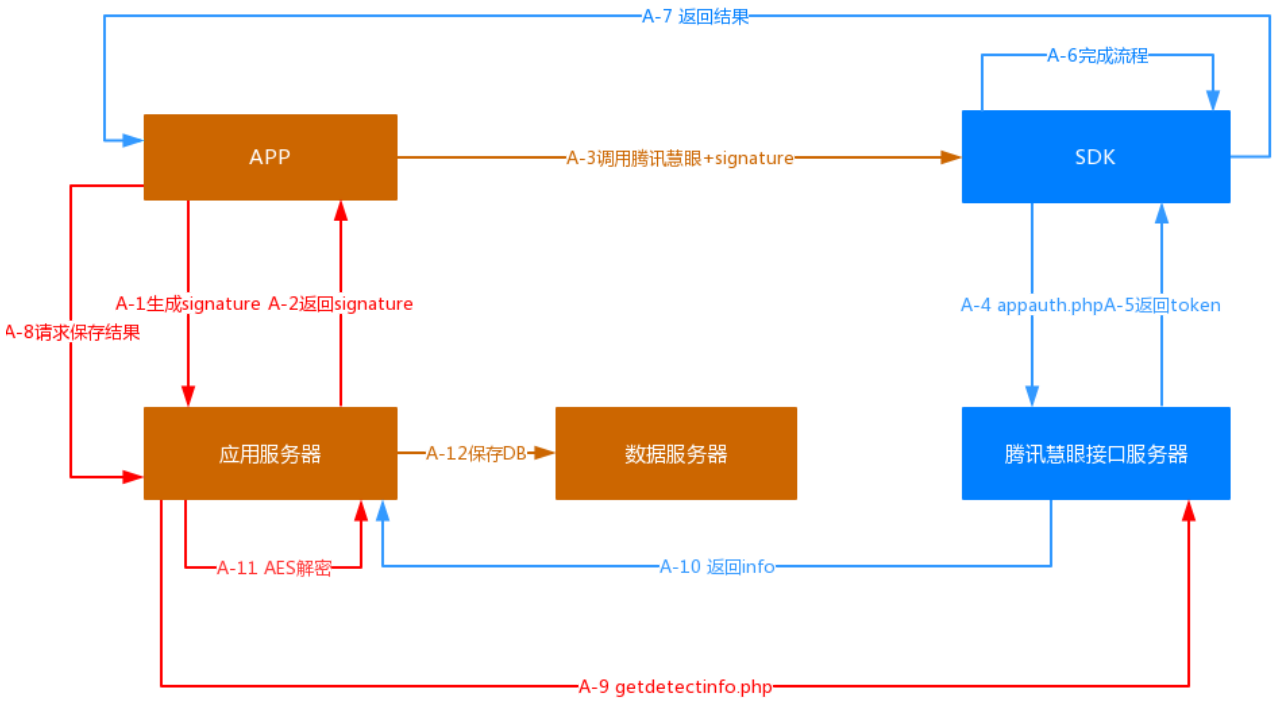
腾讯实名核身SDK说明文件 -- Android端

App业务流程框架图



其中，蓝色部分主要由腾讯侧开发，部署在腾讯侧；橙色部分由业务方开发对接，部署在调用方。

SDK功能说明

- 版本号：V1.0
- 功能：实名核身整体流程，包括身份证OCR识别和活体检测功能。

文件说明

AuthSdk.jar 封装了实名核身的流程，包括了身份证OCR识别和活体检测功能。

AuthSdk.aar 封装了实名核身的流程，包括了身份证OCR识别和活体检测功能，以及所有需要的资源。

res 包含了SDK所使用的资源文件。

assets 包含了协议内容文件。目前使用的是《测试实名核身用户须知》。

libyoutufacetrack.so 人脸识别库。

AuthDemo 提供了实名核身接口调用方法及从后台拉取活体检测详细信息的方法。接入时可参考Demo，这里提供了两个demo，对应两种接入方式。

开发运行环境

SDK版本minSdkVersion为15，支持Android4.0及以上版本

接入流程

> 接入时有两种方式；

> 一种是使用jar包+资源的方式，可以用在Eclipse和Android Studio中；

> 另外一种是使用aar方式，可以在Android Studio中使用。建议使用aar方式接入。

使用aar接入方式

参见authdemo，SDK接入流程如下：

1. 添加aar包 将AuthSdk.aar包添加到接入方APP中的libs目录下，如Demo所示

2. 进行build配置 在接入方APP的build.gradle中进行如下配置：

```
//使用aar时必须要设置的
repositories {
    flatDir {
        dirs 'libs'
    }
}

dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    compile (name:'AuthSdk', ext:'aar')
    compile 'com.android.support:appcompat-v7:23.4.0'
}
```

3.初始化SDK接口 在程序的Application中或在调用SDK之前初始化SDK，设置相关配置，包括后台服务URL及分配给业务方的AppId和SecretKey，具体参考AuthDemo。

```
  /**
     * 初始化
     * @param context app的context
     * @param serverUrl 后台server的URL
     * @param packageName 业务方的R.java的包名
     */
  AuthSDKApi.getInstance().init(this, AppConfig.IDENTITY_SDK_URL, AppConfig.APP_ID,
R.class.getPackage().getName());
```

```
    /**
     * 初始化
     * @param context app的context
     * @param serverUrl 后台server的URL
     * @param packageName 业务方的R.java的包名
     * @param phoneVerify 是否开启短信验证 true,false
     */
    AuthSDKApi.getInstance().init(this, AppConfig.IDENTITY_SDK_URL, AppConfig.APP_ID,
R.class.getPackage().getName(),true);
```

4.调用实名核身短信功能接口（更新于2017.7.19)

```
    AuthSDKApi.getInstance().setAbilityPhoneverify(boolean abilityPhoneverify);
```

参数说明： **abilityPhoneverify**：true,开启短信功能；false，不开启。该功能默认为关闭。开通appid是需申请短信功能权限。

5. 调用实名核身接口 这里提供了*五种调用方式*，可以在初始化时进行配置： 1) 正常的实名核身认证流程，包括身份证正反面OCR识别和活体检测流程

```
    AuthSDKApi.getInstance().setIdentityType(AuthSDKApi.NORMAL_IDENTITY_TYPE);
```

2) 仅有身份证正面的OCR识别和活体检测流程

```
    AuthSDKApi.getInstance().setIdentityType(AuthSDKApi.NO_BACK_OCR_IDENTITY_TYPE);
```

3) 没有OCR识别，只有活体检测流程，用户手动输入身份证号码和姓名

```
    AuthSDKApi.getInstance().setIdentityType(AuthSDKApi.NO_OCR_IDENTITY_TYPE);
```

4) 仅有活体检测流程，身份证号及姓名由外部传递

```
    AuthSDKApi.getInstance().setIdentityType(AuthSDKApi.ONLY_LIVE_DETECT_IDENTITY_TYPE);
    AuthSDKApi.getInstance().setUserInfo(idcard, name); //SDK内部有对身份证号和姓名做校验
```

5) 二次认证（更新于2017.7.24）

```
    AuthSDKApi.getInstance().setUserInfo(idcard, name); //SDK内部有对身份证号和姓名做校验
    AuthSDKApi.getInstance().setReliveSourceToken(token);//用一次验证返回的token获取比对源
    AuthSDKApi.getInstance().setIdentityType(AuthSDKApi.RELIVE_DETECT_IDENTITY_TYPE);
```

然后调用实名核身接口：

a. 启动SDK首页，需要传递一个可以*跳转到接入方中间页*的intent

```
    Intent intent = new Intent(this, OcrTestActivity.class);
    AuthSDKApi.getInstance().startMainPage(this, intent, signature, mListener);
```

在启动首页时，SDK内部会对传递来的身份证号码和姓名进行校验，只有合法的身份证号码和姓名才能启动首页及后面的实名核身流程。其中，mListener是一个IdentityCallback回调接口，调用SDK内的实名核身整体流程后，待验证成功或失败进行人工审核时，会通过回调函数返回实名核身结果和token。

b. 调用实名验证接口

```
AuthSDKApi.getInstance().startAuth(this);
```

对于前三种类型，都是直接调用SDK的入口页面即可，点击“快速验证”后会调用SDK内部的OCR流程，对于前三种类型，可进行如下调用：

```
AuthSDKApi.getInstance().startMainPage(this, intent, signature, mListener);
```

6. 根据实名核身返回的token，从后台拉取活体检测详细信息

通过token可以从后台拉取实名核身活体检测的详细信息，具体调用接口和方法参见后台接口和Demo。

7. 混淆说明

在混淆文件增加，参考demo混淆文件

```
   # Add project specific ProGuard rules here.
   # By default, the flags in this file are appended to flags specified
   # in D:\software\android-sdk_r24.3.4-windows\android-sdk-windows/tools/proguard/proguard-
android.txt
   # You can edit the include path and order by changing the proguardFiles
   # directive in build.gradle.
   #
   # For more details, see
   #   http://developer.android.com/guide/developing/tools/proguard.html

   # Add any project specific keep options here:

   # If your project uses WebView with JS, uncomment the following
   # and specify the fully qualified class name to the JavaScript interface
   # class:
   #-keepclassmembers class fqcn.of.javascript.interface.for.webview {
   #   public *;
   #}

   #不去掉无用方法，有些方法是给外部提供使用的
   #-dontshrink
   -optimizationpasses 5
   -dontusemixedcaseclassnames
   -dontskipnonpubliclibraryclassmembers
   -dontskipnonpubliclibraryclasses
   -dontpreverify
   -verbose
   -optimizations !code/simplification/arithmetic,!field/*,!class/merging/*
   -dontwarn android.support.**
   -dontwarn android.support.**

   -ignorewarnings                # 抑制警告


   # For BatteryStats
   -dontwarn android.os.**
   -dontwarn android.app.**

   -keep class android.support.** { *; }
   -keep class android.support.** { *; }


   -keep class android.** {
       <fields>;
       <methods>;
   }
   -keep class com.android.** {
       <fields>;
       <methods>;
   }



   -keep public class * extends android.app.Activity
```

```
-keep public class * extends android.app.Application
-keep public class * extends android.content.BroadcastReceiver

-keepattributes *Annotation*
-keepattributes InnerClasses
-keepattributes Signature

-keepclasseswithmembernames class * {
    native <methods>;
}

-keepclasseswithmembers class * {
    public <init>(android.content.Context, android.util.AttributeSet);
}

-keepclasseswithmembers class * {
    public <init>(android.content.Context, android.util.AttributeSet, int);
}

-keepclassmembers class * extends android.app.Activity {
   public void *(android.view.View);
}

-keepclassmembers enum * {
    public static **[] values();
    public static ** valueOf(java.lang.String);
}

-assumenosideeffects class android.util.Log {
    public static *** e(...);
    public static *** w(...);
    public static *** i(...);
    public static *** d(...);
    public static *** v(...);
}


 #不混淆资源类
-keep class **.R$* {
*;
}


# AuthSdk
-keep class com.tencent.authsdk.AuthSDKApi { *; }
-keep class com.tencent.authsdk.callback.IdentityCallback { *; }
-keep class com.tencent.youtufacetrack.** { *; }
```

使用**jar**包+资源接入方式

参见authdemo_jar，SDK接入流程如下：

1. 设置权限及**Manifest**配置

在接入方APP的AndroidManifest.xml中进行如下配置，具体可参见AuthDemo

权限设置：

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

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

<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" />

<!-- 使用音视频录制的权限-->

<uses-permission android:name="android.permission.RECORD_VIDEO" />

<uses-permission android:name="android.permission.RECORD_AUDIO"/>

<!-- 使用相机及自动对焦功能的权限-->

<uses-permission android:name="android.permission.CAMERA" />

<uses-feature android:name="android.hardware.camera" />

<uses-feature android:name="android.hardware.camera.autofocus" />

<!-- 监听来电 -->

<uses-permission android:name="android.permission.READ_PHONE_STATE" />

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

Activity添加：

```
<activity android:name="com.tencent.authsdk.activity.MainSdkActivity"
    android:screenOrientation="portrait"
    android:theme="@style/SDKAppTheme"
    android:launchMode="singleTask">
</activity>

<activity android:name="com.tencent.authsdk.activity.IdcardOcrActivity"
    android:screenOrientation="portrait"
    android:theme="@style/SDKAppTheme">
</activity>

<activity android:name="com.tencent.authsdk.activity.CameraActivity"
    android:screenOrientation="portrait">
</activity>

<activity android:name="com.tencent.authsdk.activity.IdentityDetectActivity"
    android:screenOrientation="portrait"
    android:theme="@style/SDKAppTheme">
</activity>

<activity android:name="com.tencent.authsdk.activity.RecordActivity"
    android:screenOrientation="portrait"
    android:theme="@style/SDKAppTheme">
</activity>

<activity android:name="com.tencent.authsdk.activity.LiveDetectActivity"
    android:screenOrientation="portrait"
    android:theme="@style/SDKAppTheme">
</activity>

<activity android:name="com.tencent.authsdk.activity.DetectResultActivity"
    android:screenOrientation="portrait"
    android:theme="@style/SDKAppTheme">
</activity>
```

2. 添加**jar**包和资源

参照AuthDemo，将AuthSDK.jar 添加到接入方APP中的libs目录下，将res目录下的资源文件添加到接入方APP的res下的相应目录下，以及assets目录下的文件添加到APP的assets下，将libyoutufacetrack.so添加到jniLibs下。

3. 初始化**SDK**及调用实名核身接口

具体流程跟上面aar接入方式中的3-6步骤一致，不再赘述。

SDK接口详情

初始化

```
/**
 * 初始化
 * @param context app的context
 * @param serverUrl 后台server的URL
 * @param appId     分配给业务方的AppId
 * @param packageName     业务方的包名
 */
public void init(Context context, String serverUrl, String packageName)
```

设置实名核身调用类型

```
/**
 * 设置实名认证调用类型
 * @param type 是否需要身份证OCR识别的4种类型
 */
public void setIdentityType(int type)
```

五种调用类型如下：（更新于2017.7.19）

```
public final static int NORMAL_IDENTITY_TYPE = 0;       //普通的实名认证流程，包含正反身份证OCR识别+活体检测
public final static int NO_BACK_OCR_IDENTITY_TYPE = 1; //正面身份证OCR识别+活体检测
public final static int NO_OCR_IDENTITY_TYPE = 2;       //没有OCR识别，需要用户手动输入身份证号码和姓名
public final static int ONLY_LIVE_DETECT_IDENTITY_TYPE = 3; //仅有活体检测，外部传递身份证号码
public final static int ONLY_BACK_OCR_IDENTITY_TYPE = 4;       //没有OCR识别，需要用户手动输入身份证号码和姓名
public final static int ONLY_OCR_IDENTITY_TYPE = 5; //仅OCR
public final static int RELIVE_DETECT_IDENTITY_TYPE = 6; //二次验证
```

设置用户信息

```
/**
 * 设置用户信息，包括身份证号码和姓名
 * @param idcard 外部传递过来的身份证号码
 * @param name   外部传递过来的姓名
 */
public void setUserInfo(String idcard, String name)
```

启动实名核身（两个接口）

```
/**
 * 启动实名认证首页
 * @param context 调用类
 * @param destIntent  可以跳转到接入方中间页（OCR页面）的intent
 * @param signature 从后台服务器获取的签名，用于appauth接口的请求头部签名
 * @param callback 回调监听接口
 */
public void startMainPage(Activity context, Intent destIntent, String signature,
IdentityCallback callback)
```

```
/**
 * 启动实名认证流程
 * @param context 调用类
 */
public void startAuth(Activity context)
```

其中回调接口IdentityCallback的实现如下，返回数据包括活体检测的结果及token，具体参见AuthDemo。

```
private IdentityCallback mListener = new IdentityCallback() {
    @Override
    public void onIdentityResult(Intent data) {
        String token = data.getStringExtra(AuthSDKApi.EXTRA_TOKEN);
        Log.i("test", "token = " + token);
        boolean identityStatus = data.getBooleanExtra(AuthSDKApi.EXTRA_IDENTITY_STATUS, false);
        String result = getResultStr( identityStatus);
        mSdkResult.setText(result);
    }
};
```

实名信息拉取接口

由接入方自己实现。

1）接口

https://iauth-sandbox.wecity.qq.com/new/cgi-bin/getdetectinfo.php

2) 描述

拉取实名详细信息接口。回包内容已加密，详细算法参看7 加解密算法。

3) 方法

POST

4) HTTP请求格式

a、头部信息

要求	参数名	类型	参数说明
必选	signature	String	接口签名，具体见签名算法4.1

signature生成算法，请查看签名算法。

b、请求包体

要求	参数名	类型	参数说明	取值说明
必选	token	string	上一个接口返回的token序列号	
必选	appid	string	分配的appid	
可选	info_type	int	获取信息类型	不传时,默认带上所有文件buffer；;传”0”表示获取所有信息，含文件buffer；;“1”为传文本信息，不含文件buffer。

c、请求包体示例

```
{
    "token":"{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}",
    "appid":"xxxx"
}
```

5) 返回值说明

字段	类型	说明
errorcode	int	返回状态码,0表示成功，非0值为出错
errormsg	String	返回错误描述
data	String	BASE64数据（加密数据）

b、返回示例

```
{
    "errorcode": 0,
    "errormsg": "success",
    "data": "base64(aes密文)"
}
```

解密data后对应的数据如下：

```
{
    "ID": "4501111994xxxxxxxx",

    "name": "张三",

    "phone": "159********",

    "sex": "男",

    "nation": "汉",

    "ID_address": "广东省深圳市南山区*****",

    "ID_birth": "xxxx",

    "ID_authority": "***公安局",

    "ID_valid_date": "xxxx.xx.xx-xxxx.xx.xx",

    "validatedata": 3344,//数字模式活体检测录制视频读取，动作模式此参数为空

    "frontpic": "身份证正面照片的base64编码",

    "backpic": "身份证反面照片的base64编码",

    "video": "视频的base64编码",

    "videopic1": "视频截图1的base64编码",

    "videopic2": "视频截图2的base64编码",

    "videopic3": "视频截图3的base64编码",

    "yt_errorcode":0,//最终结果错误码

    "yt_errormsg":"成功", //最终结果错误描述

    "livestatus": 0,//活体检测错误码

    "livemsg": "OK",//活体检测错误描述

    "comparestatus": 0,//活体比对错误码

    "comparemsg": "OK",//活体比对错误描述

    "type": 0//auth传入的type参数
}
```

其中type为对外接口扩展参数中的type,默认为0（即首次实名验证）

鉴权算法

签名

上述提供的API接口，通过签名来验证请求的合法性。开发者通过将签名授权给调用方，使其具备使用API接口的能力。 密钥的获取及签名的生成方法如下：

Step 1 获取应用appid、密钥secretkey和过期时间expired

应用appid: 用来标识唯一的应用；

密钥secretkey: 使用加密算法时使用的秘钥；

expired: 当次请求的时间戳的有效时间；

Step 2 拼接有效签名串

a=xxxxx&m=xxxxxxx&t=1427786065&e=600

a为appid

m为调用的apiName，

t为当前时间戳，是一个符合UNIX Epoch时间戳规范的数值，单位为秒

e为此签名的凭证有效期，是一个符合UNIX Epoch时间戳规范的数值，单位为秒,

同appid和secretkey一样，由API提供方给定。

拼接有效签名串的结果,下文称之为orignal

Step 3 生成签名串

(1)API提供方 使用 HMAC-SHA1 算法对请求进行签名。

(2)签名串需要使用 Base64 编码。

根据签名方法signature= Base64(HMAC-SHA1(secretkey, orignal) + original)，其中secretkey为Step1获取，orignal为Step2中拼接好的签名串，对orignal使用HMAC-SHA1算法进行签名，然后将orignal附加到签名结果的末尾，再进行Base64编码，得到最终的sign。

注：此处使用的是标准的Base64编码，不是urlsafe的Base64编码，请注意。 以 JAVA 语言为例,其他语言均有类似算法. JAVA语言的示例代码见附件。

Step 4 使用签名串

将Step 3生成的signature，填充到http请求的head头部的signature字段中即可。

NodeJS参考代码

```
var crypto = require('crypto');
var signExpired = 600;//有效期一般为600s
//生成签名
function getAppSign(apiName, appId, appSecretKey, signExpired) {
    if (!apiName || !appId || !appSecretKey || !signExpired)
        return '';
    var now = parseInt(Date.now() / 1000);
    var plainText = 'a=' + appId + '&m=' + apiName + '&t=' + now + '&e=' + signExpired;
    var data = new Buffer(plainText, 'utf8');
    var res = crypto.createHmac('sha1', appSecretKey).update(data).digest();
    var bin = Buffer.concat([res, data]);
    var sign = bin.toString('base64');
    return sign;
}
```

参数校验

参数签名校验算法

md5(参数1-参数2-...-私钥key)

说明

参数顺序使用“-”拼接，拼接后的字符串，再最后拼接上SIG_KEY(authkey),然后字符串md5,取32位小写字符串

NodeJS参考代码

```
var crypto = require('crypto');
var SIG_KEY = "authkey";
//生成sig 参数顺序要按照文档上的顺序
function getHashSig(postBody) {
    var datas = JSON.parse(postBody);
    var sigData = datas["sig"];
    var srcData = "";
    for (var index in datas) {
        if (index != 'sig')
            srcData += datas[index] + '-';
    }
    srcData += SIG_KEY;
    return  crypto.createHash('md5').update(srcData).digest('hex');
}
```

PHP参考代码

```php
/**
* 计算签名
* @param $apiName
* @param $secretKey 接口签名
* @param $plainText 拼接有效签名串
* @return array|string
*/
static function getAppSign($apiName,$secretKey,$plainText) {
    if (empty($apiName)) {
        return array("ret"=>-1,"msg"=>"apiName error","signature"=>"");
    }

    $bin = hash_hmac("SHA1", $plainText, $secretKey, true);
    $bin = $bin.$plainText;
    $sign = base64_encode($bin);
    return $sign;
}
```

加解密算法

AES 256算法

使用项目提供的AES解密秘钥解密

NodeJS 代码参考

AES-256-ECB + PKCS7

```
function encryptAes256ECBPKCS7(data, secretKey) {
    try {
        let iv = "";
        var clearEncoding = 'utf8';
        var cipherEncoding = 'base64';
        var cipherChunks = [];
        var cipher = crypto.createCipheriv('aes-256-ecb', secretKey, iv);
        cipher.setAutoPadding(true);
        cipherChunks.push(cipher.update(data, clearEncoding, cipherEncoding));
        cipherChunks.push(cipher.final(cipherEncoding));
        return cipherChunks.join('');
    } catch (e) {
        console.error(e);
        return "";
    }
}

function decryptAes256ECBPKCS7(data, secretKey) {
    try {
        if (!data) {
            return "";
        }
        let iv = "";
        var clearEncoding = 'utf8';
        var cipherEncoding = 'base64';
        var cipherChunks = [];
        var decipher = crypto.createDecipheriv('aes-256-ecb', secretKey, iv);
        decipher.setAutoPadding(true);
        let buff = data.replace('\r', '').replace('\n', '');
        cipherChunks.push(decipher.update(buff, cipherEncoding, clearEncoding));
        cipherChunks.push(decipher.final(clearEncoding));
        return cipherChunks.join('');
    } catch (e) {
        console.error(e);
        return "";
    }
}
```

PHP 代码参考

AES-256-ECB +PKCS7（由于 PHP 底层的 256 和 Nodejs、Java 的不一样。所以 PHP 使用的是128长度）

```
/**
 * 利用mcrypt做AES加密解密
 */
class AES{
    /**
     * 算法,另外还有192和256两种长度
     */
    const CIPHER = MCRYPT_RIJNDAEL_128;
    /**
     * 模式
     * 1. MCRYPT_MODE_ECB(electronic codebook)
     适合对小数量随机数据的加密，比如加密用户的登录密码之类的。
     * 2. MCRYPT_MODE_CBC (cipher block chaining)
     适合加密安全等级较高的重要文件类型。
     * 3. MCRYPT_MODE_CFB (cipher feedback)
     适合于需要对数据流的每一个字节进行加密的场合。
     * 4. MCRYPT_MODE_OFB (output feedback, in 8bit) 和CFB模式兼容，但比C
     FB模式更安全。CFB模式会引起加密的错误扩散，如果一个byte出错，则其后
     续的所有byte都会出错。OFB模式则不会有此问题。但该模式的安全度不是很
     高，不建议使用。
     * 5. MCRYPT_MODE_NOFB (output feedback, in nbit)
     和OFB兼容，由于采用了块操作算法，安全度更高。
     * 6. MCRYPT_MODE_STREAM
     是为了WAKE或者RC4等流加密算法提供的额外模型。
     */
    const MODE = MCRYPT_MODE_ECB;

    /**
     * pkcs7补码
     * @param string $string  明文
     * @param int $blocksize Blocksize , 以 byte 为单位
     * @return String
     */
    private function addPkcs7Padding($string, $blocksize = 16) {
        $len = strlen($string); //取得字符串长度
        $pad = $blocksize - ($len % $blocksize); //取得补码的长度
        $string .= str_repeat(chr($pad), $pad); //用ASCII码为补码长度的字符， 补足最后一段
        return $string;
    }

    /**
     * 加密然后base64转码
     * @param $str
     * @param $key
     * @return string
     */
    function aes256cbcEncrypt($str,$key ) {
        $iv = mcrypt_create_iv(mcrypt_get_iv_size(self::CIPHER,self::MODE),MCRYPT_ENCRYPT);
        return base64_encode(mcrypt_encrypt(self::CIPHER, $key, $this->addPkcs7Padding($str) ,
self::MODE, $iv));
    }
```

```
    /**
     * 除去pkcs7 padding
     *
     * @param String 解密后的结果
     *
     * @return String
     */
    private function stripPkcs7Padding($string){
        $slast = ord(substr($string, -1));
        $slastc = chr($slast);
        $pcheck = substr($string, -$slast);

        if(preg_match("/$slastc{".$slast."}/", $string)){
            $string = substr($string, 0, strlen($string)-$slast);
            return $string;
        } else {
            return false;
        }
    }
    /**
     * 解密
     * @param String $encryptedText 二进制的密文
     * @param String $key 密钥
     * @return String
     */
    function aes256cbcDecrypt($encryptedText, $key) {
        $encryptedText =base64_decode($encryptedText);
        $iv = mcrypt_create_iv(mcrypt_get_iv_size(self::CIPHER,self::MODE),MCRYPT_ENCRYPT);
        return $this->stripPkcs7Padding(mcrypt_decrypt(self::CIPHER, $key, $encryptedText,
self::MODE, $iv));
    }
}
```