将二维码数据传送到SDK的[绑定PC设备接口](#_绑定PC设备)中。
4. SDK收到绑定数据后，整理请求发送到MAuth的绑定接口，进行PC用户和集成应用用户的绑定，至此绑定过程结束。
5. PC启动过程中，在Client的协助下，登录页面产生可用于扫描的登录二维码。
6. 集成应用扫描该二维码，传送到SDK的[二维码Web登录](#_Web认证登录)接口中。
7. SDK收到登录的二维码数据，对数据进行签名后，将用户数据和二维码数据发送到MAuth服务器请求签名验证。MAuth服务器验证通过，返回验证结果和用户信息到PC登录页面。
8. PC登录页面在Client的协助下，登录到操作系统当中，至此，扫码登录流程结束。

### PDF签章

本章节描述了集成应用如何使用SDK，在MAuth服务器的协助下，利用数字签章技术对PDF文件进行签章的的业务场景和流程。

#### 前置条件

已经成功下载了证书

#### 参与成员

* 集成应用-APP
* 安全中间件-SDK，SDK本身被集成在集成应用APP中，这里为了使角色和流程清晰，单独列为一个成员
* 集成应用的业务服务器-Business
* MAuth服务器-MAuth

#### 流程图

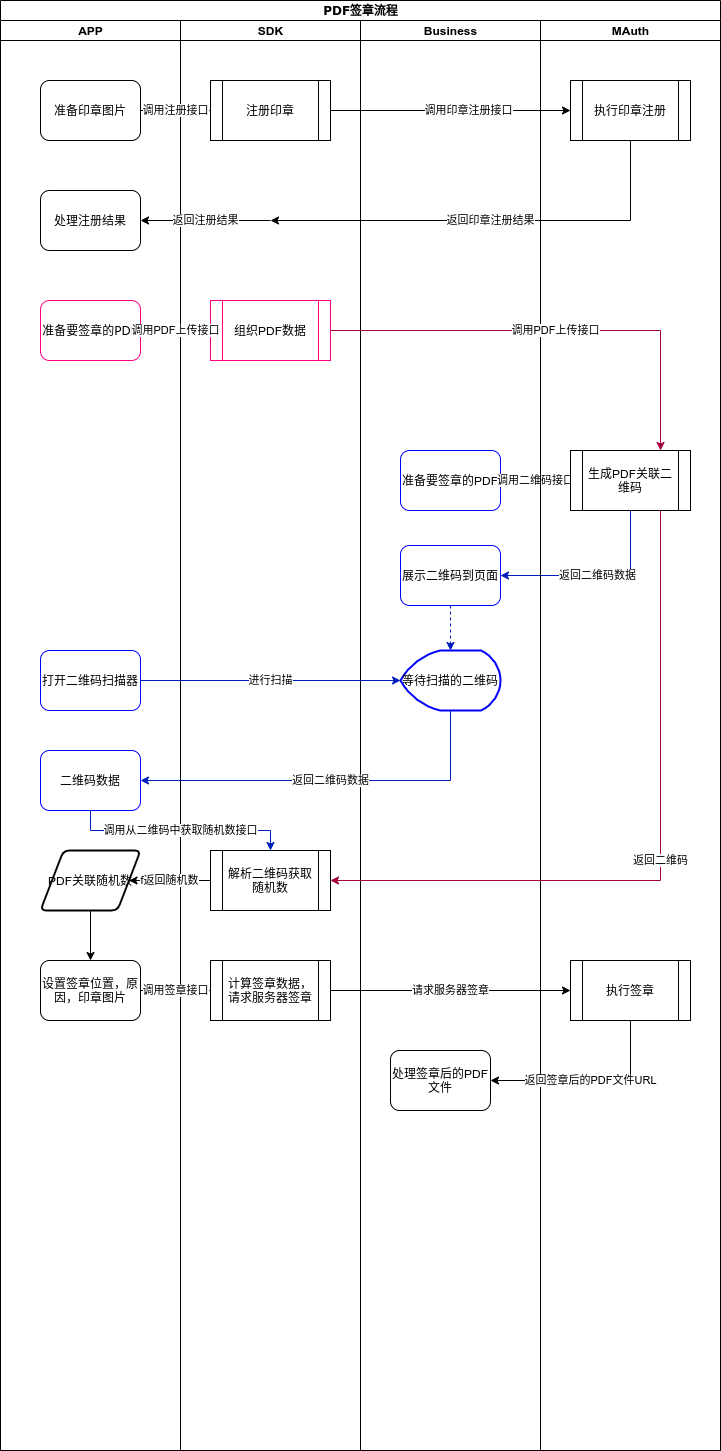


图6.2.5.3PDF签章流程

#### 流程说明

1. 如果要使用印章图片已经注册，可以跳过印章注册步骤，直接从第3步开始PDF签章流程
2. 集成应用准备要作为印章的图片，传入到SDK的[印章注册接口](#_印章图片注册)，该接口会将印章图片注册到MAuth服务器上，并返回印章注册结果。印章注册流程结束。
3. 进入PDF签章流程，PDF文件来源于服务器。
4. 流程图中的蓝色线框表示服务器PDF文件的流程，Business服务器将文件提交到MAuth服务器之后，会收到一个二维码。Business需要将该二维码展示到页面上，供集成应用扫描。
5. 集成应用扫描得到二维码数据之后，通过调用[从二维码中获取PDF关联随机数](#_从二维码中获取PDF关联随机数)接口解析出一个和PDF关联的随机数。如果集成应用需要下载展示该PDF文件，也可以通过调用[根据随机数获取PDF的下载地址](#_根据随机数获取PDF的下载地址)接口，传入随机数并得到PDF文件的下载路径。
6. 集成应用将随机数以及PDF签章的其他数据传入到[执行PDF签章](#_执行PDF签章)接口中，请求进行PDF签章，SDK计算签章数据之后，将数据发送到MAuth并请求MAuth服务器进行签章。
7. MAuth服务器签章完毕，返回签章后的PDF文件URL，集成应用或者Business可以根据实际需求决定是否下载PDF文件。至此，PDF签章流程结束。

### 对称数据加解密

本章节描述了集成应用如何使用SDK中的对称秘钥接口对业务数据或者报文数据进行加解密和Hmac计算，在MAuth服务器的协助下，可以对客户端到服务器之间的数据实现数据隐私保护和完整性计算。

#### 前置条件

已经成功下载了证书

#### 参与成员

* 集成应用-APP
* 安全中间件-SDK，SDK本身被集成在集成应用APP中，这里为了使角色和流程清晰，单独列为一个成员
* 集成应用的业务服务器-Business
* MAuth服务器-MAuth

#### 流程图

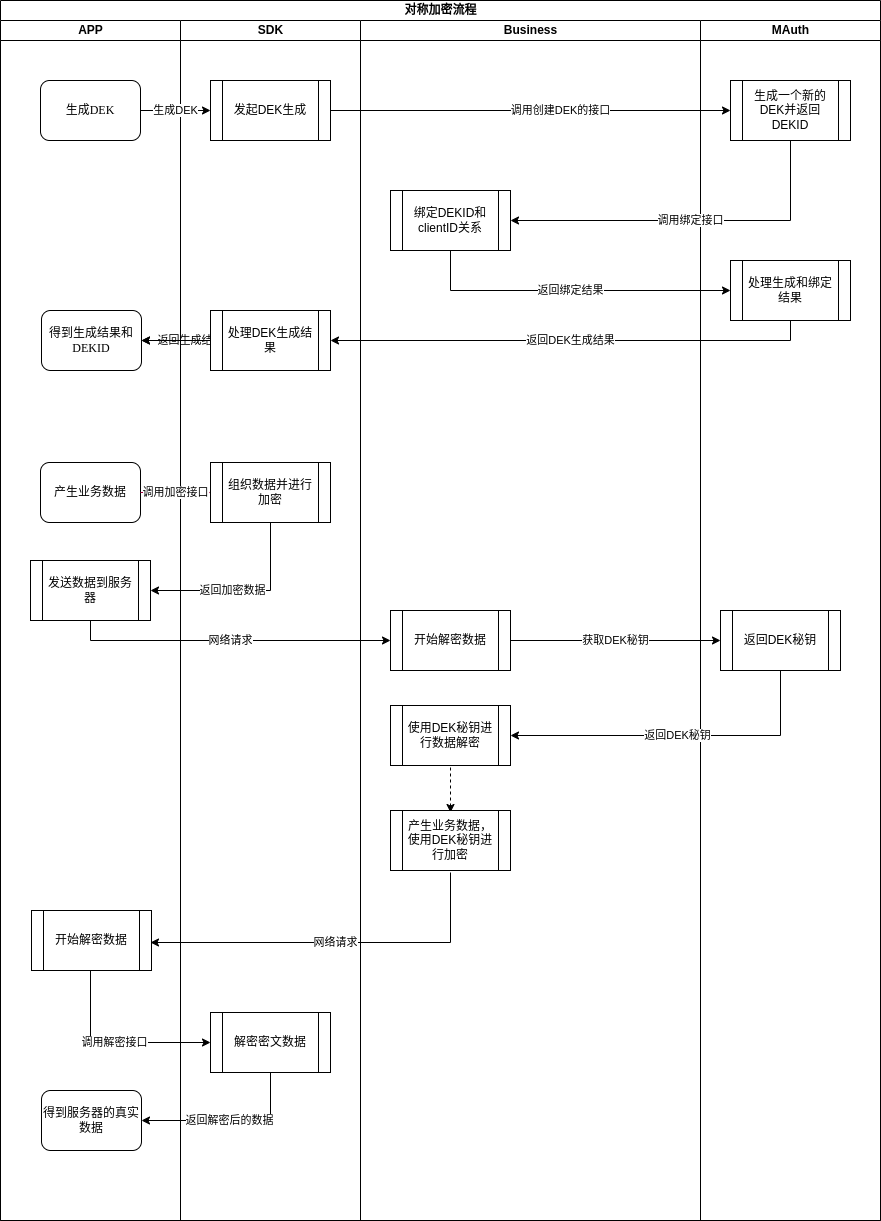


图6.2.6.3对称秘钥加解密功能

#### 流程说明

1. 如果当前用户已经生成过了DEK，可以跳过DEK生成步骤，直接从第3步开始加解密的流程，否则需要调用[生成新的对称秘钥](#_生成新的对称秘钥)接口，该接口会返回DEK是否生成成功的结果以及生成之后的DEKID。如图所示，生成新的DEK过程中MAuth服务器需要调用Business服务器进行绑定，所以业务服务器需要开放一个接口给MAuth调用
2. 集成应用可以随时调用[获取DEKID](#_获取DEKID)来获取之前是否已经生成过DEK，如果之前已经生成过了DEK则返回该DEK对应的DEKID，否则返回其他错误码
3. 档集成应用需要把业务数据加密发送到Business服务器时，将数据传入到SDK的[加密数据](#_加密数据)接口，该接口会将数据进行加密，并返回给集成应用。
4. 集成应用将密文发送给Business服务器之后，Business服务器需要从MAuth服务器上查询DEK秘钥。随后Business使用DEK秘钥对加密的数据进行解密得到原始的业务数据。
5. 当Business需要回复或者发送数据给客户端时，将业务数据使用DEK进行加密，随后通过网络传送至集成应用客户端侧，4和5步骤中涉及到的查询秘钥接口，加解密接口，由MAuth服务器提供。
6. 集成应用将得到的密文数据传入到[解密数据](#_解密数据)接口中，SDK会对数据进行解密，并返回解密后的数据。
7. 涉及到集成应用到Business之间的HMac数据完整性校验，和前面流程是一样的，只需要把原来调用的<加密数据>和<解密数据>接口替换为[对数据做Hmac](#_对数据做Hmac)接口即可。

### HTTPS功能

SDK中的HTTPS功能可以让集成应用使用HTTPS加密传输HTTP数据，提供服务的实现类为InfosecHttpsURLConnection，InfosecHttpsURLConnection类继承了HttpURLConnection，所以可以直接使用HttpURLConnection类的API进行HTTPS请求，目前，InfosecHttpsURLConnection仅提供GET和POST请求的实现。HttpURLConnection是Java和Android开发中常用的基础HTTP客户端API，如果您对HttpURLConnection还不太熟悉，请自行在网络上搜索学习HttpURLConnection的使用方法，这里不再赘述。下面是InfosecHttpsURLConnection类基于HttpURLConnection类扩展的接口以及使用流程和注意事项：

#### 设置信任证书

设置信任证书，需要使用：setTrustCert接口

信任证书用于验证服务器端证书，不论服务器证书是不是由公开信任CA颁发的，均需要调用该接口设置信任证书

详细说明参看[设置信任证书](#_设置信任证书)部分

#### 设置客户端证书

设置客户端证书，需要使用：setClientCert接口

该方法有两个重载方法，分别用于设置SDK内部已经下载好的证书，以及外部传入的证书参数，仅仅在需要执行双向HTTPS的时候，才需要设置客户端证书，所以请在调用之前明确你要执行的是单向还是双向HTTPS

详细说明参看[设置客户端证书(非SDK内部证书)](#_设置客户端证书(非SDK内部证书))和[设置客户端证书(SDK内部证书)](#_设置客户端证书(SDK内部证书))部分

#### 基本调用顺序

1. 创建InfosecHttpsURLConnection实例对象，调用[构造方法](#_构造方法)，并且需要传入要连接的URL地址
2. 设置信任证书，调用[设置信任证书](#_设置信任证书)接口，该接口支持传入CA根证书和证书链，参数是文件路径或者是证书的PEM内容均可
3. 设置客户端证书，如果使用的是外部客户端证书，调用[设置客户端证书(非SDK内部证书)](#_设置客户端证书(非SDK内部证书))，参数格式和信任证书接口要求一致。如果使用的是SDK下载的证书，调用[设置客户端证书(SDK内部证书)](#_设置客户端证书(SDK内部证书))接口，传入用户名和PIN码即可。
4. 使用标准的HttpURLConnection的API进行HTTP请求和响应操作操作

### SSLSocket功能

SDK中的SSLSocket功能可以支持集成应用使用基于Socket的安全传输协议进行安全通信，在SDK中的实现类是InfosecSocket，它继承了Socket，所以可以直接使用Socket类的API进行HTTPS请求。Socket是Java和Android开发中常用的套接字编程API，如果您对Java中的Socket编程还不太熟悉，请自行在网络上搜索学习Socket的使用方法，这里不再赘述。下面是InfosecSocket基于Socket类扩展接口以及使用流程和注意事项：

#### 设置证书

不论是信任证书的设置，还是客户端证书的设置，其中的相关要求和上面HTTPS中证书设置的要求是一样的，这里不再复述

#### 建立连接和关闭连接

InfosecSocket类的实例必须调用connect方法成功创建连接通道之后，再进行读取和写出，数据操作完毕之后，确定不再使用该InfosecSocket对象时，需要调用close方法关闭连接通道，否则有可能会造成内存泄露问题

InfosecSocket使用模拟非阻塞IO的模式进行数据的读取和写入，暂不支持阻塞IO

#### 数据IO和超时

Socket使用者应当自定义协议，用以指定读取数据数据的长度，或者数据终止位，需要强调的是，这并不是SDK的BUG或者设计问题，这是Socket编程的基础知识。

#### 基本调用顺序

1. 创建InfosecSocket对象，调用[构造方法](#_构造方法_1)接口，传入要连接的IP和端口
2. 设置信任证书，调用[设置信任证书](#_设置信任证书_1)接口，该接口支持传入CA根证书和证书链，参数是文件路径或者是证书的PEM内容均可
3. 设置客户端证书，如果使用的是外部客户端证书，调用[设置客户端证书(非SDK内部证书)](#_设置客户端证书(非SDK内部证书)_1)，参数格式和信任证书接口要求一致。如果使用的是SDK下载的证书，调用[设置客户端证书(SDK内部证书)](#_设置客户端证书(SDK内部证书)_1)接口，传入用户名和PIN码即可。
4. 调用[建立连接](#_建立连接)接口，创建到服务器端的连接。
5. 使用标准的Socket编程API进行数据收发，确定不再使用Socket时，调用标准的Socket的关闭Socket连接接口进行Socket的释放和关闭。

# 附录A 错误码

|  |  |  |
| --- | --- | --- |
| **常量** | **类型** | **说明** |
| 000000 | String | 本次操作成功 |
| IE1001 | String | 尚未进行用户注册 |
| IE1002 | String | P10创建失败 |
| IE1003 | String | 证书导入失败 |
| IE1004 | String | OTP种子导入失败 |
| IE1005 | String | 尚未下载证书 |
| IE1006 | String | 错误的二维码数据 |
| IE1007 | String | 签名失败 |
| IE1008 | String | JSON解析失败 |
| IE1009 | String | 网络不可用，无法联接服务器 |
| IE1010 | String | 证书PIN码错误 |
| IE1011 | String | 证书删除失败 |
| IE1012 | String | PIN修改失败 |
| IE1013 | String | 参数为空或者不合法 |
| IE1014 | String | 数据转换时发生异常 |
| IE1015 | String | 数据IO时发生异常 |
| IE1016 | String | 文件上传或者下载失败 |
| IE1017 | String | 没有新版本 |
| IE1018 | String | JNI库初始化失败 |
| IE1019 | String | 在线初始化失败 |
| IE1020 | String | 打开数字信封失败 |
| IE1021 | String | 制作数字信封失败 |
| IE1022 | String | 状态：未申请 |
| IE1023 | String | 状态：已生效 |
| IE1024 | String | 状态：已冻结 |
| IE1025 | String | 状态：已作废 |
| IE1026 | String | 用户签约失败 |
| IE1027 | String | 用户锁定失败 |
| IE1028 | String | 证书申请失败 |
| IE1029 | String | 获取状态失败 |
| IE1030 | String | 种子申请失败 |
| IE1031 | String | 随机数获取失败 |
| IE1032 | String | 签章注册失败 |
| IE1033 | String | 签章下载失败 |
| IE1034 | String | PDF文件下载失败 |
| IE1035 | String | PDF签章失败 |
| IE1036 | String | 二维码登录失败 |
| IE1037 | String | OS绑定失败 |
| IE1038 | String | 获取密码失败 |
| IE1039 | String | 获取PushID失败 |
| IE1040 | String | 获取PC列表失败 |
| IE1041 | String | 检查版本失败 |
| IE1042 | String | 签名验证失败 |
| IE1043 | String | 设备注册失败 |
| IE1044 | String | PIN验证失败 |
| IE1045 | String | 用户已经被锁定 |
| IE1046 | String | 生成OTP令牌失败 |
| IE1047 | String | 无效算法 |
| IE1048 | String | 无效的返回数据 |
| IE1049 | String | 用户解锁失败 |
| IE1050 | String | 当前系统不支持双证证书 |
| IE1051 | String | 图片大小超出限制 |
| IE1052 | String | 文件大小超出限制 |
| IE1053 | String | 创建用户失败 |
| IE1054 | String | 获取签约码失败 |
| IE1055 | String | 功能未获得授权 |
| IE1056 | String | 不是双证证书 |