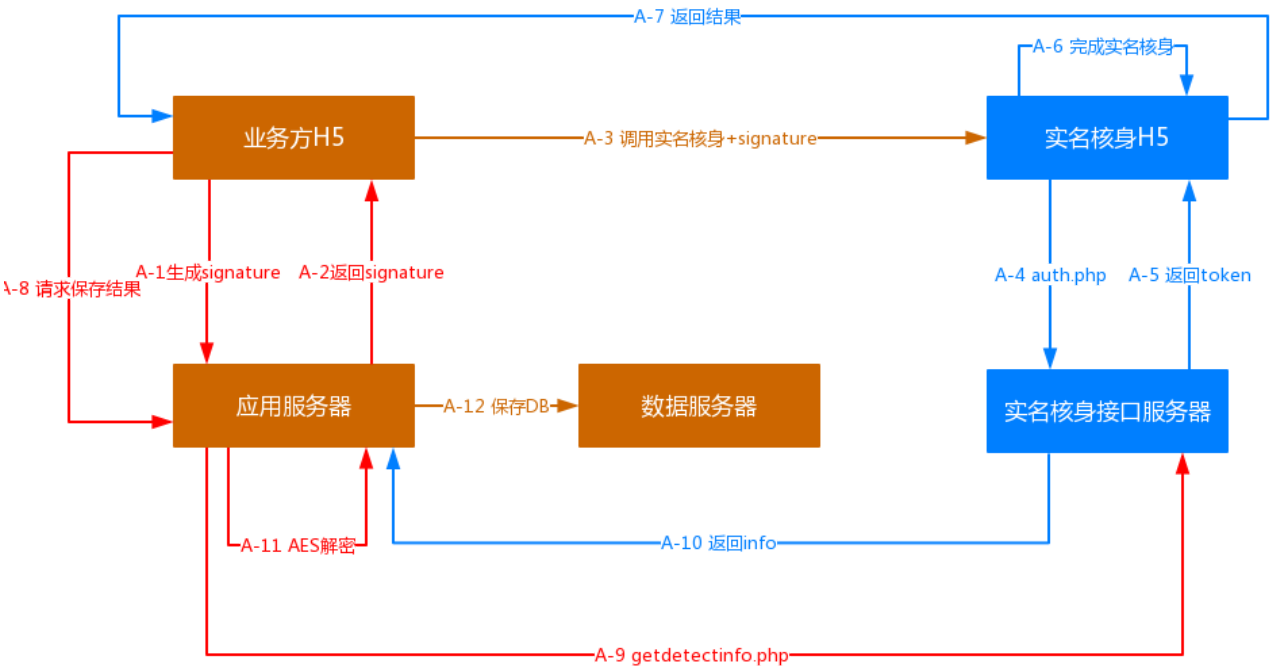
腾讯慧眼方案及接口说明文档(公有部署)

接口使用流程框架图

H5业务流程框架



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，部署在调用方侧。

实名认证详细流程

用户首先进入业务方公众号具体业务，若用户未实名，则通过表单提交到实名认证对外接口。

用户完成实名认证，包括协议确认、身份证正反面OCR、活体检测和人脸比对等步骤。完成后通过回调的方式返回用户实名信息token。

业务方调用信息拉取接口通过token获得实名信息,并将实名信息落地到业务数据库中。 其中，橙色部分需要对接业务方实现。

对外接口

接口前需要注意的点

1. 生成signature的apiName 即接口的名字，例如auth.php接口，apiName =auth。
2. 所有接口使用的都是utf8编码

实名认证对外接口（**web**端）

1) 接口

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

2) 描述

实名认证流程，拉取活体检测详细信息接口。通过表单方式post

3) 方法

POST

4) 表单请求内容

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	signature	string	接口签名	具体见签名算法
必选	redirect	string	回调地址	
必选	uid	string	一般传参用户uid	在回调地址中会带回，用于用户关系绑定
必选	type	int	请求类型	0--全流程完整验证; 1--二次验证
可选	ID	string	身份证号	type=1时提供 只做活体检测时提供
可选	name	string	姓名	type=1时提供　只做活体检测时提供
可选	pic_key	string	图片标志值	直接用首次验证完返回的token即可; type=1时提供
可选	out_trade_no	String	流水单号	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	附加数据	传入此次验证附加数据，回调地址和后台通知中会带回

c、请求包体示例

```
{
    "signature": "xxx",
    "appid": "xxxx",
    "redirect": "xxx.html",
    "uid": "xxx",
    "type": 0
}
```

请求 JS 格式例子：

```
var url = https://xxxxxx/new/cgi-bin/auth.php;

    var args = {

        "appid": "xxxxxxxxxxxxxxx",
        "signature": "xxxxxxxxx",
        "redirect": "xxxxxxxx",
        "uid": "xxxxx",
        "type": 0
    };

    var form = $("<form method='post'?</form>");

    form.attr({action: url});

    for (var arg in args) {

        var input = $("<input type='hidden'>")

        input.attr({name: arg});

        input.val(args[arg]);

        form.append(input);

    }

    form.submit();
```

5) 回调地址数据说明

a、返回主体包的内容

要求	参数名	类型	参数说明	取值说明
必选	token	string	用户实名信息token;	首次验证的token需要存储，以便二次验证的时候用作图片标志（pic_key）； 二次验证的token取完结果数据即可扔掉
必选	uid	string	请求时传参的用户uid，在回调地址中会带回	0验证成功,非0为验证失败情况
可选	out_trade_no	string	流水单号	请求时传参的out_trade_no，在回调地址中会带回
可选	out_extra	string	附加数据	请求时传参的out_extra，在回调地址中会带回

b、返回示例

redirect?uid=xxx&token=xxx&out_trade_no=xxx&out_extra=xxx

实名信息拉取接口

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 6.加解密算法。

3) 方法

POST

接口访问方式 content-type支持两种形式：

1. application/json传参数也是json格式的；
2. application/x-www-form-urlencoded 参数形式是字符串格式(如下示例所示)

token=xxx&appid=xxx&info_type=xxx

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

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(视频和图片)； 1为传文本信息，不含文件buffer(视频和图片)。

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	取数据是否成功，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;

}
```

PHP参考代码

```
/**
* 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;
}
```

错误码

1) 成功

返回值	类型	说明
0	SUCESS	成功

2) HTTP返回码

返回值	类型	说明
1	HTTP_UNSET_PARAM	请求不合法，缺少参数
2	HTTP_WRONG_PARAM	请求不合法，参数错误
3	HTTP_UNAUTHORIZED	权限验证失败
4	HTTP_NOT_SIGNATURE	请求不合法，缺少签名
5	HTTP_REQUES_ILLEGAL	业务请求不合法
6	HTTP_DATA_SIF	数据签名错误
7	ERROR_UNLOGIN	未登录
8	ERROR_SYSTEM	系统错误
9	ERROR_API_ACCESS	未授权接口
10	ERROR_OVER_INTERFACE_LIMIT	超过调用限制
11	ERROR_TOKEN_TIME_EXPIRED	无效请求
12	ERROR_TOKEN_OVER_TIMELIMIT	token超过有效期
13	ERROR_UNLEGAL_URL	非法转跳URL
14	ERROR_TOKEN_VERIFY_FINISHED	无效请求(14)

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));
    }
}
```