统一密码服务支撑平台
集成开发手册

江苏意源科技有限公司

2025 年 4 月

目 录

1. 范围 5

2. 规范性引用文件 5

3. 接入说明 6

3.1. 接入流程 6

4. 身份认证服务 7

4.1. 身份认证服务集成开发说明 7
4.2. 组件安装 7
4.3. 初始化 7
4.4. 挑战随机数 8
4.4.1. 功能说明 8
4.4.2. 接口说明 8
4.5. 身份认证客户端组件 9
4.5.1. 功能说明 9
4.5.2. 接口说明 9
4.6. 身份认证服务端组件 10
4.6.1. 功能说明 10
4.6.2. 接口说明 10
4.6.3. 身份验证结果数据格式 11
4.7. 调用示例 12
4.7.1. 调用示例 12
4.7.2. 相关依赖 12
4.8. 错误码说明 13

5. 签名验签服务 14

5.1. 签名验签服务集成开发说明 14
5.2. 初始化 15
5.2.1. 获取指定应用的实例接口 15
5.3. 获取错误信息 15
5.3.1. 功能说明 15
5.3.2. 接口说明 16
5.4. 证书操作 16
5.4.1. 读取应用服务器证书 16
5.4.2. 获得证书信息 16
5.4.3. 验证证书有效性 18
5.5. 签名验签 19
5.5.1. 设置签名算法 19
5.5.2. 获得当前签名算法 19

5.5.3. 数字签名 20
5.5.4. 验证签名 20
5.5.5. 文件签名 21
5.5.6. 验证文件签名 21
5.5.7. 消息签名 22
5.5.8. 验证消息签名 22
5.5.9. 消息 Detach 签名 23
5.5.10. 验证消息 Detach 签名 23
5.5.11. XML 签名 24
5.5.12. XML 验签 24
5.5.13. XML Detach 签名 25
5.5.14. XML Detach 验签 25
5.5.15. HMAC 摘要 26
5.6. 调用示例 26
5.7. 错误码说明 28
6. 时间戳服务 30
6.1. 时间戳服务集成开发说明 30
6.2. 初始化 31
6.2.1. 创建实例 31
6.3. 时间戳服务 31
6.3.1. 申请时间戳 31
6.3.2. 验证时间戳 32
6.3.3. 解析时间戳 32
6.4. 错误码说明 33
6.5. 调用示例 34
7. 加解密服务 35
7.1. 加解密服务集成开发说明 35
7.2. 初始化 35
7.3. 加密 36
7.3.1. 功能说明 36
7.3.2. 接口说明 36
7.4. 解密 37
7.4.1. 功能说明 37
7.4.2. 接口说明 37
7.5. 调用示例 37
7.5.1. 调用示例 37
7.5.2. 相关依赖 38
7.6. 错误码说明 39
8. 协同签名服务 41

8.1.协同签名服务集成开发说明.....41
8.2.初始化.....41
8.3.回调接口说明.....43
8.3.1. YYCallback 接口说明.....43
8.3.2. YYCertStatusCallback 接口说明.....44
8.3.3. YYResultCallback 接口说明.....44
8.4.证书申请.....45
8.4.1. 功能说明.....45
8.4.2. 接口说明.....45
8.4.3. 示例说明.....45
8.5.证书下载.....46
8.5.1. 功能说明.....46
8.5.2. 接口说明.....46
8.5.3. 示例说明.....46
8.6.证书查询.....47
8.6.1. 功能说明.....47
8.6.2. 接口说明.....47
8.6.3. 示例说明.....47
8.7.修改 PIN 码.....48
8.7.1. 功能说明.....48
8.7.2. 接口说明.....48
8.7.3. 示例说明.....48
8.8 验证 PIN 码.....49
8.8.1. 功能说明.....49
8.8.2. 接口说明.....49
8.8.3. 示例说明.....49
8.9. 签名.....49
8.9.1. 功能说明.....49
8.9.2. 接口说明.....49
8.9.3. 示例说明.....50
8.10. 验证签名.....50
8.10.1. 功能说明.....50
8.10.2. 接口说明.....50
8.10.3. 示例说明.....51
8.11. 错误码说明.....51

1. 范围

本手册适用于指导商用密码应用建设与本平台进行相关安全接口接入，手册主要进行功能操作说明及接口定义。

2. 规范性引用文件

下列文件中的条款通过本规范的引用而成为本规范的条款。凡是标注日期的引用文件，其后所有的修改（不包括勘误的内容）或者修订版均不适用于本规范。凡是不标注日期的引用文件，其最新版本适用于本规范。

GB/T 39786-2021 《信息安全技术 信息系统密码应用基本要求》

GB/T 22239-2019 《信息安全技术 网络安全等级保护基本要求》

GB/T 0028-2014 《密码模块安全技术要求》

GM/T 0009-2023 《SM2 密码算法使用规范》

GM/T 0010-2023 《SM2 密码算法加密签名消息语法规范》

GM/T 0014-2023 《数字证书认证系统密码协议规范》

GM/T 0015-2023 《数字证书格式》

GM/T 0016-2023 《智能密码钥匙密码应用接口规范》

GM/T 0017-2023 《智能密码钥匙密码应用接口数据格式规范》

GM/T 0018-2023 《密码设备应用接口规范》

GM/T 0019-2023 《通用密码服务接口规范》

GM/T 0020-2023 《证书应用综合服务接口规范》

GM/T 0033-2023 《时间戳接口规范》

GM/T 0127-2023 《移动终端密码模块应用接口规范》

GM/T 0128-2023 《数据报传输层密码协议规范》

GM/T 0132-2023 《信息系统密码应用实施指南》

3. 接入说明

3.1. 接入流程

当应用系统需要接入平台调用服务时，首先需要在平台完成应用注册，填写应用名称、密评等级、应用地址、服务端口、厂商及对接人联系方式、访问策略等基本信息，选择需要调用的服务，提交平台管理员进行审核。待审核通过后，应用系统获取到需要调用的服务相关接口文档以及应用对应的 token 鉴权信息，依据该接口文档对应用进行配置后，将应用上线，即可成功调用服务。

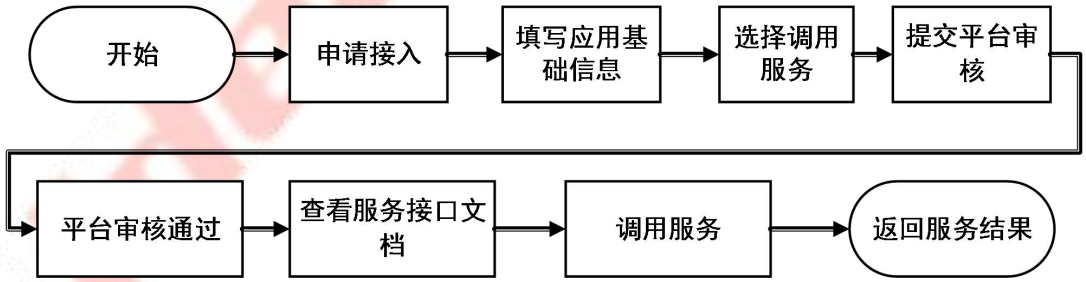

图 3-1 应用接入流程图

4. 身份认证服务

4.1. 身份认证服务集成开发说明

图 4-1 身份认证服务集成开发说明图

4.2. 组件安装

在应用系统对接时，客户端需要安装意源认证安全组件IBSapTrusAH，该服务端口为:TCP 31018，地址为:ws://127.0.0.1:31018。请求数据包格式为 Json，响应数据包格式为 Json。

4.3. 初始化

jar 包示例：

函数名：

AuthApi.getInstance().initParams(URL,appServerId,token)

功能说明： 初始化身份认证服务端调用参数。

输入参数：

URL：{camelPath}/services/testWS?wsdl（camelPath 由线路产生）

appServerId:请求方应用 ID，为 RMS 分配的资源标识

token:由服务平台分配应用后产生

备注：初始化只需要一次

4.4. 挑战随机数

4.4.1. 功能说明

向服务端请求挑战随机数。

4.4.2. 接口说明

表 4-1 挑战随机数接口说明表

接口名称	String AuthApi.getInstance().generateChallenge();		
接口参数	参数名称	类型	说明
			无参数
返回值	随机数	String	成功返回 32 位 BASE64 格式的随机数；失败抛出异常

4.5. 身份认证客户端组件

4.5.1. 功能说明

客户端用户和服务端进行基于国产数字证书的双向身份认证。

4.5.2. 接口说明

4.5.2.1. 请求报文

报文格式：

```
{
    "FuncName": "GetSignAndTokenByHttp",
    "Parames": {
        "url":
"{camelPath}/sap2000server?IDEABANK_AUTHORITY_TOKEN={token}",
        "svrRandom": "MTIzNDU2Nzg4MQ=="
    },
    "sessionID": "11111111"
}
```

表 4-2 身份认证接口请求报文说明表

参数名	取值	说明
FuncName	GetSignAndTokenByHttp	认证函数名固定
url	认证服务器 url 地址	camelPath 由线路产生,token 由应用分配, token 可在 URL 中传递也可在 header 中传递 header: {“IDEABANK_AUTHORITY_TOKEN”:XXXXXXXXXX}
svrRandom	随机数	服务器返回的 base64 编码的随机数
sessionID	uuid	前台产生的 uuid 编码

4.5.2.2. 返回成功报文

报文格式：

```
{
   "result": "success",
    "sessionID": "11111111",
    "Taken": "",
    "FuncName": " GetSignAndTokenByHttp"
}
```

表 4-3 身份认证接口返回成功报文说明表

参数名	取值	说明
result	success	字符串
sessionID	返回调用时传入的 uuid	字符串
Taken	票据	字符串
FuncName	GetSignAndTokenByHttp	字符串

4.5.2.3. 返回失败报文

报文格式：

```
{
    "result": "failed",
    "sessionID": "11111111",
    "ErrorCode": "",
    "FuncName": "GetSignAndTokenByHttp"
}
```

表 4-4 身份认证接口返回失败报文说明表

参数名	取值	说明
result	failed	字符串
sessionID	返回调用时传入的 uuid	字符串
ErrorCode	具体错误代码及内容	字符串
FuncName	GetSignAndTokenByHttp	字符串

4.6. 身份认证服务端组件

4.6.1. 功能说明

验证客户端认证结果并鉴权。

4.6.2. 接口说明

表 4-5 身份验证接口说明表

接口名称	VerifyIdTRet AuthApi.getInstance().doVerifyIdentityTicket(String challenge, String ticket);		
接口参数	参数名称	类型	说明
	challenge	String	应用服务器向应用客户端发起的随机数挑战值，应用服务器通过调用 generateChallenge 函数产生

	ticket	String	客户端通过身份认证客户端组件认证结果（身份票据和签名值）
返回值	VerifyIdTRet	Class 对象	见表 4-6 VerifyIdTRet 对象类说明

表 4-6 VerifyIdTRet 对象类说明表

对象名称	Class VerifyIdTRet
	{
	String result;
	String signature;
	}		

对象参数	参数名称	类型	说明
	result	String	认证服务器返回的结果，其格式详见身份验证结果数据格式表
	signature	String	返回结果的签名值 Base64 编码，用于验证结果的正确性和完整性

4.6.3. 身份验证结果数据格式

表 4-7 身份验证数据格式表

节点	父节点	类型	说明
verifyIdentityTicketResult	无	根节点	
version	verifyIdentityTicketResult	节点属性	验证结果版本号
result	verifyIdentityTicketResult	节点	验证结果
challenge	verifyIdentityTicketResult	节点	挑战随机数
error	verifyIdentityTicketResult	节点	验证失败时的错误信息
userinfo	verifyIdentityTicketResult	节点	令牌持有者的用户信息
rmsid	userinfo	节点	RMS 注册的用户编号
name	userinfo	节点	用户名信息
cert	userinfo	节点	用户证书
certSN	userinfo	节点	证书序列号
organization	userinfo	节点	组织

ou	userinfo	节点	单位
Email	userinfo	节点	电子邮箱
account	userinfo	节点	用户在应用系统中的账号
password	userinfo	节点	用户在应用系统中的密码
issuerinfo	verifyIdentityTicketResult	节点	身份票据签发者的证书
cert	issuerinfo	节点	身份签发者证书信息
time	verifyIdentityTicketResult	节点	验证身份时间
signature	verifyIdentityTicketResult	节点	签发“验证身份票据结果”的签名算法信息
oid	signature	节点属性	签名算法 OID
desc	signature	节点属性	签名算法描述信息

4.7. 调用示例

4.7.1. 调用示例

```
AuthApi.getInstance().initParams("http://172.21.68.71:8282/HTTP/YCVCWI7CO/services/testWS?wsdl","ff8080816cfa5154016cfa52bba50003","E7QL5QFYTBP");
AuthApi.getInstance().generateChallenge();
AuthApi.getInstance().doVerifyIdentityTicket("随机数 base64","票据");
```

4.7.2. 相关依赖

```
<dependencies>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-text</artifactId>
        <version>1.8</version>
    </dependency>
    <dependency>
        <groupId>dom4j</groupId>
        <artifactId>dom4j</artifactId>
        <version>1.6.1</version>
    </dependency>
    <dependency>
    <groupId>com.google.code.gson</groupId>
       <artifactId>gson</artifactId>
    <version>2.8.7</version>
</dependency>
```

```
    <dependency>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>httpclient</artifactId>
        <version>4.5.3</version>
    </dependency>
</dependencies>
```

4.8. 错误码说明

表 4-8 身份认证服务错误码说明表

序号	错误码	说明
1	0x00000001	票据中的证书与会话中的证书不匹配
2	0x00000002	票据中颁发者证书被破坏
3	0x00000003	票据中颁发者证书不受信任
4	0x00000004	票据中颁发者证书被吊销
5	0x00000005	票据中用户证书被破坏
6	0x00000006	票据中用户证书被吊销
7	0x00000007	票据中用户证书不可信
8	0x00000008	不在票据有效期内
9	0x00000009	不支持的证书类型
10	0x0000000a	未知证书错误
11	0x0000000b	非法参数
12	0x0000000c	内部错误
13	0x0000000d	票据签名值验证不通过
14	0x0000000e	客户端签名验证不通过
15	0x0000000f	用户不存在
16	0x00000010	用户被停用
17	0x00000011	应用服务器资源未注册
18	0x00000012	应用服务器资源被停用
19	0x00000013	用户无授权访问
20	0x00000014	授权时间不满足
21	0x00000015	参数丢失
22	0x00000016	随机数不是 base64
23	0x00000017	票据不是 base64
24	0x00000018	随机数长度错误

5. 签名验签服务

5.1. 签名验签服务集成开发说明

图 5-1 签名验签服务集成开发说明图

jar 包示例：

sof_url_jar.jar

5.2. 初始化

5.2.1. 获取指定应用的实例接口

5.2.1.1. 功能说明

初始化接口，通过 url，索引号，私钥授权码，容器名，策略名获取实例，Url 对应签名验签的 HTTP 服务，应用别名关联所配置的证书、密钥、信任证书链、算法类型、CRL 及证书验证策略等。

5.2.1.2. 接口说明

表 5-1 获取指定应用的实例接口说明表

接口名称	SOFInterface SOF_getAppInstance(String url, String keyIndex, String keyValue, String containerName, String policyname);		
接口参数	参数名称	类型	说明
	url	String	HTTP 接口服务地址
	keyIndex	String	索引号
	keyValue	String	私钥授权码
	containerName	String	容器名
	policyName	String	应用策略名称
返回值	SOFInterface	Class 对象	返回 SOFInterface 实体类

5.3. 获取错误信息

5.3.1. 功能说明

获取错误信息。

5.3.2. 接口说明

表 5-2 获取最新错误信息接口说明表

接口名称	long SOF_ getLastError ();		
接口参数	参数名称	类型	说明
			无
返回值	错误代码	long	详见错误说明

5.4. 证书操作

5.4.1. 读取应用服务器证书

5.4.1.1. 功能说明

根据证书用途获取当前应用服务器的签名证书或加密证书。

5.4.1.2. 接口说明

表 5-3 读取应用服务器证书接口说明表

接口名称	String SOF_getServerCertificateByUsage(short certUsage);		
接口参数	参数名称	类型	说明
	certUsage	short	证书用途，1:加密证书;2:签名证书
返回值	非空	String	Base64 编码的服务器证书
	空值	String	失败

5.4.2. 获得证书信息

5.4.2.1. 功能说明

获取证书信息。

5.4.2.2. 接口说明

表 5-4 获得证书信息接口说明表

接口名称	String SOF_getCertInfo(String base64EncodeCert, short type);		
接口参数	参数名称	类型	说明
	base64EncodeCert	String	Base64 编码的数字证书
	type	short	获取信息的类型，Type 参数见表 5-5 证书信息解析说明表
返回值	非空	String	证书内指定类型的信息
	空值	String	失败

表 5-5 证书信息解析说明表

序号	标识符	描述
1	0x00000001	证书版本
2	0x00000002	证书序列号
3	0x00000005	证书颁发者信息
4	0x00000006	证书有效期
5	0x00000007	证书拥有者信息
6	0x00000008	证书公钥信息
7	0x00000009	证书扩展项信息
8	0x00000011	颁发者密钥标识符
9	0x00000012	证书持有者密钥标识符
10	0x00000013	密钥用途
11	0x00000014	私钥有效期
12	0x00000015	证书策略
13	0x00000016	策略映射
14	0x00000017	基本限制
15	0x00000018	策略限制
16	0x00000019	扩展密钥用途
17	0x0000001A	CRL 发布点
18	0x0000001B	Netscape 属性
19	0x0000001C	私有的自定义扩展项
20	0x00000021	证书颁发者 CN
21	0x00000022	证书颁发者 O
22	0x00000023	证书颁发者 OU
23	0x00000031	证书拥有者信息 CN
24	0x00000032	证书拥有者信息 O
25	0x00000033	证书拥有者信息 OU
26	0x00000034	证书拥有者信息 EMAIL
27	0x00000035	证书起始日期
28	0x00000036	证书截至日期

29	0x00000081	证书 TITLE

5.4.3. 验证证书有效性

5.4.3.1. 功能说明

根据应用的策略根据验证证书有效性。

5.4.3.2. 接口说明

表 5-6 验证证书有效性接口说明表

接口名称	long SOF_validateCert(String base64EncodeCert) ;		
接口参数	参数名称	类型	说明
	base64EncodeCert	String	Base64 编码的数字证书
返回值	1	long	验证成功，证书有效
	其他	long	失败 -1 证书不被信任 -2 超过有效期范围 -3 证书已作废 -4 证书已冻结 -5 证书未生效 -6 false 错误
备注	基本的证书验证策略应包括 a） 验证 CA 信任列表，各层都进行签名和有效期验证 b） 各层证书的有效期 c） 各层证书的吊销状态，在特殊情况下（如：网络条件不允许），证书的吊销状态可采取灵活方式，有应用系统各层内部维护一个吊销列表，在证书登录认证时应用该吊销列表。验证证书有效性也可以采采取代理验证方式。		

5.5. 签名验签

5.5.1. 设置签名算法

5.5.1.1. 功能说明

设置接口在签名运算时使用的签名算法。

5.5.1.2. 接口说明

表 5-7 设置签名算法接口说明表

接口名称	void SOF_setSignMethod(long signMethod);		
接口参数	参数名称	类型	说明
	signMethod	long	签名算法标识（0x00020201）
返回值			无

5.5.2. 获得当前签名算法

5.5.2.1. 功能说明

获得控件签名使用的签名算法。

5.5.2.2. 接口说明

表 5-8 获得当前签名算法接口说明表

接口名称	long SOF_getSignMethod();		
接口参数	参数名称	类型	说明
			无
返回值	非 0	long	当前接口使用的签名算法预定义值
	0	long	没有设置签名算法

5.5.3. 数字签名

5.5.3.1. 功能说明

对字符串数据进行数字签名。

5.5.3.2. 接口说明

表 5-9 数字签名接口说明表

接口名称	String SOF_signData(byte[] inData) ;		
接口参数	参数名称	类型	说明
	inData	byte[]	签名原文
返回值	非空	String	Base64 编码签名结果
	空		失败

5.5.4. 验证签名

5.5.4.1. 功能说明

验证数字签名。

5.5.4.2. 接口说明

表 5-10 数字验签接口说明表

接口名称	boolean SOF_verifySignedData(String base64EncodeCert, String inData, String signValue) ;		
接口参数	参数名称	类型	说明
	base64EncodeCert	String	Base64 编码的证书
	inData	String	签名原文
	signValue	String	Base64 编码的签名值
返回值	true	boolean	成功
	false	boolean	失败

5.5.5. 文件签名

5.5.5.1. 功能说明

对文件数字签名。

5.5.5.2. 接口说明

表 5-11 文件签名接口说明表

接口名称	String SOF_signFile(String inFile);		
接口参数	参数名称	类型	说明
	inFile	String	待签名的文件路径
返回值	非空	String	Base64 编码签名结果
	空		失败

5.5.6. 验证文件签名

5.5.6.1. 功能说明

验证文件数字签名的完整性。

5.5.6.2. 接口说明

表 5-12 验证文件签名接口说明表

接口名称	boolean SOF_verifySignedFile(String base64EncodeCert, String inFile, String signValue) ;		
接口参数	参数名称	类型	说明
	base64EncodeCert	String	Base64 编码的证书
	inFile	String	待签名的文件路径
	signValue	String	Base64 编码的数据签名值
返回值	true	boolean	成功
	false	boolean	失败

5.5.7. 消息签名

5.5.7.1. 功能说明

对字符串消息数据进行签名。

5.5.7.2. 接口说明

表 5-13 消息签名接口说明表

接口名称	String SOF_signDataMessage(byte[] inData);		
接口参数	参数名称	类型	说明
	inData	byte[]	签名原文
返回值	非空	String	带原文消息结构的 Base64 编码签名数据
	空		失败

5.5.8. 验证消息签名

5.5.8.1. 功能说明

验证消息签名。

5.5.8.2. 接口说明

表 5-14 验证消息签名接口说明表

接口名称	boolean SOF_verifySignedDataMessage (String signedMessage) ;		
接口参数	参数名称	类型	说明
	signedMessage	String	带原文消息结构的 Base64 编码的消息签名数据
返回值	true	boolean	成功
	false	boolean	失败

5.5.9. 消息 Detach 签名

5.5.9.1. 功能说明

对字符串数据进行消息签名，签名值不带原文消息结构。

5.5.9.2. 接口说明

表 5-15 消息 Detach 签名接口说明表

接口名称	String SOF_signDataMessageDetach(byte[] inData);		
接口参数	参数名称	类型	说明
	inData	byte[]	签名原文
返回值	非空	String	不带原文消息结构的 Base64 编码签名数据
	空		失败

5.5.10. 验证消息 Detach 签名

5.5.10.1. 功能说明

验证不带原文的消息签名。

5.5.10.2. 接口说明

表 5-16 验证消息 Detach 签名接口说明表

接口名称	boolean SOF_verifySignedDataMessageDetach (byte[] inData, String signedMessage) ;		
接口参数	参数名称	类型	说明
	inData	byte[]	原文数据
	signedMessage	String	不带原文消息结构的 Base64 编码的消息签名数据
返回值	true	boolean	成功
	false	boolean	失败

5.5.11. XML 签名

5.5.11.1. 功能说明

对 XML 进行签名。

5.5.11.2. 接口说明

表 5-17 XML 签名接口说明表

接口名称	String SOF_signDataXML(byte[] inData, String xPath);		
接口参数	参数名称	类型	说明
	inData	byte[]	原文数据
	xPath	String	指定 xml 中需要签名的元素。为空时，表示签名整个 xml
返回值	非空	String	返回 Base64 编码签名数据
	空		失败

5.5.12. XML 验签

5.5.12.1. 功能说明

对 XML 签名结果进行验签。

5.5.12.2. 接口说明

表 5-18 XML 验签接口说明表

接口名称	boolean SOF_verifySignedDataXML(String signedMessage, String xPath);		
接口参数	参数名称	类型	说明
	signedMessage	String	签名数据
	xPath	String	指定 xml 中需要签名的元素。为空时，表示签名整个 xml
返回值	true	boolean	成功
	false	boolean	失败

5.5.13. XML Detach 签名

5.5.13.1. 功能说明

XML Detach 签名。

5.5.13.2. 接口说明

表 5-19 XML Detach 签名接口说明表

接口名称	String SOF_signDataXMLDetach(byte[] inData,String xPath);		
接口参数	参数名称	类型	说明
	inData	byte[]	原文数据
	xPath	String	指定 xml 中需要签名的元素。为空时，表示签名整个 xml
返回值	非空	String	返回 Base64 编码签名数据
	空		失败

5.5.14. XML Detach 验签

5.5.14.1. 功能说明

XML Detach 验签。

5.5.14.2. 接口说明

表 5-20 XML Detach 验签接口说明表

接口名称	boolean SOF_verifySignedDataXMLDetach(byte[] inData, String signedMessage, String xPath);		
接口参数	参数名称	类型	说明
	inData	byte[]	原文数据
	signedMessage	String	签名数据
	xPath	String	指定 xml 中需要签名的元素。为空时，表示签名整个 xml
返回值	true	boolean	成功
	false	boolean	失败

5.5.15. HMAC 摘要

5.5.15.1. 功能说明

HMAC 摘要。

5.5.15.2. 接口说明

表 5-21 HMAC 摘要接口说明表

接口名称	byte[] SOF_hmacSM3(byte[] key, byte[] message);		
接口参数	参数名称	类型	说明
	key	byte[]	密钥
	message	byte[]	原文数据
返回值	非空	byte[]	byte[]数组摘要数据
	空		失败

5.6. 调用示例

```
/*获取实例*/
SOFInterface svc=
SOFInterface.SOF_getAppInstance("http://172.21.68.71:8282/HTTP/963682
HCA0?IDEABANK_AUTHORITY_TOKEN=8AZMZHH1EG","7","06690d6c6f6a4d6bbbc321
75ac6af9fb","2baabe7af3ac4683bdff6a365e0f4769_7","a96e2a0c-5ec5-4df3-
831f-4367de5da19d");

/*导出证书*/
String s = svc.SOF_getServerCertificateByUsage(2);
System.out.println("证书信息:" + s);

/*设置签名算法*/
long SGD_SM3_SM2 = 0x00020201;
svc.SOF_setSignMethod(SGD_SM3_SM2);

/*获取签名算法*/
Long signMethod =svc.SOF_getSignMethod();

/*原文*/
```

```
String inData = "2345234532";
byte[] inDataByte = inData.getBytes("GBK");

/*数字签名,注意:必须先设置签名算法*/
String signString = svc.SOF_signData(inDataByte);
System.out.println((signString != null) ? "数字签名成功:" +
        signString : "数字签名失败");

/*数字验签*/
boolean ver = svc.SOF_verifySignedData(s,inDataByte,signString);
System.out.println((ver) ? "数字验签成功" : "数字验签失败");

/*不带原文消息签名，注意:必须先设置签名算法*/
String messignString = svc.SOF_signMessageDetach(inDataByte);
System.out.println((signString != null) ? "不带原文签名成功:" +
signString : "不带原文签名失");

/*不带原文消息验签*/
Boolean mesver = svc.SOF_verifySignedMessageDetach(inDataByte,
messignString);
System.out.println((mesver)？"不带原文验签成功":"不带原文验签失败");

/*消息签名，注意:必须先设置签名算法*/
String messignStr= svc.SOF_SignMessage(inDataByte);
System.out.println((messignStr != null)？"消息签名成功:"＋messignStr :
"消息签名失败");

/*消息验签*/
boolean mesverify = svc.SOF_verifySignedMessage(messignStr);
System.out.println((mesverify) ? "消息验签成功" : "消息验签失败");

/*文件签名，注意:必须先设置签名算法*/
String path="E:\\sxfConfig_yy.txt";
String fileSignStr=svc.SOF_signFile(path);
System.out.println((messignString != null) ? "文件签名成功:" +
fileSignStr : "文件签名失败");

/*文件验签*/
boolean fileVer=svc.SOF_verfySignedFile(s, path, fileSignStr);
System.out.println((fileVer) ? "文件签名成功" : "文件签名失败");

/*XML 签名，注意:必须先设置签名算法*/
Stringin XmlData=
"<PARAM><DBID>35</DBID><SEQUENCE>atgtca</SEQUENCE><PurchaseOrder><Com
```

```
pany>Widgets.Org</Company><Product>A large
widget</Product><Amount>$16.50</Amount><Due>16 January
2010</Due></PurchaseOrder></PARAM>";
byte[] inXmlDataByte = inXmlData.getBytes("GBK");
String xmlPath="/PARAM/PurchaseOrder";
String data=svc.SOF_signDataXML(inXmlDataByte,xmlPath);
System.out.println("签名数据："+data);

/*XML 验签*/
Boolean status=svc.SOF_verifySignedDataXML(data,xmlPath);
System.out.println(status?"验签成功":"验签失败");

/*XML Detach 签名，注意:必须先设置签名算法*/
String DetachData=svc.SOF_signDataXMLDetach(inXmlDataByte,xmlPath);
System.out.println("签名数据："+DetachData);

/*XML Detach 验签*/
Boolean
status1=svc.SOF_verifySignedDataXMLDetach(inXmlDataByte,DetachData,xm
lPath);
System.out.println(status1?"验签成功":"验签失败");

/*HMAC 摘要*/
byte[] key = "secret".getBytes();
byte[] message = "message".getBytes();
byte[] hmac = svc.SOF_hmacSM3(key, message);
System.out.println("HMAC-SM3: " + Base64Util.getBASE64(hmac));
```

5.7. 错误码说明

表 5-22 签名验签错误码说明表

序号	错误码	说明
1	0x00000000	正常返回
2	0x04000000	错误码起始值
3	0x04000001	错误证书标识
4	0x04000002	错误证书信息类型
5	0x04000003	CRL 或 OCSP 服务器无法连接
6	0x04000004	签名算法类型错误
7	0x04000005	签名者私钥索引值错误
8	0x04000006	签名者私钥权限标识码错误
9	0x04000007	证书非法或服务器内不存在
10	0x04000008	证书解析错误
11	0x04000009	证书过期

12	0x0400000A	证书尚未生效
13	0x0400000B	证书已被吊销
14	0x0400000C	签名无效
15	0x0400000D	数据格式错误
16	0x0400000E	系统内部错误
17	0x04100000	访问数据库失败
18	0x04100001	数据库数据不存在
19	0x04100002	时间戳证书无效或服务停用状态

6. 时间戳服务

6.1. 时间戳服务集成开发说明

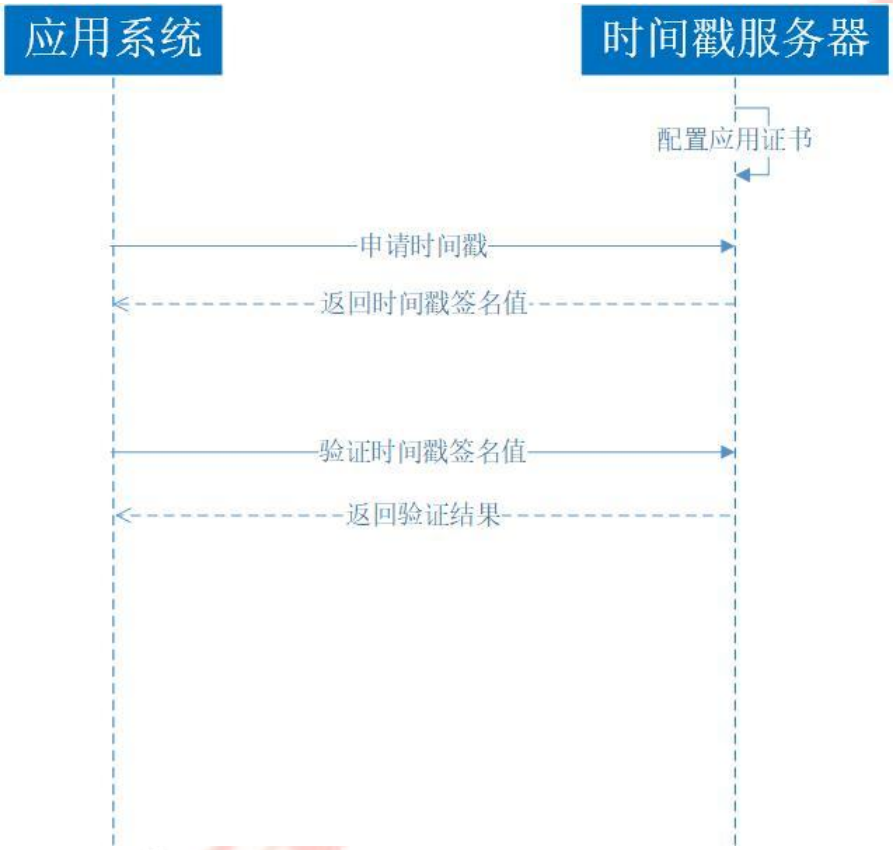

图 6-1 时间戳服务集成开发说明图

jar 包示例：

sof_url_jar.jar

6.2. 初始化

6.2.1. 创建实例

6.2.1.1. 功能说明

初始化接口，通过 url 获取实例， url 对应时间戳的 HTTP 服务。

6.2.1.2. 接口类构造方法

表 6-1 创建实例接口说明表

构造函数	TSAInterface(String url);		
参数	参数名称	类型	说明
	url	String	HTTP 接口服务地址

6.3. 时间戳服务

6.3.1. 申请时间戳

6.3.1.1. 功能说明

申请时间戳，成功返回字符串，失败抛出 TSAException 异常。

6.3.1.2. 接口说明

表 6-2 获取应用的实例接口说明表

接口名称	String createTimestamp(String data);		
接口参数	参数名称	类型	说明
	data	String	原文数据
返回值	非空	String	成功时返回 Base64 的编码时间戳数据

6.3.2. 验证时间戳

6.3.2.1. 功能说明

验证时间戳，成功返回 TSAResult 对象，失败抛出 TSAException 异常。

6.3.2.2. 接口说明

表 6-3 验证时间戳接口说明表

接口名称	TSAResult verifyTimestamp(String timestamp);		
接口参数	参数名称	类型	说明
	timestamp	String	Base64 编码的时间戳数据
返回值	TSAResult	Class 对象	见表 6-4 TSAResult 对象类说明

表 6-4 TSAResult 对象类说明表

对象名称	Class TSAResult
	{
	boolean status;
	String timeStamp;
	String statusCodeMessage;
	}		

对象参数	参数名称	类型	说明
	status	boolean	验证是否成功
	signature	String	返回时间信息
	statusCodeMessage	String	错误信息

6.3.3. 解析时间戳

6.3.3.1. 功能说明

解析时间戳，成功返回 Map<String,String>对象，失败抛出 TSAException 异常。

6.3.3.2. 接口说明

表 6-5 解析时间戳接口说明表

接口名称	Map<String,String> parseTimestamp(String timestamp);		
接口参数	参数名称	类型	说明
	timestamp	String	Base64 编码的时间戳数据
返回值	Map	map	返回 map，key 值如下： timestamp 时间信息 originData Hash 值 tsa 使用者

6.4. 错误码说明

表 6-6 时间戳错误码说明表

序号	错误码	说明
1	0x00000000	正常返回
2	0x04000000	错误码起始值
3	0x04000001	错误证书标识
4	0x04000002	错误证书信息类型
5	0x04000003	CRL 或 OCSP 服务器无法连接
6	0x04000004	签名算法类型错误
7	0x04000005	签名者私钥索引值错误
8	0x04000006	签名者私钥权限标识码错误
9	0x04000007	证书非法或服务器内不存在
10	0x04000008	证书解析错误
11	0x04000009	证书过期
12	0x0400000A	证书尚未生效
13	0x0400000B	证书已被吊销
14	0x0400000C	签名无效
15	0x0400000D	数据格式错误
16	0x0400000E	系统内部错误
17	0x00020001	参数错误
18	0x00020002	生成时间戳请求失败
19	0x00020003	生成时间戳响应失败
20	0x00020004	验证时间戳失败
21	0x00020005	解析时间戳失败
22	0x00020006	系统异常

6.5. 调用示例

```
//创建实例对象
TSAInterface tsa=new
TSAInterface("http://172.21.68.71:8282/HTTP/963682HCA0?IDEABANK_AUTHO
RITY_TOKEN=8AZMZHH1EG");

//获取时间戳
String timeStamp=tsa.createTimestamp("123456");
System.out.println(timeStamp);

//解析时间戳
Map<String,String> map= tsa.parseTimestamp(timeStamp);
System.out.println("时间戳信息："+map.get("timeStamp"));

//验证时间戳
TSAResult result=tsa.verifyTimestamp(timeStamp);
System.out.println(result.getStatus());
```

7. 加解密服务

7.1. 加解密服务集成开发说明

图 7-1 密码机服务集成开发说明图

7.2. 初始化

接口 jar 包如下：

初始化密码服务，参数列表如下：

表 7-1 初始化访问配置参数表

参数名称	参数说明	数据类型
requestUrl	请求地址	String

token	应用 token 值	String

示例：

```
InterfaceUrl.getInstance().initRequestUrl("http://172.21.68.71:8282/HTTP/QQPCF25XO", "KYLTOXNCWI");
```

7.3. 加密

7.3.1. 功能说明

对字符串数据进行加密。

7.3.2. 接口说明

表 7-2 加密接口请求参数说明表

参数名称	参数类型	说明
keyIndex	String	密钥索引，范围 1~256
mode	String	加密模式，"CBC"或者"ECB"
isPad	int	1 代表补位 0 代表不需要补位
srcData	String	待加密原文

表 7-3 加密接口返回参数说明表

参数名称	参数类型	说明
respValue	String	0 是成功，其他失败详见错误码
encryptedData	String	密文，base64 格式

7.3.2.1. 示例说明

加密接口调用示例：

```
Map encryptData = CipherUtils.postRequestEncryptData("2", "CBC", 1, "加密数据 123abc");
```

加密接口返回示例：

```
{respValue=0, encryptedData=hrMPWJgyOQuslVxRmqy++w==}
```

7.4. 解密

7.4.1. 功能说明

对加密密文进行解密得到原文。

7.4.2. 接口说明

表 7-4 解密接口请求参数说明表

参数名称	参数类型	说明
keyIndex	String	密钥索引，范围 1~256
mode	String	加密模式，"CBC"或者"ECB"
isPad	int	1 代表补位 0 代表不需要补位
encryptedData	String	待解密密文，Base64 格式

表 7-5 加密接口返回参数说明表

参数名称	参数类型	说明
plainData	String	已解密原文
respValue	String	0 是成功，其他失败详见错误码

7.4.2.1. 示例说明

解密接口调用示例：

```
Map decryptData = CipherUtils.postRequestDecryptData("2", "CBC", 1, "hrMPWJgyOQuslVxRmqy++w==");
```

解密接口返回示例：

```
{"plainData":"加密数据 123abc","respValue":"0"}
```

7.5. 调用示例

7.5.1. 调用示例

```
@Test
public void testEncryptDecrypt1() throws Exception {
/**
* url 提供的 path  http://172.21.68.71:8282/HTTP/D3N4B5RPJ0
* token
```

```
 */

InterfaceUrl.getInstance().initRequestUrl("http://172.21.68.71:8282/HTTP/QQPCF25XO", "KYLTOXNCWI");
//加密
Map encryptData = CipherUtils.postRequestEncryptData("2", "CBC", 1, "加密数据 123abc");
System.out.println("加密：" + JSON.toJSONString(encryptData));
//解密
Map decryptData = CipherUtils.postRequestDecryptData("2", "CBC", 1, (String) encryptData.get("encryptedData"));
System.out.println("解密：" + JSON.toJSONString(decryptData));
}
```

返回结果：

加密：

```
{"respValue":"0","encryptedData":"6bKwsoa+ngYe6LQp/D6nmsEWG6/OEmAqBl1xCYAf1jE="}
```

解密：

```
{"plainData":"加密数据 123abc","respValue":"0"}
```

7.5.2. 相关依赖

```
<dependency>
            <groupId>org.apache.httpcomponents</groupId>
            <artifactId>httpclient</artifactId>
            <version>4.5.3</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-lang3</artifactId>
            <version>3.10</version>
        </dependency>

        <dependency>
            <groupId>commons-codec</groupId>
            <artifactId>commons-codec</artifactId>
            <version>1.10</version>
        </dependency>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13.2</version>
```

```
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.30</version>
</dependency>
<!-- //slf4j 和 log4j 日志交换包 -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-jdk14</artifactId>
    <version>1.7.25</version>
</dependency>
<!-- //log4j 包 -->
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
    <type>zip</type>
</dependency>
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.60</version>
</dependency>
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpcore</artifactId>
    <version>4.4.13</version>
</dependency>
<dependency>
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-all</artifactId>
    <version>5.8.29</version>
</dependency>
```

7.6. 错误码说明

表 7-6 加解密接口错误码说明表

序号	错误码	描述
1	0x00000000	操作成功
2	0x01000000	错误码基础值
3	0x01000001	未知错误

4	0x01000002	不支持的接口调用
5	0x01000003	与设备通信失败
6	0x01000004	运算模块无响应
7	0x01000005	打开设备失败
8	0x01000006	创建会话失败
9	0x01000007	无私钥使用权限
10	0x01000008	不存在的密钥调用
11	0x01000009	不支持的算法调用
12	0x0100000A	不支持的算法模式调用
13	0x0100000B	公钥运算失败
14	0x0100000C	私钥运算失败
15	0x0100000D	签名运算失败
16	0x0100000E	验证签名失败
17	0x0100000F	对称算法运算失败
18	0x01000010	多步运算步骤错误
19	0x01000011	文件长度超出限制
20	0x01000012	指定的文件不存在
21	0x01000013	文件起始位置错误
22	0x01000014	密钥类型错误
23	0x01000015	密钥错误
24	0x01000016	ECC 加密数据错误
25	0x01000017	随机数产生失败
26	0x01000018	私钥使用权限获取失败
27	0x01000019	MAC 运算失败
28	0x0100001A	指定文件已存在
29	0x0100001B	文件写入失败
30	0x0100001C	存储空间不足
31	0x0100001D	输入参数错误
32	0x0100001E	输出参数错误

8. 协同签名服务

8.1. 协同签名服务集成开发说明

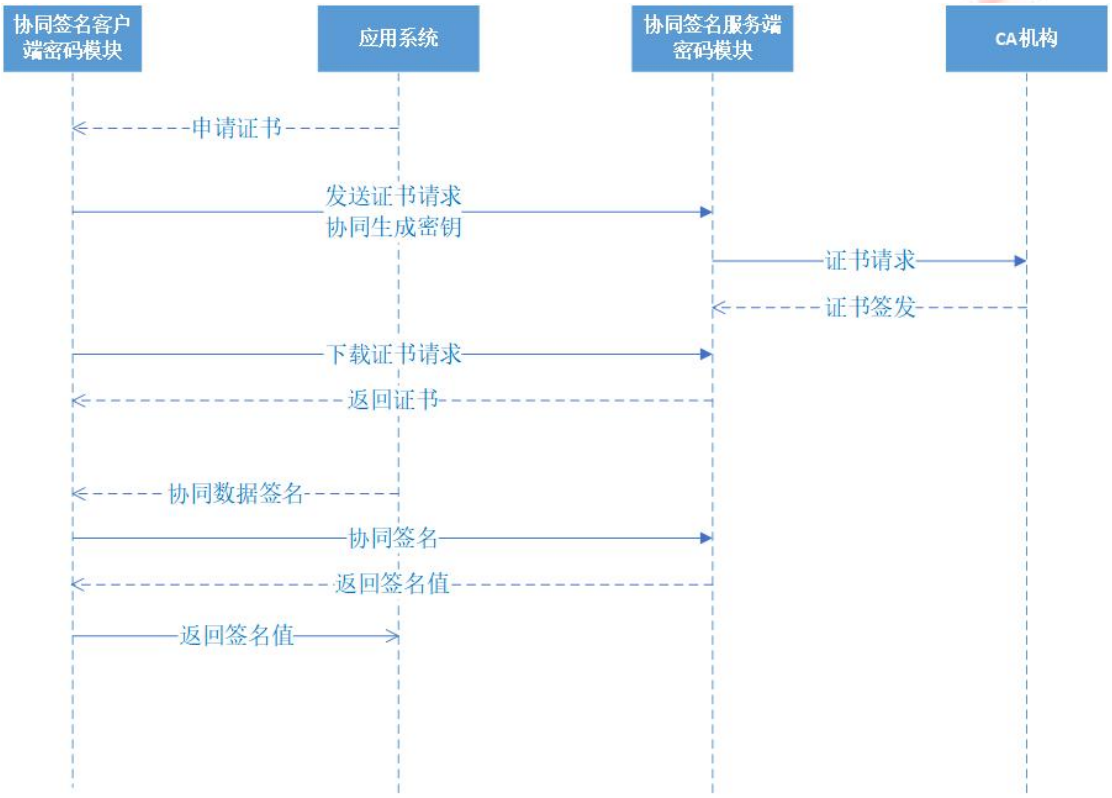

图 8-1 协同签名服务集成开发说明图

8.2. 初始化

Aar 包示例：

IBxtsdkSign-2.0926.aar

① 调用之前创建实例 YYCertManager impl = new YYCertManagerImpl（）

② 业务接口部分接口含有回调接口，回调方法中 onFinish(int status, String msg) status = 0 代表请求成功，错误参照 8.11 错误码，msg 返回具体失败信

息提示。

具体步骤如下：

步骤 1.将 aar 包 yyxtsdk-release.aar 复制到 lib 下

步骤 2. 在 app 的 build.gradle 下添加下面代码

```
defaultConfig {
    ...
    ndk {
        abiFilters "armeabi-v7a","arm64-v8a"
    }
}
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar','*.aar'])
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
    implementation 'com.squareup.retrofit2:converter-scalars:2.3.0'
    implementation "androidx.security:security-crypto:1.1.0-alpha03"
    implementation "org.jetbrains.kotlin:kotlin-stdlib:1.6.0"
}
```

步骤 3.自定 Application 继承 Application,在 onCreate 里初始化

```
XtSignSdk.getInstance().init(this, appid, appCode)
```

* appid 和 appCode 需要到后台下载（可不填）

```
XtSignConfig.getConfig().setDebug(true).setXtqmUrl("xxx")
```

*xtqmUrl 协同签名服务器地址

示例代码

```
class MyApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        XtSignSdk.init(this,”appid”,”appCode”);
        XtSignConfig.getConfig().setDebug(true).setXtqmUrl("xxx")
    }
}
```

步骤 4.在 AndroidManifest.xml 中添加相关权限

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

```
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.MANAGE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
将 application 的 name 指向自定义的 MyApplication, 并添加
android:networkSecurityConfig="@xml/network_security_config"配置
// 示例代码
<application
    android:name=".MyApplication"
    android:networkSecurityConfig="@xml/network_security_config">
    <activity>
        ...
    </activity>
</application>
```

表 8-1 初始化说明表

*适配版本： Android 5.0~Android11（Android SDK 21~30）
*重要：在用户使用 app 时，切勿把手机恢复出厂设置或清除数据，此操作会导致客户端密钥文件丢失，无法恢复，用户需要重新生成新的密钥。 *H5 调用：需在 WebViewActivity 里加入如下代码 mWebView.addJavascriptInterface(new IbBridge(this, mWebView), "ib");

8.3. 回调接口说明

8.3.1. YYCallback 接口说明

8.3.1.1. 功能说明

通用型回调接口。

8.3.1.2. 接口说明

表 8-2 YYCallback 接口说明表

接口名称	public interface YYCallback {void onFinish(int status,String msg);}		
接口参数	参数名称	类型	说明
	status	int	状态码

	msg	String	提示信息

8.3.2. YYCertStatusCallback 接口说明

8.3.2.1. 功能说明

查询证书状态回调接口。

8.3.2.2. 接口说明

表 8-3 YYCertStatusCallback 接口说明表

接口名称	public interface YYCertStatusCallback{void onFinish(int status, String msg, boolean isNewCert, String certStatusName);}		
接口参数	参数名称	类型	说明
	status	int	状态码
	msg	String	提示信息
	isNewCert	boolean	是否有新证书需要下载
	certStatusName	String	证书状态名称

8.3.3. YYResultCallback 接口说明

8.3.3.1. 功能说明

带返回值回调。

8.3.3.2. 接口说明

表 8-4 YYResultCallback 接口说明表

接口名称	public interface YYResultCallback {void onFinish(int status, String msg, String result);}		
接口参数	参数名称	类型	说明
	status	int	状态码
	msg	String	提示信息
	result	String	返回值

8.4.证书申请

8.4.1. 功能说明

进行证书申请。

8.4.2. 接口说明

表 8-5 证书申请接口说明表

接口名称	void YY_CertApply(final YYApplyCertInfo info, final YYCallback callback);		
接口参数	参数名称	类型	说明
	info	YYApplyCertInfo	申请证书需要的信息
	callback	YYCallback	回调接口
返回值			无

8.4.3. 示例说明

调用示例：

```
YYApplyCertInfo info = new YYApplyCertInfo();
info.setName(name);// 用户名 （最长 15 个字符）
info.setUnit("Test1");// 单位（最长 15 个字符）
info.setIdCard("123456789012345678");// 身份证（18 位校验）
info.setOrganization("工会组织");// 组织（最长 15 个字符）
info.setPin(pin);// PIN 码（6~16 位校验）
info.setConfirmPin(pin);// 确认 PIN 码（6~16 位校验）
info.setEmail("1234@qq.com");// 邮箱（邮箱格式校验）
info.setPhone("12345678901");// 手机号（11 位校验）
info.setTitle("Title");// Title 项（最长 15 个字符）

mImpl.YY_CertApply(info, new YYCallback() {
    @Override
```

```
    public void onFinish(int status, String msg){
        // status=0 代表申请成功
    }
});
```

8.5.证书下载

8.5.1. 功能说明

进行证书下载。

8.5.2. 接口说明

表 8-6 证书下载接口说明表

接口名称	void YY_DownloadCert(YYCallback callback);		
接口参数	参数名称	类型	说明
	callback	YYCallback	回调接口
返回值			无

8.5.3. 示例说明

调用示例：

```
mImpl.YY_DownloadCert(new YYCallback() {
    @Override
    public void onFinish(int status, String msg) {
        // status=0 下载成功
    }
});
```

8.6.证书查询

8.6.1. 功能说明

进行证书查询。

8.6.2. 接口说明

表 8-7 证书查询接口说明表

接口名称	void YY_CertStatus(YYCertStatusCallback callback);		
接口参数	参数名称	类型	说明
	callback	YYCertStatusCallback	回调接口
返回值			无

8.6.3. 示例说明

调用示例：

```
mImpl.YY_CertStatus(new YYCertStatusCallback() {
        @Override
        public void onFinish(int status, String msg, boolean isNewCert,
String certSName) {
    status： 0 请求成功 90011 证书制作状态
    msg： 证书状态相关提示
    isNewCert： 是否为新申请或者更新过的证书 true 是 false 否
    certSName： 证书状态名
        }
    }));
```

8.7. 修改 PIN 码

8.7.1. 功能说明

进行 PIN 码修改。

8.7.2. 接口说明

表 8-8 修改 PIN 码接口说明表

接口名称	void YY_ModifyPin(String oldPin, String newPin, String confirmPin, YYCallback callback);		
接口参数	参数名称	类型	说明
	oldPin	String	旧 PIN 码
	newPin	String	新 PIN 码
	confirmPin	String	确认 PIN 码
	callback	YYCallback	回调接口
返回值			无

8.7.3. 示例说明

调用示例：

```
mImpl.YY_ModifyPin(oldPin, newPin, newPin, new YYCallback() {
    @Override
    public void onFinish(int status, String msg) {
        Toast.makeText(MainActivity.this, msg,
Toast.LENGTH_SHORT).show();
    }
});
```

8.8 验证 PIN 码

8.8.1. 功能说明

进行 PIN 码验证。

8.8.2. 接口说明

表 8-9 验证 PIN 码接口说明表

接口名称	boolean YY_VerifyPin(String pin);		
接口参数	参数名称	类型	说明
	pin	String	需要验证的 pin 码
返回值	true	boolean	正确
	false	boolean	错误

8.8.3. 示例说明

调用示例：

```
boolean isTrue= impl.YY_VerifyPin(pin)
true 正确 false 错误
```

8.9. 签名

8.9.1. 功能说明

对数据进行签名。

8.9.2. 接口说明

表 8-10 签名接口说明表

接口名称	void doSign(String pin, String originalData, int signType, YYResultCallback callback);		
接口参数	参数名称	类型	说明

	pin	String	pin 码
	originalData	String	需要签名的数据
	signType	int	签名类型：1 为 P1 2 为 P7
	callback	YYResultCallback	回调接口
返回值			无

8.9.3. 示例说明

调用示例：

```
YYSignManager.getInstance().doSign(s, originalData, mSignType, new
YYResultCallback() {
    @Override
    public void onFinish(int status, String msg, String result) {
        if (i == 0) {
            binding.result.setText(result);
        } else {
            Toast.makeText(MainActivity.this, s,
Toast.LENGTH_SHORT).show();
        }
    }
});
```

8.10. 验证签名

8.10.1. 功能说明

对数据进行验签。

8.10.2. 接口说明

表 8-11 验签接口说明表

接口名称	void doVerify(String originalData, String signResult, int signType, YYCallback callback);

	参数名称	类型	说明
接口参数	originalData	String	原数据
	signResult	String	签名数据
	signType	int	签名类型：1 为 P1 2 为 P7
	callback	YYCallback	回调接口
返回值			无

8.10.3. 示例说明

调用示例：

```
YYSignManager.getInstance().doVerify(originalData, result, mSignType,
new YYCallback() {
    @Override
    public void onFinish(int status, String msg) {
        // status=0 验证成功
    }
});
```

8.11. 错误码说明

表 8-12 协同签名错误码说明表

序号	错误码	说明
1	90000	请先设置服务器地址
2	90001	请输入 11 位手机号码
3	90002	手机号码格式不正确
4	90003	登录失败
5	90004	申请证书填入项不正确
6	90005	证书已申请
7	90006	缺少权限
8	90007	证书申请失败
9	90008	本地无证书
10	90009	证书正在生成中
11	90010	证书下载失败
12	90011	证书状态
13	90012	证书延期时间为空
14	90013	证书到期时间为空

15	90014	选择的延期时间范围不对
16	90015	延期申请失败
17	90016	证书更新填入项格式不正确
18	90017	证书更新失败
19	90018	重签申请失败
20	90019	获取验证码失败
21	90020	PIN 码为空
22	90021	PIN 码不正确
23	90022	随机数的长度不正确
24	90023	随机数获取失败
25	90024	PIN 码修改失败
26	90025	两次 PIN 码输入不一致
27	90026	UUID 为空
28	90029	公钥不存在
29	90030	应用未授权
30	90031	用户不存在
31	90032	未授权
32	40001	签名值为空
33	40002	签名原数据为空
34	40003	验签失败
35	40004	需要签名的数据为空
36	40005	下载新证书提示
37	40006	R 值获取失败
38	40007	S 值获取失败
39	40008	签名失败
40	40009	PIN 码为空
41	40010	上传已盖章文件失败
42	40011	盖章失败
43	40012	未找到签章图片
44	40013	未关联印章
45	40014	印章数据查询失败
46	40015	印章获取失败