Android极密密码控件集成说明

最新版极密密码控件*SDK*压缩包下载

一． 集成方式：

采用Jar包+.so库的形式进行集成。集成文件名为passguard.Jar，libGeeMeeSDKBase.so,libgnustl_shared.so，libPassGuard.so。 集成时首先把passguard.Jar引入到工程libs目录下最后拷贝libPassGuard.so库文件到libs目录对应文件夹下，然后引用包中的PassGuardEdit类。 示例代码：

> Import cn..passguard.PassGuardEdit;

然后于layout xml中放置cn..passguard.PassGuardEdit使用，参数设置与Edittext基本一致，与EditText重合的接口请使用下面接口文档提供的接口。

二． 接口以及调用示例

A) 属性设置接口

1. 设置开发权限

```
属性名称:void setAppLicense(String license)
参数: String license
备注: 设置License开发权限。通常给用户的是临时开发权限，期限为三个月，权限过期会提示弹框。确定合同
返回值:    无
```

2. 设置输入框明文密文模式

```
属性名称:   void setInputMode(boolean need)
参数: boolean need
备注: 是否为加密状态。传入true为加密状态开启，false为普通输入框。默认为true。设置为false，将以明
返回值:    无
```

3. 设置键盘按键状态

```
属性名称:   void setKeyPressMode(boolean need)
参数: boolean need
备注: 设置键盘的按键状态。传入true为有按键状态，false为没有。默认构造false。有按键状态下，截屏容易
返回值:    无
```

4. 设置按键动画效果

```
属性名称    void setButtonPressAnim(boolean need)
参数  boolean need
备注  设置键盘按键动画状态。传入true为有动画效果，false为没有。默认为false。有动画效果的状态下，
返回值 无
```

5. 设置加密32位随机数

```
属性名称    void setCipherRandom(String key)
参数  String key
备注  设置32位随机字符串。此串用于AES/SM4等的加密。
返回值 无
```

6. 设置RSA公钥

```
属性名称    void setRSAPublicKey(String key)
参数  String key
备注  设置RSA加密公钥。此串用于RSA算法的加密。
返回值 无
```

7. 设置国密SM2公钥

```
属性名称    void setSM2PublicKey(String key)
参数  String key
备注  设置SM2加密公钥。此串用于SM2算法的加密。
返回值 无
```

8. 设置是否显示纯数字键盘

```
属性名称    void useNumberPad(boolean need)
参数  Boolean need
备注  表示此键盘是否只使用数字键盘。true为仅使用数字键盘，false为使用字母和数字全键盘（默认）。
返回值 无
```

9. 设置输入内容最大长度

```
属性名称    void setMaxLength(int length)
参数  Int length
备注  设置键盘输入的最大长度，传入大于0的数字，默认限制为100
返回值 无
```

10. 设置键盘是否乱序

```
属性名称    void setKeyOrderType(int reorder)
```

参数　Int reorder
备注　设置键盘是否乱序。有三种模式：
　　0/KEY_NONE_CHAOS，默认不乱序。
　　1/KEYCHAOS_SWITCH_VIEW，初始化后只乱一次。
　　2/KEY_CHAOS_PRESS_KEY，每次点击键盘是乱序一次。
设置示例：setReorder(PassGuardEdit.KEY_NONE_CHAOS);
返回值 无

11. 设置输入内容正则表达式

```
属性名称　　void setInputRegular(String regex)
参数　String regex
备注　设置键盘输入正则规则，默认为无限制。
返回值 无
```

12. 设置正则规则，输入完毕后字符类型判断条件的正则

```
属性名称　　void setResultRegular(String regex)
参数　String regex
备注　设置checkMatch()函数用于查看输入内容格式的正则表达式。默认为无。
返回值 无
```

13. 设置键盘显示状态

```
属性名称　　void setKeyBoardShowAction(doAction action)
参数　doAction action
备注　传入回调函数，此回调函数将在键盘显示时被调用。然后通过doAction 中回调isKeyBoardShowing()方
返回值 无
```

14. 设置隐藏键盘调用

```
属性名称　　void setKeyBoardHideAction(doAction action)
参数　doAction action
备注　传入回调函数，此回调函数将在键盘隐藏时被调用。然后通过doAction 中回调isKeyBoardShowing()方
返回值 无
```

B) 功能接口

1. 初始化键盘

```
属性名称　　void initPassGuardKeyBoard()
参数　无
备注　初始化键盘。必须在设置完键盘属性后调用一次，以使键盘能被正常调用
返回值 无
```

2. 清空密码框内容

```
属性名称    void clear()
参数   无
备注   调用后清空密码框内容
返回值 无
```

3. 启动键盘

```
属性名称    void StartPassGuardKeyBoard()
参数   无
备注   启动键盘
返回值 无
```

4. 隐藏键盘

```
属性名称    void hide()
参数   无
备注   隐藏键盘
返回值 无
```

C) 输出接口

1.获取RSA+AES密文

```
属性名称    String getRSAAESCiphertext()
参数   无
备注   获取用户输入数据的加密密文（RSA+AES），用Base64编码
返回值 String类型数据
```

2.获取对称加密AES 256 密文

```
属性名称    String getCipherWithType(PassGuardEdit.MICRODONE_ALGO_AES)
参数   PassGuardEdit.MICRODONE_ALGO_AES
备注   传入的PassGuardEdit.MICRODONE_ALGO_AES 表示加密的类型。获取用户输入数据的加密密文（AES），
返回值 String类型数据
```

3.获取国密SM2密文

```
属性名称    String getCipherWithType(PassGuardEdit.MICRODONE_ALGO_SM2)
参数   PassGuardEdit.MICRODONE_ALGO_SM2
备注   PassGuardEdit.MICRODONE_ALGO_SM2表示SM2加密类型。获取用户输入数据的加密密文（SM2），用Bas
返回值 String类型数据
```

4.获取国密SM3密文

属性名称　String getCipherWithType(PassGuardEdit.MICRODONE_ALGO_SM3)
参数　PassGuardEdit.MICRODONE_ALGO_SM3
备注　PassGuardEdit.MICRODONE_ALGO_SM3 表示SM3加密类型。获取用户输入数据的加密密文（SM3），用十
返回值 String类型数据

5.获取国密SM4密文

属性名称　String getCipherWithType(PassGuardEdit.MICRODONE_ALGO_SM4)
参数　PassGuardEdit.MICRODONE_ALGO_SM4
备注　PassGuardEdit.MICRODONE_ALGO_SM4 表示SM4加密类型。获取用户输入数据的加密密文（SM4），用Ba
返回值 返回 String数据

6.获取哈希值

属性名称　String getHashWithType(PassGuardEdit.ALGO_FLAG_SHA1)
参数　PassGuardEdit.ALGO_FLAG_SHA1
备注　根据需要获取的hash类型，传入参数。获取用户输入数据的MD5哈希值
返回值 String 类型

7.获取输入长度

属性名称　int getInputLength()
参数　无
备注　获取用户输入数据的长度
返回值 Int数据

8.检查输入内容是否匹配

属性名称　boolean checkMatch()
参数　无
备注　返回boolean值，根据setMatchRegex函数所设置的正则表达式尝试匹配当前输入框内容，匹配返回true，
返回值 Boolean 返回值

9.获取输入内容的类型

属性名称　int[] getInputLevel()
参数　无
备注　返回一个包含两个int的数组，int[0]返回当前输入框内容的组成结构：
完全为空，返回0。
仅有数字，字符或特殊符号为1。例：”1234””abcd””%#@!”
有两种组合返回2。例:”12bd””12@#””ab@#”
有三种组合返回3。例：”1@b””1@2ab3”
Int[1]返回判断当前输入框内容是否为简单密码。若内容为8位以下且是连续字符则为简单密码，返回1。否
返回值 Int[]

10.判断键盘是否显示

```
属性名称    boolean isKeyBoardShowing()
参数  无
备注  根据当时键盘是否显示返回boolean值，true为正在显示，false为没有显示。
返回值 返回boolean类型
```

11.获取键盘高度

```
属性名称    int getKeyboardHight()
参数  无
备注  获取键盘高度（px）
返回值 Int类型
```

12.获取SM2+SM4密文

```
属性名称    String getSM2SM4Ciphertext()
参数  无
备注  获取用户输入数据的加密密文（SM2+SM4），用HEX编码
返回值 String类型数据
```

GeeMeeSDK iOS密码控件集成文档

> 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
- 建议不要与其他带有安全输入的SDK同时使用，可能影响功能的正常使用；
- 集成有问题，建议查看 FAQ，或者联系技术支持人员；

步骤

1. 下载最新SDK压缩包

序号	文件	说明
1	GMPassGuardBundle.bundle	资源包
2	GMPassGuardCtrl.h	头文件
3	GMPassGuardProtocol.h	头文件
4	GMPassGuardViewController.h	头文件
5	libcrypto.a	静态库文件
6	libGMPassGuardCtrl.a	静态库文件

2. 解压缩

1. 将形如GM*PassSDK*iOS_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 | AudioToolbox.framework | libstdc++6.0.9.tbd
```

4. 设置极密开放平台密码安全控件SDK appLicense

1. 获取GMSDK AppLicense。如果你之前已经在极密开放平台注册了应用，获得了AppLicense，可以继续使用之前获得AppLicense。
2. 如果你尚未在极密开放平台注册账号，需要先注册，注册之后登录你的账号，点击添加新应用，完成新应用填写之后，将进入应用管理页面。在该页面就能得到AppLicense。
3. 在代码中设置你的极密SDK安全输入控件 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 "GMPassGuardCtrl.h"
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
            //此处license建议app客户端配置参数，从服务器端动态下发

            [GMPassGuardTextField setAppLicense:@"AppLicense"];

            return YES;
    }
```

Swift

```
func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NS

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

            GMPassGuardTextField.setAppLicense("AppLicense")

            return true
}
```

5.ViewController 中调用密码键盘

(A) Xib 方式集成

1. ViewController 中引入 **GMPassGuardCtrl.h**
2. 选中**xib**或**storyboard**
3. 在*view*内拖入一个*UITextField*
4. 设置*UITextField*中的*Custom Class*类名为**GMPassGuardTextField**
5. 在类文件中，声明IBOutle，
6. 在xib编辑器中（IB）进行关联绑定

Objective-C

```
@property(nonatomic, retain) IBOutlet PassGuardTextField *mmytextfield1;

@synthesize mmytextfield1;
- (void)viewDidLoad {
     [super viewDidLoad];

     [m_mytextfield1 setCipherRandom:@"d8mujn26fstqdozzq3iijpsvx71g74du"];//云端下发加密因
}
```

Swift

```
@IBOutlet weak var mtextfield1: GMPassGuardTextField!
override func viewDidLoad() {
    super.viewDidLoad()

    mtextfield1.setCipherRandom("12345678901234567890123456789012")
}
```

(B) 代码方式集成

1. ViewController 中引入 **PassGuardCtrl.h**
2. 开始集成开发，调用演示

Objective-C

```
- (void)viewDidLoad {
     [super viewDidLoad];
    GMPassGuardTextField *mmytextfield1 = [[GMPassGuardTextField alloc] initWithFrame:CGR
    mmytextfield1.borderStyle = UITextBorderStyleRoundedRect;
    [mmytextfield1 setMiMaxLen:32];//控件输入最大长度
    [mmytextfield1 setCipherRandom:@"d8mujn26fstqdozzq3iijpsvx71g74du"];//云端下发加密因
    [self.view addSubview:mmytextfield1];
}
```

Swift

```
override func viewDidLoad() {
    super.viewDidLoad()
    let mmytextfield1 = PassGuardTextField(frame: CGRect(x: 100.0 , y: 100.0, width: 200.
    mmytextfield1.borderStyle = UITextBorderStyle.RoundedRect
    mmytextfield1.setCipherRandom("12345678901234567890123456789012")
    self.view.addSubview(mmytextfield1)
}
```

更多使用参数说明 请参考 *iOS*极密*SDK*安全控件代码接口说明文档

6.密码键盘密文输入接口

密文输出目前支持HASH输出、密文输出、密文长度等，详见代码接口文档

```
- (NSString *)getCipherWithType:(GMKBCipherType)cipherType;
- (NSString *)getHashWithType:(GMKBHASHType)hashType;
```

编译运行 App，点击输入框，安全键盘成功弹起，键盘输入，并能获取密文，成功了！

7.密文解密请参考 极密密码控件**SDK PC**端**SDK**。

GeeMeeSDK PC密码控件集成文档

简介

> 以下以GeeMeeSDK PC密码控件集成为例进行说明，更多详细代码请参考”示例代码”中的geesdkpg.js、demo.js、login.jsp、changePass.jsp文件，控件接口及参数见《GeeMeeSDK PC密码控件JavaScript接口文档.html》

下载

> 点此下载GeeMeeSDK PC密码控件相关文件。点此下载

示例代码介绍

文件	位置	说明
/DemoGeeMeeSDKPG.war中/js目录下的gmsdkpg.js	放到您的项目js目录，在需要集成密码控件的页面引用	密码控件JavaScript接口脚本
/DemoGeeMeeSDKPG.war中/WEB-INF/lib目录下的AESWithJCE.jar	放到您的项目/WEB-INF/lib/目录下	密码控件解密jar包,提供了算法解密库，详见”密码卫士安全控件AESWithJCE.jar接口说明.docx”
/DemoGeeMeeSDKPG.war中/ocx目录	此目录放的是密码控件安装包，放到您的项目可以提供给用户下载的目录，确保可以在浏览器中通过URL地址下载	控件安装包文件：PassGuardEdge.exe Windows PassGuardEdgeMac.pkg Mac
/DemoGeeMeeSDKPG.war中/根目录	示例代码根目录下的文件只是为了演示如何集成，无需放到您的项目中	login.jsp 登录场景集成页面 changePass.jsp 修改密码场景集成页面

开始集成

> 首先将json2.js(JSON工具类)、crypto-js.js(js加密算法库)、gmsdkpg.js(GeeMeeSDK PC密码控件核心js库)引入到页面中。

第一步

> 定义密码控件对象，参数，如下:

```
<script language="javascript" src="./js/json2.js"></script>
<script language="javascript" src="./js/crypto-js.js"></script>
<script language="javascript" src="./js/gmsdkpg.js"></script>
<script type="text/javascript">
    //定义控件对象pgeditor
    var pgeditor = new GeeMeeSDKPG({
        pgID : "_ocx_password",//控件ID，不可重复且不可跟页面其他元素重复
        pgREOne : "[\\s\\S]*",//正则1，限制输入过程中的字符类型
```

```
        pgRETwo : "[\\s\\S]{6,12}",//正则2，用pwdValid()方法判断输入的密码是否匹配此正则
        pgMaxLen : 12,//最大输入长度
        pgTabIndex : 2,//tab键顺序
        pgPlaceHolder : "请输入密码",//placeholder值,不支持IE6-IE8
        pgClass : "ocx_style",//安装控件后控件框的样式
        pgInstallClass : "ocx_style",//未安装控件时控件框的样式
        pgOnkeydown :"FormSubmit()",//监听回车事件
        pgOnfocus:"pgFocus()",//监听onfocus事件
        pgOnblur:"pgBlur()",//监听onblur事件
        pgWindowID:"password" + new Date().getTime()+1//控件实例窗口ID值，不可重复且不可跟页面其他元素重复
    });
    //定义公共参数
    //定义RSA公钥
    pgUtil.settings.pgRSAPubKey = "3081890281810092D9D8D04FB5F8EF9B8374F21690FD46FDBF49B40EECCDF416B4E2AC204
    pgUtil.settings.pgPath = "./ocx/";//控件安装包下载路径，也可以定义成绝对路径：如"http://www.microdone.cn/ocx/
</script>
```

第二步

> 绘制密码框，初始化密码控件，如下：

```
//绘制密码框，初始化密码控件代码要写到window.onload里面。
window.onload = function(){
        //绘制密码框
        pgeditor.pgGetEById("_ocx_password_str").innerHTML = pgeditor.loadpgHtml();
        //申请通信加密参数
        Ajax.request( {
            url : "./getCommEncryDatas.jsp?" + pgUtil.pgGetTime(),
            type : "GET",
            async : true,
            success : function(xhr){
                var data = xhr.responseText;
                var o = data.split(",");
                pgUtil.settings.pgRandkey = o[0];
                pgUtil.settings.pgDatab = o[1];
            }
         });
         //初始化密码控件
         pgUtil.pgInit(function(isInstalled){
            //isInstalled可以判断出当前是否已经安装密码控件
            //如果您需要根据isInstalled在页面呈现不同的提示，相关代码需要在此处写。
         });
}
```

GeeMeeSDK PC密码控件解密库部署文档

简介

> 此文档对密码控件解密库的部署做了详细的阐述。密码控件的解密库用到了JNI技术。

关于JNI

> JNI是Java Native Interface的缩写，中文为JAVA本地调用。从Java1.1开始，Java Native Interface(JNI)标准成为java平台的一部分，它允许Java代码和其他语言写的代码进行交互。JNI一开始是为了本地已编译语言，尤其是C和C++而设计的，但是它并不妨碍你使用其他语言，只要调用约定受支持就可以了。使用java与本地已编译的代码交互，通常会丧失平台可移植性。

平台

> 支持Windows、Linux系统

Windows环境部署

1.检测当前Java环境，部署本地库。

将JNI库GeeMeeSDKBaseSO.dll放到JDK/JRE安装目录的bin目录下。
如果找不到JDK/JRE安装目录，可以根据以下步骤进行查找： ①执行如下代码：

```
//java.library.path 表示java lib路径
System.out.println(System.getProperty("java.library.path"));
```

得到类似以下打印信息：

```
C:\Program Files\Java\jre6\bin;C:\windows\Sun\Java\bin;C:\windows\system32;C:\windows;G:/programs/Genuitec/C
```

一般选用Program Files目录下(因为JDK/JRE默认安装目录是Program Files下)。

2.部署jar包。 将GeeMeeSDKPG.jar(密码控件核心库)、MicrodoneSMF.jar(算法依赖库)导入到项目中。

Linux环境部署

1.部署本地库。

将JNI库libGeeMeeSDKBaseSO.so放到/usr/lib64目录下，修改so库的权限。
chmod 777 /usr/lib64/libGeeMeeSDKBaseSO.so

2.部署jar包。将GeeMeeSDKPG.jar(密码控件核心库)、MicrodoneSMF.jar(算法依赖库)导入到项目中。

GeeMeeSDK PC密码控件解密库Java接口说明文档

简介

> 此文档对GeeMeeSDKPG.jar每个方法做了详细的阐述。

方法

需要下发给客户端相关方法

1.获得32位随机因子。

> 用于下发给客户端，设置给密码控件，加密时候用到。

方法定义：

```
/*
 *生成32位随机因子
 *隶属于ocx.geemeepg.GetRandom类。
 *返回值 String类型的随机因子
 */
public static String generateString(int length)
```

调用方式：

```
//生成32位随机因子
String randKey = GetRandom.generateString(32);
```

2.获得通信加密参数。

> 通信加密用到随机因子和数据b，联合方法1生成。

方法定义：

```
/**
 * AES加密接口
 * 隶属于ocx.geemeepg.GeeMeeSDKPG类。
 * transferKey 32位随机因子
 * data 待加密明文
 * @return 密文(以base64格式输出)
 */
public static String getCipher(String transferKey, String data)
```

调用方式：

```
//先获得随机因子
String  sKey = GetRandom.generateString(32);
//根据随机因子生成数据b
String enStr = GeeMeeSDKPG.getCipher(sKey,sKey);
//注意：需要将sKey&enStr成对下发到客户端。
```

密钥对生成相关方法

1.RSA算法密钥对生成。

> 用于生成RSA算法的公私钥对。

方法定义:

```
/**
 * 生成RSA公私钥
 * 隶属于ocx.geemeepg.GeeMeeSDKPG类。
 * size 密钥位数：只支持1024或2048
 * @return RSA公私钥String数组
 */
public static String[] generateRSAKeys(int size)
```

调用方式:

```
//生成2048位的RSA公私钥对
String[] strs = GeeMeeSDKPG.generateRSAKeys(2048);
//strs[0]表示公钥,将此值交给客户端集成人员，定义控件对象时设置到js代码中。
System.out.println("pubkey:" + strs[0]);
//strs[1]表示私钥
System.out.println("prikey:" + strs[1]);
```

2.SM2算法密钥对生成。

> 用于生成SM2算法的公私钥对。

方法定义:

```
/**
 * 生成SM2公私钥
 * 隶属于ocx.geemeepg.GeeMeeSDKPG类。
 * @return SM2公私钥String数组
 */
public static String[] generateSM2Keys()
```

调用方式:

```
//生成SM2公私钥String数组
String[] strs = GeeMeeSDKPG.generateSM2Keys();
String x = strs[0];//代表公钥X值
String y = strs[1];//代表公钥Y值
String z = strs[2];//代表私钥
//按x|y的格式把公钥交给客户端集成人员，定义控件对象时设置到js代码中。
System.out.println("pubkey:" + x + "|" + y);
//打印私钥
System.out.println("prikey:" + z);
```

解密相关方法

1.AES解密。

> 用于解密单层AES算法的密文。

方法定义：

```
/**
 * AES解密接口
 * 隶属于ocx.geemeepg.GeeMeeSDKPG类。
 * @param transferKey
 *            32位随机因子
 * @param data
 *            待解密密文
 * @return 1.密码明文2.当随机因子跟密文对应不上时返回乱码或空
 */
public static String getAESResult(String transferKey, String data)
```

调用方式：

```
//AES密文
```

```
String str = "k2mJmjyamZ9Te7hsJmy8Yg==";
//32位随机因子
String key = "12345678901234567890123456789012";
//执行解密
String str2 = GeeMeeSDKPG.getAESResult(key, str);
//打印明文
System.out.println("str:" + str2);
```

2.SM4解密。

> 用于解密单层SM4算法的密文。

方法定义：

```
/**
 * SM4解密接口
 * 隶属于ocx.geemeepg.GeeMeeSDKPG类。
 * @param transferKey
 *            32位随机因子
 * @param data
 *            待解密密文
 * @return 1.密码明文2.当随机因子跟密文对应不上时返回乱码或空
 */
public static String getSM4Result(String transferKey, String data)
```

调用方式：

```
//SM4密文
String str = "jcOzlSSa5HDXjvcov73JbA==";
//32位随机因子
String key = "12345678901234567890123456789012";
//执行解密
String str2 = GeeMeeSDKPG.getSM4Result(key, str);
//打印明文
System.out.println("str:" + str2);
```

3.3DES解密。

> 用于解密3DES算法的密文。

方法定义：

```
/**
 * 3DES解密接口
 * 隶属于ocx.geemeepg.GeeMeeSDKPG类。
 * @param transferKey
 *            32位随机因子
 * @param data
 *            待解密密文
 * @return 1.密码明文2.当随机因子跟密文对应不上时返回乱码或空
 */
public static String get3DESResult(String transferKey, String data)
```

调用方式：

```
//3DES密文
String str = "0Xba31owIBcpF2hTTin16A==";
//32位随机因子
String key = "12345678901234567890123456789012";
//执行解密
String str2 = GeeMeeSDKPG.get3DESResult(key, str);
//打印明文
System.out.println("str:" + str2);
```

4.RC4解密。

> 用于解密RC4算法的密文。

方法定义：

```
/**
 * RC4解密接口
 * 隶属于ocx.geemeepg.GeeMeeSDKPG类。
 * @param transferKey
 *            32位随机因子
 * @param data
 *            待解密密文
 * @return 1.密码明文2.当随机因子跟密文对应不上时返回乱码或空
 */
public static String getRC4Result(String transferKey, String data)
```

调用方式：

```
//RC4密文
String str = "3RqKlJ9by59d";
//32位随机因子
String key = "12345678901234567890123456789012";
//执行解密
String str2 = GeeMeeSDKPG.getRC4Result(key, str);
//打印明文
System.out.println("str:" + str2);
```

5.(AES+RSA)解密。

> 用于解密(AES+RSA)双层加密算法的密文。

方法定义：

```
/**
 * AES+RSA解密接口
 * 隶属于ocx.geemeepg.GeeMeeSDKPG类。
 * @param transferKey
 *            32位随机因子
 * @param data
 *            待解密密文
 *
 * @param priKey RSA私钥字符串
 * @return 1.密码明文2.当随机因子跟密文对应不上时返回乱码或空
 */
public static String getAESRSAResult(String transferKey, String data ,String priKey)
```

调用方式：

```
//(AES+RSA)密文
String str = "xOYhNuLPHburqkQ0mm8b347AS0Pul7/pVPZDuc4KEjTL4PHqCcZFJSq6XFiEVwUACzUkqmGzH4pOfCxCFmlt46vCmCbLJG
//32位随机因子
String key = "12345678901234567890123456789012";
//私钥字符串(公私钥由生成密钥对生成方法生成)
String priKey = "308204A30201000282010100A35C34C7478C8FA2AD2762FC6E46F67F86BC4F0FC79CC3B76F157ACFC92408072CA
//执行解密
String str2 = GeeMeeSDKPG.getAESRSAResult(key, str , priKey);
//打印明文
System.out.println("str:" + str2);
```

6.(SM4+SM2)解密。

> 用于解密(SM4+SM2)双层算法的密文。

方法定义：

```
/**
 * SM4+SM2解密接口
 * 隶属于ocx.geemeepg.GeeMeeSDKPG类。
 * @param transferKey
 *            32位随机因子
 * @param data
 *            待解密密文
 *
 * @param priKey RSA私钥字符串
 * @return 1.密码明文2.当随机因子跟密文对应不上时返回乱码或空
 */
public static String getSM4SM2Result(String transferKey, String data ,String priKey)
```

方法调用：

```
//(SM4+SM2)密文
String str = "YeDPUakNSzVvoNbyUhD1Sr9oXlvD6che6sl5p05oQ2bQTXmsVK4ICop+AsqxH39Ic8fdX0wtR9r9YczhB84tFKzSJVObI6
//32位随机因子
String key = "12345678901234567890123456789012";
//SM2私钥字符串(SM2私钥字符串由密钥对生成方法生成)
String priKey = "d7af99cdc5c97279dd86744eb39da6a891cb8159f3956d64c6dff765a8297bba";
//执行解密
String str2 = GeeMeeSDKPG.getSM4SM2Result(key, str , priKey);
//打印明文
System.out.println("str:" + str2);
```

GeeMeeSDK PC密码控件JavaScript接口文档

简介

> 此文档对gmsdkpg.js中每个方法和公共变量进行了详细的阐述。

方法

功能	方法名	参数说明	详述
构造方法			

构造	GeeMeeSDKPG(option)	指定控件各个参数的信息值	定义密码控件对象时用此方法。如:
			`var pgeditor = new GeeMeeSDKPG({`
			`//控件ID，不可重复且不可跟页面其他元素重复`
			`pgID : "_ocx_password",`
			`pgREOne : "[\\s\\S]*",//正则1，限制输入过程中的字符类型`
			`//正则2，用pwdValid()方法判断输入的密码是否匹配此正则`
			`pgRETwo : "[\\s\\S]{6,12}",`
			`pgMaxLen : 12,//最大输入长度`
			`pgTabIndex : 2,//tab键顺序`
			`pgPlaceHolder : "请输入密码",//placeholder值,不支持IE6-IE8`
			`pgClass : "ocx_style",//安装控件后控件框的样式`
			`pgInstallClass : "ocx_style",//未安装控件时控件框的样式`
			`pgOnkeydown :"FormSubmit()",//监听回车事件`
			`pgOnfocus:"pgFocus()",//监听onfocus事件`
			`pgOnblur:"pgBlur()",//监听onblur事件`
			`//控件实例窗口ID值，不可重复且不可跟页面其他元素重复`
			`pgWindowID:"password" + new Date().getTime()+1`
			});`

加密相关			

获得AES密文	pwdResultAES(callf)	回调函数	Base64(AES(pass));采用AES算法直接对pass加密，密文以Base64格式输出。
			调用方法:
			`pgeditor.pwdResultAES(function(enStr){`
			`//enStr就是密码的AES密文`
			`});`

获得(AES+RSA)密文	pwdResultAESRSA(callf)	回调函数	Base64(AES(Hex(RSA(pass))))，先用RSA算法对pass加密，以Hex格式输出得到密文A，再以AES算法对密文A加密，以Base64格式输出得到密文B。
			调用方法:
			`pgeditor.pwdResultAESRSA(function(enStr){`
			`//enStr就是密码的(AES+RSA)密文`
			`});`

获得SM4密文	pwdResultSM4(callf)	回调函数	Base64(SM4(pass));采用SM4算法直接对pass加密，密文以Base64格式输出。
			调用方法:
			`pgeditor.pwdResultSM4(function(enStr){`
			`//enStr就是密码的(SM4)密文`
			`});`

获得(SM4+SM2)	pwdResultSM4SM2(callf)	回调函数	Base64(SM4(Hex(SM2(pass))))，先用SM2算法对pass加密，以Hex格式输出得到密文A，再以SM4算法对密文A加密，以Base64格式输出得到密文B。 调用方法: `pgeditor.pwdResultSM4SM2(function(enStr){`

			//enStr就是密码的(SM4+SM2)密文 });

获得3DES密文	pwdResult3DES(callf)	回调函数	Base64(3DES(pass));采用3DES算法直接对pass加密，密文以Base64格式输出。
			调用方法:
			pgeditor.pwdResult3DES(function(enStr){
			//enStr就是密码的3DES密文
			});

获得RC4密文	pwdResultRC4(callf)	回调函数	Base64(RC4(pass));采用RC4算法直接对pass加密，密文以Base64格式输出。
			调用方法:
			pgeditor.pwdResultRC4(function(enStr){
			//enStr就是密码的RC4密文
			});

获得密码的SHA1 Hash值	pwdHashSHA1(callf)	回调函数	Hex(SHA1(pass));采用SHA1算法直接对pass做Hash，Hash值以Hex格式输出。
			调用方法:
			pgeditor.pwdHashSHA1(function(hashStr){
			//hashStr就是密码的SHA1 Hash值
			});

获得密码的SHA224 Hash值	pwdHashSHA224(callf)	回调函数	Hex(SHA224(pass));采用SHA224算法直接对pass做Hash，Hash值以Hex格式输出。
			调用方法:
			pgeditor.pwdHashSHA224(function(hashStr){
			//hashStr就是密码的SHA224 Hash值
			});

获得密码的SHA256 Hash值	pwdHashSHA256(callf)	回调函数	Hex(SHA256(pass));采用SHA256算法直接对pass做Hash，Hash值以Hex格式输出。
			调用方法:
			pgeditor.pwdHashSHA256(function(hashStr){
			//hashStr就是密码的SHA256 Hash值
			});

获得密码的SHA384 Hash值	pwdHashSHA384(callf)	回调函数	Hex(SHA384(pass));采用SHA384算法直接对pass做Hash，Hash值以Hex格式输出。
			调用方法:
			pgeditor.pwdHashSHA384(function(hashStr){
			//hashStr就是密码的SHA384 Hash值
			});

获得密码的SHA512 Hash值	pwdHashSHA512(callf)	回调函数	Hex(SHA512(pass));采用SHA512算法直接对pass做Hash，Hash值以Hex格式输出。
			调用方法:
			pgeditor.pwdHashSHA512(function(hashStr){
			//hashStr就是密码的SHA512 Hash值
			});

密码判断			

获得密码长度	pwdLength(callf)	回调函数	获得密码长度，以数字类型输出。
			调用方法:
			pgeditor.pwdLength(function(len){
			//len就是密码的长度值
			});

判断密码是否符合要求(匹配正则2)	pwdValid(callf)	回调函数	判断密码是否匹配正则2，以数字类型输出，匹配返回0，不匹配返回1。
			调用方法:
			`pgeditor.pwdValid(function(isValid){`
			`  //isValid就是匹配结果`
			`});`

判断密码是否是简单密码	pwdSimple(callf)	回调函数	判断密码是否是简单密码，以数字类型输出，是简单密码返回1，不是简单密码返回0。
			简单密码判定规则：
			1.密码每个字符都一样且密码长度小于等于8，如"111111"、"aaaaaa"
			2.密码是键盘连续字符且密码长度小于等于8，
			如"123456"、"asdfgh"、"!!!!!!"
			调用方法:
			`pgeditor.pwdSimple(function(isSimple){`
			`  //isSimple就是判断结果`
			`});`

获得密码字符种类个数	pwdCharTypeNums(callf)	回调函数	获得密码字符种类个数。
			种类有以下三种：字母、数字、符号。
			调用方法:
			`pgeditor.pwdCharTypeNums(function(nums){`
			`  //nums就是字符种类个数`
			`});`

获得密码组合类型	pwdComMode(callf)	回调函数	判断密码的组合类型，以数字类型输出。
			组合类型取值范围:
			0 没有字符
			1 数字
			2 小写字母
			3 小写字母、数字组合
			4 大写字母
			5 大写字母、数字组合
			6 大写字母、小写字母组合
			7 大写子母、小写字母、数字组合
			8 符号
			9 符号、数字组合
			10 符号、小写字母组合
			11 符号、小写字母、数字组合
			12 符号、大写字母组合
			13 符号、大写字母、数字组合
			14 符号、大写字母、小写字母组合
			15 符号、大写字母、小写字母、数字组合
			调用方法:
			`pgeditor.pwdComMode(function(cm){`
			`  //cm就是密码组合类型`
			`});`

其他			
获得密码控件html代码	loadpgHtml()	/	获得绘制密码控件需要的html代码。 调用方法: `pgeditor.loadpgHtml();`
清空密码	pgClear()	/	清空已输入的密码。 调用方法: `pgeditor.pgClear();`
		s:32位随机因	给密码控件设置随机因子。 调用方法: `//32位随机因子由服务器端下发，此处写死为了方便演示`

设置随机因子	pgSetSk(s,callf)	子
		callf:
		回调
		函数
			`var s = "12345678912345678912345678912345";`
			`pgeditor.pgSetSk(s,function(){`
			`//your code`
			`});`

获得计算机网卡信息	machineMACInfo(callf)	回调
		函数
			获得计算机网卡信息，以AES密文形式Base64格式输出，即Base64(AES(MacInfo))。
			调用方法:
			`pgeditor.machineMACInfo(function(info){`
			`//info就是计算机的网卡信息`
			`});`

获得计算机CPU信息	machineCPUNum(callf)	回调
		函数
			获得计算机CPU序列号信息，以AES密文形式Base64格式输出，即Base64(AES(CPUNum))。
			调用方法:
			`pgeditor.machineCPUNum(function(info){`
			`//info就是计算机的CPU序列号信息`
			`});`

获得计算机硬盘信息	machineHDNum(callf)	回调
		函数
			获得计算机硬盘序列号信息，以AES密文形式Base64格式输出，即Base64(AES(HDNum))。
			调用方法:
			`pgeditor.machineHDNum(function(info){`
			`//info就是计算机的硬盘序列号信息`
			`});`

获得计算机硬件信息列表	machineHardList(callf)	回调
		函数
			获得计算机硬件信息列表，以AES密文形式Base64格式输出，即Base64(AES(HardList))。
			调用方法:
			`pgeditor.machineHardList(function(info){`
			`//info就是计算机的硬件信息列表`
			`});`

获得计算机硬件信息列表	machineHardList(callf)	回调
		函数
			获得计算机硬件信息列表，以AES密文形式Base64格式输出，即Base64(AES(HardList))。
			调用方法:
			`pgeditor.machineHardList(function(info){`
			`//info就是计算机的硬件信息列表`
			`});`

公共变量介绍

> pgUtil(密码控件工具类，实质是GeeMeeSDKPG的一个对象)

详细

功能	方法名/参数名	参数说明	详述
获得当前时间毫秒数	pgGetTime()	/	获得当前计算机时间毫秒数。 调用方式: `var time = pgUtil.pgGetTime();`

通过ID获得页面元素	pgGetEById(ID)	页面元素ID	通过页面元素ID获得页面元素对象。 调用方式: `var obj = pgUtil.pgGetEById("_ocx_password");`
设置安装包下载路径	pgPath	安装包下载路径	控件安装包下载路径，也可以定义成绝对路径：如"http://www.microdone.cn/ocx/" 设置方式: `pgUtil.settings.pgPath = "./ocx/";`
设置RSA公钥	pgRSAPubKey	RSA公钥	给密码控件设置RSA公钥。 设置方式: `pgUtil.settings.pgRSAPubKey="3081890281810092D9D8D04FB5F8EF9B8374F21690FD46FDBF49B40EECCDF416B4E2AC2044B0CFE3BD67EB4416B26FD18C9D3833770A526FD1AB66A83ED969AF74238D6C900403FC498154EC74EAF420E7338675CAD7F19332B4A56BE4FF946B662A3C2D217EFBE4DC646FB742B8C62`
设置SM2公钥	pgSM2PubKey	SM2公钥	给密码控件设置SM2公钥。 设置方式: `pgUtil.settings.pgSM2PubKey="59d50b4230a20162308ad2acad9976e5101acc8099a541eb2c85c7e2d74482b3|a52063179c4f5709b629dae79816ce2f293a58b105b8ce315012a73c195506bd";`

设置通信加密参数	pgRandkey&pgDatab	通信加密随机因子&密文b	设置通信加密随机因子和密文b，用于通信加密。
			设置方式:
			`//32位随机因子和密文b由服务器端下发，此处写死为了方便演示`
			`pgUtil.settings.pgRandkey="42655661800176464589854034179714";`
			`pgUtil.settings.pgDatab="HZB6/0qZ92tQb5MKDbnF9mdiE+OgGN9+/gdCF4ufAjo=";`

初始化密码控件	pgInit(callf)	回调函数	初始化密码控件，最后调用。
			调用方式:
			`pgUtil.pgInit(function(isInstalled){`
			`//isInstalled可以判断出当前是否已经安装密码控件`
			`//如果您需要根据isInstalled在页面呈现不同的提示，相关代码需要在此处写。`
			`});`