IFAA

互联网金融身份认证联盟标准

T/IFAA 0001—2016

IFAA本地免密技术规范

IFAA Local Password-less Technical Specification

2016－04发布　　2016－04实施

互联网金融身份认证联盟　　发布

目录

1 范围

本文档描述了IFAA本地免密技术标准的内容。本文分为六个部分：

部分	说明	参与的角色
IFAF 协议部分	描述IFAA客户端和服务端的数据交换格式，提供IFAA Server服务的厂商及应用厂商需要参考。	IFAA服务商、应用厂商
REE客户端部分	描述了IFAA本地免密在安卓应用部分内容，需要接入IFAA客户端的应用厂商需要参考。	应用厂商
REE系统框架部分	描述IFAA本地免密在在android framework层的技术标准，TEE/OEM厂商需要重点参考。	TEE厂商、OEM厂商(使用自己的TEE产品时)
TrustZone部分	描述IFAA本地免密在TEE里技术细节，TEE厂商需要重点参考。	TEE厂商、OEM厂商(使用自己的TEE产品时)
TLV协议部分	描述IFAA本地免密的二进制协议格式,Server厂商和TEE厂商需要重点参考。	TEE厂商、OEM厂商(使用自己的TEE产品时)及IFAA服务商
安全体系部分	描述IFAA安全架构方案，TEE厂商、Server厂商、应用厂商需要重点参考。	TEE厂商、IFAA服务商及应用厂商

2 名词

2.1 缩略语

API	Application Program Interface	应用程序接口
REE	Rich Execution Environment	富应用执行环境
IFAF	IFAA Authenticator Framework	IFAA 校验器框架

3 IFAF 协议部分

IFAF协议是实现IFAA在Framework层的接口与IFAA服务器之间端到端通信的标准。

3.1 消息结构

在协议体系中，IFAA 服务器、IFAA REE端、IFAA TEE端直接处理IFAF消息。

——IFAA 服务器：在应用程序服务端运行，生成 IFAF 注册消息，处理 IFAF 响应消息；

——IFAA REE 端：在用户设备端运行，处理 IFAF 注册消息并响应；

——IFAA TEE 端：在用户设备安全区运行。

IFAF协议核心流程在IFAA 服务器和IFAA 客户端之间包含如下4个阶段：

——注册:应用程序厂商根据用户的信息注册一种校验方式,应用程序厂商可以选择校验器的类型，IFAA REE 端会根据应用厂商选择的类型执行注册操作；

——校验:用户使用注册时使用的校验器进行校验,应用程序可以随时在需要的时候进行校验操作；

——注销：删除用户的开通记录数据。

IFAF协议抽象的定义了IFAA 服务端作为请求的发起者，实际上通常是客户端程序通过HTTP请求向IFAA 服务端请求用于校验的IFAF消息。

3.2 消息定义

3.2.1 通用数据类型

本小节定义在流程中使用到的通用数据类型。

Version

标识主要和次要的版本号。

```
public class Version {
    /**
     * Major version
     */
    int mj = Setting.majorVersion;
    /**
     * Minor version
     */
    int mn = Setting.minorVersion;
}
```

Operation Header

```
/**
 * IFAF协议的请求和响应消息头
 */
public class OperationHeader {
    /**
     * Version, IFAF协议版本号，目前版本为1.0
     */
    Version ipv;

    /**
     * operation, Reg or DeReg or Auth. 操作分类：注册、注销、校验
     */
    String op;

    /**
     * operation type, Request or Response. 操作类型：Request或Response
     */
    String opType;

    /**
     * auth type, FP or iris or wear, 验证类型，大小为1个字节，mask为0xFF，
     * 目前IFAF协议1.0版本指纹为0x01、虹膜为0x02
     */
    byte authType;

    /**
     * application ID, IFAA Client用于判断app是否有权访问指定校验类型
```

```
     */
    String appID;

    /**
     * 设备型号
     */
    String deviceModel;

    /**
     * server data (optional) 服务端用于其他对服务端有用的数据(可选)
     */
    String serverData;
}
```

Extension

通用的扩展字段

```
public class Extension {
    /**
     * Extension id 扩展字段ID
     */
    String id;
    /**
     * Extension data 扩展字段数据
     */
    String data;
}
```

TrustedApps

信任的App来源。

```
public class TrustedApps {

    /**
     * version of trustedApps
     */
    Version version;

    /**
     * facetIDs
     */
    List<String> ids;
}
```

字段解析：

——version：可信 App 集合版本，当前为 1.0；

——ids：可信 app 的 facetID 列表。对于 Android 程序来说，facetID 是根据 APK 程序的 SHA1 哈希值计算出来的，样例如下：android:apk-key-hash:<sha1_hash-of-apk-signing-cert>

IFAFSignedData

```
/**
 * 证书签名后的数据包
 */
public class IFAFSignedData {
    /**
     * scheme of krd should be IFAFV1TLV
     * 定义KRD的消息格式，目前支持IFAFV1TLV。
     */
    String scheme;

    /**
     * key registration data
     * 证书签名后的二进制数据块转义之后的字符串。
     */
    String identifyData;
}
```

IFAFServerMessage

```
/**
 * IFAA Server 与 App Server 通信 Message 基类
 */
public class IFAFServerMessage {
    /**
     * 消息错误码
     */
    int code;
    /**
     * 错误消息内容
     */
    String errorMessage;
}
```

IFAFMessage

```
/**
 * IFAFMesssage基类
 * 包含消息头部,封装好的经过证书签名后的数据,以及扩展字段.
 */
public class IFAFMessage {
    /**
     * header of the message.
     * IFAF协议的请求和响应消息头，其中op字段由具体的业务决定。
     */
    OperationHeader header;
```

```
    /**
     * 证书签名后的数据包
     */
    IFAFSignedData signedData;

    /**
     * extension  (optional)
     * 校验器可以扩展的字段。
     */
    Extension[] exts;
}
```

Challenge 生成规则

IFAA Server在生成Request阶段，生成一个Challenge，用来避免重放攻击。Challenge的生成规则是：32字节的a-zA-Z0-9的随机字符。

示例：

kqOOlidpOCxH14ukbx1yDXJs5pWTa52U

Token 生成规则

Token是IFAA Server在注册阶段生成的用来标识业务的唯一ID。Token的格式如下：

IFAA Server的供应商缩写+UUID，用“-”分隔。

示例：

ABC-4e576fc1-8a7f-4357-a991-840e2747b2ba

3.3 注册流程

注册时，IFAA Client和IFAA Server通过IFAF协议协商校验密钥，服务端保存开通信息。

首先IFAA Server下发AppID、会话挑战字段Challenge、业务Token至IFAA Client。

校验器验证IFAA Server的证书签名，验签通过后，生成用户业务用的公私钥，并将该密钥对与Token保存在安全区中，之后将返回的数据用校验器预置的证书进行签名，IFAA Server通过IFAA中心验证服务器来验证客户端的来源可信性。

3.3.1 注册 Request 获取阶段

IFAA Server接受注册Request请求的数据体：

```
/**
 * 注册Request的请求数据体
 */
public class RegisterRequestReq {
    /**
     * 验证类型
     */
    byte authType;

    /**
     * IFAA 版本号
     */
    private byte ifaaVersion;
```

```
    /**
     * 客户端支持的证书编码格式
     */
    private byte certEncode;
}
```

IFAA Server生成注册Request后，将响应发回App Server的数据体：

```
/**
 * IFAA Server 在处理Request后返回给App Server的数据体
 */
public class IFAFServerRequestResp extends IFAFServerMessage {
    /**
     * Request数据体
     */
    IFAFMessage request;
}
```

3.3.2 注册 Response 响应阶段

IFAA Server 接收注册 Response 请求的消息体：参见 IFAFMessage。
IFAA Server 在处理完注册 Response 之后，返回给 App Server 的响应消息体：

```
/**
 * 注册Response的响应数据体
 */
public class RegisterResponseResp extends IFAFServerMessage {
    /**
     * 验证类型
     */
    byte authType;
    /**
     * 设备ID
     */
    String deviceId;
    /**
     * 业务令牌
     */
    String token;
}
```

3.3.3 消息处理规则

IFAA Server 注册 Request 生成规则

——创建 RegisterRequest 对象，根据规则设置 header；
——scheme 对象赋值为“IFAFV1TLV”；
——按照规则生成 identifyData 并赋值到 identifyData 字段中。

IFAA Client 注册 Request 处理规则

——处理服务端下发注册 Request 消息，根据消息类型，对 IFAA Server 下发的消息进行解析；

——根据 authType 选择身份认证方式，并进行指定类型的身份认证；
——将 identifyData 发送给 IFAA Authenticator。

IFAA Authenticator 注册请求处理规则

IFAA Authenticator处理规则参考IFAA-Authenticator。

IFAA Client 注册响应生成规则

——创建 RegistrationAssertion 对象；
——将 scheme 字段赋值为 IFAFV1TLV,将 IFAA Authenticator 返回的数据放入 identifyData 中；
——创建注册 Response 对象；
——将 RegisterRequest 中的 header 内容赋值到 response 的 header 字段中，并调整 opType 为 Response；
——将生成的消息返回给调用方。

IFAA Server 注册响应生成规则

——如果有需要，根据 header 中的 serverData 验证响应的合法性。

3.4 校验流程

校验流程IFAA Server会向IFAA Client发送校验请求验证用户注册时保存的密钥信息和用户校验器特征。

首先IFAA Server下发AppID和Token至IFAA Client。

然后IFAA Client将authData传入至校验器。

校验器验证用户特征并且和Challenge一起使用开通时和Token绑定的密钥进行签名，IFAA Server验证签名信息完成校验流程。

3.4.1 校验 Request 获取阶段

IFAA Server接受校验Request请求的数据体：

```
/**
 * 校验Request的请求数据体
 */
public class AuthenticationRequestReq extends IFAFServerMessage {
    /**
     * 业务令牌
     */
    private String token;

    /**
     * IFAA 版本号
     */
    private byte ifaaVersion;

    /**
     * 客户端支持的证书编码格式
     */
    private byte certEncode;

}
```

IFAA Server生成校验Request后，将响应发回App Server的数据体：

```
/**
 * IFAA Server 在处理Request后返回给App Server的数据体
 */
public class IFAFServerRequestResp extends IFAFServerMessage {
    /**
     * Request数据体
     */
    IFAFMessage request;
}
```

3.4.2 校验 Response 响应阶段

IFAA Server接收校验Response请求的消息体参见2.1.7 IFAFMessage。

IFAA Server在处理完校验Response之后，返回给App Server的响应消息体：

```
/**
 * IFAA 校验结果
 */
public class AuthenticationResponseResp extends IFAFServerMessage {
    /**
     * 业务令牌
     */
    String token;
    /**
     * 校验结果
     */
    boolean authenticationResult;
}
```

3.4.3 消息处理规则

IFAA Server 校验请求消息生成规则

——创建 AuthenticationRequest 对象，根据规则设置 header；

——scheme 对象赋值为“IFAFV1TLV”；

——按照规则生成 identifyData 并赋值到 identifyData 字段中。

IFAA Client 校验请求消息处理规则

——处理服务端下发 AuthenticationRequest 消息，根据消息类型，对 IFAA Server 下发的消息进行解析；

——根据 authType 选择身份认证方式，并进行指定类型的身份认证；

——将 identifyData 发送给 IFAA Authenticator。

IFAA Authenticator 校验请求消息处理规则

——IFAA Authenticator 处理规则参考 IFAA-Authenticator。

IFAA Client 校验响应消息生成规则

——创建 AuthenticationAssertion 对象；

——将 scheme 字段赋值为 IFAFV1TLV,将 IFAA Authenticator 返回的数据放入 identifyData 中；
——创建 AuthenticationResponse 对象；
——将 AuthenticationRequest 中的 header 内容赋值到 response 的 header 字段中；
——将生成的消息返回给调用方。

IFAA Server 校验响应消息处理规则

——如果有需要，根据 header 中的 serverData 验证响应的合法性；
——分别处理每一个 assertion，处理的方式详见 IFAA-server。

3.5 注销流程

在注销流程中，IFAA Server删除注册时的密钥信息并且生成注销Request下发给IFAA Client，IFAA Client响应注册请求并且删除本地开通数据。

3.5.1 注销 Request 生成阶段

IFAA Server接受注销Request请求的数据体：

```
/**
 * 注销Request的请求数据体
 */
public class DeregisterRequestReq extends IFAFServerMessage {
    /**
     * 业务令牌
     */
    private String token;

    /**
     * IFAA 版本号
     */
    private byte ifaaVersion;

    /**
     * 客户端支持的证书编码格式
     */
    private byte certEncode;

}
```

IFAA Server生成注销Request后，将响应发回App Server的数据体：

```
/**
 * IFAA Server 在处理Request后返回给App Server的数据体
 */
public class IFAFServerRequestResp extends IFAFServerMessage {
    /**
     * Request数据体
     */
    IFAFMessage request;
}
```

3.5.2 消息处理规则

IFAA Server 注销 Request 消息生成规则

——创建 DeregisterAuthenticator 对象，根据规则设置 header；
——scheme 对象赋值为“IFAFV1TLV”；
——按照规则生成 deregData 并赋值到 deregData 字段中。

IFAA Client 注销请求消息处理规则

——处理服务端下发 DeregisterAuthenticator 消息，根据消息类型，对 IFAA Server 下发的消息进行解析；
——将 deregData 发送给 IFAA Authenticator。

IFAA Authenticator 注销请求消息处理规则

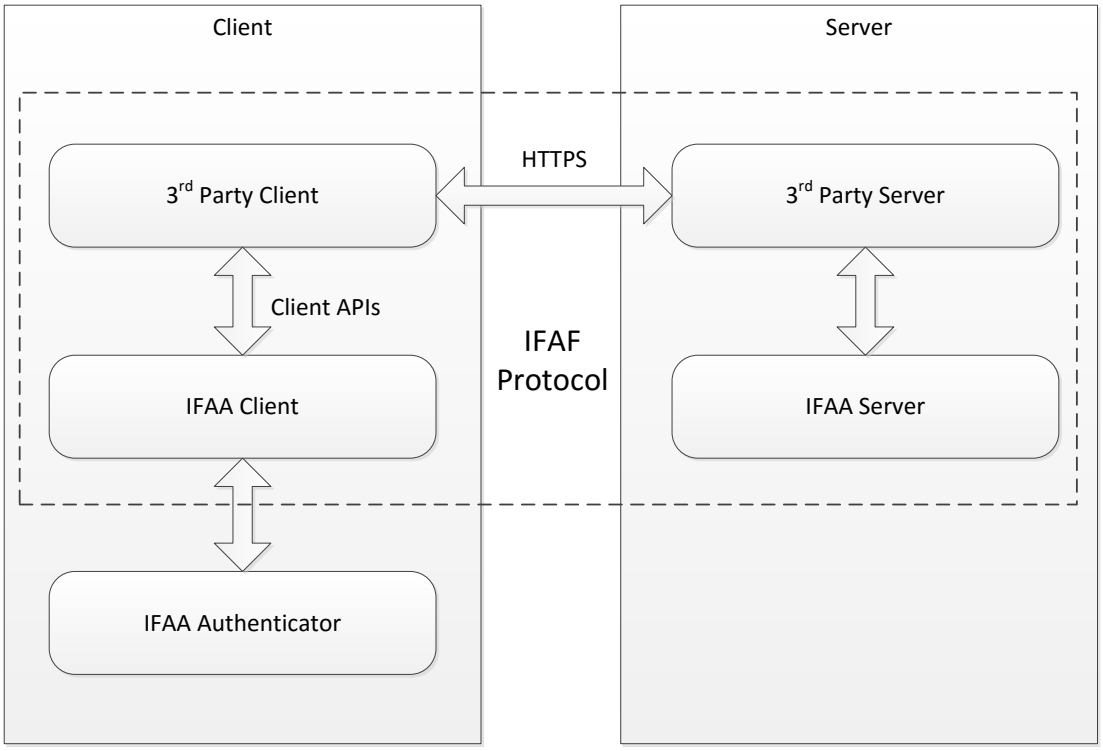
IFAA Authenticator处理规则参考IFAA-Authenticator。

4 REE 客户端部分

REE客户端部分解决Framework层接口与具体App的适配问题。该层为ifaa提供的标准实现，无需厂商介入。

4.1 系统结构

整体的IFAF协议和流程在IFAF协议说明文档中进行定义。下面这个简单的架构图说明了本文档涉及的交互流程。



IFAF流程包含如下4个阶段：
——设备检查：根据客户端可用的校验器类型判断客户端的 IFAA 协议能力
——注册：客户端生成用户密钥，并且将用户开通信息与密钥进行关联，将密钥的对应信息和校验器特征上传到 IFAA Server。
——校验：根据用户开通时候的注册的校验器特征进行用户身份确认，利用开通时生成的用户密钥进行用户身份校验。

——去注册：先通知服务端删除用户开通信息，然后删除本地开通时生成的密钥信息。

——设备检查阶段不涉及到与 IFAA Server 之间的协议交换，IFAA Client 通过客户端校验器的特性判断客户端的能力。

IFAF协议抽象的定义了IFAA Server作为请求的发起者，实际上通常是客户端程序通过HTTPS请求向IFAA Server请求用于校验的IFAF message。

从依赖IFAA Client的客户端应用程序的角度来看，注册、校验、去注册协议校验流程如下：

——客户端程序即可以直接向服务端获取 IFAF Request Message，也可以和其他的客户端消息一起向服务端请求消息。

——客户端通过调用 IFAA Client 提供的 api 将 IFAF Request Message 传递给 IFAA Client，异步的收到结果的回调。

——IFAA Client 处理请求消息并且和用户进行相应的交互，交互完成之后将 IFAF Response Message 返回给调用方，或者返回一个错误信息。

——客户端将 IFAF Response Message 通过 HTTP 协议发送回 IFAA Server。

去注册不是一个完成的IFAF协议过程，IFAA Server首先在服务端进行解绑操作，然后生成一个Deregistration request，客户端将Deregistration request传给IFAA Client进行客户端解绑操作，IFAA Client不会返回一个Deregistration reponse，客户端应用不需要将消息再次返给会服务端。

IFAF协议的消息类型使用JSON结构定义的，消息中包含了加密的数据以及签名的保护，客户端应用程序不要对IFAF协议的消息数据进行修改，修改了会导致IFAA Client或IFAA Server认为这个消息是不合法的。

4.2 系统组件

4.2.1 IFAFClient.aidl

IFAFClient是应用程序与IFAA Client之间交互的主要接口，IFAFClient提供了异步接口用于响应IFAF Message，提供了同步接口用于对IFAA Client的状态查询。

```
package cn.org.ifaa.ifaf.client;

import cn.org.ifaa.ifaf.client.IFAFCallback;

interface IFAFClient
{
    int processCommand(inout Bundle aMessage);
    int processIFAFCommandAsync(inout Bundle aMessage,
                                IFAFCallback callback);
}
```

IFAA Client在aMessage中获取IFAF request message，将reponse message同样设置在aMessage中。

request message

aMessage是Android标准的Bundle组件，提供键值对的数据结构，aMessage应该包含的信息如下，具体的值的定义参考IFAF message定义：

——KEY_OPERATIONT_TYPE：操作的类型，注册请求、校验请求、去注册请求;

——KEY_MESSAGE：IFAA Server：下发的 IFAF Request Message;

——KEY_AUTHENTICATOR_TYPE：校验器类型，指纹、虹膜等等。

callback

异步操作回调接口，IFAA Client会把处理结果通过callback实例返回给调用方，接口说明参考IFAFCallback.aidl。

4.2.2 IFAFCallback.aidl

这个接口定义了IFAA Client将处理结果回调给调用方的接口定义，这个接口在IFAA Client处理结果成功或者失败时被调用。

```
package cn.org.ifaa.ifaf.client;

import android.os.Bundle;

interface IFAFCallback {
    int callback(in Bundle bundle);
}
```

response message

aMessage是Android标准的Bundle组件，提供键值对的数据结构，aMessage应该包含的信息如下，具体的值的定义参考IFAF message定义：

——IFAA_OPERATIONT_TYPE：操作的类型，注册响应、校验响应、去注册响应；

——IFAA_RESULT：操作结果，成功、失败；

——IFAA_MESSAGE：IFAA Client 响应的 IFAF Response Message。

4.2.3 IFAFMesssage.java

这个类结构定义了IFAA Client与IFAA Server之间双向数据传输的类型。类结构包含了实际的IFAF协议消息，并且提供了额外的扩展字段用于协议的扩展或者应用程序自定义的消息。在应用程序客户端对IFAFMessage进行修改可能会导致消息不可用。消息类型的详细说明在IFAF message说明文档中进行详细定义。

扩展字段的类型为String，可以用来在应用程序客户端和IFAA Client之间交换额外的数据，这个数据类型可以由应用厂商自行定义，建议使用JSON格式。

5 REE 系统框架部分

REE系统框架部分向下封装TA和Android M的指纹接口，向上对应用客户端提供服务。

5.1 整体结构

IFAA要求所有的支持IFAF协议的系统中都应该实现IFAAManager接口，IFAAManager接口中包含对校验方式的判断，设备型号的获取，以及TEE的调用。

所有Android M指纹接口特性保持不变，对于@hide类型的接口名称与特性要与本文档保持一致。

相比于Android M的标准接口，此文档增加了两个接口，startFpManager(), processCmd(),

org.ifaa.android.manager.IFAAManagerFactory工厂类。

org.ifaa.android.manager.IFAAManager是需要实现的接口类。

同时这2个java类需要放到厂商的系统ROM中，我们会使用反射或者其他的方式来调用。

5.2 接口规定

5.2.1 IFAAManager

IFAAManager为抽象类，需要放到系统framework中，通过IFAAManager生成实例。

getDeviceModel

函数原型：String getDeviceModel () ;

功能描述：返回系统手机型号，model字段用于标记手机的型号，主要是指的一个系列。例如一个厂商的各个MODLE，例如三星的S6有可能叫SAMSUNG-SM9200， SAMSUNG-SM9201等等，其实都是指的一个系列，但是他们都是使用的同一个方案，例如TA和芯片和指纹芯片。所以对model的定义就是一个系列，他们对应了一个内置的厂商TA私钥。

getVersion

函数原型：int getVersion() ;

功能描述：返回Manager接口版本，目前为1。

getSupportBIOTypes

函数原型：int getSupportBIOTypes(Context context);

功能描述：返回手机上支持的校验方式，目前IFAF协议1.0版本指纹为0x01、虹膜为0x02，验证类型可以为多种不同方式组合，用'或'操作符拼接。

startBIOManager

函数原型：int startBIOManager(Context context, int authType) ;

功能描述：启动系统的指纹管理应用界面，让用户进行指纹录入。指纹录入是在系统的指纹管理应用中实现的，本函数的作用只是将指纹管理应用运行起来，直接进行页面跳转，方便用户录入。

参数描述：context:上下文环境

. authType: 生物特征识别类型，指纹为1，虹膜为2

返回值：0，成功启动指纹管理应用；其他值，启动指纹管理应用失败。

返回值	取值	说明
COMMAND_OK	0	成功
COMMAND_FAIL	-1	失败

processCmd

函数原型：public native byte[] processCmd(Context context, byte[] param);

功能描述：通过ifaateeclient的so文件实现REE到TAA的通道。

参数描述：byte[] param 用于传输到IFAA TA的数据buffer。

返回值：byte[] 返回IFAA TA返回REE的数据buffer。

5.2.2 IFAAMangerFactory

IFAAManagerFactory只包含一个静态方法，即生成IFAAManager实例，需要放到系统framework中。

getIFAAManager

函数原型：IFAAManager getIFAAManager (Context context, int authType) ;

功能描述：返回对应的IFAAManager实例，

参数描述：context:上下文环境

authType: 生物特征识别类型，指纹为1，虹膜为2

5.2.3 指纹验证相关接口

isHardwareDetected

函数原型：boolean isHardwareDetected () ;
功能说明：判断当前系统是否支持指纹，接口特性与Android 6.0保持一致。
参数说明：无；
返回值：true：有；false：无。

hasEnrolledFingerprints

函数原型：boolean hasEnrolledFingerprints() ;
功能说明：判断当前系统是否有已经录入的指纹，接口特性与Android 6.0保持一致。
参数说明：无；
返回值：true：有；false：无。

authenticate

函数原型：void authenticate(FingerprintManager.CryptoObject crypto, CancellationSignal cancel, int flags, FingerprintManager.AuthenticationCallback callback, Handler handler)

功能说明：开始指纹认证，接口特性与Android 6.0保持一致。异步返回，通过回调客户端的FingerprintManager.AuthenticationCallback中的onXXXX()函数向应用通报状态和认证结果。认证操作可以通过传入的CancellationSignal 对象取消。
参数说明：
crypto：object associated with the call or null if none required.
cancel：an object that can be used to cancel authentication
flags：optional flags; should be 0
callback：an object to receive authentication events
handler：an optional handler to handle callback events
返回值：无

5.2.3.1.1 CancellationSignal.cancel()

函数原型：void cancel() ;
功能说明：取消指纹校验
参数说明：无
返回值： N/A

5.2.3.1.2 FingerprintManager.AuthenticationCallback.onAuthenticationSucceeded()

函数原型：void onAuthenticationSucceeded (AuthenticationResult result) ;
功能说明：authenticate回调函数，指纹校验通过时调用
参数说明：result: 结果AuthenticationResult
返回值： N/A

5.2.3.1.3 FingerprintManager.AuthenticationCallback.onAuthenticationFailed()

函数原型：void onAuthenticationFailed() ;
功能说明：Called when a fingerprint is valid but not recognized.

参数说明：无
返回值： N/A

5.2.3.1.4 FingerprintManager.AuthenticationCallback.onAuthenticationError(int errorCode, CharSequence errString)

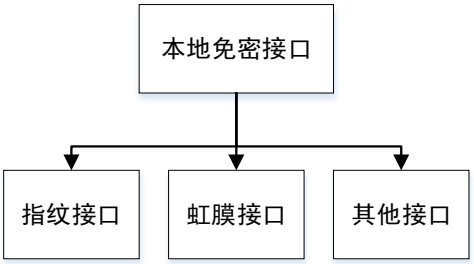
函数原型：onAuthenticationError(int errorCode, CharSequence errString);
功能说明：Called when an unrecoverable error has been encountered and the operation is complete.
参数说明：errorCode: An integer identifying the error message
errString：A human-readable error string that can be shown in UI
返回值： N/

6 TA 部分

TA部分描述IFAA在Trust Zone里的技术标准。本部分有样例代码提供。可对照参考。

6.1 基本架构

IFAA TA 将作为一个本地免密的容器，其本身仅负责做协议解析和调用底层接口做加解密和签名验签等操作。具体的生物验证由各厂商去实现。这些生物特征将继承一致的标准接口。



这些模块物理上可以被实现为独立的静态库，当 TA 启动时，由 TA 负责将这些模块的标准导出符号注入到 TA 本地免密的实现中，以便在处理不同的生物类型时分别调用

6.2 接口形式

除了强制实现的 IFAA ta 接口外，IFAA 的内部接口建议也使用统一形式。方便后续扩展。

IFAA TA 的接口形式如下:

```
IFAA_Result IFAA_TaInvokeCmd(uint8 *buf_in, size_t in_len, uint8_t *buf_out, size_t *out_len);
```

In为入参buffer,out为出参buffer,IFAA_Result为状态码。

状态码	值	说明
IFAA_ERR_SUCCESS	0x00000000	成功
IFAA_ERR_UNKNOWN	0x7A000001	未知错误
IFAA_ERR_BAD_ACCESS	0x7A000002	访存通用错误
IFAA_ERR_BAD_PARAM	0x7A000003	不合法的参数
IFAA_ERR_UNKNOWN_CMD	0x7A000004	不能识别的命令
IFAA_ERR_BUF_TOO_SHORT	0x7A000005	参数 buffer 不够
IFAA_ERR_OUT_OF_MEM	0x7A000006	内存分配失败
IFAA_ERR_TIMEOUT	0x7A000007	超时
IFAA_ERR_HASH	0x7A000008	HASH 摘要算法失败
IFAA_ERR_SIGN	0x7A000009	签名算法失败
IFAA_ERR_VERIFY	0x7A00000A	验签算法失败
IFAA_ERR_KEY_GEN	0x7A00000B	生成秘钥失败
IFAA_ERR_READ	0x7A00000C	读文件失败
IFAA_ERR_WRITE	0x7A00000D	写文件失败
IFAA_ERR_ERASE	0x7A00000E	删除文件失败
IFAA_ERR_NOT_MATCH	0x7A00000F	生物特征不匹配
IFAA_ERR_GEN_RESPONSE	0x7A000010	生成 response 失败
IFAA_ERR_GET_DEVICEID	0x7A000011	获取设备 ID 失败
IFAA_ERR_GET_LAST_IDENTIFIED_RESULT	0x7A000012	获取最近一次认证通过的 ID 失败
IFAA_ERR_AUTHENTICATOR_SIGN	0x7A000013	校验器签名失败
IFAA_ERR_GET_ID_LIST	0x7A000014	获取系统中录入的 id 列表失败
IFAA_ERR_GET_AUTHENTICATOR_VERSION	0x7A000015	获取校验器版本号失败
IFAA_ERR_UN_INITIALIZED	0x7A000016	未初始化错误
IFAA_ERR_NO_OPTIONAL_LEVEL	0x7A000017	本地无匹配的安全等级错误

6.3 IFAA TA 接口规范

6.3.1 容器管理

IFAA 的 TA 将提供一个入口函数:

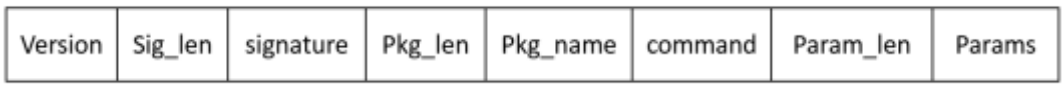
```
IFAA_Result IFAA_TaInvokeCmd(uint8 *buf_in, size_t in_len, uint8_t *buf_out, size_t *out_len);
```

其中

buf_in 为输入 buffer,**in_len** 为输入 buffer 的长度；

buf_out 为输出 buffer,**out_len** 为输出 buffer 的长度指针;

输入 buffer 的形式如下:



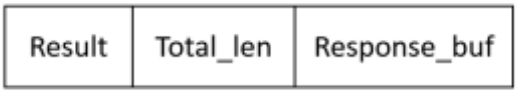
各字段语义如下:

字段	类型	长度	说明
version	uint8_t*	4 bytes	版本
sig_len	size_t	4 bytes	签名长度
signature	void*	sig_len	签名
pkg_len	size_t	4 bytes	包名长度
package_name	void*	pkg_len	包名
command	uint32_t	4 bytes	命令类型
param_len	size_t	4 bytes	参数长度
params	void*	param_len	参数

其中command的取值有以下几种：

COMMAND	值	说明
IFAA_TA_CMD_GET_DEVICE_ID	0x01	获取设备 ID
IFAA_TA_CMD_REGISTER	0x02	注册
IFAA_TA_CMD_AUTHENTICATE	0x03	验证
IFAA_TA_CMD_DEREGISTER	0x04	注销
IFAA_TA_CMD_QUERY_STATUS	0x05	查询开通状态
IFAA_TA_CMD_GEN_ASYMMETRIC_KEY	0x06	异步生成密钥
IFAA_TA_CMD_GET_PROTOCL_VERSION	0x07	获取 ta 的协议版本号

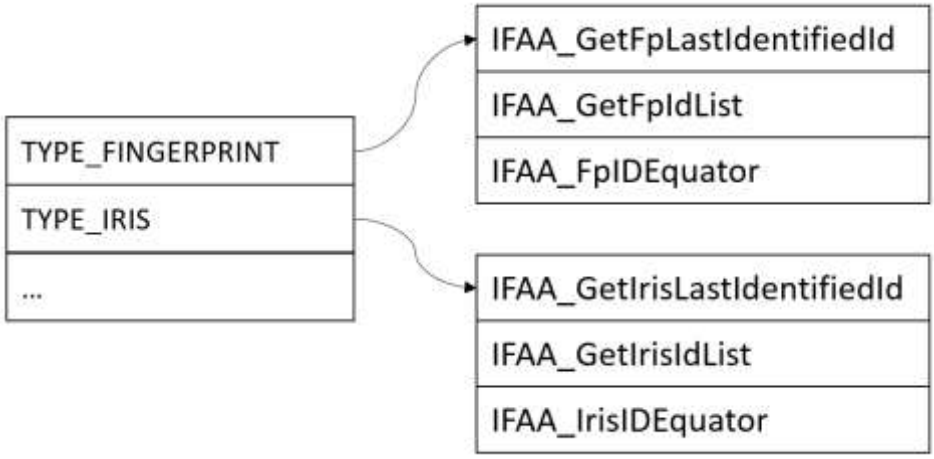
输出 buffer 的形式如下:



其中各个字段的语义为:

字段	类型	长度	说明
Result	uint8_t*	4 bytes	结果
Total_len	size_t	4 bytes	出参 buffer 总长度
Response_buf	uint8_t*	Total_len	出参 buffer

下面分别对这些 command 做分别诠释。
容器将维护下面一个表。



其中记录了各种生物验证类型的获取上次验证的入口地址。这些均在TA启动时由TA负责完成初始化。

容器提供下面接口，由 TA 负责调用注入实现(实现来源于 TA 中包含的生物特征识别模块)

```
typedef enum {
    IFAA_BIO_FINGERPRINT   = 1 << 0,              //指纹
    IFAA_BIO_IRIS          = 1 << 1,              //虹膜
    IFAA_BIO_FACE          = 1 << 2,              //人脸
    //more
} IFAA_BioType;

typedef enum {
    IFAA_ENTRYLAST_IDENTIFIED_RESULT_GETTER     = 0x01, //获取最近一次认证通过的id
    IFAA_ENTRY_AUTHENTICATOR_VERSION_GETTER,            //获取对应Authenticator的版本
    IFAA_ENTRY_ID_LIST_GETTER,                          //获取系统中录入的生物特征的id列表
    IFAA_ENTRY_EQUATOR                                  //生物特征id的比较符
} IFAA_TaEntry;

//TA启动时调用该函数完成注入
TEE_Result IFAA_TaAddEntry(IFAA_BioType bioType,IFAA_TaEntry entry, void* func);

//TA运行时通过该接口获取对应函数入口
TEE_Result IFAA_TaGetEntry(IFAA_BioType bioType, IFAA_TaEntry entry, void**func);
```

至于表的实现方式则保留给具体容器自行设计。

6.3.2 获取上一次认证结果的模板 Id(强制接口)

用于获取最近一次认证成功的生物特征模板id(如指纹的Index)。

```
typedef IFAA_Result (*IFAA_LastIdentifiedResultGetter)(uint8_t *buf_id, uint32_t *id_len);
```

结果通过 buf_id 传出参数返回。(虽然对于指纹，目前的标识就是一个整型的 id,但对于以后出现的其他生物特征可能不符合该假设，所以抽象的以 buffer 表示)

参数:

	类型	长度	说明
buf_id	uint8_t*	*id_len	最近一次认证通过的模板 id
Id_len	size_t*	sizeof((size_t*)0)	输入 buffer 的长度

返回值：是否成功的标志；
指纹/虹膜等都需要实现该接口并导出该符号。由 ta 启动时完成符号表注入。
注意:针对指纹，要求模板 ID 的有效期为 3s，超过 3s 则作废，返回失败。

6.3.3 获取对应 Authenticator 版本号（强制接口）

用于获取对应的 Authenticator 的版本号（指纹/虹膜等）

```
typedef IFAA_Result (*IFAA_AuthenticatorVersionGetter)(uint32_t *version);
```

结果通过出参 buffer_version 获取。同样由 ta 启动时完成符号注入。
参数:

参数	类型	长度	说明
buf_version	uint8_t*	*id_len	版本对应的 buffer
version_len	size_t*	sizeof((size_t*)0)	输入 buffer 的长度

返回值: 是否成功的标志；

6.3.4 获取对应 IDlist（强制接口）

用于获取对应的已经录入的 id 列表（指纹/虹膜等）

```
typedef IFAA_Result (*IFAA_IdListGetter)(uint8_t *buf_id_list, uint32_t *id_list_len);
```

参数:

参数	类型	长度	说明
buf_id_list	uint8_t *	*id_list_len	已经录入的 id 列表,形式如下 
id_list_len	size_t*	sizeof((size_t*)0)	出参 buffer 的长度

返回值: 是否成功的标志

6.3.5 生物特征比较符(可选接口)

用于完成对生物特征的本地比对。

```
typedef IFAABoolean (*IFAA_IdEquator)(const uint8_t *buf_l, const uint8_t *buf_r, uint32_t buf_len);
```

结果通过 id 传出参数返回。
参数:

参数	类型	长度	说明
buf_l	uint8_t*	buf_len	待比较的 id
buf_r	uint8_t*	buf_len	被比较的 id

返回值: 是否相等；
指纹/虹膜等都需要实现该接口并导出该符号。由 ta 启动时完成符号表注入。
如不提供，则默认使用按位比对的方式进行。

6.3.6 获取 Ta 的协议版本号版本(建议接口)

获取本 TA 的版本:

```
IFAA_Result (*)(vlb_t *buf_version);
```

参数:

参数	类型	长度	说明
buf_version	uint8_t*	version_len	Ta 的协议版本号

返回值: 是否成功的标志

6.3.7 查询本地状态(建议接口)

用于查询本地免密的开通状态。

```
IFAA_Result (*)(vlb_t *token, vlb_t *response);
```

参数:

参数	类型	长度	说明
token	vlb_t*	token->len	服务端下发的用于关联密钥的 id
response	vlb_t*	Response->len	开通状态集合。其中按位标识了各个本地免密类型状态。 0 代表关闭 1 代表开通 该接口为标准实现，可参考 sample.

返回值: 是否成功的标志；

6.3.8 注册(建议接口)

用于注册一个生物特征的本地免密。

```
IFAA_Result (*)(vlb_t *request_tlv, vlb_*response_tlv);
```

该接口为标准实现。可参考我们后续提供的 sample.

参数:

参数	类型	长度	说明
request_tlv	vlb_t*	request_tlv ->len	Ifaa server 下发的 tlv 格式的注册数据
response_tlv	vlb_t*	response_tlv->len	返回的 tlv buffer

返回值: 是否成功的标志；

6.3.9 注销(建议接口)

用于注销一个生物特征的本地免密。

```
IFAA_Result (*)(vlb_t *request_tlv);
```

该接口同样为标准实现。可参考我们后续提供的 sample.

参数:

参数	类型	长度	说明
request_tlv	vlb_t*	request_tlv ->len	Ifaa server 下发的 tlv 格式的注册数据

返回值: 是否成功的标志；

6.3.10 验证(建议接口)

用于完成一次本地的生物特征验证。

```
IFAA_Result (*)(vlb_t *request_tlv, vlb_t *response_tlv);
```

该接口为标准实现。可参考我们后续提供的 sample。

参数:

参数	类型	长度	说明
request_tlv	vlb_t*	request_tlv ->len	Ifaa server 下发的 tlv 格式的注册数据
response_tlv	vlb_t*	response_tlv->len	返回的 tlv buffer

返回值: 是否成功的标志；

6.3.11 加解密部分

SHA256:

```
IFAA_Result IFAA_Sha256(const uint8_t *msg, uint32_t msg_len,
                        uint8_t digest, uint32_t *digest_len);
```

RSA签名:

```
IFAA_Result IFAA_RsaSignDigest(const IFAA_RsaKey *key,
                               const uint8_t *digest, uint32_t digest_len,
                               uint8_t *sig, uint32_t *sig_len);
```

RSA验签:

```
IFAA_Result IFAA_RsaVerifyDigest(const IFAA_RsaKey *key,
                                 const uint8_t *digest, uint32_t digest_len,
                                 const uint8_t *sig, uint32_t siglen);
```

Hmac SHA1:

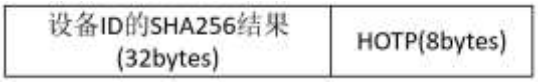
```
IFAA_Result IFAA_HmacSha1(const uint8_t *msg, uint32_t msg_len,
                          const uint8_t *key, uint32_t key_len,
                          uint8_t *mac);
```

6.3.12 获取设备 ID

该接口用于获取本机的设备 ID.(RPMB)

```
IFAA_Result IFAA_GetDeviceId(uint8_t *device_id, uint32_t *device_id_len);
```

Response 中包含了 40bytes 长度的 deviceid。其组成为:



6.3.13 校验器签名接口

使用每个 RPMB 中烧入的密钥(一机一密)进行签名。

```
IFAA_Result IFAA_AuthenticatorSignDigest(const uint8_t *digest, uint32_t digest_len,
                                         uint8_t *signature, uint32_t *sig_len);
```

参数:

参数	类型	长度	说明
digest	uint8_t*	digest_len	待验证的消息摘要

signature	uint8_t*	sig_len	产生的签名

返回值: 是否成功的标志；

6.3.14 验签接口

使用 RPMB 中烧入的 IFAA 根证书验证证书链。

```
IFAA_Result IFAA_AuthenticatorVerifyDigest(const uint8_t *digest, uint32_t digest_len,
                                           const uint8_t *signature, uint32_t sig_len,
                                           const IFAA_Certificate *cert_chain,
                                           uint32_t cert_count);
```

参数:

参数	类型	长度	说明
digest	uint8_t*	digest_len	待验证的消息摘要
signature	uint8_t*	sig_len	待验证的签名
cert_chain	uint8_t*	cert_count	证书链

返回值: 是否成功的标志

7 TLV 协议部分

TLV协议定义了IFAA TA到IFAA Server的端到端的通信格式。在TA和IFAA Server均需要TLV协议的封装和解析实现。

7.1 协议内容

7.1.1 结构

IFAA 客户端与服务端通信均基于 TLV(tag-length-value)的消息格式进行。TLV 是一种序列化方法。顾名思义，tlv 的 buffer 形式如下。

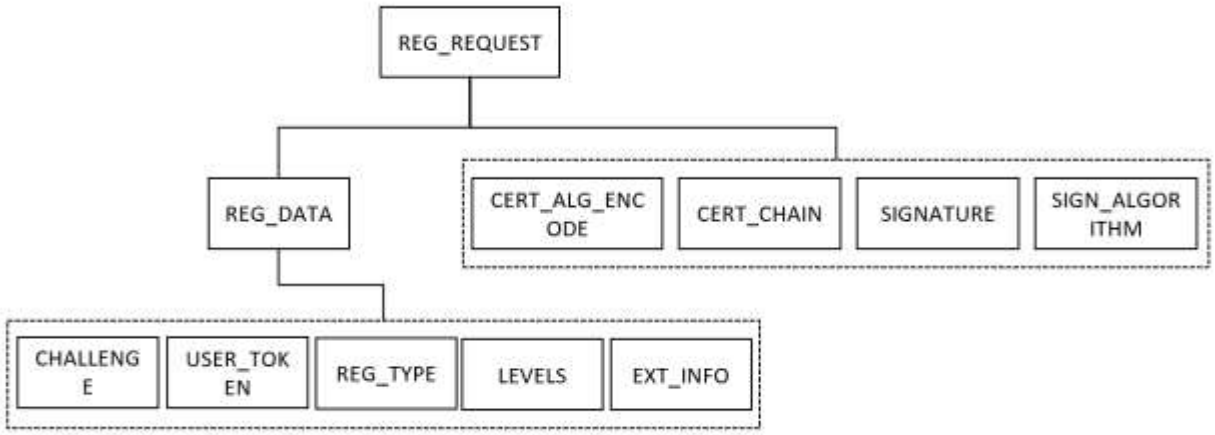
对应各个字段的意义如下:

字段	类型	长度	说明
TAG_VALUE	uint16_t	2 bytes	一个表明该节点类型的标记。规定叶子节点的 TAG_VALUE >= 128. 其余值均为非叶子节点
BUFFER_LEN	uint16_t	2 bytes	该 tlv 节点有效负载的长度。
BUFFER	unsigned char*	BUFFER_LEN 指定的长度	该 tlv 节点的有效负载。如果该节点为叶子节点，则该 buffer 为实际负载。否则为各子节点的负载之和。

注: tlv 节点中的数据采用大端模式存储。

注册 request

客户端发起注册请求时，需要向 IFAA Server 获取 request。该 request 为 tlv 的 buffer。反序列化之后，其结构如下。



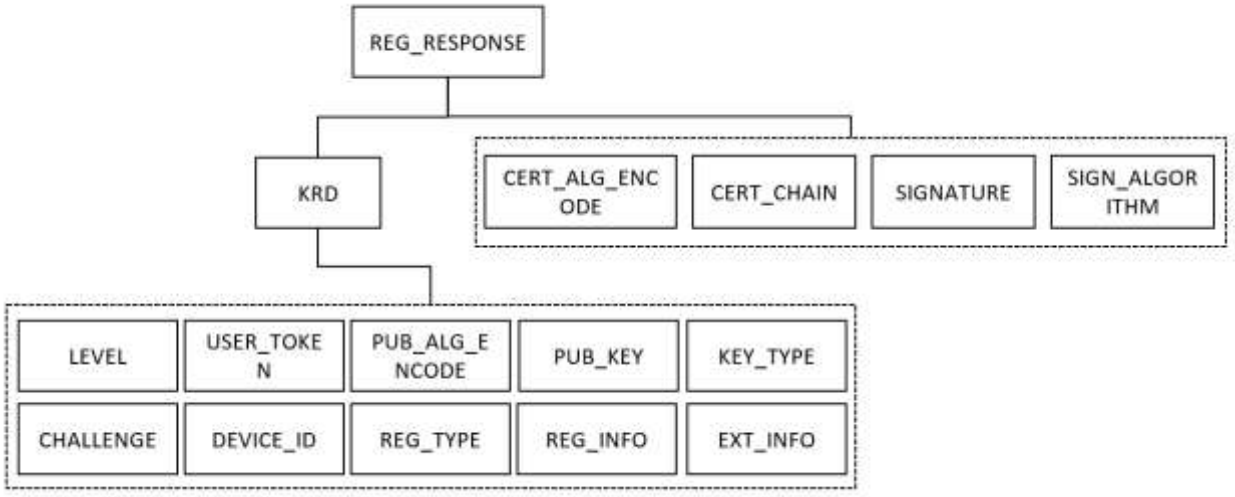
各节点的语义如下:

字段	类型	值	说明
REG_REQUEST	uint16_t	0x01	根节点
REG_DATA	uint16_t	0x02	注册 request 的主要数据
CERT_ALG_ENCODE	uint16_t	0x8005	证书链编码格式
CERT_CHAIN	uint16_t	0x8006	IFAA 服务商证书链
SIGN_ALGORITHM	uint16_t	0x8008	签名的算法
SIGNATURE	uint16_t	0x8007	签名
CHALLENGE	uint16_t	0x8001	IFAA 服务端下发的一种有时效限制的防重放挑战码
USER_TOKEN	uint16_t	0x8002	唯一标识用户的 id
REG_TYPE	uint16_t	0x8003	生物验证类型(指纹/虹膜等)
LEVELS	uint16_t	0x8011	服务端支持的安全等级
EXT_INFO	uint16_t	0x8004	其他扩展信息(可选)

关于签名：签名由 IFAA 服务商申请的二级证书进行，下发 request 时，需要将二级甚至三级证书放置在 CERT_CHAIN 里，便于客户端进行证书链的校验。而证书链的根证书将会被内置在 TA。

注册 response

客户端完成必要的操作后，将会生成 response TLV。需要透传到 IFAA Server 解析并验证认证客户端的认证结果。其格式如下：

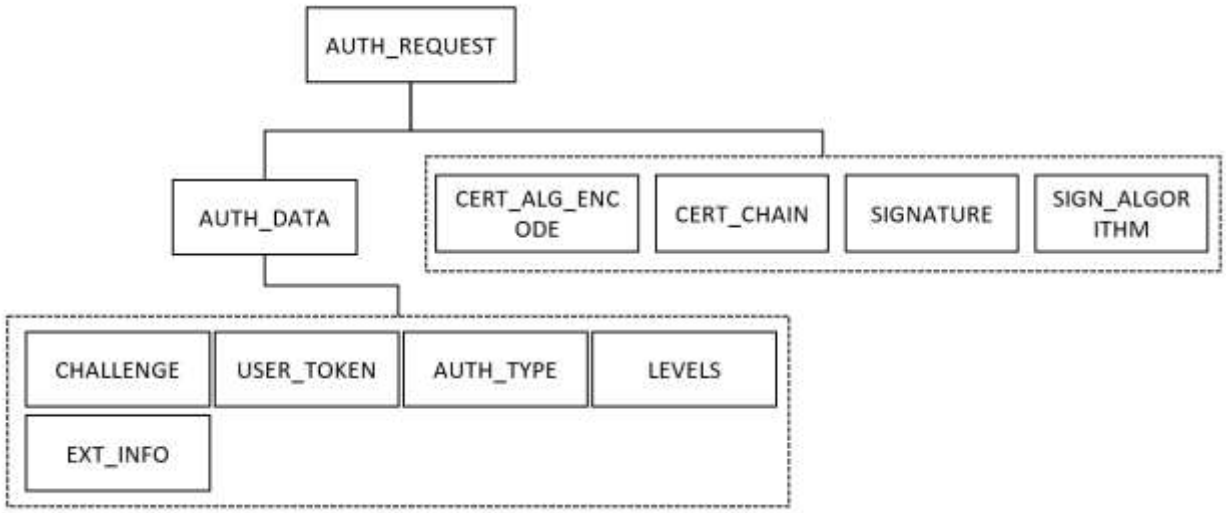


各节点的语义如下:

字段	类型	值	说明
REG_RESPONSE	uint16_t	0x03	注册结果的 response
KRD	uint16_t	0x04	Response 有效信息
CERT_ALG_ENCODE	uint16_t	0x8005	证书链编码格式
CERT_CHAIN	uint16_t	0x8006	证书链
SIGN_ALGORITHM	uint16_t	0x8008	签名算法，支持 RSA2048,ECC256 和国密
SIGNATURE	uint16_t	0x8007	签名内容
LEVEL	uint16	0x8011	客户端支持的安全等级
USER_TOKEN	uint16_t	0x8002	唯一标识用户的 id
PUB_ALG_ENCODE	uint16_t	0x800A	公钥编码格式
PUB_KEY	uint16_t	0x800B	公钥
KEY_TYPE	uint16	0x800C	非对称密钥类型。支持 RSA2048,ECC256 和国密
CHALLENGE	uint16_t	0X8001	IFAA 服务端下发的防止重发攻击的挑战码
DEVICE_ID	uint16_t	0X800D	设备 id
REG_TYPE	uint16_t	0x8003	注册类型
REG_INFO	uint16_t	0x800E	注册内容(如注册的指纹模板 ID)
EXT_INFO	uint16_t	0x8004	其他扩展信息(可选)

校验 request

当客户端发起校验时，需要请求IFAA Server下发一个校验的request，其消息格式如下：



各节点语义如下:

字段	类型	值	说明
AUTH_REQUEST	uint16_t	0x05	根节点
AUTH_DATA	uint16_t	0x06	注册 request 的主要数据
CERT_ALG_ENCODE	uint16_t	0x8005	证书链编码格式
CERT_CHAIN	uint16_t	0x8006	IFAA 服务商证书链
SIGN_ALGORITHM	uint16_t	0x8008	签名算法，支持 RSA2048,ECC256 和国密
SIGNATURE	uint16_t	0x8007	签名
CHALLENGE	uint16_t	0x8001	IFAA 服务端下发的一种有时效限制的放重放挑战码
USER_TOKEN	uint16_t	0x8002	唯一标识用户的 id

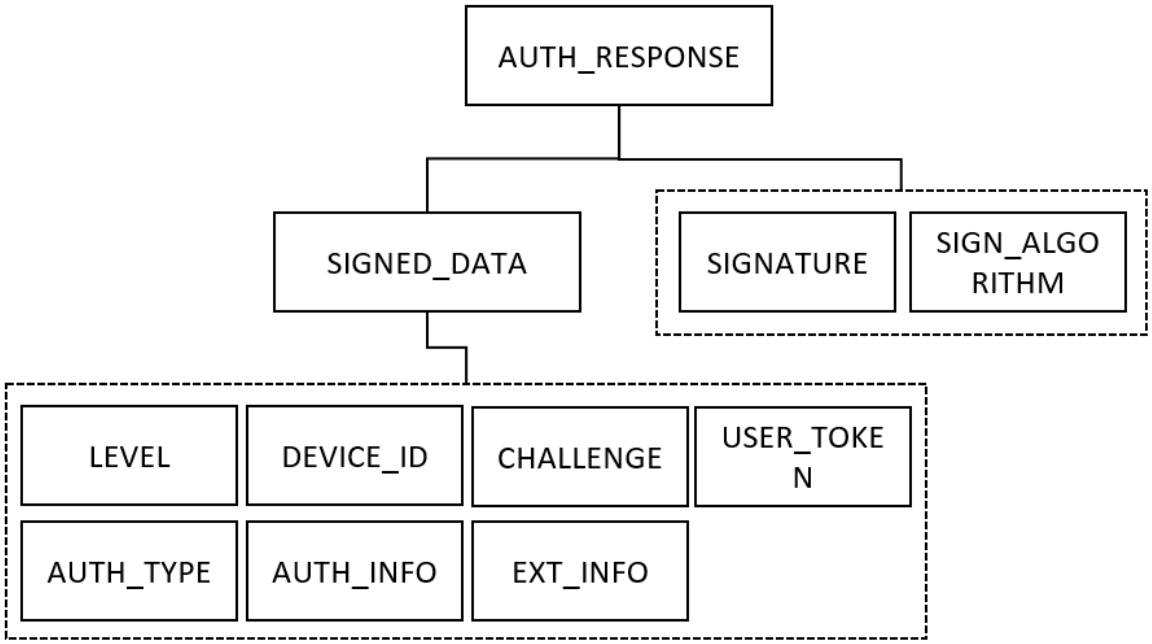
AUTH_TYPE	uint16_t	0x800F	验证类型
LEVELS	uint16	0x8011	服务端支持的安全等级
EXT_INFO	uint16_t	0x8004	其他扩展信息（可选）

关于签名：签名由 IFAA 服务商的二级证书进行，下发 request 时，需要将二级甚至三级证书放置在 CERT_CHAIN 里，便于客户端进行证书链的校验。而证书链的根证书将会被内置在客户端里。

校验 response

客户端进行本地校验，如果客户端的生物验证通过，则客户端会返回 response.由应用将该 response 上传到相应的 IFAA 服务器做验证，以 IFAA 服务器的验证作为结果。

该 response 的结构如下:

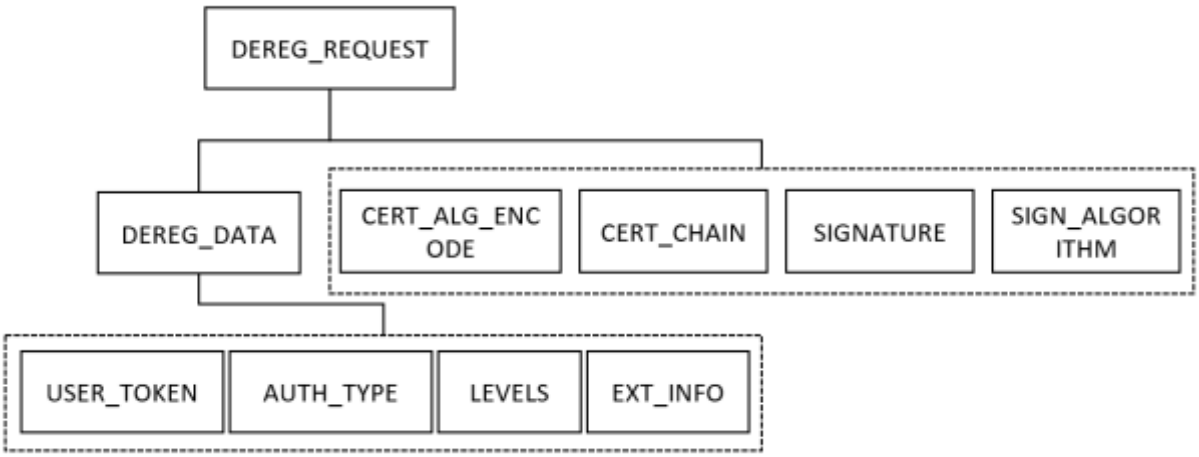


各节点语义如下:

字段	类型	值	说明
AUTH_RESPONSE	uint16_t	0x05	注册结果的 response
SIGNED_DATA	uint16_t	0x08	Response 有效信息
SIGN_ALGORITHM	uint16_t	0x8008	签名算法，支持 RSA2048,ECC256 和国密
SIGNATURE	uint16_t	0x8007	签名内容
LEVEL	uint16_t	0x8011	客户端支持的安全等级
DEVICE_ID	uint16_t	0x800D	设备 id
CHALLENGE	uint16_t	0x8001	IFAA 服务端下发的防止重发攻击的挑战码
USER_TOKEN	uint16_t	0x8002	唯一标识用户的 id
AUTH_TYPE	uint16_t	0x800F	注册类型
AUTH_INFO	uint16_t	0X8010	生物认证注册内容，取决于类型
EXT_INFO	uint16_t	0X8004	其他扩展信息（可选）

注销 request

客户端发起注销时，需要向服务端获取一个用户注销的 request 消息。服务端先删除用户注册记录，然后将 request 消息下发到客户端。客户端根据 request 中的 USER_TOKEN 完成数据清理。



各节点语义如下:

字段	类型	值	说明
DEREG_REQUEST	uint16_t	0x09	注销 request 的根节点
DEREG_DATA	uint16_t	0x0A	注销 request 的有效信息
CERT_ALG_ENCODE	uint16	0X8005	客户端签名的证书编码格式
CERT_CHAIN	uint16_t	0X8006	证书链
SIGN_ALGORITHM	uint16_t	0X8008	签名算法
SIGNATURE	uint16_t	0X8007	签名
USER_TOKEN	uint16_t	0X8002	注册时绑定的 token
AUTH_TYPE	uint16_t	0X800E	验证类型
LEVELS	uint16_t	0x8011	服务端支持的安全等级
EXT_INFO	uint16_t	0x8004	其他扩展信息

关于签名：签名由 IFAA 服务商的二级证书进行，下发 request 时，需要将二级甚至三级证书放置在 CERT_CHAIN 里，便于客户端进行证书链的校验。而证书链的根证书将会被内置在客户端里。

证书链验证

注册 request,注册 response,校验 request,注销 request 中的签名将使用证书连签名的方式。
考虑到不同平台对 x509 和 pkcs#7 的支持情况，目前预留了两种证书链编码格式。

```
typedef enum {
    CERT_ENCODE_ALG_X509 = 0x01,     //X509 格式
    CERT_ENCODE_ALG_IFAA,           //自定义格式，使用简单的公钥+签名的方式
    //more here...
} IFAA_CertEncodeAlgorithm;
```

证书链中将包含从签名证书(包含)到 IFAA 根证书之间的所有公钥证书。

用户密钥

注册时生成的用户公私钥，其中私钥由客户端加密存储。公钥上传服务端时需要指定公私钥类型

其中密钥类型将预留以下类型：

```
typedef enum {
    ASYM_KEY_RSA2048 = 0x01,       //RSA2048 秘钥
    ASYM_KEY_ECDSA256,             //ECC256 秘钥
    ASYM_KEY_SM2,                  //国密算法
    //more here
} IFAA_AsymmetricKeyType;
```

目前建议使用 RSA2048 的方式。
公钥的编码格式则预留以下方式:

```
typedef enum {
    KEY_ENCODE_ALG_ECC_NISTP256R1_X962_RAW = 0x01,  //ECC 裸数据
    KEY_ENCODE_ALG_ECC_NISTP256R1_X962_DER,         //ECC DER 格式
    KEY_ENCODE_ALG_RSA_2048_PSS_RAW,                //RSA2048 的裸数据
    KEY_ENCODE_ALG_RSA_2048_PSS_DER,                 //RSA2048 的 DER 格式
    KEY_ENCODE_ALG_RSA_2048_PSS_IFAA,                //N 和 E 的大整数方式
    //more here...
} IFAA_PubKeyEncodeAlgorithm;
```

如果为 RSA 密钥，建议使用其中的 N/E 大整数组织方式。

签名算法

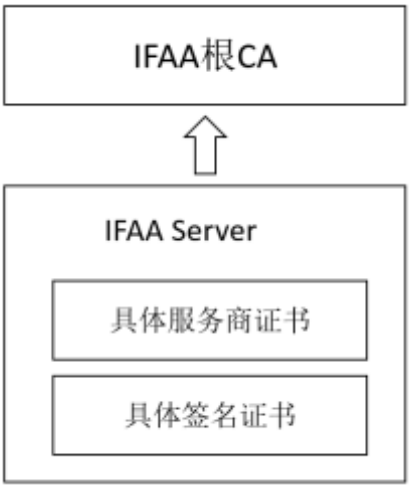
签名算法规范了 IFAA Client 和 IFAA Server 之间进行身份确认时的签名验签规则，目前 IFAA 将预留以下种类的签名算法:

```
typedef enum {
    SIGN_ALG_ECDSA_SHA256_RAW = 0x01,   //ECDSA+SHA256 的裸数据
    SIGN_ALG_ECDSA_SHA256_DER,          //ECDSA+SHA256 的 der 格式签名
    SIGN_ALG_RSA_SHA256_RAW,            //RSA+SHA256 的裸数据
    SIGN_ALG_RSA_SHA256_DER,            //RSA+SHA256 的 DER 格式签名
    SIGN_ALG_SHA256_PKCS7,              //PKCS7 格式的证书链签名
    //more here ...
} IFAA_SignAlgorithm;
```

8 安全体系部分

安全体系部分描述从IFAA TA到Framework，到REE客户端，到应用服务端，到IFAA Server再到最终的IFAA认证中心整条链路上的安全架构。

8.1 服务端架构



IFAA 将建立一个顶级根证书，并向各服务商颁发二级证书。IFAA 服务商可根据具体情况再派发三级等，具体证书和私钥作为 IFAA Server 的身份凭证。

8.2 客户端安全

厂商根据实际能力决定使用一机一密或是一型一密，体现在注册时的上传IFAA Server的安全等级字段。

对于一机一密的方式，手机在产线上将按照 IFAA 标准为每个手机生成唯一的公私钥，私钥存入 RPMB 中。公钥连同设备 ID,手机型号等一起上传到 IFAA 认证中心。IFAA 认证中心负责保存所有本地免密设备的公钥，并向外提供安全验证服务。

而对于一型一密的方式，需要厂商严格按照一型一密的原则为每个型号配置一对公私钥，并将私钥扫入 RPMB，公钥上传到 IFAA 网站。

两种方式的安全级别有所区分，并将可能实际体现在业务的应用上。

8.3 IFAA 认证中心

IFAA 将提供一个集中的鉴定手机身份的服务。主要完成注册时客户端的签名验证等操作。对于发现安全风险的客户端，将控制其使用。该服务将向 IFAA Server 会员提供服务。IFAA Server 在向 IFAA 认证中心发起调用请求时，需要使用自己的证书做签名，签名将包含完整的证书链。IFAA 认证中心将使用 IFAA 根证书来验证请求中的证书链，如果验证通过，则响应其请求。否则拒绝访问。

IFAA 认证中心维护下面的表关系：

设备 ID	设备型号	公钥	日期	其他

这些信息是在手机产线上生成私钥后，将公钥及其对应的设备信息（无论是一机一密还是一型一密）上传到 IFAA 认证中心保存。由 IFAA 认证中心为 IFAA Server 的会员提供验证手机身份的服务。产线阶段秘钥的上传细则，请参考 IFAA 提供的《生产阶段秘钥上传》文档。

8.4 客户端消息格式

客户端上传的消息将经过RPMB中的私钥签名，形式如：

8.5 服务端消息格式

服务端下发的数据需经过业务证书签名，并附加对应的证书链信息，形式如：

8.6 验证方式

8.6.1 客户端验证服务端的消息合法性

客户端通过IFAA根证书来验证服务端消息的证书链，并用具体业务证书验证消息签名的真伪。

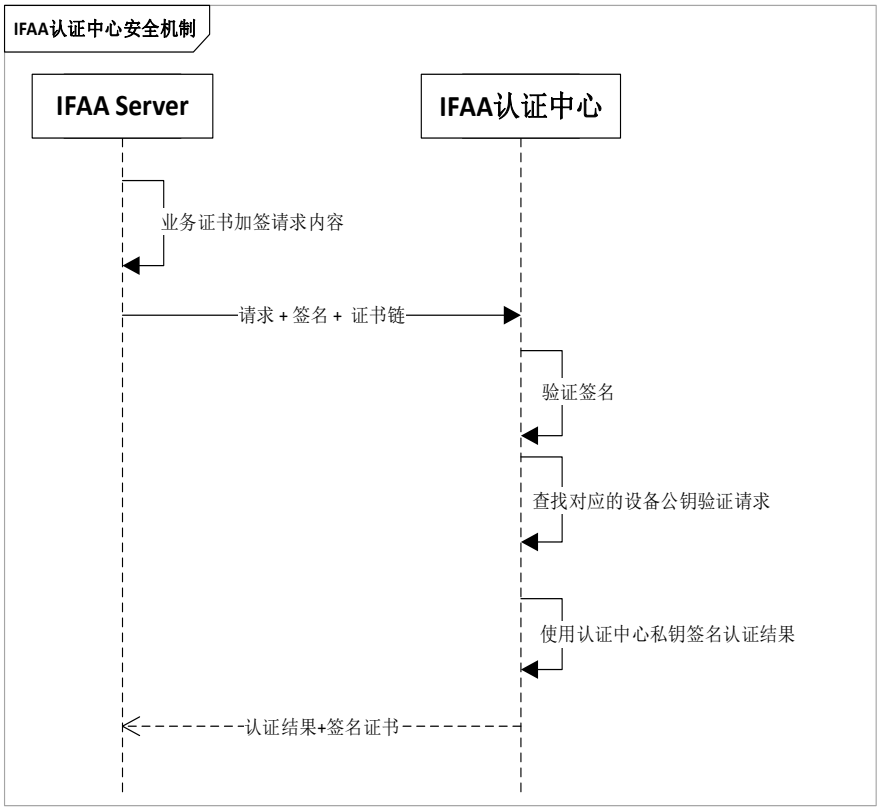
8.6.2 服务端验证客户端消息合法性

注册时:
IFAA Server将客户端上传的消息和签名连同deviceid一起发到IFAA认证中心进行验证。IFAA认证中心根据deviceid来索引到具体的设备公钥，并使用该公钥验证签名是否匹配。

校验时:
IFAA Server根据注册时保存的用户公钥来验证用户校验结果的签名，如果验证通过则继续验证用户的生物特征是否匹配。两步均验证通过后，则返回给应用服务器结果。否则返回失败，应用服务器应该信任IFAA Server的验证结果，而不是客户端的验证结果。

8.6.3 IFAA Server 与 IFAA 认证中心之间的安全验证

IFAA Server 发往 IFAA 认证中心的消息必须经过自己的私钥签名，并附加上具体的证书链，消息到达 IFAA 认证中心后，IFAA 认证中心先使用附加的证书验证签名，再使用 IFAA 根证书验证该附加证书的合法性。IFAA 认证中心回复给具体 IFAA Server 的消息将经过 IFAA 认证中心自己持有的私钥的签名。并将证书附加签名上。以防止认证结果的伪造和篡改。



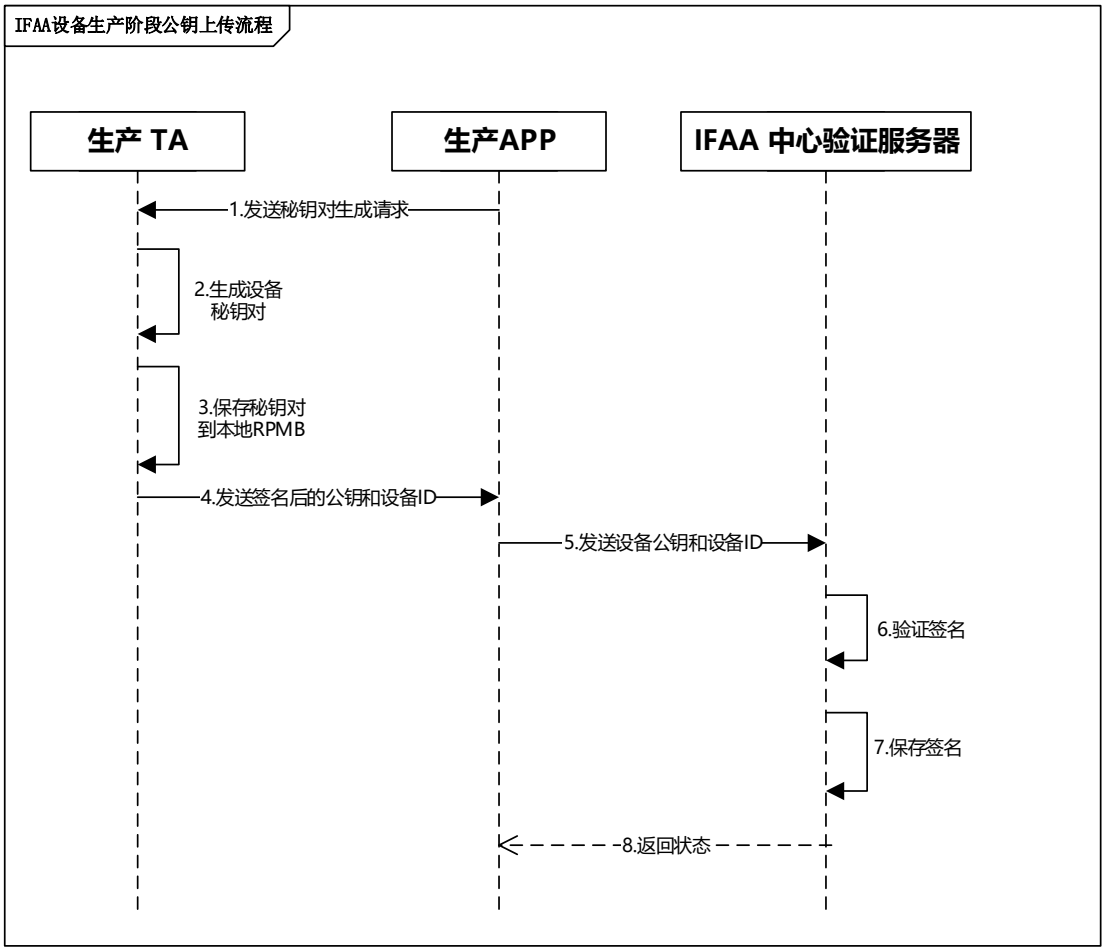
8.7 生产阶段密钥置入

8.7.1 原理

符合 IFAA 规范的设备在生产阶段需要写入一对验证机器身份的秘钥对。该秘钥对的公钥信息会通过生产用 APP 发送到 IFAA 证书服务器。IFAA 中心验证服务器会在该产品的注册阶段使用该公钥信息来判断该设备的合法性。
对于 IFAA 目前的四个安全等级，所有的等级都要求实现下面的流程，只是在设备密钥对的生成方式，和保存密钥对的位置有区别。

8.7.2 流程

在 IFAA 安全级别 Level3 的情况下，IFAA 设备生产阶段公钥上传流程如下:



流程明细:

1. 产线生产用 APP 向生产 TA 发送设备秘钥对生成请求；
2. 生产 TA 生成设备秘钥对；
3. 生产 TA 将生成的秘钥对保存到设备的 RPMB 中；
4. 生产 TA 将生成的秘钥对的公钥和设备的 ID 用私钥签名后发送到生产用 APP 中；
5. 生产用 APP 将签名后的公钥和 ID 发送到 IFAA 证书服务器中；
6. IFAA 证书服务器用收到的签名数据中的公钥验证该签名数据的完整性
7. IFAA 证书服务器保存

注：由于 RPMB 本身实现不并加密，且在部分 MTK 芯片上，REE 也可以读取 RPMB 中的内容。所以约定，存入 RPMB 中的秘钥必须经过 TEE 的加密处理。

其他安全级别在流程上与 Level 3 的区别如下：
在安全级别为 Level 1 的情况下：
步骤 2 中，不为每一个设备生成独立的密钥对，而是一种型号共用同一个密钥对；
步骤 3 中，设备的密钥对信息不存放在 RPMB 中，而是存放在 TA 中。

在安全级别为 Level 2 的情况下：
步骤 2 中，不为每一个设备生成独立的密钥对，而是一种型号共用同一个密钥对；

在安全级别为 Level 4 的情况下：
步骤 3 中，设备的密钥对信息不存放在 RPMB 中，而是存放在 SE 中。

所有级别，都要求在生产阶段将设备的秘钥信息与设备 ID 上传到 IFAA 的中心验证服务器中。

向 IFAA 证书服务器发送设备公钥信息的接口为：

```
http://IFAA_KEY_SERVER_ADDRESS/ifaaserver/ifaaSendDevicePubKey
```

其中 pubKeyData 的形式如下：

```
{
     “devicedata”: “xxxx” //签名后的 pubkey 和 device id（转义后）
}
```

返回值的形式为：

```
{
     “code”: xxxx,
     “status”: xxxx
}
```

其中 code 描述错误码。如：

```
ERR_SUCCESS = 0,
ERR_TIMEOUT,
ERR_WRONG_PARAMS,
  …,
ERR_UNKNOWN
```

当 code 为 ERR_SUCCESS 时，status 里包含了公钥数据的上传结果：

```
true  //上传成功
false //上传失败
```