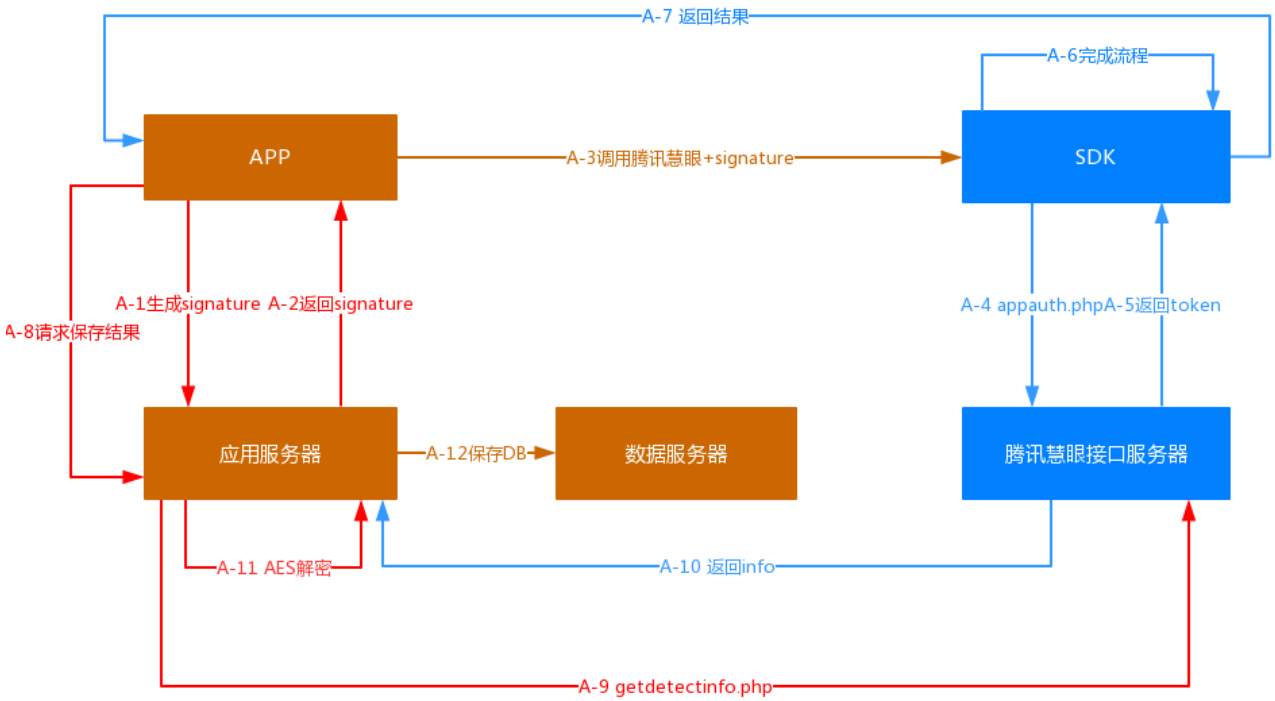
腾讯实名核身SDK说明文件 -- iOS端

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；橙色部分由也无妨开发对接，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.framewore 封装了实名认证的流程，包括了身份证OCR识别和活体检测功能。

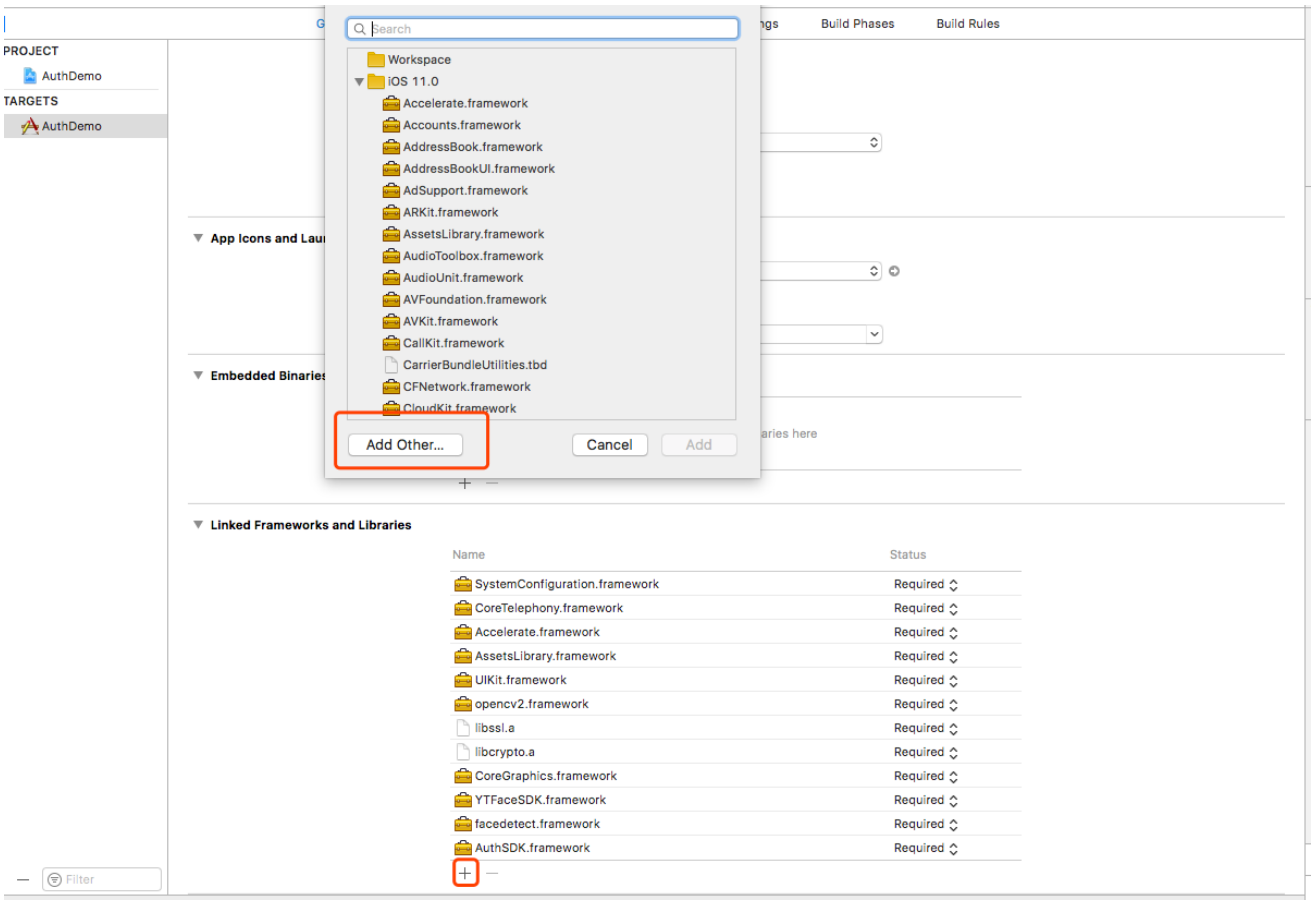
AuthDemo 提供了实名认证接口调用方法及从后台拉取活体检测详细信息的方法。

注: 协议文件在AuthSDK.framework/authsdk.bundle/protocol.txt中，可以根据需要修改。（其他资源文件，如首页banner也可根据需求替换authsdk.bundle中图片）

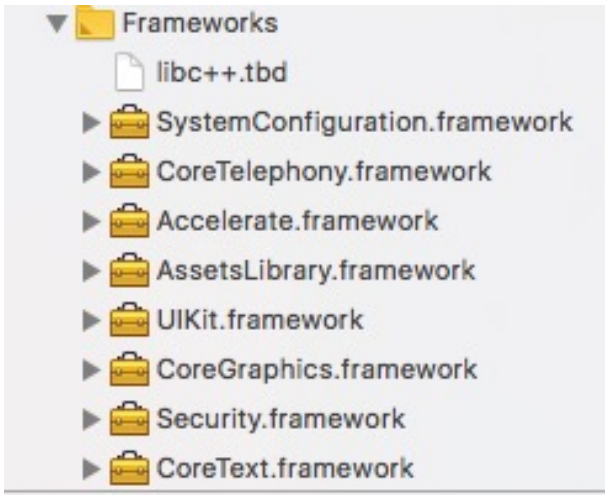
接入流程

添加framework

将AuthSDK.framework拷贝到项目根目录Framework中（目录可自定义），在TARGETS-Build Phases-Link Binary with Libraries 点击“+”，弹出添加列表后，点击“Add Other...”,从Framework文件夹中添加AuthSDK.framework到工程中。同理，添加SDK中的所有文件添加到工程中。

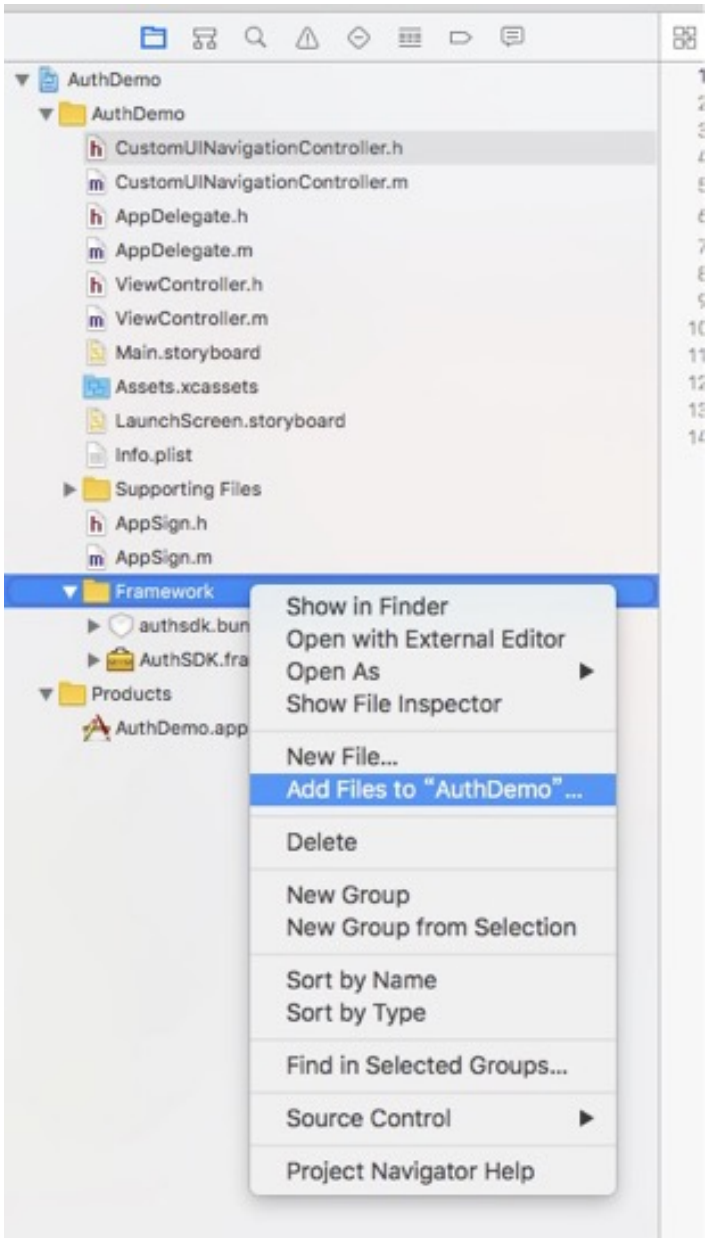


保证下图中的内部framework添加到工程中：



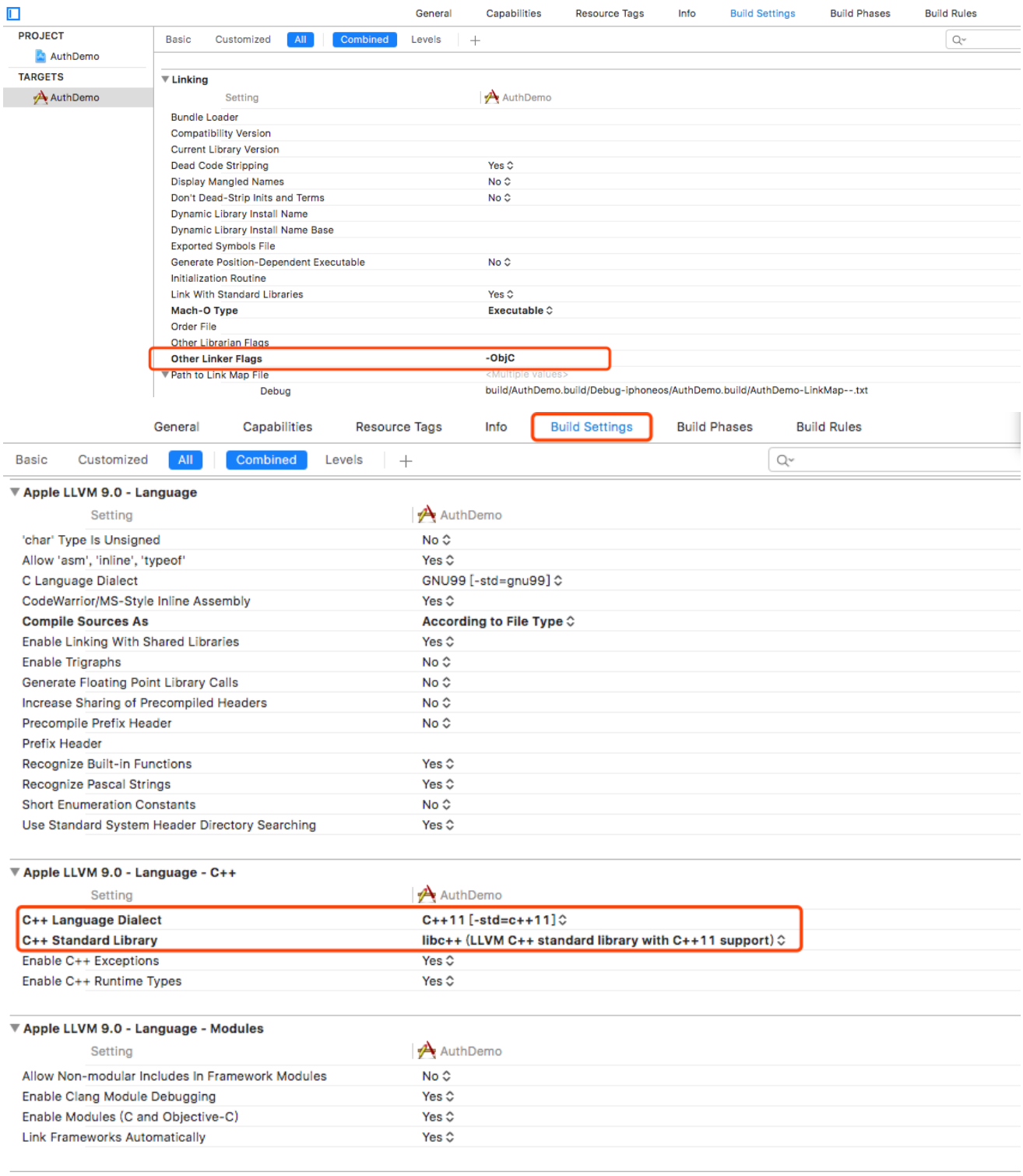
添加AuthSDK.framework中资源文件

在工程界面右键弹出菜单中选择"Add Files To...", 从文件夹Framework->AuthSDK.framework中将资源文件authsdk.bundle添加到工程中。



添加需要的编译选项

在TARGETS-Build Settings-Other Linker Flags 中添加如下内容： -ObjC 。 C++ Language Dialect设置为C++11 [-std=c++11] C++ Standard Library设置为libc++



添加权限声明

在工程info.plist中增加如下字串:

```
    <key>NSPhotoLibraryUsageDescription</key>
    <string>需要您的同意才能读取照片库</string>
    <key>NSCameraUsageDescription</key>
    <string>需要您的同意才能使用摄像头</string>
    <key>NSMicrophoneUsageDescription</key>

    <string>需要您的同意才能使用麦克风</string>
```

Xcode9以上版本需要添加：

```
    <key>NSPhotoLibraryAddUsageDescription</key>
    <string>需要您的同意才能读取照片库</string>
```

以下配置当服务器使用http访问时需要配置，否则不需要

```
    <key>NSAppTransportSecurity</key>
    <dict>
        <key>NSAllowsArbitraryLoads</key>
        <true/>
    </dict>
```

Licence 使用说明

您需要将授权给您的 licence 以文件名 YTFaceSDK.licence 加入您的工程中，确保在 Xcode->Build Phases->Copy Bundle Resources 清单中显示。您需要将应用程序的 Bundle Identifier 设置为 licence 对应的 Bundle Identifier。

初始化SDK接口

在程序中import sdk，并初始化，设置分配的APPID。

```
#import <AuthSDK/AuthSDK.h>
```

注：目前Demo中使用的是测试URL，接入时可修改成正式服务URL（传nil时，使用测试服务器）

```
@property (nonatomic) AuthSDK * sdk;
```

```
_sdk = [[AuthSDK alloc] initWithServerURL:nil]
[_sdk setAppId:@"4396" appSecretKey:@"416df0300bd4043fb4d4df5c00b3e280"];
```

获取signature

接入方需要在服务端实现获取signature的接口，客户端通过接口拉取到signature，在下面调用SDK接口的时候需要传入。

调用实名认证接口

这里提供了四种调用方式，可以在初始化时进行配置：

- 正常的实名认证认证流程，包括身份证正反面OCR识别和活体检测流程

```
[_sdk startAuth:AuthTypeOcrAll name:nil idNum:nil parent:self delegate:self
signature:_signature];
```

- 仅有身份证正面的OCR识别和活体检测流程

```
[_sdk startAuth:AuthTypeOcrLight name:nil idNum:nil parent:self delegate:self
signature:_signature];
```

- 没有OCR识别，只有活体检测流程

```
[_sdk startAuth:AuthTypeOcrNone name:nil idNum:nil parent:self delegate:self
signature:_signature];
```

- 没有OCR识别，只有活体检测流程

```
[_sdk startAuth:AuthTypeWithIDCardInfo name:_nameTextField.text idNum:_textField.text token:nil
parent:self delegate:self signature:_signature];
```

- 二次验证

```
[_sdk startAuth:AuthTypeWithIDCardInfo name:_nameTextField.text idNum:_textField.text token:nil
parent:self delegate:self signature:_signature];
```

根据实名认证返回的**token**，从后台拉取活体检测详细信息

通过token从后台拉取详细信息的接口需要在接入方服务器调用。

接口详情

启动实名认证接口

```
/**
 开始OCR流程

 @param type 需要跳转的流程类型
 @param name 身份证名字，不需要时可传入nil
 @param idNum 身份号，不需要时可传入nil
 @param token 需要传入的token，不需要时可传入nil
 @param vc 父controller
 @param delegate 需要实现回调的Controller
 @param signature 签名，由接入方后台生成的签名
 */
-(void)startAuth:(AuthType)type name:(NSString*)name idNum:(NSString*)idNum token:(NSString
*)token parent:(UIViewController *)vc delegate:(id<AuthSDKDelegate>)delegate signature:
(NSString*)signature;
```

在启动实名认证接口时，可传入四种实名认证流程类型

```
typedef NS_ENUM(NSInteger, AuthType) {
    AuthTypeOcrAll = 0,          /**< 有Ocr，需要用户操作上传身份证正面及反面图片 */
    AuthTypeOcrLight,            /**< 有Ocr，需要用户操作上传身份证正面图片 */
    AuthTypeOcrBack,             /**< 有Ocr，需要用户操作上传身份证反面图片 */
    AuthTypeOcrNone,             /**< 没有Ocr，需要用户手动输入身份信息 */
    AuthTypeWithSmsVerify,       /**< 有OCR，需要短信验证 */
    AuthTypeWithIDCardInfo,      /**< 仅活体检测，需要传入姓名和身份证号 */
    AuthSecondVerification,      /**< 二次验证 */
};
```

实名认证回调接口类

```
@protocol AuthSDKDelegate <NSObject>

-(void)onResultBack:(NSDictionary *)result;

@end
```

设置验证成功提示语（不设置此接口时，使用默认提示语）

```
-(void)setVerifySuccessTipLable:(NSString *)label;
```

设置验证失败提示语（不设置此接口时，使用默认提示语）

```
-(void)setVerifyFailTipLable:(NSString *)label;
```

设置SDK页面标题（默认为“实名认证”）

```
-(void)setViewTitle:(NSString *)title;
```

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

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)"
}
```

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,//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）

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');
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;
}
```

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）

```php
/**
 * 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));
    }
}
```