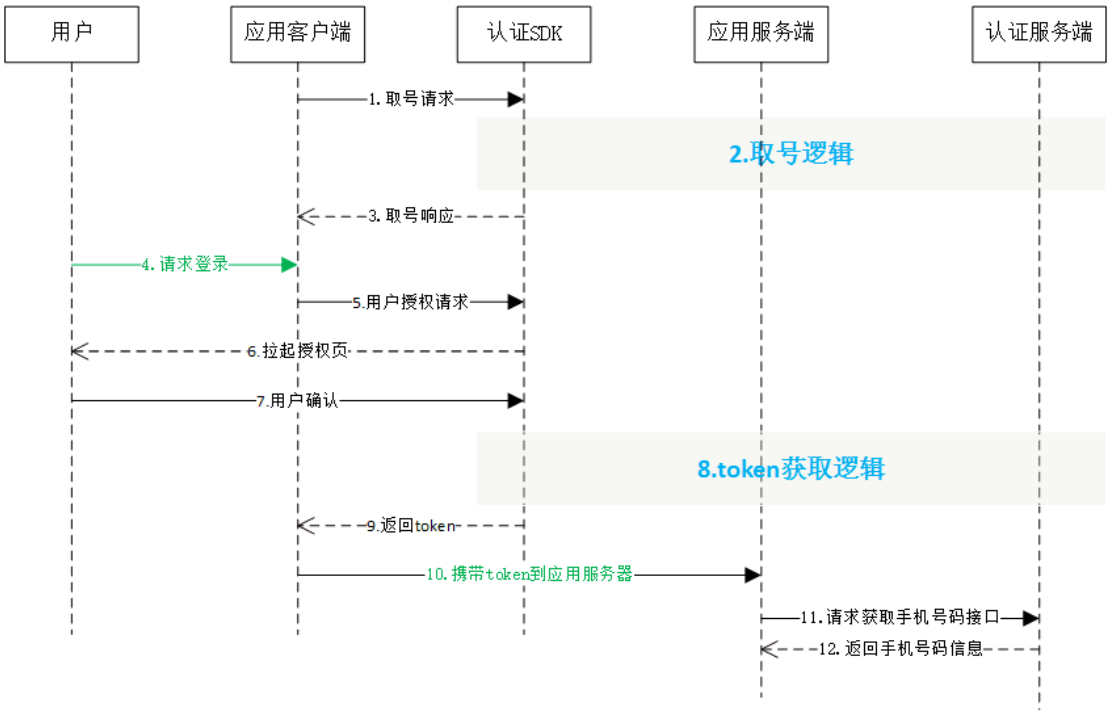
移动认证服务端接口文档-5

1. 一键登录接口

1.1. 业务流程



1.2. 接口说明

主要地址：

https://onekey2.cmpassport.com/unisdk/rsapi/loginTokenValidate

备用地址:

https://www.cmpassport.com/unisdk/rsapi/loginTokenValidate

协议： HTTPS

请求方法： POST+json，Content-type 设置为 application/json

注意：建议开发者使用主要地址进行调用，主要地址能提供更灵活稳定的运维保障

1.3. 参数说明

1、json 形式的报文交互必须是标准的 json 格式

2、发送时请设置 content type 为 application/json

3、参数类型都是 String

请求参数

参数	是否必填	说明
version	是	2.0 或 3.5。3.5 版本会返回运营商信息
msgid	是	标识请求的随机数即可(1-36 位)
systemtime	是	请求消息发送的系统时间，精确到毫秒，共 17 位，格式：20121227180001165
strictcheck	是	填写“1”，对服务器 IP 白名单进行强校验
appid	是	业务在社区申请的应用 id
expandparams	否	扩展参数
token	是	业务凭证
sign	是	请求签名，** 1）当 encryptionalgorithm="RSA"**，开发者使用在社区配置的验签公钥即应用公钥 1 对应的私钥进行签名，签名算法为 SHA256withRSA，参数组装格式为(appid+token)，签名后使用 hex 编码。2）当 encryptionalgorithm="SM"，开发者使用在社区配置的客户签名公钥对应的私钥进行签名，签名算法为 SM2，参数组装格式为(appid + version + msgid + systemtime + strictcheck + token + APPSecret)3）当 encryptionalgorithm 等于其它字符或空时，开发者使用社区生成的 APPSecret 进行签名，签名算法为 MD5，参数组装格式为(appid + version + msgid + systemtime + strictcheck + token + APPSecret)；具体签名算法可参考 1.5 提供的算法示例

encryptionalgorithm	否	加密算法类型，其中，1）当 encryptionalgorithm=RSA 时，加密类型为 RSA 算法； 2）当 encryptionalgorithm=SM 时，加密类型为国密算法；3）当****encryptionalgorithm 等于其它字符或空时，签名使用 MD5 算法，手机号码 msisdn 字段将以明文形式回传（不推荐）
clientipmd5	否	可选，用户登录时的源 IP 的 MD5 值

注 1： 对于 2018 年 12 月 26 日前申请、验签方式为 appkey 的 appid，可选择升级为使用 APPSecret 验签。但**由于升级一经提交立即生效，因此在提交前请确保已与您的服务端研发进行了充分沟通**，以免服务端生成 sign 的参数未替换导致验签失败。使用 RSA 验签加密的开发者不受影响。 **注 2：** 使用 RSA 验签加密时，生成 RSA 公私钥对请使用 PKCS#8 算法，在公钥报备时，请使用 Base64 格式

响应参数

参数	说明
inresponseto	对应的请求消息中的 msgid
systemtime	响应消息发送的系统时间，精确到毫秒，共 17 位，格式: 20121227180001165
resultCode	返回码
msisdn	用户手机号码，其中，1）当 encryptionalgorithm=RSA，开发者使用在社区配置的加密公钥，即应用公钥 2 对应的私钥进行解密；2）当 encryptionalgorithm=SM，开发者使用在社区配置的客户加密公钥对应的私钥进行解密。密文格式: Base64(0x04+c1+c3+c2)3）当****encryptionalgorithm 等于其它字符或空时，msisdn 字段将以明文形式回传（不推荐）具体签名算法可参考 1.5 提供的算法示例
taskId	话单流水号
isIPmatch	请求中携带 clientipmd5 时返回。用户获取 token 和登录的 IP 地址是否一致。0-不一致，1-一致，2-不兼容的版本。（是否

	一致不影响正常的手机号结果返回）
operatortype	运营商 0:未知 1:移动 2:联通 3:电信 version 为 3.5 时才返回

1.4. 返回码

返回码	返回码描述
103000	返回成功
103101	签名错误
103113	token 格式错误
103119	appid 不存在
103133	sourceid 不合法（服务端需要使用调用 SDK 时使用的 appid 去换取号码）
103211	其他错误
103412	无效的请求
103414	参数校验异常
103511	请求 ip 不在社区配置的服务器白名单内
103811	token 为空
104201	token 失效或不存在
105018	用户权限不足（使用了本机号码校验的 token 去调用本接口）
105019	应用未授权（开发者社区未勾选能力）
105312	套餐已用完
105313	非法请求

1.5. 示例

RSA 算法示例（Java）

```
Java
       /**
     * 私钥签名
     * @param data appid + token
     * @param algorithm SHA256withRSA
     * @param privateKeyStr
     * @return 不区分大小写
     */
  public static String sign(byte[] data, String privateKeyStr) {
    Signature signature;
    try {
      RSAPrivateKey privateKey =
getprivateKeyBykey(privateKeyStr);
      signature = Signature.getInstance("SHA256withRSA");
      signature.initSign(privateKey);
      signature.update(data);
      return byte2hex(signature.sign());
    } catch (Exception e) {
      return null;
    }

  }

  /**
   * Description：私钥解密
   *
   * @param encodeData
   *            需要解密的数据 hex
   * @return 解密后的数据
   * @author 拜力文
   */
  public static String privateKeyDecrypt(String encodeData, String
privateKeyStr) {
    Cipher cipher = null;
    try {
      RSAPrivateKey privateKey =
getprivateKeyBykey(privateKeyStr);
      byte[] data = hexStr2byte(encodeData);
      cipher = Cipher.getInstance("RSA");
      cipher.init(Cipher.DECRYPT_MODE, privateKey);
```

```
      byte[] output = cipher.doFinal(data);
      return new String(output, "UTF-8");
    } catch (Exception e) {
      e.printStackTrace();
    }
    return null;
  }

  /**
   * 将私钥字符串转换为 RSAPrivateKey 对象
   *
   * @param privatekey 私钥字符串
   * @return RSAPrivateKey 对象
   * @throws Exception 异常
   */
  public static RSAPrivateKey getprivateKeyBykey(String
privatekey) throws Exception {
    KeyFactory keyFactory = KeyFactory.getInstance("RSA");
    byte[] privateKeyByte =
Base64.getDecoder().decode(privatekey);
    PKCS8EncodedKeySpec priKeySpec = new
PKCS8EncodedKeySpec(privateKeyByte);
    RSAPrivateKey privateKey = (RSAPrivateKey)
keyFactory.generatePrivate(priKeySpec);
    return privateKey;
  }

  /**
   * Description：将二进制转换成 16 进制字符串
   *
   * @param b
   * @return
   * @return String
   * @author name：
   */
  public static String byte2hex(byte[] b) {
    String hs = "";
    String stmp = "";
    for (int n = 0; n < b.length; n++) {
      stmp = (java.lang.Integer.toHexString(b[n] & 0XFF));
      if (stmp.length() == 1) {
        hs = hs + "0" + stmp;
      } else {
        hs = hs + stmp;
```

```
      }
    }
    return hs.toUpperCase();
  }

  /**
   * Description：将十六进制的字符串转换成字节数据
   *
   * @param strhex
   * @return
   * @return byte[]
   * @author name：
   */
  public static byte[] hexStr2byte(String strhex) {
    if (strhex == null) {
      return null;
    }
    int l = strhex.length();
    if (l % 2 == 1) {
      return null;
    }
    byte[] b = new byte[l / 2];
    for (int i = 0; i != l / 2; i++) {
      b[i] = (byte) Integer.parseInt(strhex.substring(i * 2, i * 2
+ 2), 16);
    }
    return b;
    }
```

MD5 算法示例（Java）

```
Java
    public static String MD5(String data) throws Exception{
            MessageDigest md = MessageDigest.getInstance("MD5");
            md.update(data.getBytes("UTF-8"));
            //不区分大小写
            return bytes2Hex(md.digest());
    }
```

国密算法示例（Java）

1. SM2 签名示例（Java）

```
Java
```

```
// 格式：32 字节的私钥 D、96 字节的 私钥 D + 公钥 X + 公钥 Y。使用 base64 编码后传输
    String privateKeyBase64 = "base64 编码的私钥";
    // 构建私钥对象。AllSMUtils 也支持直接传 base64 编码的公私钥的方法。
    com.cmcc.idmp.rhsdk.gm.PrivateKey gmPriKey = AllSMUtils.createPrivateKey(privateKeyBase64);
    String appkey = "appkey1";
    String appid = "appid1";
    String version = "version1";
    String msgid = "msgid1";
    String systemtime = "systemtime1";
    String strictcheck = "1";
    String token = "token1";
    String appSecret = "APPSecret1";
    String doSignStr = appid + version + msgid + systemtime + strictcheck + token + appSecret;
    byte[] cipherBytes = AllSMUtils.sm2Sign1(gmPriKey, doSignStr.getBytes(StandardCharsets.UTF_8));
    //或使用这种方式
    //AllSMUtils.sm2Sign2(privateKeyBase64, doSignStr.getBytes(StandardCharsets.UTF_8));
    String sign = java.util.Base64.getEncoder().encodeToString(cipherBytes);
```

1. SM2 解密手机号（Java）

```
Java
// 密文格式： Base64(0x04+c1+c3+c2)
    String msisdnCipherBase64 = "base64 编码的手机号密文";
    // 先用 base64 解码
    byte[] msisdnCipherBytes = java.util.Base64.getDecoder().decode(msisdnCipherBase64);
    // 最后一个参数兼容其他厂商的密文格式。一般使用 false 就行
    byte[] msisdnBytes = AllSMUtils.sm2Dec2(privateKeyBase64, msisdnCipherBytes, false);
    // 明文手机号
    String msisdn = new String(msisdnBytes, StandardCharsets.UTF_8);
```

2. 本机号码校验接口

开发者获取 token 后，需要将 token 传递到应用服务器，由应用服务器发起本机号码

校验接口的调用。

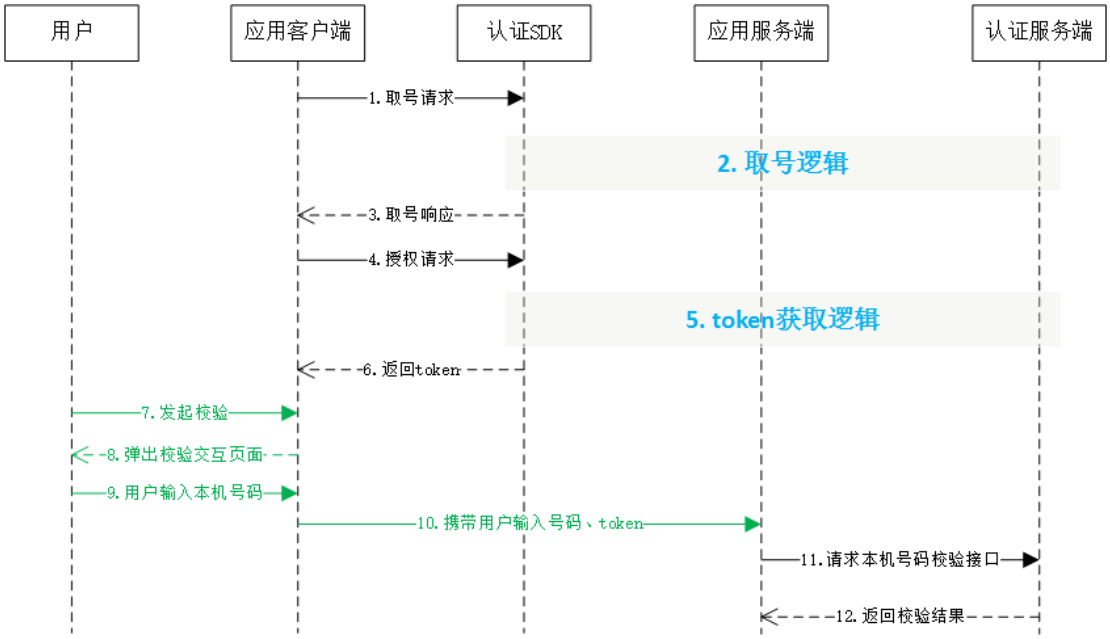
调用本接口，必须保证：

1. token 在有效期内（2 分钟）。
2. token 还未使用过。
3. 应用服务器出口 IP 地址在开发者社区中配置正确。

对于本机号码校验，需要注意：

1. 本产品属于收费业务，开发者未签订服务合同前，每天总调用次数有限，详情可咨询商务。
2. 签订合同后，将不再提供每天免费的测试次数。

2.1. 业务流程



2.2. 接口说明

调用次数说明： 本产品属于收费业务，开发者未签订服务合同前，每天总调用次数有限，详情可咨询商务。

主要地址：

https://onekey2.cmpassport.com/openapi/rs/tokenValidate

备用地址：

https://www.cmpassport.com/openapi/rs/tokenValidate

协议： HTTPS

请求方法： POST+json

注意：建议开发者使用主要地址进行调用，主要地址能提供更灵活稳定的运维保障

2.3. 参数说明

1、json 形式的报文交互必须是标准的 json 格式

2、发送时请设置 content type 为 application/json

3、参数类型都是 String

请求参数

参数	层级	是否必填	说明
****header	1	是	
version	2	是	版本号,初始版本号 1.0。2.5 版本号会返回运营商信息
msgId	2	是	使用 UUID 标识请求的唯一性
timestamp	2	是	请求消息发送的系统时间，精确到毫秒，共 17 位，格式：20121227180001165
appId	2	是	应用 ID
****body	1	是	
openType	2	否，requestertype 字段为 0 时是	运营商类型：1：移动;2：联通;3：电信;0：未知
requesterType	2	是	请求方类型：0：APP；
message	2	否	接入方预留参数，该参数会透传给通知接口，此参数需 urlencode 编码
expandPar	2	否	扩展参数格式：param1=value1

ams			
keyType	2	否	加密算法类型，其中，1）****keyType=0 时，加密类型为 SHA；2）****keyType=1 时，加密类型为 RSA；3）****keyType=2 时，加密类型为国密算法；受影响的字段包括 phoneNum、sign 和 respSign，具体算法可阅读对应的字段说明。（注：keyType 不等于其它值或空时，按 keyType=0 处理）
phoneNum	2	是	加密的手机号码，其中，1）****keyType=0 时，使用社区生成的 appkey 对手机号码进行摘要处理，加密算法是 SHA256，字母大写，参数组装格式为（待校验的手机号码 + appKey+ timestamp）。2）****keyType=1 时，使用社区提供的 RSA 公钥（即“平台公钥”）对手机号码进行加密，加密算法是 RSA，参数组装格式为（待校验的手机号码 + appKey+ timestamp）3）keyType=2 时，使用开发者社区中的国密签名公钥作为 SM3 哈希加密的 key，加密算法是 SM3，参数组装格式为（待校验的手机号码 + appKey+ timestamp）。具体签名算法可参考 2.5 提供的算法示例（注：建议开发者对用户输入的手机号码的格式进行校验，增加校验通过的概率）
token	2	是	业务凭证
sign	2	是	请求签名，其中，1）keyType=0 时，使用社区生成的 appkey 作为秘钥进行签名，加密算法是 HMACSHA256，输出 64 位大写字母。参数名做自然排序，参数组装格式为(appId + msgId + phoneNum + timestamp + token + version)。 2）keyType=1 时，使用社区中的客户公钥所对应的私钥加签，加密算法是 RSA，参数

			组装格式为(appId + msgId + phoneNum + timestamp + token + version)。3）keyType=2 时，使用开发者社区中的国密签名公钥所对应的私钥加签，加密算法是 SM2，参数组装格式为(appId + msgId + phoneNum + timestamp + token + version)。具体签名算法可参考 2.5 提供的算法示例

注： 使用 RSA 验签加密时，生成 RSA 公私钥对请使用 PKCS#8 算法，在公钥报备时，请使用 Base64 格式

响应参数

参数	层级	说明
****header	1	
msgId	2	对应的请求消息中的 msgid
timestamp	2	响应消息发送的系统时间，精确到毫秒，共 17 位，格式：20121227180001165
appId	2	应用 ID
resultCode	2	平台返回码
****body	1	
resultDesc	2	平台返回码
message	2	接入方预留参数，该参数会透传给通知接口，此参数需 urlencode 编码
expandParams	2	扩展参数格式：param1=value1
taskId	2	话单流水号话单流水号
respSign	2	响应签名，可选项，目前仅支持国密算法类型（即

		keyType=2）响应结果时，移动侧将使用社区配置的平台验签公钥对应的私钥加签，业务方可以使用平台验签公钥进行验签，确保本次请求的合法性。具体签名算法可参考 2.5 提供的算法示例。组装格式：Base64 (SM2_SIGN(msgId+timestamp+appId+resultCode+taskId))（注：“+”号为合并意 思，不包含在被加密的字符串中，字符串编码使用 utf-8）
operatortype	2	运营商 0:未知 1:移动 2:联通 3:电信 version 为 2.5 时才返回

2.4. 返回码

返回码	返回码描述
000	是本机号码（纳入计费次数）
001	非本机号码（纳入计费次数，允许使用短验辅助）
002	取号失败
003	调用内部 token 校验接口失败
004	加密手机号码错误
102	参数无效
124	白名单校验失败
302	sign 校验失败
303	参数解析错误
606	验证 Token 失败
999	系统异常
103211	其他错误

103420	本机号码校验能力使用权限不足

2.5. 示例

RSA 示例（Java）

业务方使用私钥签名

```
Java
 /**
    * @param data utf-8 bytes. appId + msgId + phoneNum +
timestamp + token + version 组成
    * @param privateKey rsa 私钥. base64 编码(NO_WRAP 格式)
    **/
   public static String sign(byte[] data, String privateKey)
throws Exception {
    byte[] keyBytes = decryptBASE64(privateKey);
    PKCS8EncodedKeySpec pkcs8KeySpec = new
PKCS8EncodedKeySpec(keyBytes);
    KeyFactory keyFactory = KeyFactory.getInstance(KEY_ALGORITHM);
    PrivateKey privateK =
keyFactory.generatePrivate(pkcs8KeySpec);
    Signature signature =
Signature.getInstance(SIGNATURE_ALGORITHM);
    signature.initSign(privateK);
    signature.update(data);
    return encryptBASE64(signature.sign());
  }
public static byte[] decryptBASE64(String key) throws Exception {
    return Base64.getDecoder().decode(key);
  }
public static String encryptBASE64(byte[] key) throws Exception {
    return Base64.getEncoder().encodeToString(key);
  }
```

业务方使用公钥加密待校验信息

```
Java
/**
   * RSA 最大加密明文大小
   */
  private static final int MAX_ENCRYPT_BLOCK = 117;
        /**
```

```
          * java 端公钥加密
           * @param data 待加密的原始字符串
            * @param PUBLICKEY 公钥
            * @return 加密的字符串
            * @throws Exception 异常
          */
  public static String encryptedDataOnJava(String data, String
PUBLICKEY) throws Exception {
    data = encryptBASE64(encryptByPublicKey(data.getBytes(),
PUBLICKEY));
    return data;
  }

  public static String encryptBASE64(byte[] key) throws Exception
{
    return Base64.getEncoder().encodeToString(key);
  }

  public static byte[] encryptByPublicKey(byte[] data, String
publicKey) throws Exception {
    byte[] keyBytes = decryptBASE64(publicKey);
    X509EncodedKeySpec x509KeySpec = new
X509EncodedKeySpec(keyBytes);
    KeyFactory keyFactory = KeyFactory.getInstance("RSA");
    Key publicK = keyFactory.generatePublic(x509KeySpec);
    // 对数据加密
    Cipher cipher = Cipher.getInstance(keyFactory.getAlgorithm());
    cipher.init(Cipher.ENCRYPT_MODE, publicK);
    int inputLen = data.length;
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    int offSet = 0;
    byte[] cache;
    int i = 0;
    // 对数据分段加密
    while (inputLen - offSet > 0) {
      if (inputLen - offSet > MAX_ENCRYPT_BLOCK) {
        cache = cipher.doFinal(data, offSet, MAX_ENCRYPT_BLOCK);
      } else {
        cache = cipher.doFinal(data, offSet, inputLen - offSet);
      }
      out.write(cache, 0, cache.length);
      i++;
      offSet = i * MAX_ENCRYPT_BLOCK;
    }
```

```
    byte[] encryptedData = out.toByteArray();
    out.close();
    return encryptedData;
  }

  public static byte[] decryptBASE64(String key) throws Exception
{
    return Base64.getDecoder().decode(key);
  }
```

SHA256 算法示例（Java）

```
Java
 /**
    @param src
    *@return 返回大写字母
    **/
    public static String sha256(String str) throws
NoSuchAlgorithmException, UnsupportedEncodingException{
        java.security.MessageDigest messageDigest;
        String encdeStr = "";
            messageDigest =
java.security.MessageDigest.getInstance("SHA-256");
            byte[] hash = messageDigest.digest(str.getBytes("UTF-
8"));
            // 可以使用其它 Hex 编码器
            encdeStr = Hex.encodeHexString(hash);
        return encdeStr.toUpperCase();
    }
```

HMACSHA256 算法示例（Java）

```
Java
 /**
     * HMACSHA256 加密
     * @param data utf-8 bytes. appId + msgId + phoneNum +
timestamp + token + version 组成
     * @param key  appkey. utf-8 bytes.
     * @return 返回大写字母
     */
    public static String HMACSHA256(byte[] data, byte[] key)
    {
          try  {
             javax.crypto.spec.SecretKeySpec signingKey = new
```

```
javax.crypto.spec.SecretKeySpec(key, "HmacSHA256");
            javax.crypto.Mac mac =
javax.crypto.Mac.getInstance("HmacSHA256");
            mac.init(signingKey);
            return bytesToHexString(mac.doFinal(data));
          } catch (NoSuchAlgorithmException e) {
            logger.error("", e);
          } catch (InvalidKeyException e) {
            logger.error("", e);
          }
          return null;
    }

    /**
    * hex 大写
    **/
    public static String bytesToHexString(byte[] src) {
        StringBuilder stringBuilder = new StringBuilder("");
        if (src == null || src.length <= 0) {
            return null;
        }
        for (int i = 0; i < src.length; i++) {
            int v = src[i] & 0xFF;
            String hv =
Integer.toHexString(v).toUpperCase(Locale.CHINA);
            if (hv.length() < 2) {
                stringBuilder.append(0);
            }
            stringBuilder.append(hv);
        }
        return stringBuilder.toString();
    }
```

国密算法示例（Java）

1. phoneNum 密文计算（Java）

```
Java
// 2. phoneNum 密文计算 Base64(SM3(手机号码+appKey+timestamp))
    String willCheclMsisdn = "159xxxxxxxx";
    //base64 格式，代表 65 个字节（0x04 + x + y）其中第一个字节为是否压缩标识，固定为 0x04。
    String publicKeyBase64 = "base64 编码公钥";
    // 得到加密的 phoneNum
```

```
    String  phoneNum =
java.util.Base64.getEncoder().encodeToString(AllSMUtils.sm3HashWit
hPublicKey2(publicKeyBase64, (willCheclMsisdn + appkey +
systemtime).getBytes(StandardCharsets.UTF_8)));
```

1. SM2 签名示例（Java）

```
Java
// 格式：32 字节的私钥 D、96 字节的 私钥 D + 公钥 X + 公钥 Y。使用 base64
编码后传输
    String privateKeyBase64 = "base64 编码的私钥";
    // 构建私钥对象。AllSMUtils 也支持直接传 base64 编码的公私钥的方法。
    com.cmcc.idmp.rhsdk.gm.PrivateKey gmPriKey =
AllSMUtils.createPrivateKey(privateKeyBase64);
    String appid = "appid1";
    String msgId = "version1";
    String phoneNum = "sm3 摘要的手机号";
    String timestamp = "timestamp1";
    String token = "token1";
    String version = "version1";
    String doSignStr = appid + msgId + phoneNum + timestamp +
token + version;
    byte[] cipherBytes = AllSMUtils.sm2Sign1(gmPriKey,
doSignStr.getBytes(StandardCharsets.UTF_8));
        //或使用这种方式
        //AllSMUtils.sm2Sign2(privateKeyBase64,
doSignStr.getBytes(StandardCharsets.UTF_8));
        String sign =
java.util.Base64.getEncoder().encodeToString(cipherBytes);
```

1. 响应验签（Java）

```
Java
// 3. 响应验签.
Base64(SM2Sign(msgId+timestamp+appId+resultCode+taskId))
    String respSign = "base64 编码的签名";
    // 解码 base64
    byte[] respSignBytes =
java.util.Base64.getDecoder().decode(respSign);
    String msgId = "msgId1";
    String timestamp = "timestamp1";
    String appId = "appId1";
    String resultCode = "resultCode1";
```

```
    String taskId = "taskId1";
    // 待验签的 bytes
    byte[] srcBytes =
(msgId+timestamp+appId+resultCode+taskId).getBytes(StandardCharset
s.UTF_8);
    // 返回是否签名正确
    boolean isGoodSign = AllSMUtils.sm2Verify2(publicKeyBase64,
srcBytes, respSignBytes);
```