bitfactory Documentation

Release 1.0

bitfactory.cn

Apr 07, 2023

星火印接入指南

星火印是星火链网主链上的一项基础性公共服务，基于 BID 标识体系，面向骨干节点用户及可信企业用户提供便捷的电子数据存证和核验服务。

CHAPTER

ONE

简介

欢迎使用星火 · 链网星火印 API 接口文档。

1、本文档用于指导可信用户使用星火印完成数据上链。

2、本文档的接口采用 HTTPS 加密方式请求。

CHAPTER

TWO

请求

访问星火 · 链网星火印 API 需要使用 https 协议，并进行数字签名。

2.1 路径

API 地址：

测试网络环境：https://test-stamp.bitfactory.cn/api

生产环境：https://stamp.bitfactory.cn/api

请求路径 =API 地址 + 接口名称，比如 hash 存证接口的名称是/evidence/hash，则请求路径为：`https://test-stamp.bitfactory.cn/api/evidence/hash`

2.2 方法

所有的请求遵循 RESTFUL 方法

2.3 参数

2.3.1 通用请求头 headers

字段名	描述
request_id	请求号，由接入客户创建唯一字符串，长度不超过 32 位
access_key	访问识别码，当您在星火印成功上传 SM2 公钥后会获得一个 access_key
nonce	请求时间, 必须以 Unix Time 的格式发送, nonce 与服务器时间不得超过正负 900 秒，否则请求将视为无效
signature	使用你的 SM2 私钥进行签名后的字符串，具体签名的方法后面会进一步描述

例如:

```
"request_id": "2XiTgZ2oVrBgGqKQ1ruCKh"
"access_key": "2y7cg8kmoGDrDBXJLaizoD"
"nonce": "1464594744"
"signature": "cdVtQ52evi4YDIuygRRiGhosn5XZyDH63LhNMk10I0LFBAamfuEBb6A2vlynVYll2ASzC/
↪yolU/pbEAZ0zxdtg=="
```

2.3.2 通用应答参数

字段名	描述
code	状态码
data	返回数据
message	错误描述

例如:

```
{
    "data": {
        "attestationId": "rBgGqKQ1ruCKhXiTgZ2oVr",
    },
    "message": "string",
    "code": "string"
}
```

2.3.3 错误码

错误码	描述
200	操作成功
500	系统内部错误
401	暂未登录或 token 已经过期
403	没有相关权限
1001	参数检验失败
1002	账户余额不足
1003	存证不存在
1004	存证失败
1005	存证处理中
1006	提取码错误
1007	存证文件已经过期
1008	请输入最大 6 个字符的标签
1009	请输入最大 20 个字符的文件名称
1010	请输入正确格式的文件 hash 值
1011	该文件已经做过存证
1012	请勿提交相同文件 hash
1013	存证失败, 已存在相同文件 hash 的存证数据
1014	存证失败, 请勿提交相同文件
1015	存证失败, 文件有安全问题
2001	存证文件不存在
2002	上传文件失败
3001	用户不存在
3002	验证码错误
4001	业务流水不唯一
4002	权限校验失败

CHAPTER

THREE

接口

3.1 初始化请求参数

```
// 构建请求
HttpRequest httpRequest = HttpUtil.createPost(uri + apiName);
// 构建请求头
Map<String ,String> headers = new HashMap<>();
headers.put("request_id", requestId);
headers.put("access_key", accessKey);
headers.put("nonce",nonce);
headers.put("signature",signatureData);
httpRequest.addHeaders(headers);
```

3.2 文件存证（分两步）:

3.2.1 上传文件 - /file/upload

客户可以通过该接口上传文件并获取文件 id，文档类文件最大限制 150M，图片类文件最大 10M，音频和视频类文件最大 550M，文件未存证时数据会在存储 7 天后删除。

form-data

参数名	描述	是否可选
file	文件	必选
type	doc: 文档 pic: 图片 audio: 音频 video: 视频	必选

返回的 data

字段名	描述
fileKey	文件 id

以 java 为例:

```
// 构建请求参数
httpRequest.form("file",new File("/tmp/背景图.png"));
httpRequest.form("type","pic");
HttpResponse httpResponse = httpRequest.execute();
String result = httpResponse.body();
```

返回结果示例: a. 接口调用成功，则返回 JSON 数据示例为：：

```
{
    "code": "200",
    "data": {
        "fileKey": "1544567382363930624"
    },
    "message": "操作成功"
}
```

b. 接口调用失败，则返回 JSON 数据示例为：：

```
{
    "code": "1001",
    "message": "fileLabel 文件标签不能为空"
}
```

3.2.2 文件存证 - /evidence/file

用户进行文件存证

请求参数

参数名	描述	是否可选
fileLabel	文件标签	必选
files	文件 id 列表	必选
files[0]	文件 id	必选

返回的 data

调用文件接口成功后会返回文件 id 对应的存证 id

字段名	描述
list	bean 对象列表
bean.id	文件 id
bean.attestationId	存证 id

以 java 为例:

```
// 构建请求参数
List<Long> list = new ArrayList<>();
list.add(1529663660129480704L);
EvidenceFileParam evidenceFileParam = new EvidenceFileParam();
evidenceFileParam.setFileLabel("标签");
evidenceFileParam.setFiles(list);
httpRequest.body(JSONUtil.toJsonStr(evidenceFileParam));
HttpResponse httpResponse = httpRequest.execute();
String result = httpResponse.body();
```

返回结果示例: a. 接口调用成功，则返回 JSON 数据示例为：：

```
{
    "code":"200",
    "data":[
        {
            "attestationId":"did:bid:ef23cydtVMQit888kfwqrZAJCccet2qQM",
            "id":"1544567382363930624"
        }
    ],
    "message":"操作成功"
}
```

b. 接口调用失败，则返回 JSON 数据示例为：：

```
{
    "code": "500",
    "message": "文件类型不支持"
}
```

3.3 hash 存证（sha256）- /evidence/hash

用户进行 hash 存证。

3.3.1 请求参数

参数名	描述	是否可选
fileLabel	文件标签	必选
list	HashInfo 对象列表	必选
HashInfo.filename	文件名	必选
HashInfo.fileHash	文件 hash	必选

3.3.2 返回的 data

调用 hash 存证接口成功后会返回存证 id 列表

字段名	描述
list	bean 对象列表
bean.hash	文件 hash
bean.attestationId	存证 id

以 java 为例:

```
// 构建请求参数
List<EvidenceHashParam.HashInfo> list = new ArrayList<>();
EvidenceHashParam.HashInfo hashInfo1 = new EvidenceHashParam.HashInfo();
hashInfo1.setFilename("test1");
hashInfo1.setFileHash(
↪"98df1f1dfb3b1a123c1517912dc70447aa61c6be532ac99de973abb6219e1653");
list.add(hashInfo1);
EvidenceHashParam evidenceHashParam = new EvidenceHashParam();
evidenceHashParam.setFileLabel("标签");
evidenceHashParam.setList(list);
httpRequest.body(JSONUtil.toJsonStr(evidenceHashParam));
HttpResponse httpResponse = httpRequest.execute();
String result = httpResponse.body();
```

返回结果示例: a. 接口调用成功，则返回 JSON 数据示例为：：

```
{
    "code":"200",
    "data":[
        {
            "attestationId":"did:bid:efaE9e45apUbuA87y7Y6zjMTaGfHt7WX",
            "hash":"98df1f1dfb3b1a123c1517912dc70447aa61c6be532ac99de973abb6219e1653"
        }
    ],
    "message":"操作成功"
}
```


b. 接口调用失败，则返回 JSON 数据示例为：：


```
{
    "code":"1010",
    "message":"请输入正确格式的文件 hash 值"
}
```


3.4 存证列表 - /evidence/list

获取存证列表

3.4.1 请求参数

参数名	描述	是否可选
evidenceType	存证类型 1. 文件存证 2.hash 存证	非必选
evidenceChannel	存证方式 1. 自助 2.API	非必选
state	3. 待支付 4. 上链中 5. 存证成功 6. 存证失败	非必选
startTime	开始时间	非必选
endTime	结束时间	非必选
pageNumber	当前页码	非必选
pageSize	每页显示数量最大 50	非必选
filename	文件名称	非必选

3.4.2 返回的 data

调用存证获取列表接口成功后会返回存证列表

字段名	描述
totalPage	当前页
pageSize	每页显示数量
pageNum	总页数
rows	存证数据对象 info
info.evidenceChannel	存证方式 1. 自助 2.API
info.attestationId	存证 id
info.auditTime	审核时间
info.auditResult	审核结果
info.fileHash	文件 hash
info.userId	用户 id
info.fileLabel	文件标签
info.filename	文件名
info.fileSize	文件大小
info.createTime	创建时间
info.upChainTime	上链时间
info.evidenceType	存证类型 1: 文件存证, 2:hash 存证
info.state	1. 待审核 2. 待复审 3. 待支付 4. 上链中 5. 存证成功 6. 存证失败
info.username	用户名称

以 java 为例:

```
// 构建请求参数
Map<String ,Object> body = new HashMap<>();
body.put("evidenceType",1);
httpRequest.body(JSONUtil.toJsonStr(body));
HttpResponse httpResponse = httpRequest.execute();
String result = httpResponse.body();
```

返回结果示例: a. 接口调用成功，则返回 JSON 数据示例为：：

```
{
    "code":"200",
    "data":{
        "totalPage":"1",
        "pageSize":"10",
        "rows":[
            {
```

(continues on next page)

(continued from previous page)


```
                "evidenceChannel":2,
                "attestationId":"did:bid:efaE9e45apUbuA87y7Y6zjMTaGfHt7WX",
                "fileHash":
↪"98df1f1dfb3b1a123c1517912dc70447aa61c6be532ac99de973abb6219e1653",
                "userId":"did:bid:zfGUkdqhxEamsPvpqAH2iRHk1ifhcW61",
                "fileLabel":"标签",
                "filename":"test1",
                "createTime":"2022-07-07 11:10:19",
                "evidenceType":2,
                "upChainTime":"2022-07-07 11:10:59",
                "state":4,
                "username":"陈诚"
            },
            {
                "evidenceChannel":2,
                "attestationId":"did:bid:ef23cydtVMQit888kfwqrZAJCccet2qQM",
                "fileHash":
↪"46d1f4f65279641891c13eb1cfba0f4a93cdd1c9e5d7cca31cd1860dbe7ca463",
                "userId":"did:bid:zfGUkdqhxEamsPvpqAH2iRHk1ifhcW61",
                "fileLabel":"标签",
                "filename":"背景图.png",
                "fileSize":"1306418",
                "createTime":"2022-07-07 11:08:51",
                "evidenceType":1,
                "state":2,
                "username":"陈诚"
            }
        ],
        "pageNum":"1",
        "total":"2"
    },
    "message":"操作成功"
}
```


b. 接口调用失败，则返回 JSON 数据示例为：：


```
{
    "code": "500",
    "message": "系统错误"
}
```

3.5 存证详情 - /evidence/detail

查询存证详情。

3.5.1 请求参数

参数名	描述	是否可选
attestationId	存证 id	必选

3.5.2 返回的 data

调用存证详情成功后会返回详情数据

字段名	描述
attestationId	存证 id
evidenceShareCode	证据提取码
pdfFileKey	pdf 文件 id
fileHash	存证文件 hash
dataExpireTime	存证文件过期时间
attestationType	存证类型 1. 文件 2.hash
dataExpireFlag	存证文件是否已过期
userId	用户 id
fileLabel	文件标签
auditTime	审核时间
auditResult	审核结果
filename	文件名
createTime	创建时间
upChainTime	上链时间
attestationChannel	数据来源 1. 自助 2.API
dataFileKey	存证文件的文件 id
username	用户名称
checkBean	链信息
checkBean.blockHash	交易 hash
checkBean.fileName	文件名称
checkBean.evidenceTime	存证时间
checkBean.flag	是否上链
checkBean.attestationId	存证 id
checkBean.confirmTime	出块时间
checkBean.confirmHash	区块 hash
checkBean.ledgerSeq	区块高度
checkBean.hash	文件 hash

以 java 为例:

```
    // 构建请求参数
Map<String ,Object> body = new HashMap<>();
body.put("attestationId","did:bid:efsRrRCTEmA7ZWodWFPkjMW2u5Y4hikv");
httpRequest.body(JSONUtil.toJsonStr(body));
HttpResponse httpResponse = httpRequest.execute();
String result = httpResponse.body();
```

返回结果示例: a. 接口调用成功，则返回 JSON 数据示例为：：

```
{
    "code":"200",
    "data":{
        "checkBean":{
            "blockHash":
↪"ec879f484d5aed9d598c3d615ea70f8246272b3d4c5796dcedc3e67a402d0905",
            "fileName":"test1",
            "evidenceTime":"2022-07-07 11:10:59",
            "flag":true,
            "attestationId":"did:bid:efaE9e45apUbuA87y7Y6zjMTaGfHt7WX",
            "confirmTime":"2022-07-07 11:11:01",
            "confirmHash":
↪"106f9a90a4ac78a45acdfe203a353562f3779ff1c6f3fc35d8914dd6a7ec06da",
            "ledgerSeq":"1113290",
            "hash":"98df1f1dfb3b1a123c1517912dc70447aa61c6be532ac99de973abb6219e1653"
        },
        "attestationId":"did:bid:efaE9e45apUbuA87y7Y6zjMTaGfHt7WX",
        "evidenceShareCode":"KD8TCISG",
        "pdfFileKey":"1544881909048279040",
        "fileHash":"98df1f1dfb3b1a123c1517912dc70447aa61c6be532ac99de973abb6219e1653",
        "attestationType":2,
        "dataExpireFlag":false,
        "userId":"did:bid:zfGUkdqhxEamsPvpqAH2iRHk1ifhcW61",
        "fileLabel":"标签",
        "filename":"test1",
        "createTime":"2022-07-07 11:10:19",
        "attestationChannel":2,
        "upChainTime":"2022-07-07 11:10:59",
        "id":"1544881469589377024",
        "username":"陈诚"
    },
    "message":"操作成功"
}
```


b. 接口调用失败，则返回 JSON 数据示例为：：


```
{
    "code": "500",
    "message": "系统错误"
}
```

3.6 下载存证或 pdf 文件 - /file/download/{fileKey}

存证原文件或 pdf 下载

3.6.1 Path

参数名	描述	是否可选
fileKey	文件 id	必选

3.6.2 返回的文件

该接口会返回存证文件以及文件名，文件就是 http 返回结果的 body，文件名存放在 http 的 header 中，header 的名称是 Content-Disposition，header 值形如:

```
form-data; name=Content-Disposition; filename=5Yhus2mVSMnQRXobRJCYgt.zip
```

以 java 为例:

```
String apiName = "/file/download/1529707935276466176";
HttpRequest httpRequest = createRequestGet(apiName);

HttpResponse httpResponse = httpRequest.execute();
String header = httpResponse.header("Content-Disposition");
Pattern pattern = Pattern.compile(".*filename=\"(.*)\".*");
Matcher matcher = pattern.matcher(header);
String fileName = "";
if (matcher.matches()) {
    fileName = matcher.group(1);
}
byte[] bytes = httpResponse.bodyBytes();
IoUtil.write(new FileOutputStream("/tmp/" + fileName),true,bytes);
```

返回结果示例: a. 接口调用成功，则返回文件流：：

```
byte[]
```

b. 接口调用失败，则返回 JSON 数据示例为：：

```
{
    "code": "2001",
    "message": "文件不存在"
}
```

CHAPTER

FOUR

签名

假定待签名数据头为:

```
"request_id": "2XiTgZ2oVrBgGqKQ1ruCKh",
"access_key": "2y7cg8kmoGDrDBXJLaizoD",
"nonce": 1464594744
```

签名过程用 Java 代码描述如下:

```
import cn.hutool.core.util.HexUtil;
import cn.hutool.core.util.IdUtil;
import cn.hutool.crypto.BCUtil;
import cn.hutool.crypto.SecureUtil;
import cn.hutool.crypto.asymmetric.SM2;
import org.bouncycastle.crypto.engines.SM2Engine;
import org.bouncycastle.jcajce.provider.asymmetric.ec.BCECPublicKey;

import java.nio.charset.StandardCharsets;
import java.security.KeyPair;
import java.util.Base64;

public class SignatureUtil {
    //签名
    public static String sign(byte[] data, String privateKey) {
        SM2 sm2 = new SM2(privateKey,null);
        sm2.setMode(SM2Engine.Mode.C1C2C3);
        sm2.usePlainEncoding();
        //签名使用 Base64 编码后得到的值即为请求头中 signature 字段的值
        return Base64.getEncoder().encodeToString(sm2.sign(data));
    }
    //验签
    public static boolean verify(byte[] data, String publicKey, String sign) {
        SM2 sm2 = new SM2(null, publicKey);
        sm2.setMode(SM2Engine.Mode.C1C2C3);
```

(continues on next page)

(continued from previous page)

```
        sm2.usePlainEncoding();
        return sm2.verify(data,Base64.getDecoder().decode(sign));
    }
    //测试用例
    public static void main(String[] args) {
        KeyPair keyPair = SecureUtil.generateKeyPair("SM2");
        String publicKey = HexUtil.encodeHexStr(((BCECPublicKey)keyPair.getPublic()).
↪getQ().getEncoded(false));
        //RSA 私钥文件路径
        String privateKey = HexUtil.encodeHexStr(BCUtil.encodeECPrivateKey(keyPair.
↪getPrivate()));
        System.out.println(publicKey);
        System.out.println(privateKey);
        long requestId = IdUtil.getSnowflakeNextId();//2XiTgZ2oVrBgGqKQ1ruCKh
        String accessKey = "2y7cg8kmoGDrDBXJLaizoD";
        long nonce = System.currentTimeMillis()/1000;//1464594744
        String data = requestId + accessKey + nonce;
        //签名
        String sign = sign(data.getBytes(StandardCharsets.UTF_8),privateKey);
        System.out.println(sign);
        //验签
        boolean verify = verify(data.getBytes(StandardCharsets.UTF_8),publicKey,sign);
        System.out.println(verify);
    }
}
```

Note: 签名所用的方法是 SM2，签名数据字符串转换成 bytes 时要用 UTF-8 编码格式

CHAPTER

FIVE

JAVA

如果使用 maven，可以加入如下依赖:

```
    <dependency>
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-all</artifactId>
    <version>5.7.22</version>
</dependency>
<dependency>
    <groupId>org.bouncycastle</groupId>
    <artifactId>bcprov-jdk15to18</artifactId>
    <version>1.71</version>
</dependency>
```

5.1 示例程序

java:

```
import cn.hutool.core.io.IoUtil;
import cn.hutool.core.util.IdUtil;
import cn.hutool.crypto.asymmetric.SM2;
import cn.hutool.http.HttpRequest;
import cn.hutool.http.HttpResponse;
import cn.hutool.http.HttpUtil;
import cn.hutool.json.JSON;
import cn.hutool.json.JSONUtil;
import org.junit.Test;

import java.io.*;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.regex.Matcher;
```

(continues on next page)

(continued from previous page)

```
import java.util.regex.Pattern;

public class ApiRequestTest {

    static class EvidenceHashParam {
        private String fileLabel;
        private List<HashInfo> list;
        static class HashInfo {
            private String filename;
            private String fileHash;

            public String getFilename() {
                return filename;
            }

            public void setFilename(String filename) {
                this.filename = filename;
            }

            public String getFileHash() {
                return fileHash;
            }

            public void setFileHash(String fileHash) {
                this.fileHash = fileHash;
            }
        }

        public String getFileLabel() {
            return fileLabel;
        }

        public void setFileLabel(String fileLabel) {
            this.fileLabel = fileLabel;
        }

        public List<HashInfo> getList() {
            return list;
        }

        public void setList(List<HashInfo> list) {
            this.list = list;
        }
```

(continues on next page)

(continued from previous page)

```
    }

    static class EvidenceFileParam {
        private String fileLabel;
        private List<Long> files;

        public String getFileLabel() {
            return fileLabel;
        }

        public void setFileLabel(String fileLabel) {
            this.fileLabel = fileLabel;
        }

        public List<Long> getFiles() {
            return files;
        }

        public void setFiles(List<Long> files) {
            this.files = files;
        }
    }

    private String uri = "http://127.0.0.1:18848/api";

    /**
     *
     * @throws Exception
     */
    @Test
    public void detail() throws Exception {
        String apiName = "/evidence/detail";
        HttpRequest httpRequest = createRequestPost(apiName);
        // 构建请求参数
        Map<String ,Object> body = new HashMap<>();
        body.put("attestationId","did:bid:efaE9e45apUbuA87y7Y6zjMTaGfHt7WX");
        httpRequest.body(JSONUtil.toJsonStr(body));
        HttpResponse httpResponse = httpRequest.execute();
        String result = httpResponse.body();
        JSON json = JSONUtil.parse(result);
        System.out.println(json.toString());
    }
```

(continues on next page)

(continued from previous page)

```
    @Test
    public void list() throws Exception {
        // API path
        String apiName = "/evidence/list";
        HttpRequest httpRequest = createRequestPost(apiName);
        // 构建请求参数
        Map<String ,Object> body = new HashMap<>();
//        body.put("attestationId","");
        httpRequest.body(JSONUtil.toJsonStr(body));
        HttpResponse httpResponse = httpRequest.execute();
        String result = httpResponse.body();
        JSON json = JSONUtil.parse(result);
        System.out.println(json.toString());
    }

    @Test
    public void hash() throws Exception {
        // API path
        String apiName = "/evidence/hash";
        HttpRequest httpRequest = createRequestPost(apiName);
        // 构建请求参数
        List<EvidenceHashParam.HashInfo> list = new ArrayList<>();
        EvidenceHashParam.HashInfo hashInfo1 = new EvidenceHashParam.HashInfo();
        hashInfo1.setFilename("test1");
        hashInfo1.setFileHash(
↪"98df1f1dfb3b1a123c1517912dc70447aa61c6be532ac99de973abb6219e1653");
        list.add(hashInfo1);
        EvidenceHashParam evidenceHashParam = new EvidenceHashParam();
        evidenceHashParam.setFileLabel("标签");
        evidenceHashParam.setList(list);
        httpRequest.body(JSONUtil.toJsonStr(evidenceHashParam));
        HttpResponse httpResponse = httpRequest.execute();
        String result = httpResponse.body();
        JSON json = JSONUtil.parse(result);
        System.out.println(json.toString());
    }

    @Test
    public void file() throws Exception {
        // API path
        String apiName = "/evidence/file";
        HttpRequest httpRequest = createRequestPost(apiName);
        // 构建请求参数
```

(continues on next page)

(continued from previous page)

```
        List<Long> list = new ArrayList<>();
        list.add(1544567382363930624L);
        EvidenceFileParam evidenceFileParam = new EvidenceFileParam();
        evidenceFileParam.setFileLabel("标签");
        evidenceFileParam.setFiles(list);
        httpRequest.body(JSONUtil.toJsonStr(evidenceFileParam));
        HttpResponse httpResponse = httpRequest.execute();
        String result = httpResponse.body();
        JSON json = JSONUtil.parse(result);
        System.out.println(json.toString());
    }
    @Test
    public void uploadFile() throws Exception {
        // API path
        String apiName = "/file/upload";
        HttpRequest httpRequest = createRequestPost(apiName);
        httpRequest.form("file",new File("/tmp/背景图.png"));
        httpRequest.form("type","video");

        HttpResponse httpResponse = httpRequest.execute();
        String result = httpResponse.body();
        JSON json = JSONUtil.parse(result);
        System.out.println(json.toString());
    }
    @Test
    public void download() throws Exception {
        // API path
        String apiName = "/file/download/1529707935276466176";
        HttpRequest httpRequest = createRequestGet(apiName);

        HttpResponse httpResponse = httpRequest.execute();
        String header = httpResponse.header("Content-Disposition");
        Pattern pattern = Pattern.compile(".*filename=\"(.*)\".*");
        Matcher matcher = pattern.matcher(header);
        String fileName = "";
        if (matcher.matches()) {
            fileName = matcher.group(1);
        }
        byte[] bytes = httpResponse.bodyBytes();
        IoUtil.write(new FileOutputStream("/tmp/" + fileName),true,bytes);
    }

    private HttpRequest createRequestPost(String apiName) throws Exception {
```

(continues on next page)

(continued from previous page)

```
        // 构建请求
        HttpRequest httpRequest = HttpUtil.createPost(uri + apiName);
        setHttpRequestHeaders(httpRequest);
        return httpRequest;
    }
    private HttpRequest createRequestGet(String apiName) throws Exception {
        // 构建请求
        HttpRequest httpRequest = HttpUtil.createGet(uri + apiName);
        setHttpRequestHeaders(httpRequest);
        return httpRequest;
    }

    private HttpRequest setHttpRequestHeaders(HttpRequest httpRequest) throws␣
↪Exception {
        // RSA 私钥文件路径
        String privateKey =
↪"308193020100301306072a8648ce3d020106082a811ccf5501822d04793077020101042ab398da2bb9268c226f4c5908e
↪";
        // 请求头
        String requestId = IdUtil.simpleUUID();
        String accessKey = "9d82aeae8c9b4c479715fc2923619472";
        String nonce = String.valueOf(System.currentTimeMillis() / 1000);

        //待签名数据 = requestId+accessKey+nonce
        String data = requestId + accessKey + nonce;
        // 开始签名
        SM2 sm2 = new SM2(privateKey,null);
        sm2.setMode(SM2Engine.Mode.C1C2C3);
        sm2.usePlainEncoding();
        // 签名使用 Base64 编码后得到的值即为请求头中 signature 字段的值
        String signatureData = Base64.getEncoder().encodeToString(sm2.sign(data.
↪getBytes(StandardCharsets.UTF_8)));
        // 构建请求头
        Map<String ,String> headers = new HashMap<>();
        headers.put("request_id", requestId);
        headers.put("access_key", accessKey);
        headers.put("nonce",nonce);
        headers.put("signature",signatureData);
        httpRequest.addHeaders(headers);
        return httpRequest;
    }


}
```