接口文档

Go

签名示例

utils.go

商户对接说明

接入流程

1. 联系商务开通商户账号
2. 登录商户后台绑定google身份验证器、配置商户公钥
3. 发送服务器IP给客服**添加IP白名单**
4. 技术联调对接

接口交互

1. API地址:
2. 请求方式: post json格式
3. 金额格式: 保留小数点后两位
4. 签名算法: RSA 非对称加密 为了请求安全，请求、响应及我们回调的参数都需要进行签名验证

密钥说明

平台采用非对称加密的方式（RSA）对交易接口的数据进行验签。

- 商户私钥 通过工具地址自行生成，商户请求API接口时使用**商户私钥**对请求参数 进行**加密**。
- 商户公钥 通过工具地址自行生成，平台使用商户在后台配置的**商户公钥**进行**解密**。
- 平台公钥 商户对获取的响应数据使用商户后台显示的**平台公钥**进行**解密**signature字段。

RSA密钥工具

使用RSA加密(密钥长度1024位,密钥格式PKCS#8方式)加密

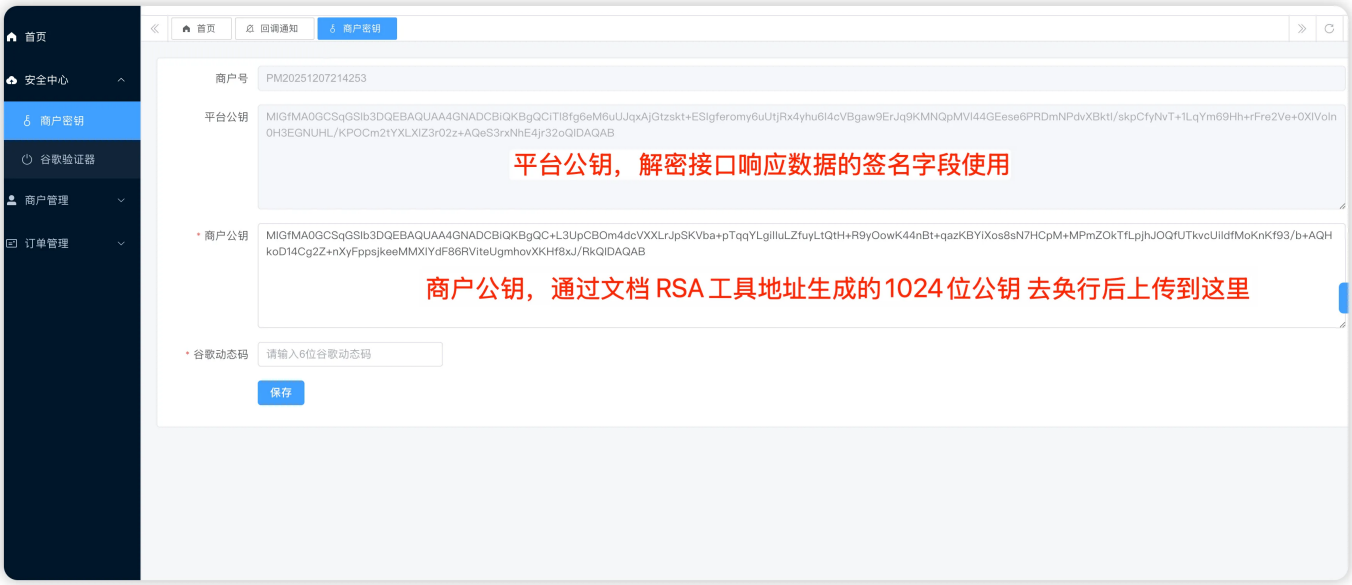
工具地址：http://web.chacuo.net/netrsakeypair

工具使用：

公钥上传之前，需要去掉空行，可以参考工具
http://www.txttool.com/WenBen_StrCompress.asp

配置密钥

商户的密钥信息不可泄露。



填写好公钥信息配置后，即完成配置。

加密规则

本平台提供的交易接口均使用HTTP协议对外提供服务，对于协议报文内容，采用以下方式进行RSA加密验签。

假设所有发送或者接收到的数据为集合 M，将集合 M 内参数值的参数**按照参数名 ASCII**码从小到大排序(字典序)，对**参数值**(即 value1value2value3…)拼接成字符串 **strX**。

1. 参数名ASCII 码从小到大排序(字典序)
2. 签名原始串(strX)中，参数值均采用原始值，不进行 URL Encode
3. signature 是签名后产生的，在接收平台回调时，解密验签比对时，不要讲该字段纳入比对

开发人员可参考文档提供的示例Demo来了解具体的加解密过程，下文对加密的过程进行详细说明。

假定用户发往服务器的原始报文内容如下：

Plain Text

```
{
    "orderNo":"T1685116153579",
    "redirectUrl":"https://google.com",
    "money":"100",
    "phone":"01234561234",
    "name":"tt",
    "description":"test",
    "callbackUrl":"http://google.com/gateway/callback",
    "expiredPeriod":"1440",
    "email":"test@gmail.com",
    "merchantNo":"S123456"
}
```

先按键值排序，然后将参数值进行拼接，该报文内容产生待签名报文内容 strX 为：

Plain Text

```
http://google.com/gateway/callbacktesttest@gmail.com1440S123456100ttT168511615357901234561234https://google.com
```

然后使用商户生成的私钥（参考配置商户）对报文内容进行加密：

Plain Text

```
BIomKoPN/m0wcDQd09zEfLNWFSbkCQCjUkZrQR+grNBM01k7zoLFEpk/0YrYL51Qa+80A1Ch2sz7v+l5wxWfoBIE7K7Yup/dzLfgbo6Ds3QEeB4NtO4ki7Flw4q/+izd+bnVMt+F8PAdM0j3ZXfw/DAGKr7S30AUXeE5tHiE0Ik=
```

最后，将新生成的报文作为补充字段添加至原始报文中。完整的发送给服务器的数据报文为：

```
{
    "orderNo":"T1685116153579",
    "redirectUrl":"https://google.com",
    "money":"100",
    "phone":"01234561234",
    "signature":"BIomKoPN/m0wcDQd09zEfLNWFSbkCQCjUkZrQR+grNBM01k7zoLFEpk/0YrYL51Qa+80A1Ch2sz7v+l5wxWfoBIE7K7Yup/dzLfgbo6Ds3QEeB4NtO4ki7Flw4q/+izd+bnVMt+F8PAdM0j3ZXfw/DAGKr7S30AUXeE5tHiE0Ik=",
    "name":"tt",
    "description":"test",
    "callbackUrl":"http://google.com/gateway/callback",
    "expiredPeriod":"1440",
    "email":"test@gmail.com",
    "merchantNo":"S123456"
}
```

代收

信用卡代收（USD）

请求URL

- /gateway/US/payIn

请求方式

- POST
- Content-Type: application/json; charset=utf-8

参数

参数名	类型	必填	说明
merchantNo	String	是	商户号（商户后台—安全中心—商户信息—商户号）
orderNo	String `<=32`	是	商户订单号 请保证唯一
money	String	是	交易金额 单位元，支持两位小数 如 0.01

description	String <=32	是	交易描述
email	String <=32	是	用户Email，如果获取不到，可以随便，格式要符合邮箱格式 如 163@gmail.com,不要有test
expiredPeriod	String	是	过期时间
phone	String <=32	是	用户电话，如果获取不到，可以随意10位手机号
callbackUrl	String <=200	是	回调通知Url，代收为异步请求，最终订单状态会通过此回调URL通知到商户接口
signature	String	是	请求加密串，校验请求参数是否合法，见加密规则
currency	String	是	币种：例如美元：USD
email	String	是	邮箱
firstName	String	是	姓
lastName	String	是	名
street	String	是	详细地址（街道）
city	String	是	城市
state	String	是	城市编码
country	String	是	国家编码
postcode	String	是	邮政编码

请求示例

{ "merchantNo": "No123456", "orderNo": "M123456", "money": "100.01", "description": "Test", "name": "Test", "email": "test@gmail.com", "phone": "081730013801", "callbackUrl": "https://www.google.com", "signature": "..." }

返回示例

{ "code": 200, "data": { "url": "支付url", "platformOrderNo": "PS17652456173762778", "orderNo": "T1765245611615", "description": "test", "fee": "18.0", "money": "100" }, "msg": "" }

返回参数说明

参数名	类型	说明
code	int	请求响应状态，非订单状态。 200为成功 其它为失败
msg	String	请求响应信息
orderNo	String	商户订单号
platOrderNo	String	平台订单号
url	String	收银台URL
money	String	交易金额
fee	String	交易手续费
description	String	交易描述
signature	String	平台加密串，校验合法性，使用商户后台平台公钥解密

代付

印度代付:

印度代付

请求URL

- `/gateway/IN/payout`

请求方式

- POST
- Content-Type: application/json; charset=utf-8

参数

参数名	类型	必填	说明
merchantNo	String	是	商户号（商户后台—安全中心—商户信息—商户号）
orderNo	String `<=32`	是	商户订单号 请保证唯一
money	String	是	交易金额 单位元，支持两位小数 如 0.01
description	String `<=32`	是	交易描述
name	String `<=32`	是	收款人账户名
bankAccount	String `<=32`	是	银行账号/UPI账号 银行账号为纯数字 如 81226678300000 UPI账号包含@字符 如 12345678xxx@indus
bankName	String `<=128`	是	银行名称 没有可取IFSC码前4位， UPI出款可不填写
ifscCode	String `=11`	是	IFSC， UPI出款可不填写
bankCode	String `=32`	否	账号类型编码 `IMPS` 或 `UPI` 不传则默认IMPS 银行账号出款
email	String `<=64`	是	邮箱
phone	String `<=32`	是	手机号

callbackUrl	String `<=200`	是	回调通知Url，代收为异步请求，最终订单状态会通过此回调URL通知到商户接口
signature	String	是	请求加密串，校验请求参数是否合法，见加密规则

请求示例

Plain Text

```
{
  "merchantNo": "No123456",
  "orderNo": "M123456",
  "money": "20.18",
  "name": "Test",
  "bankAccount": "123456",
  "callbackUrl": "https://www.google.com",
  "feeType": 1,
  "description": "Test",
  "signature": "..."
}
```

返回示例

```
{
  "responseCode": "SUCCESS",
  "responseMessage": "SUCCESS",
  "orderNo": "M123456",
  "platOrderNo": "P123456",
  "name": "Test",
  "bankAccount": "123456",
  "money": "20000.11",
  "fee": "5000.11",
  "description": "test",
  "feeType": "1",
  "status": "0",
  "signature": "..."
}
```

返回参数说明

参数名	类型	说明
code	int	请求响应状态，非订单状态。 200为成功 其它为失败
msg	String	请求响应信息
orderNo	String	商户订单号
platOrderNo	String	平台订单号
name	String	银行用户名
bankAccount	String	银行账号
money	String	交易金额
fee	String	交易手续费
status	String	订单状态，0:处理中

备注

查询

查询订单状态

请求URL

- /gateway/query

请求方式

- POST
- Content-Type: application/json; charset=utf-8

参数

参数名	类型	必填	说明
merchantNo	String	是	商户号（商户后台—安全中心—商户信息—商户号）
orderNo	String	否	商户订单号
signature	String	是	请求加密串，校验合法性，使用商户自己生成的商户私钥加密，见加密规则

请求示例

{ "merchantNo": "S2020190027344", "orderNo": "P123456", "signature": "..." }

返回示例

{ "orderNo": "M123456", "platOrderNo": "P123456", "money": "10000.12", "fee": "10.12", "status": 10, "message": "...", "utr": "...", "signature": "..." }

返回参数说明

字段名	类型	内容
orderNo	String	商户订单号
platOrderNo	String	平台订单号
money	String	交易金额
fee	String	交易手续费

status	int	状态: 10 成功
message	String	交易描述
signature	String	平台加密串，校验合法性，使用商户后台平台公钥解密

status参数详细说明

- 未支付状态：0
- 支付成功状态：10
- 订单闭关:30

备注

回调

代收回调

代收回调 POST— 格式为 JSON 形式回调

{ "orderNo": "M123456", "platOrderNo": "P123456", "money": "10000.12", "fee": "10.12", "status": 10, "message": "...", "utr": "...", "signature": "..." }

代收返回参数说明

字段名	类型	内容
orderNo	String	商户订单号
platOrderNo	String	平台订单号
money	String	交易金额
fee	String	交易手续费
status	int	状态: 10 成功
message	String	交易描述
signature	String	平台加密串，校验合法性，使用商户后台平台公钥解密

特别说明

平台回调参数以 JSON 格式，收到我方通知后请返回字符串 SUCCESS ,否则我方会每隔一段时间通知一次(一共通知 5 次)。通知的间隔时间依次为 15S, 30s, 60s, 3m,5m, 10m, 30m, 1h

签名示例

Java

签收示例

Java

```java
package com.test;

import cn.hutool.http.HttpRequest;
import cn.hutool.json.JSONObject;
import cn.hutool.json.JSONUtil;
import com.test.util.SignUtil;

import java.util.HashMap;
import java.util.Map;

public class PayIn {
    public static final String payUrl = "";

    // 商户号
    public static final String MCH_CODE = "";

    // 平台公钥
    public static final String PLAT_PUBLIC_KEY = "";

    // 商户私钥
    private static final String MCH_PRIVATE_KEY = "";

    public static void main(String[] args) {

        Map<String, Object> createMap = new HashMap<>();
        createMap.put("merchantNo", MCH_CODE);
        createMap.put("orderNo", "T" + System.currentTimeMillis());
        createMap.put("money", "100");
        createMap.put("description", "test");
        createMap.put("name", "test");
        createMap.put("email", "test@gmail.com");
        createMap.put("callbackUrl", "http://google.com/gateway/callback")
;
        createMap.put("phone", "7383442114");
        createMap.put("expiredPeriod", "1440");
        createMap.put("redirectUrl", "https://google.com");

        String signedStr = SignUtil.getSignStr(createMap, MCH_PRIVATE_KEY)
;
        createMap.put("signature", signedStr);

        String postStr = JSONUtil.toJsonStr(createMap);
        System.out.println("postJson:" + postStr);
        Long start = System.currentTimeMillis();
        String response = HttpRequest.post(payUrl)
```

```
                .header("Content-Type","application/json")
                .body(postStr).execute().body();
        JSONObject returnObj = JSONUtil.parseObj(response);
        Long end = System.currentTimeMillis();
        System.out.println("耗时:" + (end - start));
        System.out.println("return:" + returnObj);

//        System.out.println(SignUtil.verifySign(returnObj, PLAT_PUBLIC_KEY));
    }

}
```

SignUtil

Java

```java
package com.test.util;

import cn.hutool.json.JSONObject;
import lombok.extern.log4j.Log4j2;
import org.apache.commons.codec.binary.Base64;
import org.apache.commons.io.IOUtils;
import org.apache.commons.lang3.ArrayUtils;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import java.io.ByteArrayOutputStream;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;

@Log4j2
public class SignUtil {

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

    // 不仅可以使用DSA算法，同样也可以使用RSA算法做数字签名
    private static final String KEY_ALGORITHM = "RSA";

    public static String encryptByPrivateKey(String str, String privateKey)
    throws InvalidKeySpecException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
    IllegalBlockSizeException, BadPaddingException, UnsupportedEncodingException {
        // base64编码的公钥
        byte[] keyBytes = decryptBASE64(privateKey);

```

```
43          RSAPrivateKey priKey = (RSAPrivateKey) KeyFactory.getInstance(KEY_ALGORITHM)
44                                  .generatePrivate(new PKCS8EncodedKeySpec(keyBytes));
45          // RSA加密
46          Cipher cipher = Cipher.getInstance(KEY_ALGORITHM);
47          cipher.init(Cipher.ENCRYPT_MODE, priKey);
48
49          byte[] data = str.getBytes("UTF-8");
50          // 加密时超过117字节就报错。为此采用分段加密的办法来加密
51          byte[] enBytes = null;
52          for (int i = 0; i < data.length; i += MAX_ENCRYPT_BLOCK) {
53              // 注意要使用2的倍数，否则会出现加密后的内容再解密时为乱码
54              byte[] doFinal = cipher.doFinal(ArrayUtils.subarray(data, i, i + MAX_ENCRYPT_BLOCK));
55              enBytes = ArrayUtils.addAll(enBytes, doFinal);
56          }
57          String outStr = encryptBASE64(enBytes);
58          return outStr;
59      }
60
61      private static String encryptBASE64(byte[] data) {
62          return new String(Base64.encodeBase64(data));
63      }
64
65      private static byte[] decryptBASE64(String data) {
66          return Base64.decodeBase64(data);
67      }
68
69
70
71      /**
72       * Verify signature
73       *
74       * @param params
75       * @return
76       */
77      public static boolean verifySign(JSONObject params, String publickey) {
78          String platSign = params.getStr("signature"); // sign
79          log.info("signature:" + platSign);
80          List<String> paramNameList = new ArrayList<>();
81          for (String key : params.keySet()) {
82              if (!"signature".equals(key)) {
83                  paramNameList.add(key);
84              }
85          }
86          Collections.sort(paramNameList);
```

```
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < paramNameList.size(); i++) {
            String name = paramNameList.get(i);
            stringBuilder.append(params.getStr(name));
        }
        log.info("keys:" + stringBuilder);

        String decryptSign = "";
        try {
            decryptSign = publicDecrypt(platSign, getPublicKey(publickey)
);
        } catch (Exception e) {
            System.out.println(e.toString());
        }
        log.info("decryptSign:" + decryptSign);

        if (!stringBuilder.toString().equalsIgnoreCase(decryptSign)) {
            return false;
        }
        return true;
    }

    public static String getSign(Map<String, String> createMap, String MC
H_PRIVATE_KEY){
        List<String> paramNameList = new ArrayList<>();
        for (String key : createMap.keySet()) {
            paramNameList.add(key);
        }
        Collections.sort(paramNameList);  // 排序key
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < paramNameList.size(); i++) {
            String key = paramNameList.get(i);
            stringBuilder.append(createMap.get(key));  // 拼接参数
        }
        String keyStr = stringBuilder.toString();  // 得到待加密的字符串
        log.info("keyStr:" + keyStr);
        String signedStr = "";
        try {
            signedStr = privateEncrypt(keyStr, getPrivateKey(MCH_PRIVATE_
KEY));  // 私钥加密
        } catch (Exception e) {
            log.error(e);
        }
        return signedStr;
    }

    public static String getSignStr(Map<String, Object> createMap, String
 MCH_PRIVATE_KEY){
```

```
        List<String> paramNameList = new ArrayList<>();
        for (String key : createMap.keySet()) {
            paramNameList.add(key);
        }
        Collections.sort(paramNameList);  // 排序key
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < paramNameList.size(); i++) {
            String key = paramNameList.get(i);
            stringBuilder.append(createMap.get(key));  // 拼接参数
        }
        String keyStr = stringBuilder.toString();  // 得到待加密的字符串
        log.info("keyStr:" + keyStr);
        String signedStr = "";
        try {
            signedStr = privateEncrypt(keyStr, getPrivateKey(MCH_PRIVATE_KEY));  // 私钥加密
        } catch (Exception e) {
            log.error(e);
        }
        return signedStr;
    }

    /**
     * private key encryption
     *
     * @param data
     * @param privateKey
     * @return
     */
    public static String privateEncrypt(String data, RSAPrivateKey privateKey) {
        try {
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.ENCRYPT_MODE, privateKey);
            return Base64.encodeBase64String(rsaSplitCodec(cipher, Cipher.ENCRYPT_MODE, data.getBytes("UTF-8"), privateKey.getModulus().bitLength()));
        } catch (Exception e) {
            throw new RuntimeException("Exception encountered while encrypting string [" + data + "]", e);
        }
    }

    /**
     * public key decryption
     *
     * @param data
```

```
     * @param publicKey
     * @return
     */
    public static String publicDecrypt(String data, RSAPublicKey publicKey) {
        try {
            Cipher cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.DECRYPT_MODE, publicKey);
            return new String(rsaSplitCodec(cipher, Cipher.DECRYPT_MODE, Base64.decodeBase64(data), publicKey.getModulus().bitLength()), "UTF-8");
        } catch (Exception e) {
            throw new RuntimeException("An exception was encountered while decrypting the string[" + data + "]", e);
        }
    }

    /**
     * get private key
     *
     * @param privateKey key string (base64 encoded)
     * @throws Exception
     */
    public static RSAPrivateKey getPrivateKey(String privateKey) throws NoSuchAlgorithmException, InvalidKeySpecException {
        //Get the private key object through the PKCS#8 encoded Key instruction
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        PKCS8EncodedKeySpec pkcs8KeySpec = new PKCS8EncodedKeySpec(Base64.decodeBase64(privateKey));
        RSAPrivateKey key = (RSAPrivateKey) keyFactory.generatePrivate(pkcs8KeySpec);
        return key;
    }

    /**
     * get the public key
     *
     * @param publicKey key string (base64 encoded)
     * @throws Exception
     */
    public static RSAPublicKey getPublicKey(String publicKey) throws NoSuchAlgorithmException, InvalidKeySpecException {
        //Get the public key object through the X509 encoded Key instruction
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(Base64.decodeBase64(publicKey));
```

```
        RSAPublicKey key = (RSAPublicKey) keyFactory.generatePublic(x509KeySpec);
        return key;
    }

    private static byte[] rsaSplitCodec(Cipher cipher, int opmode, byte[] datas, int keySize) {
        int maxBlock = 0;
        if (opmode == Cipher.DECRYPT_MODE) {
            maxBlock = keySize / 8;
        } else {
            maxBlock = keySize / 8 - 11;
        }
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        int offSet = 0;
        byte[] buff;
        int i = 0;
        try {
            while (datas.length > offSet) {
                if (datas.length - offSet > maxBlock) {
                    buff = cipher.doFinal(datas, offSet, maxBlock);
                } else {
                    buff = cipher.doFinal(datas, offSet, datas.length - offSet);
                }
                out.write(buff, 0, buff.length);
                i++;
                offSet = i * maxBlock;
            }
        } catch (Exception e) {
            throw new RuntimeException("An exception occurred when encrypting and decrypting data whose threshold is [" + maxBlock + "]", e);
        }
        byte[] resultDatas = out.toByteArray();
        IOUtils.closeQuietly(out);
        return resultDatas;
    }

}
```

PHP

签名示例

```php

<?php
  // 平台公钥，从密钥配置中获取
  // platform public key, from Secret key config
  $platPublicKey = '';
// 商户私钥，商户自己生成
// mchchant private key
$mchPrivateKey = '';
// 商户ID，从商户信息中获取
// merchent ID from vntask, from User info
$merchantNo = 'YOUR_MERCHANT_CODE_HERE';
// 支付金额 pay money
$money = '100';

// 商户订单号
// Merchant system unique order number
$orderNo = 'T'.date("YmdHis",time());
// 描述
// The virtual account description
$description = 'Test Pay';
// 邮箱
// Customer's email address
$email = 'test@gmail.com';
// 手机号码
// Customer's mobile number
$phone = '082112345678';
// 在付款确认页面显示的转账对象
// Display name on bank confirmation display
$name = 'Neo';
// 回调地址
// url for callback
$callbackUrl = 'http://example.com/callback';
// 重定向地址
// url for redirect
$redirectUrl = 'http://example.com/redirect';
// 订单过期时间 Order expiration time
// 非必填
$expiredPeriod = '4320';

$params = array(
  'merchantNo' => $merchantNo,
  'money' => $money,
  'orderNo' => $orderNo,
  'description' => $description,
  'name' => $name,
```

```
    'email' => $email,
    'phone' => $phone,
    'callbackUrl' => $callbackUrl,
    'redirectUrl' => $redirectUrl,
    'expiredPeriod' => $expiredPeriod
);

ksort($params);
$params_str = '';
foreach ($params as $key => $val) {
    $params_str = $params_str . $val;
}


$sign = pivate_key_encrypt($params_str, $mchPrivateKey);

$params['signature'] = $sign;

$params_string = json_encode($params);
//需要找商户获取地址
$url = 'https://xxx/gateway/IN/payin';
$ch = curl_init();

curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "POST");
curl_setopt($ch, CURLOPT_POSTFIELDS, $params_string);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
curl_setopt($ch, CURLOPT_HTTPHEADER, array(
    'Content-Type: application/json',
    'Content-Length: ' . strlen($params_string))
            );
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, FALSE);

//execute post
$request = curl_exec($ch);
$httpCode = curl_getinfo($ch, CURLINFO_HTTP_CODE);

if($httpCode == 200)
{
    $result = json_decode($request, true);
    echo "responseCode :". $result['responseCode'] . "\n";
    echo "responseMessage :". $result['responseMessage'] . "\n";
    echo "platOrderNo :". $result['platOrderNo'] . "\n";
    echo "orderNo :". $result['orderNo'] . "\n";
    echo "url :". $result['url'] . "\n";
    echo "money :". $result['money'] . "\n";
    echo "fee :". $result['fee'] . "\n";
    echo "description :". $result['description'] . "\n";
```

```
  echo "signature :". $result['signature'] . "\n";

  $decryptStr = public_key_decrypt($result['signature'], $platPublicKey);
  echo "decryptStr :". $decryptStr . "\n";
}
else {
  echo $httpCode;
}

function pivate_key_encrypt($data, $pivate_key)
    {
        $pivate_key = '-----BEGIN PRIVATE KEY-----'."\n".$pivate_key."\n"
.'-----END PRIVATE KEY-----';
        $pi_key = openssl_pkey_get_private($pivate_key);
        $crypto = '';
        foreach (str_split($data, 117) as $chunk) {
            openssl_private_encrypt($chunk, $encryptData, $pi_key);
            $crypto .= $encryptData;
        }

        return base64_encode($crypto);
    }

    function public_key_decrypt($data, $public_key)
    {
        $public_key = '-----BEGIN PUBLIC KEY-----'."\n".$public_key."\n".
'-----END PUBLIC KEY-----';
        $data = base64_decode($data);
        $pu_key =  openssl_pkey_get_public($public_key);
        $crypto = '';
        foreach (str_split($data, 128) as $chunk) {
            openssl_public_decrypt($chunk, $decryptData, $pu_key);
            $crypto .= $decryptData;
        }

        return $crypto;
    }
```

Python

签名示例

Python

```python


import requests
import json
import time

from M2Crypto import RSA
from M2Crypto import BIO
import base64

def load_pub_key_string(string):
    bio = BIO.MemoryBuffer(string)
    return RSA.load_pub_key_bio(bio)


def block_data(texts, block_size):
    for i in range(0, len(texts), block_size):
        yield texts[i:i + block_size]


def encrypt(texts, pri_key):
    ciphertext = b""
    block_size = 117

    for text in block_data(texts.encode('utf-8'), block_size):
        current_text = pri_key.private_encrypt(text, RSA.pkcs1_padding)
        ciphertext += current_text

    data = base64.b64encode(ciphertext).decode("utf-8")
    return data

def decrypt(texts, pub_key):
    plaintext = b""
    block_size = 117

    for text in block_data(texts.decode("base64"), block_size):
        current_text = pub_key.public_decrypt(text, RSA.pkcs1_padding)
        plaintext += current_text

    return plaintext


# 商户ID
MCH_CODE = 'xxx'
# 平台公钥
```

```python
PLAT_PUBLIC_KEY = 'xxx'
# 商户私钥
MCH_PRIVATE_KEY = 'xxx'

payUrl = 'xxx'

params = {}
params['merchantNo'] = MCH_CODE
params['orderNo'] = 'T'+str(time.time_ns())
params['money'] = '10000'
params['description'] = 'test'
params['name'] = 'tt'
params['email'] = 'test@gmail.com'
params['callbackUrl'] = 'https://google.com'
params['phone'] = '082100138001'
params['expiredPeriod'] = '4320'
params['redirectUrl'] = 'https://google.com'

keyStr = ''
# 根据ascii码大小排序
sorted_key_list = sorted(params)
print(sorted_key_list)

for key in sorted_key_list:
    keyStr += str(params[key])

# 加密前字符串
print('加密前: ' + keyStr)


PUBLIC_KEY = "-----BEGIN PUBLIC KEY-----\n" + PLAT_PUBLIC_KEY + "\n-----END PUBLIC KEY-----"
PRIVATE_KEY = "-----BEGIN RSA PRIVATE KEY-----\n" + MCH_PRIVATE_KEY + "\n-----END RSA PRIVATE KEY-----"

pri_key = RSA.load_key_string(PRIVATE_KEY.strip('\n').encode('utf-8'))
pub_key = load_pub_key_string(PUBLIC_KEY.strip('\n').encode('utf-8'))

ciphertext = encrypt(keyStr, pri_key)
print('加密后: ' + str(ciphertext))

params['signature'] = str(ciphertext)

response = requests.post(payUrl, json.dumps(params))
# 请求内容
print('请求参数: ' + response.request.body)
# 响应内容
print('响应内容: ' + response.text)
```

```
92
```

Go

签名示例

签名示例

Go

```go
package main

import (
    "encoding/base64"
    "encoding/json"
    "github.com/farmerx/gorsa"
    "go-demo/utils"
    "log"
    "sort"
)

type PayResp struct {
    Code int    `json:"responseCode"` // 返回代码，SUCCESS（成功）
    Msg  string `json:"responseMessage"`  // 支付失败消息
}

func Sign(params map[string]string, privateKeypem string) string {
    // 对请求参数按照字母顺序进行排序并组合
    keys := make([]string, 0, len(params))
    for k := range params {
        keys = append(keys, k)
    }
    sort.Strings(keys)

    var signature string = ""
    i := 0
    for _, k := range keys {
        v := params[k]
        signature += v
        i++
    }
    log.Println("pay 签名数据：" + signature)
    gorsa.RSA.SetPrivateKey(privateKeypem)
    prienctypt, _ := gorsa.RSA.PriKeyENCTYPT([]byte(signature))
    signatureAfter := base64.URLEncoding.EncodeToString(prienctypt)
    return signatureAfter
}

func main() {

    mchCode := "xxx"

    //platPublicKey := "-----BEGIN PUBLIC KEY-----\nMIGfMA0GCSqGSIb3DQEBAQ
UAA4GNADCBiQKBgQCSLDoQwCw/HrqK46LMwYro5ASPZaJ202PrRtQM87fZnucGtSzqiPo867IU
```

```
z36/MqbZ1bkZGDZcaqV1ztBHGqq6x+ELln3tL5anKi7A/M27j1mrejbVO2lo0lLmL74pzjfEWK
bnlZ2u/qfY8JyArj2kD1t8gejGxfO6tHoFQiIa7QIDAQAB\n-----END PUBLIC KEY-----"
    mchPrivateKey := "-----BEGIN RSA PRIVATE KEY-----\nxxx\n-----END RSA P
RIVATE KEY-----"

    params := make(map[string]string)
    params["merchantNo"] = mchCode
    params["orderNo"] = "T" + time.Now().Format("20060102150405")
    params["money"] = "100"
    params["description"] = "test"
    params["name"] = "Marcelacatrinckfausto"
    params["email"] = "testpay@gmail.com"
    params["callbackUrl"] = "https://google.com/gateway/callback"
    params["phone"] = "082100138001"
    params["expiredPeriod"] = "4320"
    params["redirectUrl"] = "https://google.com"

    params["signature"] = Sign(params, mchPrivateKey)

    reqData, _ := json.Marshal(params)
    log.Println("pay 请求数据: " + string(reqData))
    body, err := utils.HttpPost("https://xxxx/gateway/IN/payin", nil, reqD
ata)
    log.Println("pay 响应结果: " + string(body))

    if err != nil {
        log.Println("pay failed", "error", err)
    }

    resp := &PayResp{}
    err = json.Unmarshal(body, resp)

    if err != nil {
        log.Println("pay, Json parse failed", "error", err)
    }
}
```

utils.go

Go

```go
package main

import (
    "encoding/base64"
    "encoding/json"
    "github.com/farmerx/gorsa"
    "go-demo/utils"
    "log"
    "sort"
)

type PayResp struct {
    Code int    `json:"responseCode"` // 返回代码，SUCCESS（成功）
    Msg  string `json:"responseMessage"`  // 支付失败消息
}

func Sign(params map[string]string, privateKeypem string) string {
    // 对请求参数按照字母顺序进行排序并组合
    keys := make([]string, 0, len(params))
    for k := range params {
        keys = append(keys, k)
    }
    sort.Strings(keys)

    var signature string = ""
    i := 0
    for _, k := range keys {
        v := params[k]
        signature += v
        i++
    }
    log.Println("pay 签名数据: " + signature)
    gorsa.RSA.SetPrivateKey(privateKeypem)
    prienctypt, _ := gorsa.RSA.PriKeyENCTYPT([]byte(signature))
    signatureAfter := base64.URLEncoding.EncodeToString(prienctypt)
    return signatureAfter
}

func main() {

    mchCode := "xxx"

    //platPublicKey := "-----BEGIN PUBLIC KEY-----\nMIGfMA0GCSqGSIb3DQEBAQ
UAA4GNADCBiQKBgQCSLDoQwCw/HrqK46LMwYro5ASPZaJ202PrRtQM87fZnucGtSzqiPo867IU
```

```
z36/MqbZ1bkZGDZcaqV1ztBHGqq6x+ELln3tL5anKi7A/M27j1mrejbVO2lo0lLmL74pzjfEWK
bnlZ2u/qfY8JyArj2kD1t8gejGxfO6tHoFQiIa7QIDAQAB\n-----END PUBLIC KEY-----"
    mchPrivateKey := "-----BEGIN RSA PRIVATE KEY-----\nxxx\n-----END RSA P
RIVATE KEY-----"

    params := make(map[string]string)
    params["merchantNo"] = mchCode
    params["orderNo"] = "T" + time.Now().Format("20060102150405")
    params["money"] = "100"
    params["description"] = "test"
    params["name"] = "Marcelacatrinckfausto"
    params["email"] = "testpay@gmail.com"
    params["callbackUrl"] = "https://google.com/gateway/callback"
    params["phone"] = "082100138001"
    params["expiredPeriod"] = "4320"
    params["redirectUrl"] = "https://google.com"

    params["signature"] = Sign(params, mchPrivateKey)

    reqData, _ := json.Marshal(params)
    log.Println("pay 请求数据: " + string(reqData))
    body, err := utils.HttpPost("https://xxxx/gateway/IN/payin", nil, reqD
ata)
    log.Println("pay 响应结果: " + string(body))

    if err != nil {
        log.Println("pay failed", "error", err)
    }

    resp := &PayResp{}
    err = json.Unmarshal(body, resp)

    if err != nil {
        log.Println("pay, Json parse failed", "error", err)
    }
}
```