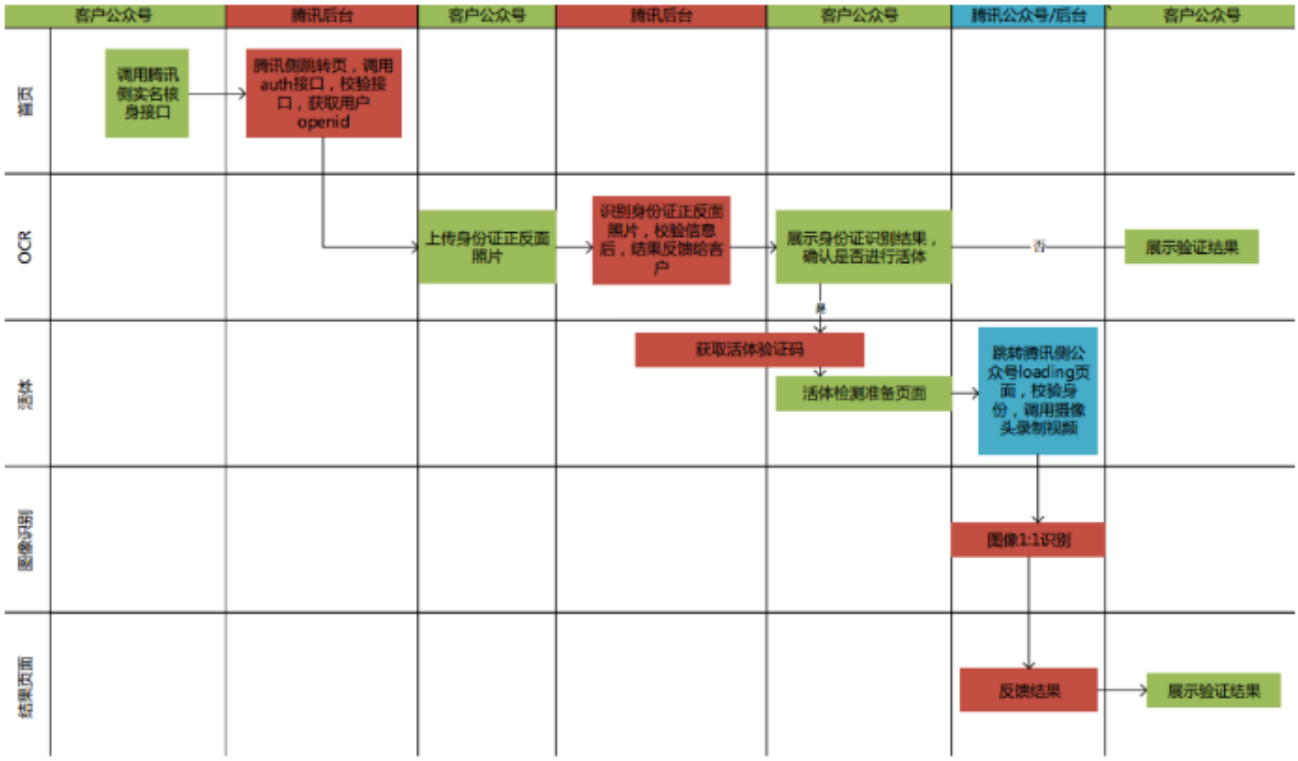
腾讯慧眼活体+1v1比对接口说明文档

流程说明



接口说明

1) 所有以下开放接口调用均需要带上头部信息：

如调用的接口结果将重定向，请将signature放入body中

参数名称	必选	类型	描述
signature	是	String	接口签名，具体见签名算法

生成signature的apiName 即接口的名字，例如auth.php接口，apiName =auth;

2) 以下所有接口回包格式：

参数名称	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	接口所需数据，具体参数参考各个接口的返回内容说明

实名登录接口

1) 接口

```
https://xxxx.xxxx.xxx/new/cgi-bin/api_auth.php
```

2) 描述

```
实名认证登录接口。
```

3) 方法

```
POST
```

4) HTTPS请求格式

```
a、头部信息
```

参数名称	取值
Content-Type	application/x-www-form-urlencoded

```
b、请求包体
```

此接口调用成功后将**302**重定向，请使用前端**form**表单请求。

要求	参数名	类型	参数说明	取值说明
必选	appid	string	分配的appid	
必选	uid	string	用户uid标识	用户唯一标识,（如微信的用户openid）
可选	redirect	string	回调地址	
可选	uid	string	一般传参用户uid，在回调地址中会带回	
可选	ID	string	身份证号	
可选	name	string	姓名	
可选	phone	string	电话号码	
可选	out_trade_no	string	传入此次验证流水单号，回调地址和后台通知中会带回	
可选	out_extra	string	传入此次验证附加数据，回调地址和后台通知中会带回	

a. 包体示例

```
{
    "appid":"xxxx",
    "uid":"xxxxxxx",
    "ID":"xxxxxxxxxxxxxxx",
    "name":"张三",
    "phone":"135xxxxxxxx"
}
```

1)返回值说明

接口成功调用后将跳转链接 `redirect?token=xxx&uid=xxx`

示例代码：

```
var args = {
    appid,
    uid,
    signature,
    redirect
    };
var form = $("<form method='post'></form>");
form.attr({ "action": host + '/new/cgi-bin/api_auth.php' });
for (var arg in args) {
    var input = $("<input type='hidden'>");
    input.attr({ "name": arg });
    input.val(args[arg]);
    form.append(input);
}
$(document.body).append(form);
form.submit();
```

动作活体+1v1接口

1) 接口

```
https://xxx.xxx.xxx/new/cgi-bin/startactionliveness.php
```

2) 描述

```
开始进行活体检测（动作）+1v1对比，接口调用成功后将重定向到腾讯侧页面，并进入活体检测流程。
```

3) 方法

```
POST
```

4) HTTPS请求格式

```
a、头部信息
```

要求	参数名称	类型	说明
必选	appid	String	分配的appid
必选	token	String	auth接口返回的token
必选	name	String	姓名
必选	ID	String	身份证
必选	validate_data	String	活体检测动作序列[1,2]or[2,1];张嘴=1，眨眼=2
必选	redirect	String	验证结果通知的客户端跳转URL

5) 返回值说明

```
完成活体检测流程后将跳转redirect，并附带token、uid、state参数
```

参数名称	参数说明
token	auth接口返回的token(可以通过此token拉取用户数据)
uid	用户uid 标识
state	重定向状态(可能为空) 1=重新验证 2=人工认证

注意，重定向参数可能为空或被恶意修改，请注意代码健壮性

数字活体验证码接口

1) 接口

```
https://xxxx.xxx.xxx/new/cgi-bin/api_getlivecode.php
```

2) 描述

```
拉取活体比对4字验证码。
```

3) 方法

```
POST
```

4) HTTPS请求格式

```
a、头部信息
```

要求	参数名	类型	参数说明
必选	appid	string	分配的appid
必选	token	string	auth接口返回的token

```
b、请求包体示例
```

```
{
    "appid":"xxxx",
    "token":"xxxxxxx"
}
```

5) 返回值说明

```
a、返回示例
```

```
{
    "errorcode": 0,
    "errormsg":"success",
    "data": {
        "validate_data":"1166"
    }
}
```

数字活体+1v1接口

1) 接口

```
https://xxx.xxx.xxx/new/cgi-bin/startlivedetectfour.php
```

2) 描述

```
开始进行活体检测（读数）+1v1对比，接口调用成功后将重定向到腾讯侧页面，并进入活体检测流程。
```

3) 方法

```
POST
```

4) HTTPS请求格式

```
a、请求包体
```

要求	参数名	类型	参数说明
必选	appid	String	分配的appid
必选	token	String	auth接口返回的token
必选	name	String	姓名
必选	ID	String	身份证
必选	validate_data	String	四位数字验证码，必须由api_getlivecode接口获取
必选	redirect	String	验证结果通知的客户端跳转URL

5) 返回值说明

```
完成活体检测流程后将跳转redirect，并附带token、uid、state参数
```

参数名称	参数说明
token	auth接口返回的token(可以通过此token拉取用户数据)
uid	用户uid 标识
state	重定向状态(可能为空) 1=重新验证 2=人工认证

注意，重定向参数可能为空或被恶意修改，请注意代码健壮性

实名信息拉取接口

1) 接口

```
https://xxx.xxx.xxx/new/cgi-bin/api_getdetectinfo.php
```

2) 描述

```
拉取实名详细信息接口。
```

3) 方法

```
POST
```

4) HTTPS请求格式

要求	参数名	类型	参数说明
必选	token	String	上一个接口返回的token序列号
必选	appid	String	分配的appid

```
c、请求包体示例
```

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

5) 返回值说明

```
a、返回示例
```

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

aes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",
    "frontpic":"身份证正面照片的base64编码",
    "backpic":"身份证反面照片的base64编码",
    "videopic1":"视频截图1",
    "videopic2":"视频截图2",
    "videopic3":"视频截图3",
    "video":"视频的base64编码",
    "yt_errorcode":0,
    "yt_errormsg":"通过",
    "livestatus":0,
    "livemsg":"OK",
    "comparestatus":0,
    "comparemsg":"OK",
    "type":0
}
```

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）

签名算法

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');
//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;
}
```

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, resultKey) {

    try {

        let iv = ;

        var clearEncoding = 'utf8';

        var cipherEncoding = 'base64';

        var cipherChunks = [];

        var cipher = crypto.createCipheriv('aes-256-ecb', resultKey, 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, resultKey) {

    try {

        if (!data) {

            return ;

        }

        let iv = ;

        var clearEncoding = 'utf8';

        var cipherEncoding = 'base64';

        var cipherChunks = [];

        var decipher = crypto.createDecipheriv('aes-256-ecb', resultKey, 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));
    }
}
```