GeeMee Android 指纹认证集成说明

一． 集成方式：

采用 Jar 包加 so 库文件的形式进行集成。集成文件名为 **myfingerprintsdkX.X.X.Jar**，**libNativefp.so** 库文件。集成时首先把 myfingerprintsdkX.X.X.Jar 引入到工程 libs 目录下，libNativefp.so 添加至 lib/armeabi 文件夹下，最后添加最 myfingerprintsdkX.X.X.Jar 包 的引用，然后引用包中的 Do0.java 类。Do0.java 为回调接口类。

示例代码： import cn.geemee.fingerprint.Do0;

```
Activity implements getauthcallback @Override public void getauthresult(String coed, final String result) { 此处为请求加密数据操作回调方法 } @Override public void getdataresult(String coed, String result) { 此处为请求解密数据操作回调方法 }
```

具体操作过程：

1. 实例化 Do0.java 类。
2. 初次使用指纹功能，需调用 register() 注册方法。（* - 此方法暂时不需集成调用*）
3. do0.setCallauthBack(code1, ac, Activity.this);
   code1 为操作码 ac 为用户唯一标识 Activity.this 为 getauthcallback 回调接口实现类，此处操作会调用刷取指纹操作。
4. 在回调接口 getauthresult 内判断 code1，得到相应返回值 result1，由用户发送给服务器，得到返回值 reData。
5. do0.setCalldataBack(code1, reData, Activity.this); code1 同上（需一致），reData 为服务器返回加密数据，Activity.this 为 getauthcallback 回调接口实现类，此处操作为解析服务器返回的加密数据。
6. 在回调接口 getdataresult 内判断 code1 返回的服务器解析值 reData1
7. do0.setCallauthBack(code2, ac, Activity.this); code2 为操作码 ac 为用户唯一标识 Activity.this 为 getauthcallback 回调接口实现类 此处操作为加密需上传服务器数据。
8. 回调接口 getauthresult 内判断 code2 返回值 result2 发送给服务器得到 reData2
9. do0.setCalldataBack(code2, reData2, Activity.this); code2 同上（需一致），reData2 为服务器返回加密数据，Activity.this 为 getauthcallback 回调接口实现类，此处操作为解析服务器返回的加密数据。

注：code1 与 code2 为同一操作内细分操作 code1 为调用刷取指纹与加密第一次上传数据 code2 为调用加密第二次上传数据

Code 对应操作：

10
`开启指纹验证功能（调用指纹刷取页面），加密首次上传服务器数据`
11
`开启指纹验证功能,加密二次上传服务器数据`
20
`调用指纹验证功能（调用指纹刷取页面），加密首次上传服务器数据`
21
`调用指纹验证功能,加密二次上传服务器数据`
30
`关闭指纹验证功能（调用指纹刷取页面），加密首次上传服务器数据`
31
`关闭指纹验证功能,加密二次上传服务器数据`
60
`获取当前账号设备指纹状态`

具体业务流程

如 开启：

1. *do0.setCallauthBack("10", ac, Activity.this)*
2. *getauthresult(String code,String result){ If(code. equals("10")) http 请求服务器 发送 result }*（此方法内应先判断 result 是否为错误码）
3. 请求服务器返回成功后 得到返回值 httpData0 调用 *do0.setCalldataBack("10", reData, Activity.this);*
4. 在 *getdataresult(String code,String result){ If(code. equals("10")) 获取服务器返回明文 }*（此方法内应先判断 result 是否为错误码）
5. *do0.setCallauthBack("11", ac, Activity.this)*，此处会调取刷指纹页面功能
6. *getauthresult(String code,String result){ If(code. equals("11")) http 请求服务器 发送 result }*（此方法内应先判断 result 是否为错误码）
7. 请求服务器返回成功后 得到返回值 httpData0 调用 *do0.setCalldataBack("11", reData, Activity.this);*
8. 在 *getdataresult(String code,String result){ If(code. equals("11")) 获取服务器返回明文，得到处理结果 }*（此方法内应先判断 result 是否为错误码）

验证：

1. *do0.setCallauthBack("20", ac, Activity.this)*

2. *getauthresult(String code,String result){ If(code. equals(“20”)) http 请求服务器 发送 result }* （此方法内应先判断 result 是否为错误码）

3. 请求服务器返回成功后 得到返回值 httpData0 调用 *do0.setCalldataBack(“20”, reData, Activity.this);*

4. 在 *getdataresult(String code,String result){ If(code. equals(“20”)) 获取服务器返回明文 }* （此方法内应先判断 result 是否为错误码）

5. *do0.setCallauthBack(“21”, ac, Activity.this)* ，此处会调取刷指纹页面功能

6. *getauthresult(String code,String result){ If(code. equals(“21”)) http 请求服务器 发送 result }*（此方法内应先判断 result 是否为错误码）

7. 请求服务器返回成功后 得到返回值 httpData0 调用 *do0.setCalldataBack(“21”, reData, Activity.this);*

8. 在 *getdataresult(String code,String result){ If(code. equals(“21”)) 获取服务器返回明文，得到处理结果 }* （此方法内应先判断 result 是否为错误码）

关闭：

1. *do0.setCallauthBack(“30”, ac, Activity.this)*

2. *getauthresult(String code,String result){ If(code. equals(“30”)) http 请求服务器 发送 result } *（此方法内应先判断 result 是否为错误码）

3. 请求服务器返回成功后 得到返回值 httpData0 调用 *do0.setCalldataBack(“30”, reData, Activity.this);*

4. 在 *getdataresult(String code,String result){ If(code. equals(“30”)) 获取服务器返回明文 }* （此方法内应先判断 result 是否为错误码）

5. *do0.setCallauthBack(“31”, ac, Activity.this)*

6. *getauthresult(String code,String result){ If(code. equals(“31”)) http 请求服务器 发送 result }*（此方法内应先判断 result 是否为错误码）

7. 请求服务器返回成功后 得到返回值 httpData0 调用 *do0.setCalldataBack(“31”, reData, Activity.this);*

8. 在 *getdataresult(String code,String result){ If(code. equals(“31”)) 获取服务器返回明文，得到处理结果 }* （此方法内应先判断 result 是否为错误码）

查询：

1. *do0.setCallauthBack(“60”, ac, Activity.this)*

2. *getauthresult(String code,String result){ If(code. equals(“60”)) http 请求服务器 发送 result }* （此方法内应先判断 result 是否为错误码）

3. 请求服务器返回成功后 得到返回值 httpData0 调用 *do0.setCalldataBack(“60”, reData, Activity.this);*

4. 在 *getdataresult(String code,String result){ If(code. equals(“60”)) 获取服务器返回明文 }*（此方法内应先判断 result 是否为错误码）

AndroidManifest.xml 内应用权限

```
<uses-permission android:name="android.permission.INTERNET" /> <uses-permission
android:name="android.permission.USE_FINGERPRINT" /> <uses-permission
android:name="android.permission.VIBRATE" /> <uses-permission
android:name="android.permission.READ_PHONE_STATE" /> <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" /> <uses-permission
android:name="android.permission.ACCESS_WIFI_STATE" />
```

GeeMee iOS 指纹认证中间件客户端集成开发流程

说明：此文档为客户端 iOS 接口调用开发文档，服务器端接口文档请参考服务器端接口文档。

第一步:导入指纹加密库

1. 解压压缩包
2. copy 库文件夹导入工程目录文件下
3. 工程里 Add File 到对应的目录
4. 到 Build Phases -> Link Binary With Libaries 下添加引用库 *LocalAuthentication.framework* | *libstdc++6.0.9.tbd* |openssl.framework | libFPGuard.a | SystemConfiguration.framework |libDeviceTrustAnalytics.a
5. 配置 build setting -> Link -> Other Linker Flags 参数值 -Objc
6. 当前控件最低支持 iOS 版本为 *9.0*+，带 TouchID 指纹识别机器

第二步:接口调用

01）*ViewController 中引入 FPGuard.h*

02）本地指纹开启状态查询

```
    //获取本地指纹开启状态只需要 UserID 既可查询
    //返回值 GMLA_OK 本地已经开启 ，其他 失败
    //参考代码
    GMLAStatus iret =[[FPGuard shareInstance]
canEvaluatePolicyForUserID:@"15252565244"];
```

03）开启、登录/交易、关闭指纹加密

```
//开启、登录/交易、关闭指纹加密首先需要根据使用方式生成命令上传服务器
端，然后服务器端下发客户端操作指令
    //参考：生成上传服务器端指令 1
    NSString *requestStr =  [[FPGuard shareInstance]
evaluatePolicyServerMessageS:@"18911377982"
gmLAPolicy:GMLACreate];
    //客户端上传 requestStr 之后，服务器端下发客户端操作指令，
    //客户端调用接口解析指令，返回客户端操作结果
    //参考代码
    NSArray *msg = [[FPGuard
shareInstance ]evaluatePolicyServerMessageR:@"服务器端下发客户端
操作指令];
    //返回值 arr[0] GMLAStatus 标识 arr[1]返回数据 NSString 格式
    //如果 arr[0]等于 GMLAStatusOK 需要再次将arr[1]数据提交服务器验
证，服务器返回客户端数据，客户端解析
    GMLAStatus rResult = [[FPGuard shareInstance ]
evaluatePolicyServerMessageR2:data];
    //返回值 参考 GMLAStatus
```

04） 其他

```
开启、关闭、交易、登录（此处开启、交易、关闭 解密流程一致，参考开启流
程）
```