

GeeMeeSDK Android 手势密码控件集成说明

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 **geemee-lockpatternsdk.X.X.Jar**, **libGeeMeeSDKBase.so**, **libGesture.so**, **libgnustlshared.so** 库文件。集成时首先把 **geemee-lockpatternsdkX.X.X.Jar** 引入到工程 **libs** 目录下, **libGeeMeeSDKBase.so**、**libGesture.so**、**libgnustlshared.so** 添加至 **lib/armeabi** 文件夹下, 最后添加 **geemee-lockpatternsdkX.X.X.Jar** 包的引用, 然后引用包中的 **cn.geemee.lockpatternsdk.LockPattern.java** 类。

LockIndicator.java 类为手势轨迹提示框。

GestureContentView.java 为手势密码容器类。

GestureCallBack.java 为手势密码回调类

引用示例代码:

```
import cn.geemee.lockpattern.view.GestureCallBack; import
cn.geemee.lockpattern.view.GestureContentView; import
cn.geemee.lockpattern.view.LockIndicator; import
cn.geemee.lockpattern_sdk.LockPattern; public class VerificationActivity extends
Activity { static { System.loadLibrary("gnustl_shared");
System.loadLibrary("Gesture"); } }
```

具体操作过程:

1. 布局文件添加控件 View, 详见 demo。
2. 实例化 **LockPattern.java** 类。
3. 设置相关手势密码属性及相关加密方式与密钥。
4. 实例化 **LockIndicator.java** 与 **GestureContentView.java** 类。
5. 添加回调支持, 重写相关方法。
6. 在回调中获取刷手势相关加密密文及明文长度、hash 值。

相关接口

LockPattern 类

1. 滑动时是否显示手势轨迹

```
public void IsGesturetrack(boolean bool) true 为显示手势轨迹 false 为不显示手势轨迹 默认为显示轨迹
```

2. 设置最短长度

```
public void minLength(int length) length 应小于 9 大于 1
```

3. 设置是否显示提示框

```
public void IsTracePromptBox(boolean bool) true 为显示, 此时需传入提示框实例 false 为不显示 默认不显示
```

4. 设置密文为 AES 加密方式及 key

```
public void EncryptionModeAES(String AES_Key) AES_Key 为 AES 加密随机数 (不可为空) 此方法设置后, 加密返回 AES 加密密文 base64 编码
```

5. 设置密文为 AES+RSA 加密方式及 key

```
public void EncryptionModeAESRSA(String AES_Key,String RSA_Key) AES_Key 为 AES 加密随机数 (不可为空) RSA_Key 为 RSA 加密 key (不可为空 此方法设置后, 加密返回 AES+RSA 加密密文 RSA 密文 16 进制编码 AES 密文 base64 编码
```

6. 设置密文为 SM2 加密方式及 key

```
public void EncryptionModeSM2(String SM2_Key) SM2_Key 为 SM2 加密 key (不可为空) 此方法设置后, 加密返回 SM2 加密密文 16 进制编码
```

7. 设置密文为 SM4 加密方式及 key

```
public void EncryptionModeSM4(String SM4_Key) SM4_Key 为 SM4 加密随机数 (不可为空) 此方法设置后, 加密返回 SM4 加密密文 base64 编码
```

8. 设置密文为 SM2+SM4 加密方式及 key

```
EncryptionModeSM2SM4(String SM2_Key,String SM4_Key) SM2_Key 为 SM2 加密 key (不可为空) SM4_Key 为 SM4 加密随机数 (不可为空) 此方法设置后, 加密返回 SM2+SM4 加密密文 SM2 密文 16 进制编码 SM4 密文 base64 编码
```

9. 设置正确时轨迹颜色

```
public void setLineColor(int color) color 为 Color 类 int 颜色值 默认为黑色 Color.BLACK
```

10. 设置错误时轨迹颜色

```
public void setWrongLineColor(int color) color 为 Color 类 int 颜色值 默认为红色 Color.RED
```

11. 获取手机设备唯一编码

```
public static String getAndroidID() 返回 32 为手机唯一编码
```

12. 设置连接线粗细（默认为 10）

```
public void setLineWidth(int width) 默认为 10dp 可根据实际情况调整
```

GestureContentView 类

1. 实例化

```
public GestureContentView(Context context, boolean isVerify, LockIndicator mLockIndicator, GestureCallback callBack) context isVerify 是否为验证 mLockIndicator 手势轨迹提示框实例（无提示框时请传入 null） callBack GestureCallback 类回调实现类
```

GestureCallback 类

1. 获取用户输入了手势密码的长度

```
public abstract void onGestureCodeInputLength(int inputCodeLength)
```

2. 获取用户输入了手势密码的加密值

```
public abstract void onGestureCodeInputEncrypt(String inputCodeEncrypt)
```

3. 获取密码的 hash 值

```
public abstract void onGestureCodeInputHash(String inputCodeHash)
```

AndroidManifest.xml 内应用权限

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

GeeMeeSDK iOS 手势密码集成文档

极密 SDK 所有版本均完美兼容 IPv6，大家可以放心使用。

注意：

- iOS 7.1+
- Xcode 7.3+
- ARC, BitCode 支持
- 集成有问题，建议查看 [FAQ](#)，或者联系技术支持人员；

步骤

1. 解压压缩包

序号	文件	说明
1	GMGuardBundle.bundle	资源包
2	GMGuardView.h	头文件
3	GMGuardViewStatus.h	头文件
4	libGMGuardView.a	静态库文件
5	libJMSDKPassGuardCrypto.a	静态库文件

2. 解压缩

1. 将形如 GMGuardSDKiOS_x.x.x 的文件夹拖入工程目录
2. 确认勾选了“Copy items to destination's group folder”选项，并选择你要添加到的 Target

3. 系统依赖库配置

XCode APP 配置, [Build Phases](#) -> [Link Binary With Libraries](#) 里添加以下 [framework](#):

```
Security.framework | libstdc++6.0.9.tbd
```

4. 设置极密开放平台手势密码控件 SDK appLicense

1. 获取 GMSDK AppLicense, 请联系微通新成公司人员获取。
2. 在代码中设置你的极密 SDK 手势密码控件 AppLicense, 在 [AppDelegate](#) 文件内设置你的 AppLicense:

如果是 Swift 项目, 请在对应的 [bridging-header.h](#) 中导入

Objective-C

```
#import "GMGuardView.h"
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    //此处 license 建议 app 客户端配置参数, 从服务器端动态下发

    [GMGuardView setGMGVLICENSE:@"AppLicense"];

    return YES;
}
```

Swift

```
func application(application: UIApplication,
didFinishLaunchingWithOptions launchOptions: [NSObject:
AnyObject]?) -> Bool {

    //此处 license 建议 app 客户端配置参数, 从服务器端动态下发

    GMGuardView.setGMGVLICENSE("AppLicense")

    return true
}
```

5. ViewController 中调用手势密码

(A) Xib 方式集成

1. ViewController 中引入 **GMGuardView.h**(如需手势提示再引入 **GMGuardViewStatus.h**)
2. 选中 **xib** 或 **storyboard**
3. 在 *view* 内拖入一个 *UIView*
4. 设置 *GMGuardView.h* 中的 *Custom Class* 类名为 **GMGuardView.h**
5. 在类文件中, 声明 *IBOutlet*,
6. 在 xib 编辑器中 (IB) 进行关联绑定

Objective-C

```
@property(n nonatomic, retain) IBOutlet GMGuardView *glView;

@synthesize glView;
- (void)viewDidLoad {
    [super viewDidLoad];

    [glView
setm_strInput1:@"d8mujn26fstqdozzq3iijpsvx71g74du"]; //云端下发
加密因子
}
```

Swift

```
@IBOutlet weak var glView: GMGuardView!
override func viewDidLoad() {
    super.viewDidLoad()

    glView.setm_strInput1("12345678901234567890123456789012")
}
```

(B) 代码方式集成

1. ViewController 中引入 **GMGuardView.h**(如需手势提示再引入 **GMGuardViewStatus.h**)
2. 开始集成开发, 调用演示

Objective-C

```
- (void)viewDidLoad {
```

```

    [super viewDidLoad];
    GMGuardView * glView = [[GMGuardView alloc]
initWithFrame:CGRectMake(0, 0, 100, 100)];
    glView.mleapointnum = 4;//手势连接最少点数
    glView.delegate = self;//设置回调代理
    glView.mstate = YES;//显示手势轨迹
    glView.mlinecolor = [UIColor redColor];//手势连线颜色
    glView.mlinewidth = 2.0f;//手势连线宽度
    glView.m_mode = EncryptionAES;//加密方式
    [self.view addSubview:glView];
}

```

Swift

```

override func viewDidLoad() {
    super.viewDidLoad()
    let glView = GMGuardView(frame: CGRect(x: 100.0 , y: 100.0,
width: 200.0, height: 200.0))
    glView.mleapointnum = 4;
    glView.delegate = self;
    glView.mstate = YES;
    glView.mlinewidth = 2.0f;
    glView.mmode = EncryptionAES;
    self.view.addSubview(glView)
}

```

更多使用参数说明 请参考 [iOS 极密 SDK 手势密码代码接口说明文档](#)

6.手势密码回调输出接口

手势密码回调代理接口

```

-(void)getactionSHA1:(NSString *)sha1 output:(NSString
*)password errCode:(int)err length:(int)len;

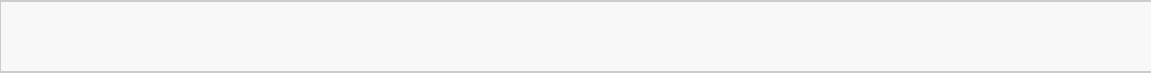
```

设备唯一性获取接口

```

+(NSString *)getTrustUDID;

```



编译运行 App，滑动手指，回调代理能获取密文，成功了！

7.密文解密请参考 极密密码控件 SDK PC 端 SDK。