CFCA 国密接口协议（313_H5 活体识别+人像比对）

版本号 V3

1 文档说明

1.1 描述

本文档为第三方商户平台对接金信反欺诈服务平台提供接口接入规范，指导各渠道系统通过金信反欺诈服务平台实现 **H5 活体识别+人像比对**功能。

1.2 阅读对象

对接金信反欺诈服务平台的第三方机构或平台商户的开发人员、需求人员。

1.3 业务术语

H5 活体识别+人像比对：根据在手机端网页进行基于屏幕炫彩，通过**面部反光**及**瞳孔反光**对核验人员进行活体判断并确认活体本人信息。此方案支持**实时检测**形式，**无需拍摄上传视频**，可直接在前端完成检测流程，提升整体核验流程的流畅度及用户体验。相比于行业内传统的动作活体和视频活体检测方式，通过率大大提升，使用效率更加流畅便捷，有效拦截视频、图片伪造、3D 面具、合成图等黑产攻击。本接口适用于金融、数娱、电商等行业在开户、绑卡、申请交易时做实名认证信息。

2 通讯方式与报文格式

2.1 报文协议

商户和金信平台之间使用 HTTPS 建立连接以及通信，需要经过双向 SSL 认证，金信平台提供建立双向 SSL 连接的客户端通信证书。

2.2 报文格式

为了保证数据传输安全性，金信平台使用国密（以下简称 SM2）算法对重要数据进行加密，同时为了保证传输数据的真实性与有效性，调用方需对请求数据签名，本系统在接收请求后会进行签名校验，签名校验失败将拒绝请求，通讯报文采用标准的 JSON 数据格式，字符编码格式为 UTF-8。

2.3 请求报文格式

以下为系统中某个业务的请求报文格式

```
{ // 外层
    "transcode": "923",
```

```
    "version": "0200",
    "ordersn": "88ed393869dd4ef3...",
    "dsorderid": "a3a9afeca51d46ce93...",
    "merchno": "0000000000000000",
    "timestamp": "1643170610550",
    "sign": "1F2B306F6DECF...",
    "data": "BHSR6Y0XYQyxCxOqVszt19YhifhkBUo..." // data 层为 JSON 格式，使用 SM2 加密后转 Base 64 的字符串
}

// data 为 JSON 格式，解密后明文如下
{
    "username": "...",
    "mobile": "...",
    "sCustomerName": "...",
    "sceneCode": "..."
}
```

请求报文为客户端上送给服务端的信息，上面示例请求 JSON 报文包含两层，外层存放请求业务编码、订单号、签名等，外层数据无需加密，data 层存放业务相关的要素信息，要素信息为敏感数据，需要使用 SM2 进行加密，服务端接收到报文后会对其进行验签、解密等操作，下面为请求报文参数介绍

参数名称	参数含义	是否必填	数据类型	说明
transcode	交易码	是	String	交易码，如 106，每个业务交易码不同，可参对应业务的接口文档。
version	版本号	是	String	0200
ordersn	商户请求流水号	是	String(32)	唯一值，每次请求都唯一，限 32 位长度
dsorderid	商户订单号	是	String	每笔交易唯一，建议 32 位长度
merchno	商户号	是	String	认证系统分配
timestamp	时间戳	是	String	时间戳，单位毫秒，30 秒有效

sign	数据签名	是	String	需为小写
data	业务相关的参数	是	JSON	格式为 JSON 字符串，需要使用 SM2 加密后转成 Base64 上送，具体内容需参考对应业务接口文档

2.4 响应报文格式

响应报文为服务端给客户端的结果信息，客户端需要对服务端的响应报文进行验签和解密，响应报文格式为 JSON 格式，其数据如下

```
{
    "dsorderid": "...",
    "merchno": "...",
    "transcode": "313",
    "ordersn": "...",
    "orderid": "20220125014074021",
    "sign": "...",
    "platformCode": "001000040",
    "platformDesc": "交易成功",
    "data": "BHSR6Y0XYQyxCxOqVszt19YhifhkBUo..."
}

// data 为 JSON 格式，解密后如下(有些业务没有 data 返回，业务不同 data 内容也不同)
{
    "confidence": "0.91"
}
```

以下为返回参数介绍

参数名称	参数含义	是否必填	数据类型	说明
transcode	交易码	是	String	请求时的交易码
ordersn	商户请求流水号	是	String(32)	请求时传入的 ordersn
dsorderid	商户订单号	是	String	请求是传入的 dsorderid

merchno	商户号	是	String	认证系统分配
orderid	平台交易流水号	是	String(32)	平台唯一流水号
platformCode	平台返回码	是	String	
platformDesc	平台返回信息	是	String	描述交易具体情况
sign	数据签名	是	String	小写
data	查询结果数据	是	JSON	查询结果，为 JSON 格式字符串，需要使用 SM2 解密，注意：某些业务可能没有 data

2.5 报文加密、加签验签流程说明

简述：使用 SM2 非对称加密，私钥加签，公钥验签。
以下步骤为加签、验签以及加密详细说明

1. 通过金信反欺诈服务平台提供的工具生成客户端的 SM2 加密公钥、私钥字符串（如已生成过，则略过此步，以下简称**客户端公钥**、**客户端私钥**），生成后**客户端公钥**给到金信反欺诈服务平台，**客户端私钥**自行保管，并向客户经理获取金信反欺诈服务平台的公钥（以下简称**平台公钥**）、商户号、商户 key 等信息
2. 使用**平台公钥**对请求报文中的 data（某些业务是单个）进行加密
3. 对请求报文按字母排序进行拼接，后对拼接的字符串使用**客户端私钥**进行签名
4. 构造 HTTPS 请求，提交请求报文到金信服务端
5. 金信服务端接收到报文数据后，使用**客户端公钥**先验签，验签通过后使用**平台私钥**解密开始处理业务
6. 金信服务端处理完后，向客户端响应结果
7. 客户端收到响应结果后使用**平台公钥**验签，并使用**客户端私钥**对 data（如对应的业务有 data 返回，详看文档）进行解密

2.6 数据加密及解密

报文中的 data 字段为 JSON 格式且使用 SM2 加密后转成 Base64 的字符串，客户端和服务端的加密、解密逻辑是一致的，只需要注意加密、解密时使用的密钥即可。
文档 2.9 节中提供了 BC 库实现的 SM2 加密、解密实现，如需要可以参考（**因篇幅关系，代码可能不全，建议参考提供的 Demo**）

2.7 数据签名及验签

签名过程主要分为两个步骤，如下：

1. 生成需要签名的原始字符串
2. 使用 SM2 对原始字符串进行签名

签名原始字符串=请求报文按名称字母顺序排序拼接后的字符串（sign 不参与签名）
例如请求报文 Map 对象为（模拟数据）

```
Map<String, Object> data = new HashMap<>();
data.put("data", " IX34cjEcpG");
data.put("transcode", "118");
data.put("version", "0200");
```

对 Map 对象的 key 排序，并获取 key 和 value 进行拼接，拼接后的原始字符串为（参数名和参数值使用=号拼接）

```
data=IX34cjEcpGtranscode=118version=0200
```

对拼接后的字符串做签名（拼接待签名字符串和签名代码详看 2.8、2.9 签名工具），得到签名值如下：

```
3045022100c4e082d22cb4be2b3c8ffa2ae9edd7dc12485897b1924bded298f91c28de63cd022067d2ca15
ba83ae24abd599ced014b4c3350d5d4507dcdd9bb51ebb248dd96f8c
```

至此，我们获取到了 data、sign，则相应的请求报文如下（模拟数据，参数可能缺失）

```
{
    "data": "IX34cjEcpG",
    "transcode": "118",
    "version": "0100",
    "sign": "3045022100c4e082d22cb4be2b3c8ffa2ae9edd7dc12485897b1924bded298f91c28de63cd0220
67d2ca15ba83ae24abd599ced014b4c3350d5d4507dcdd9bb51ebb248dd96f8c"
}
```

2.8 生成待签名字符串

```
/**
 * 根据 map 的 key 排序拼接 key、value，生成待签名字符串
 */
```

```
public String getSortStr(Map<String, Object> sortedParams) {
    StringBuilder signSrc = new StringBuilder();
    List<String> keys = new ArrayList<String>(sortedParams.keySet());
    Collections.sort(keys);
    for (String key : keys) {
        Object value = sortedParams.get(key);
        if (key != null && !"".equals(key) && value != null && !"sign".equals(key)) {
            signSrc.append(key).append("=").append(value);
        }
    }
    return signSrc.toString();
}
```

生成待加签字符串后，使用 BC（工具类 BC_SM2.sign()方法）库对其进行签名即可

2.9 BC 实现 SM2 加解密、验签

注意：因篇幅原因，示例代码可能不全，且代码中只包含了对单独字符串进行加解密、验签等操作，如需完整的接口请求加解密、验签代码实现，建议参考 Demo

需要引入 BC 库

```
<dependency>
    <groupId>org.bouncycastle</groupId>
    <artifactId>bcprov-jdk15on</artifactId>
    <version>1.65</version>
</dependency>
```

实现类

```
public class BC_SM2 {

    private BouncyCastleProvider provider;
    private X9ECParameters parameters;
    private ECParameterSpec ecParameterSpec;
    private KeyFactory keyFactory;

    public BC_SM2(){
        try {
            provider = new BouncyCastleProvider();
            parameters = GMNamedCurves.getByName("sm2p256v1");
            ecParameterSpec = new ECParameterSpec(parameters.getCurve(),
                    parameters.getG(), parameters.getN(), parameters.getH());
```

```
            keyFactory = KeyFactory.getInstance("EC", provider);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 生成密钥对
     */
    public KeyPair generateSm2KeyPair() throws InvalidAlgorithmParameterException, NoSuchAl
gorithmException {
        final ECGenParameterSpec sm2Spec = new ECGenParameterSpec("sm2p256v1");
        final KeyPairGenerator kpg = KeyPairGenerator.getInstance("EC", provider);
        SecureRandom random = new SecureRandom();
        kpg.initialize(sm2Spec, random);
        KeyPair keyPair = kpg.generateKeyPair();
        return keyPair;
    }

    /**
     * 加密
     *
     * @param input  待加密文本
     * @param pubKey 公钥
     */
    public String encode(String input, String pubKey)
            throws NoSuchPaddingException, NoSuchAlgorithmException,
            BadPaddingException, IllegalBlockSizeException,
            InvalidKeySpecException, InvalidKeyException {
        X9ECParameters parameters = GMNamedCurves.getByName("sm2p256v1");
        ECParameterSpec ecParameterSpec = new ECParameterSpec(parameters.getCurve(), parame
ters.getG(), parameters.getN(), parameters.getH());
        ECPoint ecPoint = parameters.getCurve().decodePoint(Hex.decode(pubKey));
        KeyFactory keyFactory = KeyFactory.getInstance("EC", provider);
        BCECPublicKey key = (BCECPublicKey) keyFactory.generatePublic(new ECPublicKeySpec(e
cPoint, ecParameterSpec));
        Cipher cipher = Cipher.getInstance("SM2", provider);
        cipher.init(Cipher.ENCRYPT_MODE, key);
```

```
        return Base64.toBase64String(cipher.doFinal(input.getBytes()));
    }

    /**
     * 解密
     *
     * @param input  待解密文本
     * @param prvKey 私钥
     */
    public String decoder(String input, String prvKey) throws NoSuchPaddingException, NoSuc
hAlgorithmException,
            InvalidKeySpecException, InvalidKeyException, BadPaddingException, IllegalBlock
SizeException {
        Cipher cipher = Cipher.getInstance("SM2", provider);
        BigInteger bigInteger = new BigInteger(prvKey, 16);
        BCECPrivateKey privateKey = (BCECPrivateKey) keyFactory.generatePrivate(new ECPriva
teKeySpec(bigInteger,
                ecParameterSpec));
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
        return new String(cipher.doFinal(Base64.decode(input)));
    }

    /**
     * 签名
     *
     * @param plainText 待签名文本
     * @param prvKey    私钥
     */
    public String sign(String plainText, String prvKey) throws NoSuchAlgorithmException, In
validKeySpecException,
            InvalidKeyException, SignatureException {
        // 创建签名对象
        Signature signature = Signature.getInstance(GMObjectIdentifiers.sm2sign_with_sm3.to
String(), provider);
        // 将私钥 HEX 字符串转换为 X 值
        BigInteger bigInteger = new BigInteger(prvKey, 16);
        BCECPrivateKey privateKey = (BCECPrivateKey) keyFactory.generatePrivate(new ECPriva
teKeySpec(bigInteger,
```

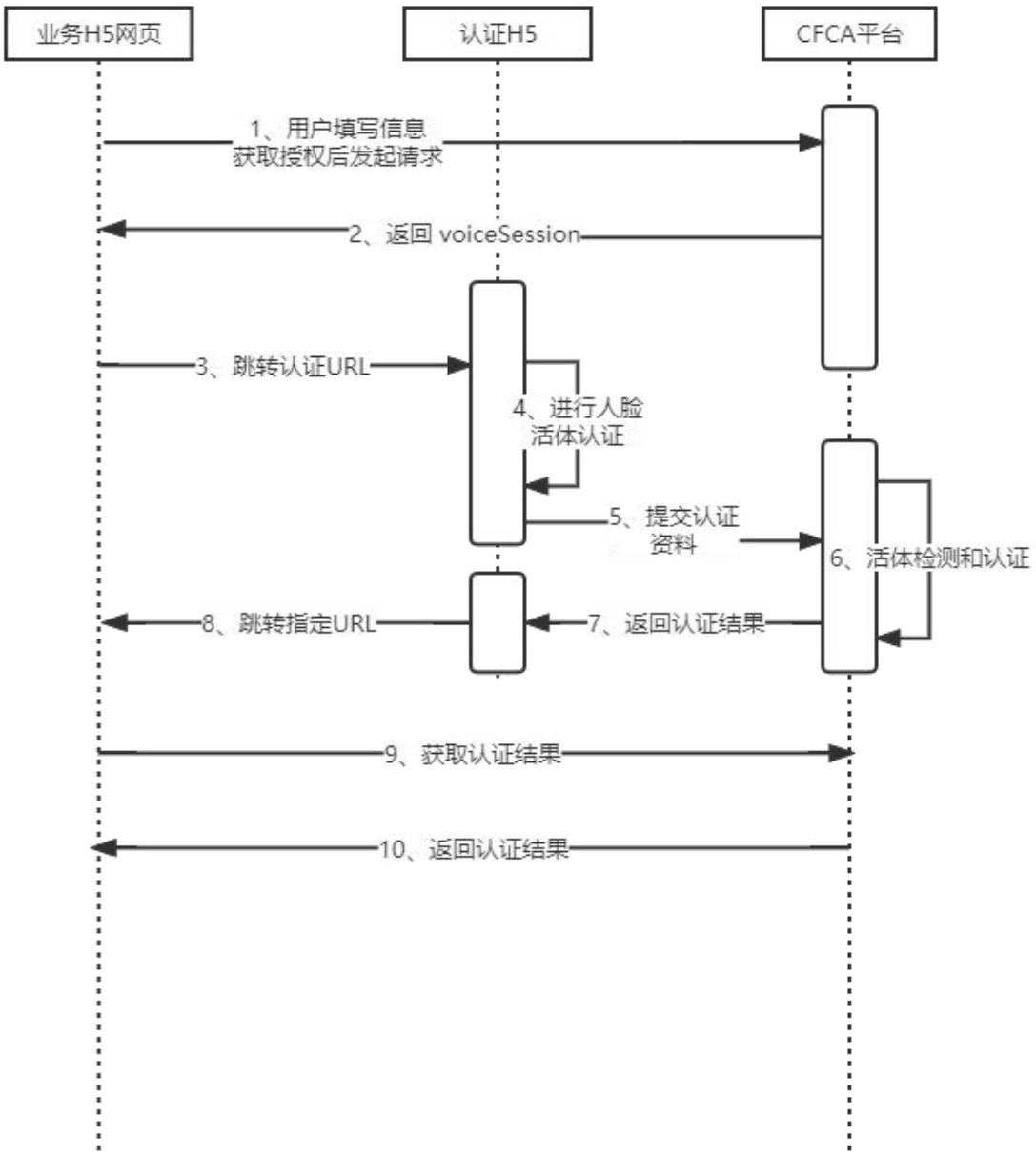
```
                ecParameterSpec));
        signature.initSign(privateKey);
        signature.update(plainText.getBytes());
        return Hex.toHexString(signature.sign());
    }

    public String sign(Map<String, Object> map, String prvKey) throws NoSuchAlgorithmExcept
ion, InvalidKeySpecException,
            InvalidKeyException, SignatureException {
        return sign(getSortStr(map), prvKey);
    }

    /**
     * 根据 map 的key 排序拼接 key、value，生成待签名字符串
     */
    public String getSortStr(Map<String, Object> sortedParams) {
        StringBuilder signSrc = new StringBuilder();
        List<String> keys = new ArrayList<String>(sortedParams.keySet());
        Collections.sort(keys);
        for (String key : keys) {
            Object value = sortedParams.get(key);
            if (key != null && !"".equals(key) && value != null && !"sign".equals(key)) {
                signSrc.append(key).append("=").append(value);
            }
        }
        return signSrc.toString();
    }

    /**
     * 验证签名
     */
    public boolean verify(String plainText, String signatureValue, String pubKey) throws No
SuchAlgorithmException, InvalidKeySpecException,
            InvalidKeyException, SignatureException {
        Signature signature = Signature.getInstance(GMObjectIdentifiers.sm2sign_with_sm3.to
String(), provider);
        ECPoint ecPoint = parameters.getCurve().decodePoint(Hex.decode(pubKey));
        BCECPublicKey key = (BCECPublicKey) keyFactory.generatePublic(new ECPublicKeySpec(e
```

```
cPoint, ecParameterSpec));
        signature.initVerify(key);
        signature.update(plainText.getBytes());
        return signature.verify(Hex.decodeStrict(signatureValue));
    }

    public static void main(String[] args) throws InvalidAlgorithmParameterException, NoSuc
hAlgorithmException {
        String str = "data=IX34cjEcpGtranscode=118version=0100aMhunM2oyKbaQa7";
        BC_SM2 sm2 = new BC_SM2();
        KeyPair keyPair = sm2.generateSm2KeyPair();
        BCECPrivateKey privateKey = (BCECPrivateKey) keyPair.getPrivate();
        BCECPublicKey publicKey = (BCECPublicKey) keyPair.getPublic();

        System.out.println("获取公私钥----------------------");
        // 公钥
        String pubKey =  "04" + (publicKey.getQ().getXCoord() + "" + publicKey.getQ().getYC
oord());
        // 私钥
        String prvKey = privateKey.getD().toString(16);

        System.out.println("公钥: " + pubKey);
        System.out.println("私钥: " + prvKey);

        System.out.println("加解密测试----------------------");
        // 加解密测试
        try {
            System.out.println("加密前：" + str);
            String encode = sm2.encode(str, pubKey);
            System.out.println("加密后：" + encode);
            String decoder = new String(sm2.decoder(encode, prvKey));
            System.out.println("解密后：" + decoder);
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("加解密错误");
        }
        System.out.println("加签验签测试----------------------");
        try {
```

```
            System.out.println("签名源数据：" + str);
            String signStr = sm2.sign(str, prvKey);
            System.out.println("签名后数据：" + signStr);
            boolean verify = sm2.verify(str, signStr, pubKey);
            System.out.println("签名验证结果：" + verify);
        } catch (Exception e) {
            System.out.println("签名和验签错误");
        }
    }
}
```

3 H5 活体识别+人像比对

H5 活体识别流程图：



3.1 获取 H5 活体识别地址

3.1.1 接口地址

测试环境请求地址：（请联系运营人员）

https://IP:端口/jxdata/api/livenessurl2.do

生产环境：https://jinxincredit.com:4446/jxdata/api/livenessurl2.do

3.1.2 请求报文

参数名称	参数含义	是否必填	数据类型	说明
transcode	交易码	是	String	交易码，如 106，每个业务交易码不同，可参对应业务的接口文档。
version	版本号	是	String	0200
ordersn	商户请求流水号	是	String(32)	唯一值，每次请求都唯一，限 32 位长度
dsorderid	商户订单号	是	String	每笔交易唯一，建议 32 位长度
merchno	商户号	是	String	平台分配
timestamp	时间戳	是	String	时间戳，单位毫秒，30 秒有效
sign	数据签名	是	String	需为小写
data	**业务相关的参数**	**是**	JSON	**格式为 JSON 字符串格式，需要使用 SM2 加密后转成 Base64 上送**
idcard	证件号码	是	String(32)	要素信息，暂只支持身份证
username	姓名	是	String(50)	要素信息

redirectUrl	回调页面地址	是	String	用户完成或取消验证后网页跳转的目标 URL（回调方法为 Post）
configId	页面配置 ID	否	String	页面配置 id，此 id 需向客户经理确认需求后获取。
isShowDefaultResultPage	是否显示结果页面	否	String	备用字段
sceneCode	交易业务场景	是	String	发起交易商户业务场景编码，枚举见最后交易场景编码
sCustomerName	二级商户名称	是	String	xxx 商户名-yyy 产品名称-zzz 使用方法
scUsePurpose	使用目的	是	String(2-64)	中文，最终使用数据的终端业务机构的使用目的
protocolVerNm	授权协议版本号	是	String(2-40)	终端商户与 C 端客户签订的电子 协议版本号(要能够查询定位到 签署授权协议的具体内容)(授权协议模版应事前备案)
serialNm	授权流水号	是	String(8-64)	C 端客户向终端商户进行信息核验(或查询）的个人信息使用授权协议签署的流水号(通过该流水号要能够定位到签署协议人员 以及日期)
reqIp	使用 IP	否	String(2-64)	网络出口 IP
remark	备注	否	String	预留字段

示例

```
{
  "transcode": "313",
  "sign": "...",
  "version": "0200",
  "ordersn": "...",
  "dsorderid": "...",
  "merchno": "...",
  "timestamp": "1643170610550",
  "data": {
    "username": "",
    "configId": "",
    "idcard": "",
    "redirectUrl": "",
    "sceneCode": "0X",
    "sCustomerName": "XX-XX-XX",
    "scUsePurpose": "XXXXX",
    "protocolVerNm": "XXXX",
    "serialNm": "XXXX",
    "reqIp": "",
    "remark": ""
  }
}
```

3.1.3 响应报文

注意：接口 data 需使用 SM2 解密

参数名称	参数含义	是否必填	数据类型	说明
transcode	交易码	是	String	交易码，如106，每个业务交易码不同，可参对应业务的接口文档。

ordersn	商户请求流水号	是	String(32)	唯一值，每次请求都唯一，限 32 位长度
dsorderid	商户订单号	是	String	每笔交易唯一，建议 32 位长度
merchno	商户号	是	String	认证系统分配
orderid	平台交易流水号	是	String(32)	平台唯一流水号
platformCode	平台返回码	是	String	描述交易具体情况
platformDesc	平台返回信息	是	String	描述交易具体情况
sign	数据签名	是	String	-
data	**业务相关的参数**	**是**	**JSON**	**格式为 JSON 字符串格式，需要使用 SM2 解密**
url	H5 活体调用地址	是	String(300)	H5 活体调用地址
voiceSession	获取到的流水号	是	String(300)	随机返回，调用【获取 H5 验证结果】接口需要上送改字段

示例

```
{
  "merchno": "xxx",
  "sign":
```

```
"522D949E300A9B6795024CDDEF5C98FD3EC548B5CBD92BFB284CA52D4AE66AD7",
  "platformCode": "001000000",
  "platformDesc": "验证通过",
  "data": {
    "url": "xxx ",
    "voiceSession": "642c112ca9d4479aa6b03fdf2e26d465"
  }
}
```

3.2 H5 调用地址的 url 对应页面展示

3.3 获取 H5 验证结果

3.3.1 请求地址

测试环境请求地址：（请联系运营人员）

https://IP:端口/jxdata/api/auth/jm/execute2.do

生产环境：https://jinxincredit.com:4446/jxdata/api/auth/jm/execute2.do

3.3.2 请求报文

参数名称	参数含义	是否必填	数据类型	说明

transcode	交易码	是	String	交易码，如 106，每个业务交易码不同，可参对应业务的接口文档。
version	版本号	是	String	0100：返回分数 0200：返回分数+相片信息
ordersn	商户请求流水号	是	String(32)	唯一值，每次请求都唯一，限 32 位长度
dsorderid	商户订单号	是	String	每笔交易唯一，建议 32 位长度
merchno	商户号	是	String	平台分配
timestamp	时间戳	是	String	时间戳，单位毫秒，30 秒有效
sign	数据签名	是	String	-
data	**业务相关的参数**	**是**	**JSON**	**格式为 JSON 字符串格式，需要使用 SM2 加密后转成 Base64 上送**
voiceSession	流水号	是	String	与 3.1 获取到的流水号对应
sceneCode	交易业务场景	是	String	发起交易商户业务场景编码，枚举见最后交易场景编码
sCustomerName	二级商户名称	是	String	xxx 商户名-yyy 产品名称-zzz 使用方法
scUsePurpose	使用目的	是	String(2-64)	中文，最终使用数据的终端业务机构的使用目的
protocolVerNm	授权协议版本号	是	String(2-40)	终端商户与 C 端客户签订的电子 协议版本号(要能够查询定位到 签署授权协议的具体内容)(授权协议模版应事前备案)
serialNm	授权流水号	是	String(8-64)	C 端客户向终端商户进行信息核验(或查询）的个人信息使用授

				权协议签署的流水号(通过该流水号要能够定位到签署协议人员以及日期)
reqIp	使用 IP	否	String(2-64)	网络出口 IP
remark	备注	否	String	预留字段

示例

```
{
  "transcode": "313",
  "sign": "...",
  "version": "0200",
  "ordersn": "...",
  "dsorderid": "...",
  "merchno": "...",
  "timestamp": "1643170610550",
  "data": {
    "voiceSession": "***",
    "sceneCode": "0X",
    "sCustomerName": "XX-XX-XX",
    "scUsePurpose": "XXXXX",
    "protocolVerNm": "XXXX",
    "serialNm": "XXXX",
    "reqIp": "",
    "remark": ""
  }
}
```

请求规则

1. 建议严格按照页面要求提示操作

3.3.3 响应报文

注意：接口 data 需使用 SM2 解密

参数名称	参数含义	是否必填	数据类型	说明

transcode	交易码	是	String	交易码，如106，每个业务交易码不同，可参对应业务的接口文档。
ordersn	商户请求流水号	是	String(32)	唯一值，每次请求都唯一，限 32 位长度
dsorderid	商户订单号	是	String	每笔交易唯一，建议 32 位长度
merchno	商户号	是	String	认证系统分配
orderid	平台交易流水号	是	String(32)	平台唯一流水号
platformCode	平台返回码	是	String	描述交易具体情况
platformDesc	平台返回信息	是	String	描述交易具体情况
sign	数据签名	是	String	-
data	**业务相关的参数**	**是**	**JSON**	**格式为 JSON 字符串格式，需要使用 SM2 解密**
confidence	置信度	是	String	置信度。值为 0~1，值越大表示本人的可能性越大，值低于 0.70 会返回

				照片比对不正确,高于或等于 0.7 返回通过
base64Image	图片数据	是	String	验证通过后会返回人脸裁剪图的 base64 编码串

示例

```
{
  “dsorderid”: ”47e6dXXXXXXXXXXXdacd8b”,
  “merchno”: ”000000000000XXXX”,
  “transcode”: ”313”,
  “ordersn”: ”47e6dXXXXXXXXXXXdacd8b”,
  “orderid”: ”201912XXXXXXXX0881”,
  “sign”: ”879ed23b7da7de282f357de454a6f04b”,
  “platformCode”: ”001000000”,
  “platformDesc”: ”验证通过”,
     "data": {
         “base64Image”: “/9j/4AAQSkZJRgA……UEn/2==”,
         “confidence”: ”0.91”
  }
}
```

3.4 移动端兼容性说明

3.4.1 移动端的 RTC 兼容性说明

webRTC 应用在炫彩活体及静默活体中。因 webRTC 技术 在 2017 年首先提出，对浏览器和手机系统存在兼容性要求，经过大规模测试，H5 活体的兼容性情况如下：

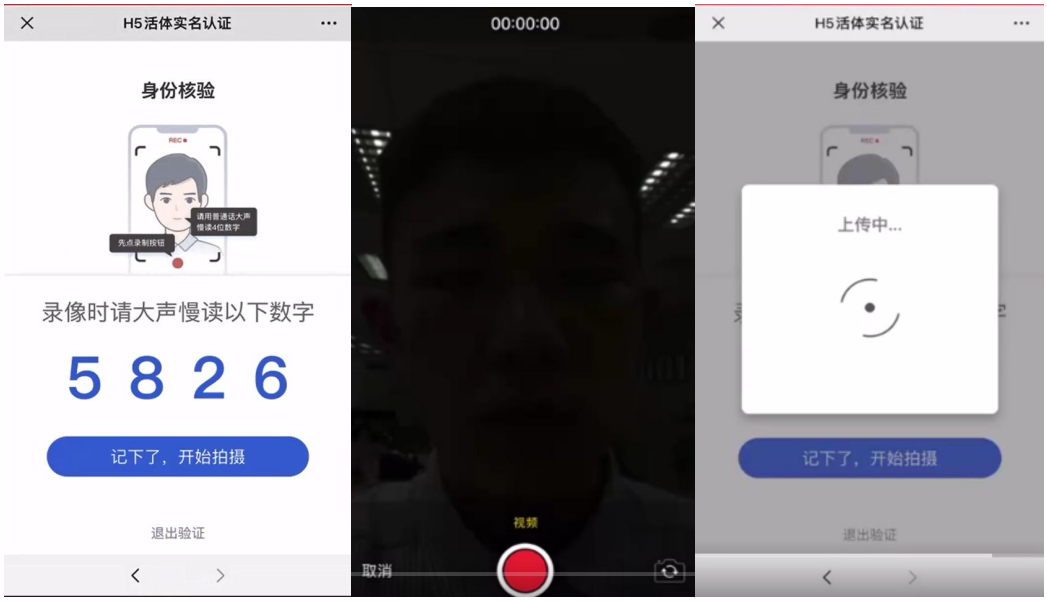
手机平台	应用端	应用端说明	兼容性要求	机型支持率
iOS	微信浏览器	在微信浏览器中使用，适用于在微信	iOS 系统版本 14.3 及以上，	95%

		中触达用户的业务场景（支持公众号，暂不支持小程序）	微信版本 6.5 及以上	
	自带浏览器	在手机浏览器使用，适用于发送链接触达用户的业务场景	iPhone7 以上，iOS 系统版本 11.1.2 及以上，Safari 浏览器版本 11 及以上	100%
	其他三方浏览器	在手机第三方主流浏览器使用，适用于发送链接触达用户的业务场景	iPhone7 以上，iOS 系统版本 11.1.2 及以上，QQ、百度、Chorme 浏览器均支持，UC 浏览器不支持	QQ 浏览器 100% 百度浏览器 100% Chorme100% UC 内核暂不支持
Android	微信浏览器	在微信浏览器中使用，适用于在微信中触达用户的业务场景（支持公众号，暂不支持小程序）	Android 系统版本 5 及以上，微信版本 6.5 及以上	95%
	自带浏览器	在手机系统自带浏览器使用，适用于发送链接触达用户的业务场景	Android 系统版本 5 及以上， 华为、VIVO、荣耀、三星等自带浏览器兼容性较好（支持率 80%），一加、小米、OPPO 自带浏览器兼容性一般（支持率 40%）	60%

	其他三方浏览器	在手机第三方主流浏览器使用，适用于发送链接触达用户的业务场景	Android 系统版本 5 及以上，QQ 浏览器支持 100%，不支持百度浏览器、UC 浏览器	QQ 浏览器 100% 百度内核暂不支持 UC 内核暂不支持

（温馨提示：当用户机型不支持时，我们仍会提供降级策略，保证用户能够通过视频活体的方式完成活体验证，流程见 3.4.2）

3.4.2 降级策略的视频录制验证操作和手机浏览器兼容列表



降级策略客户操作:

1、查看页面 4 位数字，点击【记下了，开始拍摄】

2、点击【录制】按钮开始录制视频

3、录制过程中用【普通话】读出刚才的 4 位数字

4、读完 4 位数字后，点击【录制】按钮完成录制，并上传活体视频进行验证

使用规则：建议严格按照页面要求提示操作

降级策略的视频录制应用中，若通过手机浏览器来使用移动端 H5 活体服务产品进行验证，则参考下列手机浏览器及版本的兼容性列表信息，建议在流程设计上通过更友好的提示来引导客户使用正确的浏览器来完成验证。若通过手机 APP 来进行集成移动端

H5 产品进行验证，则不存在浏览器兼容问题，兼容性主要取决于手机 APP 权限相关的实现。

若通过微信的公众号等 H5 方式进行集成，当微信版本小于 6.5 时会提示跳出到系统浏览器中；当微信版本号是 6.5 及以上版本时，则无需跳出到系统浏览器。

以下列表是经过测试的第三方浏览器：

浏览器	版本	拍摄照片	拍摄视频	本地上传照片和视频	完成 FaceID Lite 人脸核身流程
猎豹	5.13.3	是	是	是	支持
QQ	9.9.1.5730	是	是	否	支持
欧朋	12.2.0.11	是	是	否	支持
Chrome	78.0.3904.96	是	是	否	支持
360 浏览器	9.0.0.124	是	是	是	支持
2345	12.1.1	是	是	是	支持
UC	12.7.6.1056	是	是	否	支持
搜狗	5.24.6.79304	是	是	否	支持
百度	7.20	是	是	否	支持
海豚	11.3.3	是	是	是	支持
遨游云	5.2.3.3254	否	是	否	支持
Firefox	68.3.0	是	是	是	支持

注：上述测试数据是基于某型号安卓系统的手机得出的结论，可能会由于系统的差异性、软件自身适配、或软件更新所带来的差异而导致与测试不一致的情况。

以下列表是经过测试的安卓手机设备和浏览器：

机型	Android 版本	系统版本	拍摄照片	拍摄视频	本地上传照片和视频	完成 H5 活体流程
Vivo X7	5.1.1	vivo rev 1.15.2	是	是	否	支持
乐视 x620	6.0	5.6.013S 稳定版	是	是	否	支持
华为 P9	6.0	EMUI 4.1	否	否	是	不支持
红米 Note4	6.0	MIUI 8.1 稳定版	是	是	否	支持
红米 Note3	5.1.1	MIUI 7.9 稳定版	是	是	否	支持
Moto X Style	5.1.1	23.11.22	否	否	是	不支持
vivo X20	8.1	Funtouch OS_4.0	是	是	否	支持
红米 K20	10	MIUI 11.0.2 稳定版	是	是	否	支持

OPPO Find X	9	ColorOS v6.0.1	是	是	否	支持
小米 CC9	9	MIUI 11.3.1 稳定版	是	是	否	支持
小米 9	10	MIUI 11.0.5 稳定版	是	是	否	支持
红米 4X	7.1.2	MIUI 9.5 稳定版	是	是	否	支持

注 1：上述测试数据可能会由于系统自身的差异性、或软件更新所带来的差异而导致与测试不一致的情况。在遇到系统自带浏览器不支持的情况下，建议引导客户切换到第三方浏览器或者升级系统自带浏览器。

注 2：上述测试数据主要为安卓系统自带浏览器，iPhone 手机 iOS 系统版本在 6.0 之上的自带 Safari 均支持 H5 活体流程 。

4 附录

4.1 交易代码

交易代码	说明
313	H5 活体识别+人像比对

4.2 响应代码

CFCA 对外响应码	CFCA 对外响应码描述	是否收费
001000000	验证正确	是
001010034	验证信息不正确	是

001012002	证件号码或类型有误	是
001010336	照片比对不正确	是
001050337	证件照片不存在	是
001050338	未检测到人脸特征	是
001050339	活体识别失败	是
001050042	认证信息不存在	是
001050341	当前会话已失效	否
001050342	会话获取失败	否
001030050	网络超时，请稍后重试	否
001050040	交易失败，请稍后重试	否
001050060	请求参数格式错误	否
001050061	请求信息不全，请补齐	否
001050062	交易码或请求地址错误	否
001052003	订单重复，请更换商户订单号	否
001052011	参数解密失败	否
001052012	参数校验异常	否
001052013	验签失败	否
001052014	公钥错误	否
001052031	商户未注册	否
001052032	商户被禁用	否
001052033	通道权限受限,请联系客服	否
001052034	设置了黑名单，禁止访问	否
001052999	业务路由没有配置	否
001058000	验证次数超限，请次日重试	否

001058002	请求过于频繁，请稍后重试	否
001059000	渠道错误，请联系客服	否
001059001	渠道请求流水号重复，请联系客服	否
001063000	系统问题导致失败，请联系客服	否
001050050	账户余额不足，请及时充值	否
001050070	voiceSession 错误或过期，请重新发起验证	否
001009999	新增平台响应码	否

4.3 场景编码

场景代码	说明
01	直销银行
02	消费金融
03	银行二三类账户开户
04	征信
05	保险
06	基金
07	证券
08	租赁
09	海关申报
99	其他