

Treasury API OpenAM 版文档

本文档旨在协助中国大陆地区用户接入汇丰银行 API 服务。

目录

1. Treasury – 付款发起 API	3
1.1 API 信息	3
1.1.1 描述	3
1.1.2 版本历史	3
1.2 API 安全与认证	4
1.2.1 安全概况	4
1.2.2 传输层安全	4
1.2.3 消息级别的加密	4
1.2.4 API 身份验证 - JWT 验证	6
1.2.5 JWT Header 中的字段	7
1.2.6 JWT Payload 中的字段	7
1.3 实施指南	8
1.3.1 步骤 1-准备请求正文	9
1.3.2 步骤 2-加密 HTTP 请求正文	9
1.3.3 步骤 3-生成和签名客户令牌	12
1.3.4 步骤 4-添加签名的 JWT(JWS)到 HTTP 头	13
1.3.5 步骤 5-添加保留的头部并提交	13
1.3.6 处理响应	13
1.4 POST /bulk-payments/instant-receipt	19
1.4.1 描述	19
1.4.2 API 请求头	19
1.4.3 API 请求正文	19
1.4.4 API 响应正文	20
1.5 POST /payments/instant-receipt	21
1.5.1 描述	21
1.5.2 API 请求头	21
1.5.3 API 请求主体	22
1.5.4 API 响应主体	22
1.6 POST /payments/instant-settlement	23
1.6.1 描述	24
1.6.2 API 请求头	24
1.6.3 API 请求主体	25
1.6.4 API 响应主体	25
1.7 POST /payments/instant-settlement (IMPS)	26
1.7.1 描述	26
1.7.2 API 请求头	27

1.7.3 API 请求主体.....	27
1.7.4 API 响应主体.....	29
1.8 POST /payments/status.....	31
1.8.1 描述	31
1.8.2 API 请求头	32
1.8.3 API 请求主体.....	32
1.8.4 API 响应主体.....	33
1.9 Treasury API 沙盒	37
2. Treasury – 账户信息 API.....	38
2.1 API 信息	38
2.1.1 描述	39
2.1.2 版本历史	39
2.2 API 安全与认证.....	39
2.2.1 安全概况	39
2.2.2 传输层安全.....	40
2.2.3 消息级别的加密	40
2.2.4 API 身份验证 - JWT 验证.....	41
2.2.5 JWT Header 中的字段	42
2.2.6 JWT Payload 中的字段	43
2.3 实施指南	44
2.3.1 步骤 1-准备请求正文.....	44
2.3.2 步骤 2-加密 HTTP 请求正文.....	44
2.3.3 步骤 3-生成和签名客户令牌.....	47
2.3.4 步骤 4-添加签名的 JWT(JWS)到 HTTP 头	49
2.3.5 步骤 5-添加保留的头部并提交	49
2.3.6 处理响应	49
2.4 POST /balances	54
2.4.1 描述	54
2.4.2 API 请求正文.....	55
2.4.3 API 响应正文.....	57
2.5 POST /transactions.....	59
2.5.1 描述	59
2.5.2 API 请求正文.....	60
2.5.3 API 响应标头.....	61
2.5.4 API 响应正文.....	62
2.6 Treasury API 沙盒	67
2.6.1 使用指南	67
2.6.2 沙盒的响应.....	68
2.6.3 智能沙盒支持的测试账户.....	68
2.6.4 Postman Collection.....	69

1. Treasury – 付款发起 API

1.1 API 信息

版本: v3

协议: HTTPS

沙盒环境 URL: <https://sandbox.corporate-api.hsbc.com/>

生产环境 URL: **生产环境 – 请联系您的客户经理**

ISO pain.001.001.03 规范: [HSBC pain.001.001.03 HighValuePayment](#)

1.1.1 描述

付款发起 API 用于发起单一支付指令或大宗支付指令，通过该 API 客户可从指定帐户发起付款操作，付款对象可以是单个或多个受益人。

API 支持的付款类型和相应的 URL 路径：

- 即时单笔支付，即时结算 - /payments/instant-settlement
- 带有即时收款收据的单次付款 - /payments/instant-receipt
- 带有即时收款收据的批量付款 - /bulk-payments/instant-receipt
- 付款状态查询 - /payment/status

使用付款发起 API 提交的付款在发送到付款处理系统之前需要经过格式化、付款规则校验等处理。

对于单笔或批量即时付款操作，你将收到 API 已成功接受、验证、处理成功的确认信息。

我们的即时支付 API 能为你提供受益行接受、结算处理完成的确认信息。

除非另有说明，否则这些 API 接受和返回的信息遵循 ISO 20022 Payment Initiation (pain) 报文传输标准。

pain.001.001.03 客户信用转账发起报文用于请求将资金从借方帐户转至贷方账户。

对于印度地区的实时支付，有效载荷必须是专有的 XML 格式(XMLPF)，即当提供的支付国家/地区为印度时，才可以使用该消息格式发送 RTP 支付和进行支付状态查询。

本文档为企业、组织在其系统和流程中集成 HSBC 支付操作，处理请求响应提供了操作指南

1.1.2 版本历史

版本	发布日期	状态	描述
V1	2018 年 12 月	启用	支付 API
V3	2023 年 9 月	启用	OpenAM 实施规范 — 付款启动操作

1.2 API 安全与认证

本节内容将帮助你：

- 了解保护和加密与 HSBC 连接所需的步骤
- 查看 JWT 头部和负载中要提供的字段
- 了解如何生成所需 JSON Web 令牌 (JWT) 的代码示例

1.2.1 安全概况

要在汇丰银行和使用方之间建立安全连接，需要采取及使用以下措施和组件：

类别	定义	组件
传输层安全协议(TLS)	HTTPS 连接	数字证书(SSL)
消息级别安全	数据加密	PGP 私钥和公钥
API 认证	对 API 请求进行认证	签名和 Base64 编码的 JWT (JWS) - 由客户创建
客户身份认证	客户的身份信息	客户 Profile ID – 由汇丰银行方提供

1.2.2 传输层安全

为了通信的安全，信息传输会采用加密方式进行，即客户和汇丰银行 API 之间的连接需要使用 HTTPS 协议和传输层安全协议 (TLS)。 TLS 是一种加密协议，目的是在计算机数据传输层面提供通信安全。

当客户请求与汇丰银行建立 HTTPS 连接时，需要使用数字证书。该证书包含开启安全套接字层 (SSL) 握手所需的加密技术，其中涉及生成共享密钥以在客户和汇丰银行之间建立特定的安全连接。

在尝试连接到我们的 API 之前，请确保您的组织拥有来自公认的证书颁发机构颁发的有效数字证书。

1.2.3 消息级别的加密

除了传输层安全性之外，汇丰银行还在会话层的消息级别采用了额外的安全性措施。消息级加密使用了基于公钥的非对称加密方案，为企业、组织和汇丰银行之间提供 API 消息负载级别的安全性增强。

公钥加密涉及一对称为公钥和私钥的密钥。这些密钥分配给需要以电子方式进行身份验证、或要对数据进行签名或加密的企业实体或组织。每个公钥都是公开的，而相应的私钥是保密的。使用公钥加密的数据只能使用相应的私钥解密。

公钥加密可实现加密、解密和不可否认性。汇丰银行的 API 支持非对称加密，即使用公钥加密数据，使用私钥解密数据。公钥在受信任的证书中可用，而私钥是保密的且不可以共享的。

出于消息加密的目的，您的企业、组织和汇丰银行都需要交换公钥，以便双方可以通过 API 加密和解密消息。

PGP 加密

PGP 是一种加密方案，为数据通信提供加密隐私和身份验证。PGP 用于对文本、电子邮件、文件、目录和整个磁盘分区进行签名、加密和解密，并提高电子邮件通信的安全性。

1. PGP 加密的目的是什么？

出于 API 消息加密的目的，必须在 API 消费者（客户端）和 API 提供商（HSBC）之间交换公共 PGP 密钥。

2. 怎样获取 PGP 密钥对？

客户可以使用 GnuPG 或 GPG 等密钥生成工具生成自己的私钥/公钥对。

公共 PGP 密钥规范：

- 签名算法：2048 位
- RSA 哈希算法：SHA-256

您的证书准备就绪后，请联系您的 HSBC 客户端集成联系人以触发密钥交换程序。

3. 怎么使用 PGP 密钥对？

客户端每次向汇丰银行发送消息时都使用汇丰银行的公钥来加密消息。每次从汇丰银行收到消息时，客户都会验证汇丰银行的数字签名。汇丰银行使用客户的公钥对从客户发送和接收的消息执行相同的操作。

出于安全目的，汇丰银行的公钥每年都会更新，并且会触发证书更新流程。

4. PGP 密钥的保存和有效期

组件	存储	有效期
客户方私钥	应使用尽可能最安全的方法维护和处理私钥。最常见且安全的方法是：密钥密码 - 不要以纯文本或硬编码的形式将密码保存在应用程序中。我们建议使用密码加密工具/密钥存储对其进行加密 - 存储在受密码保护的密钥存储库中，例如 JKS 或 PKCS12 密钥存储库。密钥库密码也应该加密。	没有有效期。然而，如果客户怀疑密钥可能被泄露，或任何其他安全原因，则应生成新的私钥及其关联的公钥。
客户方公钥	由于公钥是公开分发的，因此中等安全的存储方法是可以接受的。客户端可以将物理文件存储在任何计算机的文件系统中，或者用于集中密钥管理 - 将所有密钥和证书存储在一个密钥存储库中。	对于自签名 PGP 公钥，适用与上述相同的条件。但是，CA 签名的证书的有效期限取决于颁发 CA 的购买计划。最常见的标准是 1 到 2 年。
汇丰银行方公钥	同上	有效期通常为 1 年，另加 1 至 2 个月。额外的时间是一个缓冲区，使客户端能够在 PGP 密钥更新过程中将“即将过期”的证书切换到新的证书。

1.2.4 API 身份验证 - JWT 验证

API 身份验证是使用签名的 JSON Web 令牌 (JWT) 执行的。每次调用汇丰 API 服务时，都必须在 HTTP 请求标头中提供令牌。这确保只有经过授权和预先批准的组织才能访问和使用我们的 API 服务。JWT 由客户生成，而不是由汇丰银行提供。有许多实用程序可以为所有常用的编程语言提供 JWT 生成和验证逻辑。以下示例使用“io.jsonwebtoken”java 库。

创建 JWT 你需要：

- 客户私钥
- 客户公钥
- 客户 Profile ID

客户 Profile ID 是 HSBC 在创建新用户过程中提供给您标识符。

私钥是加密密钥对的一部分需妥善保管避免泄露，公钥需与汇丰银行共享，是客户创建流程的一部分。

对于沙盒访问，用到的加密密钥和 Profile ID 将在创建沙盒项目过程中提供。

代码样例

```
//1. Build the JWT structure
Map<String, Object> claims = new HashMap<>();

long nowSeconds = System.currentTimeMillis() / 1000;
claims.put("iat", nowSeconds);
claims.put("jti", UUID.randomUUID().toString());
claims.put("sub", "BS0000111111"); //your profile id
claims.put("aud", "baas");

//2 Generate hash code for request payload and put into JWT Token
claims.put("payload_hash", "038c3d8437c49840591baf7eb2a2f292c4f959acfe048444da48fc376df36b87");
//your payload hash code
claims.put("payload_hash_alg", "RSASHA256");

//3 Set JWT header
Map<String, Object> header = new HashMap<String, Object>();
header.put("kid", "71F28C2FB8886368"); //key id
header.put("typ", "JWT");
header.put("ver", "1.0");
header.put("key_typ", "PGP");
header.put("alg", "PS256");

//4 Generate and Sign the JWT (JWS)
PrivateKey rsaKey = pgpHelper.convertToRsaKey(getPrivateKey());
String token = Jwts.builder().setHeader(header).setClaims(claims).signWith(rsaKey,
SignatureAlgorithm.PS256).compact();
```

1.2.5 JWT Header 中的字段

下表定义了 JWT Header 中要用到的字段

键名	名称	是否必填	描述	示例
typ	令牌类型	是	必须设置为 JWT	JWT
kid	PGP Key ID	是	HSBC 用于验证 JWT 签名的公钥 ID。汇丰银行从客户提供的 PGP 密钥文件中通过此密钥 ID 读取公钥,以验证 JWT 签名。	900864F8C11EB743
alg	签名算法	是	用于验证令牌上的签名的算法。支持以下算法: RS256、RS384、RS512、PS256、PS384、PS512。	PS256
ver	版本	是	汇丰私有声明。目前客户须将其设置为“1.0”。	1.0

注: 我们推荐 PS256、PS384 和 PS512 算法, 而不是其他 RSA 算法。PKCS#1.5 填充方案 (RS256、RS384 和 RS512) 没有提供所需的安全级别, 通常应由 PSS 填充方案取代。

示例:

```
{  
  "typ": "JWT",  
  "kid": "A9248D0E4A463426",  
  "alg": "PS256"  
  "ver": "1.0"  
}
```

1.2.6 JWT Payload 中的字段

下表定义了 JWT Payload 中要用到的字段

键名	名称	是否必填	描述	示例
jti	JWT ID	是	令牌是区分大小写的唯一标识符, 即使在不同发行者之间也是如此。将相同的值意外分配给不同数据对象的概率可以忽略不计; 如果应用程序使用多个发布者, 则必须在不同发布者生成的值之间避免冲突。“jti”声明可用于防止 JWT 被重播。“jti”值是区分大小写的字符串。	74760410-f963-11e8-b2a3-1bb26e1e5b69

iat	签发时间	是	标识 JWT 签发的时间，值必须是日期的数值形式	1565150776
sub	主体	是	标识 JWT 的主体，必须是客户的 Profile ID	PC000012345
aud	受众	是	标识 JWT 的接收方，一直设置成 baas	baas
payload_hash	Payload 哈希码	是	HTTP 请求正文的 hash 码，用于 Mule Policy 做校验	1dab559f1f4bf4f3ca...
payload_hash_alg	Payload 哈希算法	是	Payload 的哈希算法，支持以下算法： ● SHA-256 (RSASHA256) ● SHA-384 (RSASHA384) ● SHA-512 (RSASHA512)	RSASHA256

示例：

```
{
  "jti": "5ccfd3a0-36a1-11ea-b780-eeee0af2723c",
  "iat": 1578987726,
  "sub": "PC000012345",
  "aud": "baas",
  "payload_hash": "a00ccb...",
  "payload_hash_alg": "RSASHA256",
}
```

1.3 实施指南

本节内容将帮助你：

- 准备 API 请求有效负载并执行有效负载加密
- 生成并签署客户端令牌 (JWT)
- 处理 API 响应所需的步骤

对于每个请求，请准备以下内容：

1. 准备 JSON 结构中的普通 HTTP 请求正文。
2. 使用 HSBC 的公钥加密 HTTP 请求正文。
3. 生成客户端令牌(JWT)。
4. 在相关的授权请求标头中添加已签名 JWT(JWS)。
5. 添加关联 ID 以唯一标识系统中的请求并提交。

以下部分包含带有示例代码的实施指南。

1.3.1 步骤 1-准备请求正文

使用 API 参考文档根据特定的服务的 API 数据格式准备 payload 数据。

API 参考文档中的每个 API 服务地址都提供的请求正文供参考 — 请参见端点菜单。

1.3.2 步骤 2-加密 HTTP 请求正文

组件	描述	备注
明文	HTTP 请求正文 (JSON 格式)	参考 API 断点页
密钥	HSBC 提供的公钥	密钥交换时由 HSBC 提供的公钥

消息正文加密步骤：

- 使用第一步准备的请求正文
- 使用 HSBC 提供的公钥加密消息体
- 使用 Base64 编码加密的消息体

代码示例：

为了了解如何构造加密消息，让我们以下面的 Java 程序为例来描述所有必需的组件

```
/**
 * Returns the encrypted and signed message
 *
 * @param plainText message which need to be encrypted
 * @param encryptKey the public key used for encryption provided by HSBC
 * @param signKey your own private key for signing
 * @return Encrypted message encoded by base64
 * @throws Exception
 */
public String encryptAndSign(String plainText, PGPPublicKey encryptKey, PGPPrivateKey signKey, String
signer) throws Exception {
    if (plainText == null || plainText.length() <= 0) {
        throw new IllegalArgumentException("Plain text is empty.");
    }
    ByteArrayInputStream in = new ByteArrayInputStream(plainText.getBytes());
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    encryptAndSign(out, in, encryptKey, signKey, signer);
    byte[] encrypted = out.toByteArray();
    String encryptedBody = new String(Base64.getEncoder().encode(encrypted), Charset.defaultCharset());
    in.close();
    out.close();
    return encryptedBody;
}

/**
 * Returns the encrypted and signed message
```

```

* @param outputStream Encrypted and signed output stream
* @param inputStream Input stream that needs to be encrypted
* @param encryptKey the public key used for encryption provided by HSBC
* @param signKey your own private key for signing
* @param signer the name of signer
* @throws Exception
*/
public void encryptAndSign(OutputStream outputStream, InputStream inputStream, PGPPublicKey
encryptKey, PGPPrivateKey signKey, String signer) throws Exception {
    OutputStream out = new ArmoredOutputStream(outputStream);
    try {
        //generate encrypted data packets
        PGPEncryptedDataGenerator encryptedDataGenerator = new PGPEncryptedDataGenerator(new
JcePGPDataEncryptorBuilder(PGPEncryptedData.AES_256)
            .setWithIntegrityPacket(true)
            .setSecureRandom(new SecureRandom())
            .setProvider(bcp));
        encryptedDataGenerator.addMethod(new
JcePublicKeyKeyEncryptionMethodGenerator(encryptKey).setProvider(bcp));
        OutputStream encryptedOutputStream = encryptedDataGenerator.open(out, new
byte[DEFAULT_BUFFER_SIZE]);

        PGPCompressedDataGenerator compressedDataGenerator = new
PGPCompressedDataGenerator(PGPCompressedData.ZIP);
        OutputStream compressedData = compressedDataGenerator.open(encryptedOutputStream);

        //generate complete OpenPGP signed packets
        PGPSignatureGenerator signatureGenerator = new PGPSignatureGenerator(new
JcaPGPContentSignerBuilder(signKey.getPublicKeyPacket().getAlgorithm(),
PGPUtil.SHA512).setProvider(bcp));
        signatureGenerator.init(PGPSignature.BINARY_DOCUMENT, signKey);

        //generate signed sub packets, such as signer, sign time, sign key id etc
        PGPSignatureSubpacketGenerator subpacketGenerator = new
PGPSignatureSubpacketGenerator();
        subpacketGenerator.setSignatureCreationTime(false, new Date());
        subpacketGenerator.setSignerUserID(false, signer);
        subpacketGenerator.setIssuerKeyID(false, signKey.getKeyID());

        signatureGenerator.setHashedSubpackets(subpacketGenerator.generate());
        signatureGenerator.generateOnePassVersion(false).encode(compressedData);

        //generate literal data, specify metadata such as data type, date, and file name
        PGPLiteralDataGenerator literalDataGenerator = new PGPLiteralDataGenerator();
        OutputStream lOut = literalDataGenerator.open(compressedData, PGPLiteralData.BINARY,
"Sample-Data", new Date(), new byte[DEFAULT_BUFFER_SIZE]); //bOut

        // convert input stream into byte array, then write into output stream wrapped by encryption,

```

compress algorithm

```
byte[] data = IOUtils.toByteArray(inputStream);
IOOut.write(data);
signatureGenerator.update(data);

IOOut.close();
literalDataGenerator.close();

signatureGenerator.generate().encode(compressedData);

compressedDataGenerator.close();
compressedData.close();

encryptedOutputStream.close();
encryptedDataGenerator.close();

out.close();

logger.info("encryption key id: {}, sign key id: {}", encryptKey.getKeyID(), signKey.getKeyID());
} catch (PGPException e) {
    logger.error("PGP Exception", e);
    if (e.getUnderlyingException() != null) {
        logger.error("PGP Exception ", e.getUnderlyingException());
    }
} catch (Throwable e) {
    logger.error("unknown PGP Error", e);

    if (inputStream != null) {
        inputStream.close();
    }
}
}
```

@Test

```
public void testEncryptionAndSign() throws Exception {
    String plainText = "Trouble will find you no matter where you go.";
    String r = pgpHelper.encryptAndSign(plainText, getPublicKey(), getPrivateKey(), "Customer A");
    Assertions.assertTrue(r.length() > 0);
}
```

代码说明：

- 1) 将你需要加密的负载明文转换为字符串
- 2) 使用 IOUtils.toInputStream 将负载转换为输入流
- 3) 使用 HSBC 公钥作为加密器对明文负载进行加密
- 4) 使用 base64 对加密结果进行编码

下一步，生成 JWS

1.3.3 步骤 3-生成和签名客户令牌

JWT 组件

下表定义了 JWT Payload 中要用到的字段，所有字段都必填。字段的定义请参考“[1.2.6 JWT Payload](#)”部分

编码	名字	位置	示例
typ	Token 类型	JWT 头	JWT – 必须设置为“JWT”
kid	PGP Key Id	JWT 头	900864F8C11EB743
alg	签名算法	JWT 头	PS256
ver	版本	JWT 头	1.0 – 必须设置为“1.0”
jti	JWT ID	JWT 负载主体	74760410-f963-11e8-b2a3-1bb26e1e5b69
iat	发布时间	JWT 负载主体	1565150776
sub	主体	JWT 负载主体	PC000012345
aud	受众	JWT 负载主体	baas
payload_hash	负载主体 hash 值	JWT 负载主体	1dab559f1f4bf4f3ca...
payload_hash_alg	负载主体 hash 算法	JWT 负载主体	RSASHA256

JWT 准备步骤：

- 1) 使用上表提供的步骤构建 JWT 结构
- 2) 为用于步骤 2 中的加密请求负载生成 hash 码
- 3) 构建 JWT 头
- 4) 生成和签名 JWT(JWS)

代码示例：

```
//1. Build the JWT structure
Map<String, Object> claims = new HashMap<>();

long nowSeconds = System.currentTimeMillis() / 1000;
claims.put("iat", nowSeconds);
claims.put("jti", UUID.randomUUID().toString());
claims.put("sub", "BS0000111111"); //your profile id
claims.put("aud", "baas");

//2 Generate hash code for request payload and put into JWT Token
claims.put("payload_hash", "038c3d8437c49840591baf7eb2a2f292c4f959acfe048444da48fc376df36b87");
//your payload hash code
claims.put("payload_hash_alg", "RSASHA256");
```

```
//3 Set JWT header
```

```
Map<String, Object> header = new HashMap<String, Object>();
```

```
header.put("kid", "71F28C2FB8886368"); //key id
```

```
header.put("typ", "JWT");
```

```
header.put("ver", "1.0");
```

```
header.put("key_typ", "PGP");
```

```
header.put("alg", "PS256");
```

```
//4 Generate and Sign the JWT (JWS)
```

```
PrivateKey rsaKey = pgpHelper.convertToRsaKey(getPrivateKey());
```

```
String token = Jwts.builder().setHeader(header).setClaims(claims).signWith(rsaKey,  
SignatureAlgorithm.PS256).compact();
```

代码说明：

- 1) 构建 JWT 结构
- 2) 为加密的请求负载主体生成 hash 码
- 3) 构建 JWT 头
- 4) 生成和签名 JWT(JWS)

下一步，添加 JWS 到 HTTP 头

1.3.4 步骤 4-添加签名的 JWT(JWS)到 HTTP 头

添加 JWS 到 HTTP 头部 Authorization 字段

```
Authorization=JWS eyJ2ZXIiOiIxLjAiLCJraWQiOiJBNjI4RjhBRDY3NERCQz..
```

下一步，根据每个服务 API URL 的规范和要求添加其它字段

1.3.5 步骤 5-添加保留的头部并提交

根据每个具体的 API URL 规范确定是否要添加其它必要的 header 字段

关联 ID 是一个唯一 ID，它由你的系统生成用来唯一标识每一个请求

1.3.6 处理响应

响应由 HTTP 响应码和加密的响应消息组成

响应处理步骤：

- 1) 一个被成功处理的请求的响应消息是一个加密的且用 base64 进行编码了的负载
- 2) 使用 base64 解码响应消息负载
- 3) 使用你的私钥解密解码后的消息负载
- 4) 解密后的负载主体每个 API URL 可能有所不同，具体请参考你调用的 API URL 文档

一个典型的加密和编码的响应消息：

```
{
  "profileId": "PC12345678",
  "referenceId": "b7e0fdacd8bd",
  "statusCode": "ACCP",
  "statusDesc": "Accepted Technical Validation (L2)",
  "reportBase64": "PFJlc3BvbnNlPg0KICA8UGF5bWVudElkPml3ZTBmZGFjLWQ4YmQtNDk1MC1i
ZDNGljSWQ+UEMxMjM+"
}
```

代码示例:

```
/**
 * Decrypt and verify encrypted message with your private key and HSBC public key
 *
 * @param ciphertext encrypted message encoded with Base64
 * @param privateKeys your own private key used for decryption
 * @param publicKeys public key provided by HSBC
 * @return Decrypted message
 * @throws Exception
 */
public String decryptAndVerify(String ciphertext, List<PGPPrivateKey> privateKeys, List<PGPPublicKey>
publicKeys) throws Exception {
    ByteArrayInputStream inputStream = new
ByteArrayInputStream(Base64.getDecoder().decode(ciphertext));

    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

    decryptAndVerify(inputStream, outputStream, privateKeys, publicKeys);

    return outputStream.toString("UTF-8");
}

/**
 * Decrypt and verify encrypted message with your private key and HSBC public key
 * @param in encrypted input stream
 * @param out decrypted output stream
 * @param privateKeys your own private key used for decryption
 * @param publicKeys public key provided by HSBC
 * @throws Exception
 */
public void decryptAndVerify(InputStream in, OutputStream out, List<PGPPrivateKey> privateKeys,
List<PGPPublicKey> publicKeys) throws Exception {
```

```

if (in == null) {
    throw new IllegalArgumentException("The encrypted data is null");
}

PGPEncryptedDataList encryptedDataList;
PublicKeyDataDecryptorFactory dataDecryptorFactory = null;
PGPPublicKeyEncryptedData encryptedData = null;
try {
    // create PGPObjFactory obj, parse and handle input stream, get encrypted data list obj
    PGPObjFactory pgpObjFactory = new PGPObjFactory(PGUtil.getDecoderStream(in),
keyFingerPrintCalculator);

    // #1 get data obj from PGPObjFactory check if it's PGPEncryptedDataList
    Object obj = pgpObjFactory.nextObject();
    if (obj instanceof PGPEncryptedDataList) {
        encryptedDataList = (PGPEncryptedDataList) obj;
    } else {
        // the first object might be a PGP marker packet.
        encryptedDataList = (PGPEncryptedDataList) pgpObjFactory.nextObject();
    }

    // data list should only include one encrypted data obj
    if (encryptedDataList.size() > 1) {
        logger.warn("There are {} encrypted data, only decrypt the last one",
encryptedDataList.size());
    }

    // #2 get the encrypted data, only process the last one
    Iterator<PGPPublicKeyEncryptedData> it = encryptedDataList.getEncryptedDataObjects();
    while (it.hasNext()) {
        encryptedData = it.next();
    }

    // #3 Find the matching private key to create PublicKeyDataDecryptorFactory obj
    for (PGPPublicKey privateKey : privateKeys) {
        if (encryptedData.getKeyID() == privateKey.getKeyID()) {
            dataDecryptorFactory = new
JcePublicKeyDataDecryptorFactoryBuilder().setProvider(bcp).setContentProvider(bcp).build(privateKey);

            break;
        }
    }
}

```

```

    } catch (Throwable e) {
        logger.error("Parse and handle input stream failed", e);
    }

    if (dataDecryptorFactory == null) {
        throw new PGPEException("Provided key id does not match the key client used on this payload");
    }

    PGPCompressedData compressedData;
    PGPOnePassSignatureList onePassSignatureList = null;
    PGPSignatureList signatureList = null;
    ByteArrayOutputStream plainOutput = new ByteArrayOutputStream();
    InputStream decryptedDataStream = null;
    byte[] output = null;
    try {
        ///#4 Get the plain text by using your private key to decrypt the encrypted text
        decryptedDataStream = encryptedData.getDataStream(dataDecryptorFactory);

        ///// parse and handle decrypted data stream, get compressed data obj
        PGPObjectFactory pgpObjectFactory2 = new PGPObjectFactory(decryptedDataStream,
keyFingerPrintCalculator);
        Object dataObj = pgpObjectFactory2.nextObject();

        while (dataObj != null) {
            if (dataObj instanceof PGPCompressedData) {
                compressedData = (PGPCompressedData) dataObj;

                ///// parse out compressed data
                pgpObjectFactory2 = new PGPObjectFactory(compressedData.getDataStream(),
keyFingerPrintCalculator);
                dataObj = pgpObjectFactory2.nextObject();
            }

            if (dataObj instanceof PGPLiteralData) {
                ///// have to read it and keep it somewhere.
                Streams.pipeAll(((PGPLiteralData) dataObj).getInputStream(), plainOutput);
            } else if (dataObj instanceof PGPOnePassSignatureList) {
                onePassSignatureList = (PGPOnePassSignatureList) dataObj;
            } else if (dataObj instanceof PGPSignatureList) {
                signatureList = (PGPSignatureList) dataObj;
            } else {
                throw new PGPEException("Unknown message type.");
            }
        }
    }

```

```

    }

    // get and handle the next data obj from factory
    dataObj = pgpObjectFactory2.nextObject();
}

    output = plainOutput.toByteArray();
} catch (Throwable e) {
    logger.error("Parse and handle decrypted data stream failed", e);
} finally {
    plainOutput.close();
    if (decryptedDataStream != null) {
        decryptedDataStream.close();
    }
}

if (onePassSignatureList == null || signatureList == null) {
    throw new SignatureException("Poor PGP. Signatures not found.");
}

boolean signatureVerified = false;
try {
    //#5 Verify the signature using HSBC's Public Key, should only include one signature in encrypted
data
    for (int i = 0; i < onePassSignatureList.size(); i++) {
        PGPOnePassSignature ops = onePassSignatureList.get(i);
        if (publicKeys != null) {
            for (PGPPublicKey publicKey : publicKeys) {
                ops.init(new JcaPGPContentVerifierBuilderProvider().setProvider(bcp), publicKey);
                ops.update(output);

                PGPSignature signature = signatureList.get(i);
                if (ops.verify(signature)) {
                    signatureVerified = true;
                }
            }
        }
    }
} catch (Throwable e) {
    logger.error("verification failed", e);
}

```

```

if (!signatureVerified) {
    throw new SignatureException("Signature verification failed");
}

if (encryptedData.isIntegrityProtected() && !encryptedData.verify()) {
    throw new PGPDDataValidationException("Data is integrity protected but integrity is lost.");
}

try {
    // write decrypted byte array into output stream
    out.write(output);
    out.flush();
} catch (Throwable e) {
    logger.error("Write output stream failed", e);
} finally {
    if (out != null) {
        out.close();
    }
    in.close();
}
}

```

@Test

```

public void decryptAndVerifyTest() throws Exception {
    String expected = "Trouble will find you no matter where you go.";
    String ciphertext = "LS0tLS1CRUdJTiBQR1AgTUVTU0FHRS0tLS0tDQpWZXJzaW9uOiBCQ1BHI...";
    String actual = pgpHelper.decryptAndVerify(ciphertext, getBankPrivateKeys(), getClientPublicKeys());
    Assertions.assertEquals(expected, actual);
}

```

@Test

```

public void decryptAndVerifyTest2() throws Exception {
    String expected = "Trouble will find you no matter where you go.";
    String ciphertext = "LS0tLS1CRUdJTiBQR1AgTUVTU0FHRS0tLS0tDQpWZXJzaW9uOiBCQ1BHI...";
    byte[] decodedData = Base64.getDecoder().decode(ciphertext);
    InputStream inputStream = new ByteArrayInputStream(decodedData);
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    pgpHelper.decryptAndVerify(inputStream, outputStream, getBankPrivateKeys(),
    getClientPublicKeys());
    String actual = outputStream.toString();
    Assertions.assertEquals(expected, actual);
}

```

代码说明：

- 1) 解码 base64 编码的响应
- 2) 使用自己的私钥解密响应消息，注意该私钥是你与 HSBC 交换的密钥对中的私钥
- 3) 使用 HSBC 的公钥验签消息

1.4 POST /bulk-payments/instant-receipt

批量支付发起操作服务 URL

1.4.1 描述

- 异步处理的批量支付发起操作，即时返回批量操作处理的确认信息。
- 每个 API 负载最多支持 500 条支付指令。
- 请求的有效载荷由 ISO 20022 XML 版本 3(pain.001.001.03)格式提供。
- 该 API 提供的响应是系统验证处理支付指令结果状态信息。
- 客户应调用支付状态查询 API 来查询最终的支付处理状态。

1.4.2 API 请求头

根据下表准备请求的 HTTP 头部参数

键名	类型	是否必填	示例	描述
Content-Type	String	是	application/json	指明 HTTP 资源媒体类型，值必须是 application/json
x-hsbc-client-id	String	是		被授权的 API 凭据 CLIENT_ID
x-hsbc-client-secret	String	是		被授权的 API 凭据 CLIENT_SECRET
x-hsbc-profile-id	String	是	PIC12345678	Profile ID，由 HSBC 提供
x-payload-type	String	是	pain.001.001.03	支付操作负载的 XML 类型

1.4.3 API 请求正文

ISO pain.001.001.03 规范: [HSBC pain.001.001.03 HighValuePayment](#)

怎么准备构建请求负载

- 准备构建的负载的格式。批量支付操作的指令数最大为 500
- 使用你的私钥签名负载消息，再用 HSBC 提供的公钥加密
- 使用 base64 编码加密后的消息

- 使用键名 `paymentBase64` 构建加密后的消息为 JSON 格式

请求负载示例：

```
{  
  "paymentBase64": "PD94bWwgdMvYc2lrbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCjxEb2N1bWVz="
```

1.4.4 API 响应正文

成功处理的请求的响应主体包含下列字段：

字段名	描述
referenceId	Bulk Payment API 请求的唯一标识
profileId	HSBC API Profile ID
statusCode	请求处理结果的状态码，例如：ACCP
statusDesc	状态码描述信息
responseBase64	请求处理结果的 XML 消息体，经过加密、签名及 base64 编码处理

响应负载示例：

200 - ACTC (Technical Acceptance)

```
{  
  "referenceId": "b7e0fdac-d8bd-4950-bd3a-38fac5cc25df",  
  "profileId": "PC12345678",  
  "statusCode": "ACTC",  
  "statusDesc": "Accepted Technical Validation (L1) - Authentication and syntactical and semantical validation are successful.",  
  "responseBase64": "+DQogIDxTdGF0dXNDb2RlPlNVQk1JVFRFRDwvU3RhdHVzQ28U3RhdHVzRGVzYz5QYXltZW5"
```

200 - ACCP (Accepted)

```
{  
  "referenceId": "b7e0fdac-d8bd-4950-bd3a-38fac5cc25df",  
  "profileId": "PC12345678",  
  "statusCode": "ACCP",  
  "statusDesc": "Accepted Instruction Validation (L2) - Description: Preceding check of instruction validation was successful. Customer profile check was also successful.",  
  "responseBase64": "+DQogIDxTdGF0dXNDb2RlPlNVQk1JVFRFRDwvU3RhdHVzQ29kZT4gIA0KICA8U3RhdHVzRZm9"
```

200 - PART (Partially Accepted)

```
{
  "referenceId":"b7e0fdac-d8bd-4950-bd3a-38fac5cc25df",
  "profileId":"PC12345678",
  "statusCode":"PART",
  "statusDesc":"Partially Processed - A number of transactions have been accepted,
whereas another number of transactions have not yet achieved 'accepted' status.",
  "responseBase64":"+DQogIDxTdGF0dXNDb2RlP1NVQk1JVFRFRDwvU3RhZHVzQ29kZT4gzRGVzYz5Q
YXlGtzaHZhZm9"
}
```

如何处理响应负载:

- 从响应的 json 消息中提取出 **responseBase64** 值
- 使用 base64 解码出负载主体
- 使用你的私钥解密出负载并验签

如果请求的支付操作指令验证处理成功, 解码解密出来的响应消息包含相关确认信息。

1.5 POST /payments/instant-receipt

单一支付启动操作服务 URL

1.5.1 描述

异步处理的单一支付启动操作, 即时返回处理确认信息。

请求有效载荷由 ISO 20022 XML 版本 3(pain.001.001.03)格式提供。

该 API 提供的响应是系统验证处理支付指令结果状态信息。

客户应调用支付状态查询 API 来查询最终的支付处理状态。

对于印度卢比(INR)vostro 帐户, 需要在 HTTP 请求的头部添加特殊的字段来标识该种跨境汇款操作。在这种业务场景下, 请在 HTTP 请求头部添加 CBRR 的付款类型。其他币种、帐户或付款类型不需要这样操作。

1.5.2 API 请求头

键名	类型	是否必填	示例值	描述
Content-Type	String	Yes	application/json	指明 HTTP 资源媒体类型, 值必须是 application/json
x-hsbc-client-id	String	Yes		被授权的 API 凭据 CLIENT_ID
x-hsbc-client-	String	Yes		被授权的 API 凭据 CLIENT_SECRET

secret				
x-hsbc-profile-id	String	Yes	PIC12345678	Profile ID, 由 HSBC 提供
x-payload-type	String	Yes	pain.001.001.03	支付操作负载的 XML 类型
x-hsbc-payment-type	String	No	CBRR	对于印度卢比(INR)Vostro 帐户, 需要提供特殊头部(值为 CBRR)字段来标识该种跨境支付操作。对于其它币种、账户和支付类型不需要提供该 header 信息。 对 INR Vostro 之外的币种、账户和支付类型请忽略该 header 信息

1.5.3 API 请求主体

ISO pain.001.001.03 规范: [HSBC pain.001.001.03 HighValuePayment](#)

怎么准备构建请求负载

- 准备构建的负载的格式。批量支付操作的指令数最大为 500
- 使用你的私钥签名负载消息, 再用 HSBC 提供的公钥加密
- 使用 base64 编码加密后的消息
- 使用键名 `paymentBase64` 构建加密后的消息为 JSON 格式

请求负载示例:

```
{
  "paymentBase64": "PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCjxEb2N1bWVz="
}
```

1.5.4 API 响应主体

成功处理的请求的响应主体包含下列字段:

字段名	描述
referenceId	Payment API 请求的唯一标识
profileId	HSBC API Profile ID
statusCode	请求处理结果的状态码, 例如: ACCP
statusDesc	状态码描述信息
responseBase64	请求处理结果的 XML 消息体, 经过加密、签名及 base64 编码处理

响应负载示例:

200 - ACTC (Technical Acceptance)

```
{
  "referenceId": "b7e0fdac-d8bd-4950-bd3a-38fac5cc25df",
  "profileId": "PC12345678",
  "statusCode": "ACTC",
  "statusDesc": "Accepted Technical Validation (L1) - Authentication and syntactical and semantical validation are successful.",
  "responseBase64": "+DQogIDxTdGF0dXNDb2RlPlNVQk1JVFRFRDwvU3RhdHVzQ28U3RhdHVzRGVzYz5QYXltZW50"
}
```

200 - ACCP (Accepted)

```
{
  "referenceId": "b7e0fdac-d8bd-4950-bd3a-38fac5cc25df",
  "profileId": "PC12345678",
  "statusCode": "ACCP",
  "statusDesc": "Accepted Instruction Validation (L2) - Description: Preceding check of instruction validation was successful. Customer profile check was also successful.",
  "responseBase64": "+DQogIDxTdGF0dXNDb2RlPlNVQk1JVFRFRDwvU3RhdHVzQ29kZT4gIA0KICA8U3RhdHVzRZhm9"
}
```

200 - PART (Partially Accepted)

```
{
  "referenceId": "b7e0fdac-d8bd-4950-bd3a-38fac5cc25df",
  "profileId": "PC12345678",
  "statusCode": "PART",
  "statusDesc": "Partially Processed - A number of transactions have been accepted, whereas another number of transactions have not yet achieved 'accepted' status.",
  "responseBase64": "+DQogIDxTdGF0dXNDb2RlPlNVQk1JVFRFRDwvU3RhdHVzQ29kZT4gzRGVzYz5QYXltZm9"
}
```

如何处理响应负载:

- 从响应的 json 消息中提取出 **responseBase64** 值
- 使用 base64 解码出负载主体
- 使用你的私钥解密出负载并验签

如果请求的支付操作指令验证处理成功, 解码解密出来的响应消息是该支付操作相关的确认信息。

1.6 POST /payments/instant-settlement

1.6.1 描述

- 发起异步处理的、即时结算的单笔付款操作。
- 目前支持英国快速支付、SEPA 即时支付、新加坡快速支付和美国实时支付。
- 英国快速支付要求在 HTTP 请求头中提供国家/地区代码“GB”和支付类型“FPS”，详情见下表。
- SEPA 即时支付要求在 HTTP 请求头中提供支付类型“SCT”。
- 美国实时支付要求在 HTTP 请求头中提供美国的国家代码“US”和支付类型“RTP”
- 请求有效载荷由 ISO 20022 XML 版本 3(pain.001.001.03)格式提供。
- 对于英国 FPS、SEPA Instant 和 Singapore Fast 来说，API 提供的成功回应是支付操作已经与收款银行结算了（ACSC — 已接受且结算完成）。
- 对于 US RTP - API 提供的成功回应是 HSBC 已接受付款指示（ACCP — 已接受指示验证）。为了收到 US RTP 的 ACSC — 已接受结算完成确认，客户应订阅付款状态通知（推送通知），并在付款操起的 HTTP 请求头中提供订阅 ID，详情如下表。

1.6.2 API 请求头

键名	类型	是否必填	示例值	描述
Content-Type	String	Yes	application/json	指明 HTTP 资源媒体类型，值必须是 application/json
x-hsbc-client-id	String	Yes		被授权的 API 凭据 CLIENT_ID
x-hsbc-client-secret	String	Yes		被授权的 API 凭据 CLIENT_SECRET
x-hsbc-profile-id	String	Yes	PIC12345678	Profile ID, 由 HSBC 提供
x-payload-type	String	Yes	pain.001.001.03	支付操作负载的 XML 类型
x-hsbc-payment-type	String	No	FPS	英国快速支付: FPS SEPA 即时支付: SCT 美国实时支付: RTP
x-hsbc-country-code	String	No	GB	英国快速支付: GB 美国实时支付: US
x-hsbc-subscription-id	String	No		推送的订阅 ID, 仅适用于美国实时支付

1.6.3 API 请求主体

ISO pain.001.001.03 规范: [HSBC pain.001.001.03 HighValuePayment](#)

怎么准备构建请求负载

- 准备构建的负载的格式
- 使用你的私钥签名负载消息，再用 HSBC 提供的公钥加密
- 使用 base64 编码加密后的消息
- 使用键名 `paymentBase64` 构建加密后的消息为 JSON 格式

请求负载示例：

```
{
  "paymentBase64": "PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluz0iVVRGLTgiPz4NCjxEb2N1bWVz="
}
```

1.6.4 API 响应主体

成功处理的请求的响应主体包含下列字段：

字段名	描述
referenceId	Payment API 请求的唯一标识
profileId	HSBC API Profile ID
statusCode	请求处理结果的状态码，例如：ACCP
statusDesc	状态码描述信息
responseBase64	请求处理结果的 XML 消息体，经过加密、签名及 base64 编码处理

响应负载示例：

200 - ACTC (Technical Acceptance)

```
{
  "referenceId": "b7e0fdac-d8bd-4950-bd3a-38fac5cc25df",
  "profileId": "PC12345678",
  "statusCode": "ACTC",
  "statusDesc": "Accepted Technical Validation (L1) - Authentication and syntactical and semantical validation are successful.",
  "responseBase64": "+DQogIDxTdGF0dXNDb2RlPlNVQk1JVFRFRDwvU3RhdHVzQ28U3RhdHVzRGVzYz5QYXltZW5"
}
```

200 - ACCP (Accepted)

```
{
  "referenceId": "b7e0fdac-d8bd-4950-bd3a-38fac5cc25df",
```

```
"profileId":"PC12345678",
"statusCode":"ACCP",
"statusDesc":"Accepted Instruction Validation (L2) - Description: Preceding check of instruction validation was successful. Customer profile check was also successful.",
"responseBase64":"+DQogIDxTdGF0dXNDb2RlPlNVQk1JVFRFRDwvU3RhdHVzQ29kZT4gIA0KICA8U3RhdHVzRZhm9"
}
```

200 - PART (Partially Accepted)

```
{
  "referenceId":"b7e0fdac-d8bd-4950-bd3a-38fac5cc25df",
  "profileId":"PC12345678",
  "statusCode":"PART",
  "statusDesc":"Partially Processed - A number of transactions have been accepted, whereas another number of transactions have not yet achieved 'accepted' status.",
  "responseBase64":"+DQogIDxTdGF0dXNDb2RlPlNVQk1JVFRFRDwvU3RhdHVzQ29kZT4gzRGVzYz5QYXlGtzaHZhm9"
}
```

如何处理响应负载：

- 从响应的 json 消息中提取出 **responseBase64** 值
- 使用 base64 解码出负载主体
- 使用你的私钥解密出负载并验签

如果结算已完成并被收款银行接受，解码解密出来的响应消息是该支付操作相关的确认信息。

1.7 POST /payments/instant-settlement (IMPS)

印度实时支付发起操作

1.7.1 描述

通过印度即时支付系统的、异步处理的、可即时结算的单笔支付交易

本节文档专用于通过印度即时支付系统(IMPS)启动的实时支付

印度即时支付系统要求在 HTTP 请求头中提供国家代码“IN”，付款类型“IMPSRTP”

请求负载遵循下面指定的专有 XMLPF 格式

API 提供的成功响应表示“结算已接受并完成”，即付款已被受益行接受。

1.7.2 API 请求头

键名	类型	是否必填	示例值	描述
Content-Type	String	Yes	application/json	指明 HTTP 资源媒体类型，值必须是 application/json
x-hsbc-client-id	String	Yes		被授权的 API 凭据 CLIENT_ID
x-hsbc-client-secret	String	Yes		被授权的 API 凭据 CLIENT_SECRET
x-hsbc-profile-id	String	Yes	PIC12345678	Profile ID，由 HSBC 提供
x-payload-type	String	Yes	pain.001.001.03	支付操作负载的 XML 类型
x-hsbc-unique-ref-id	String	Yes	UETR11E	支付发起的唯一 ID
x-hsbc-payment-type	String	Yes	IMPSRTP	印度即时支付系统：IMPSRTP
x-hsbc-country-code	String	Yes	IN	印度即时支付系统：IN

1.7.3 API 请求主体

本 API 文档介绍的 XML (XMLPF)格式仅适用于印度即时支付系统 API

该格式支持“即时结算的实时支付发起操作”和“支付状态查询”

只有国家/地区是印度时，才支持使用此格式发送的实时支付的‘即时结算实时支付启动’和‘支付状态查询’。

XMLPF IMPS 即时支付请求参数：

所有参数都区分大小写，并且需要按照规范顺序组织。可选参数可以从 XML 结构中忽略

字段	描述	类型	最大长度	是否必填	示例
SERVICEPROVIDERID	移动服务提供商名称	字母数字	12	必填	PC000058828
REQUESTID	汇款人参考号 - 汇款网站上的任何唯一参考号	字母数字	50	必填	06MAR20200001
REQUESTTYPE	R: Request, 期望值是 R	字母数字	1	必填	R
REMITTORACOUNTNO	客户借记账户	字母数字	50	必填	002164549004
REMITTORNAME	汇款人名字	字母数字	50	可选	

ME					
REMITTORMOBILENUMBER	注册的汇款人手机号，发送数值或不发送标签	Numeric	12	可选	
REMITTORMMID	汇款人 MMID	字母数字	7	可选	
BENEFICIARYBANKIFSC	受益人 IFSC 码	字母数字	11	必填	DNSB0000001
BENEFICIARYACCOUNTNO	受益人账号	字母数字	50	必填	123456789012
AMOUNT	金额，卢比	字母数字	12	必填	1
REMARK	备注	字母数字	50	可选	
CHANNELID	期望值: CDP	字母数字	4	必填	CDP
AUTHENTICATIONTOKEN	捕获的虚拟值	字母数字	40	必填	1234
ADDITIONALINFO1	额外描述	字母数字	50	可选	
ADDITIONALINFO2	如果为客户设置了智能路由则必填，期望值：受益人名字	字母数字	50	可选	P3ACustomer
ADDITIONALINFO3	额外描述	字母数字	50	可选	
ADDITIONALINFO4	额外描述	字母数字	50	可选	
ADDITIONALINFO5	额外描述	字母数字	50	可选	
ADDITIONALINFO6	未来使用	字母数字	50	可选	
ADDITIONALINFO7	如果为客户设置了 Penny 交易则必填 可能的值：“P”并且当金额为 1 时	字母数字	6	可选	P
BENEFICIARYEMAILID	未来使用	字母数字	100	可选	
BENEFICIARYMOBILENUMBER	未来使用，或发送数值或不发送标签	Numeric	12	可选	

MERCHANTID	未来使用	字母数字	20	可选	
SUBMERCHANTCODE	未来使用	字母数字	20	可选	
ORDERINGPARTYID	未来使用	字母数字	50	可选	
SOURCECHANNEL	未来使用	字母数字	20	可选	

请求主体示例：

```
<?xmlversion='1.0'encoding='windows-1252'?>
<p2aRequest>
  <SERVICEPROVIDERID>PC000016070</SERVICEPROVIDERID>
  <REQUESTID>IMPS0001000001D073000005050025</REQUESTID>
  <REQUESTTYPE>R</REQUESTTYPE>
  <REMITTORACCOUNTNO>073048084001</REMITTORACCOUNTNO>
  <BENEFICIARYBANKIFSC>HDFC0000007</BENEFICIARYBANKIFSC>
  <BENEFICIARYACCOUNTNO>073048084001</BENEFICIARYACCOUNTNO>
  <AMOUNT>1</AMOUNT>
  <CHANNELID>CDP</CHANNELID>
  <AUTHENTICATIONTOKEN>1234</AUTHENTICATIONTOKEN>
</p2aRequest>
```

怎么准备构建请求负载

- 准备构建的负载的格式
- 使用你的私钥签名负载消息，再用 HSBC 提供的公钥加密
- 使用 base64 编码加密后的消息
- 使用键名 `paymentBase64` 构建加密后的消息为 JSON 格式

请求负载示例：

```
{
  "paymentBase64": "PD94bWwgdmVyc2lvcj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCjxEb2N1bWVz="
}
```

1.7.4 API 响应主体

成功处理的请求的响应主体包含下列字段：

字段名	描述
referenceId	Payment API 请求的唯一标识
profileId	HSBC API Profile ID
statusCode	请求处理结果的状态码，例如：ACSC

statusDesc	状态码描述信息
responseBase64	请求处理结果的 XML 消息体，经过加密、签名及 base64 编码处理

响应负载示例：

200 - ACSC (Accepted Settlement Completed)

```
{
  "referenceId": "b7e0fdac-d8bd-4950-bd3a-38fac5cc25df",
  "profileId": "PC12345678",
  "statusCode": "ACSC",
  "statusDesc": "AcceptedSettlementCompleted - Settlement on the account has been completed.",
  "responseBase64": "PFJ1c3BvbnNlPg0KICA8UGF5bWVudElkPmI3ZTBmZGFjLWQ4YmQtNDk1hMjV  
kZjwvUGF5bW+"
}
```

如何处理响应负载：

- 从响应的 json 消息中提取出 **responseBase64** 值
- 使用 base64 解码出负载主体
- 使用你的私钥解密出负载并验签

如果结算已完成并被收款银行接受，解码解密出来的响应消息是该支付操作相关的确认信息。

解码解密出的响应负载

字段名	描述	类型	最大长度	是否必填	示例
REQUESTID	汇款人参考号 - 汇款网址上的任何唯一标识	数字字母	50	必填	06MAR20200001
REQUESTTYPE	R-Request	数字字母	1	必填	R
REMITTORACCOUNTNO	账号	数字字母	50	必填	002164549004
MERCHANTID	未来使用	数字字母	20	可选	HSBC000000000288
RESPONSECODE	IMPS 内部代码	数字字母	2	必填	00
RESPONSEDESCRIPTION	响应码描述	数字字母	50	必填	Successful Transaction
RRN	参考号	数字	12	可选	

BENEFICIARYNAME	受益人名字	数字字母	50	可选	
TRANSACTIONDATE	交易日期时间, 格式: DDMMYYYYhhmms	数字字母	14	必填	20200403182414
AUTHENTICATION	捕获的虚拟值	数字字母	40	必填	1234
ADDITIONALINFO1	未来使用	数字字母	50	可选	
ADDITIONALINFO2	未来使用	数字字母	50	可选	
ADDITIONALINFO7	未来使用	数字字母	6	可选	

解码解密出的负载示例:

```
<p2aResponse>
  <REQUESTID>zwCjcJuci7nIjk2a</REQUESTID>
  <REQUESTTYPE>R</REQUESTTYPE>
  <REMITTORACCOUNTNO>002197994001</REMITTORACCOUNTNO>
  <MERCHANTID>123</MERCHANTID>
  <RESPONSECODE>00</RESPONSECODE>
  <RESPONSEDESCRIPTION>Successful</RESPONSEDESCRIPTION>
  <RRN>012119135399</RRN>
  <BENEFICIARYNAME>P3ACustomer</BENEFICIARYNAME>
  <TRANSACTIONDATE>20200430194741</TRANSACTIONDATE>
  <AUTHENTICATION>@@@</AUTHENTICATION>
  <ADDITIONALINFO1>INDIA VIX</ADDITIONALINFO1>
  <ADDITIONALINFO2>Creditor Name Test Bene Account EQ 20</ADDITIONALINFO2>
  <ADDITIONALINFO7>A</ADDITIONALINFO7>
</p2aResponse>
```

1.8 POST /payments/status

支付状态查询

1.8.1 描述

通过汇丰银行或结算银行提供从支付发起到支付结束的状态查询跟踪功能。

使用付款发起操作返回的相关参考字段(一般是 reference id)查询相关支付状态。

请使用此 API 检索通过 /payments/instant-receipt 和 /bulk-payments/instant-receipt 发起的单笔和批量支付操作的状态信息。

1.8.2 API 请求头

根据下表准备请求的 HTTP 头部参数

键名	类型	是否必填	示例	描述
Content-Type	String	是	application/json	指明 HTTP 资源媒体类型, 值必须是 application/json
x-hsbc-client-id	String	是		被授权的 API 凭据 CLIENT_ID
x-hsbc-client-secret	String	是		被授权的 API 凭据 CLIENT_SECRET
x-hsbc-profile-id	String	是	PIC12345678	Profile ID, 由 HSBC 提供
x-hsbc-country-code	String	可选	IN	如果借方账户所在国家/地区是印度则必填

1.8.3 API 请求主体

根据以下消息结构构建请求消息主体, 注意字段名是大小写敏感的

```
<StatusEnquiry>
  <ProfileID>PC000001111</ProfileID>
  <Key>
    <ReferenceID>0137fb84-b66d-40fb-9a4f-1c7d62288ff9</ReferenceID>
    <MessageID></MessageID>
    <PmtInfID></PmtInfID>
    <InstructionID></InstructionID>
    <EndToEndID></EndToEndID>
  </Key>
</StatusEnquiry>
```

参数定义如下:

参数(大小写敏感)	是否必填	描述
ProfileID	Yes	HSBC 提供的访问支付 API 的 Profile ID
ReferenceID	按条件	支付发起操作 API 生成并返回的参考 ID
MessageID	按条件	对应原支付请求报文 XML 中的<MsgID>字段
PmtInfID	按条件	对应原支付请求报文 XML 中的<PmtInfID>字段
InstructionID	按条件	对应原支付请求报文 XML 中的<InstrID>字段
EndToEndID	按条件	对应原支付请求报文 XML 中的<EndToEndID>字段

怎么准备构建请求负载

- 准备构建的负载的格式
- 使用你的私钥签名负载消息，再用 HSBC 提供的公钥加密
- 使用 base64 编码加密后的消息
- 使用键名 `paymentEnquiryBase64` 封装加密后的消息为 JSON 格式

请求负载示例：

```
{
  "paymentEnquiryBase64": "PD94bWwgdGVyc2lrbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCjxEb2N1bWVz="
}
```

1.8.4 API 响应主体

成功处理的请求的响应主体包含下列字段：

字段名	描述
referenceId	Payment API 请求的唯一标识
profileId	HSBC API Profile ID
statusCode	请求处理结果的状态码，例如：ACCP
statusDesc	状态码描述信息
responseBase64	请求处理结果的 XML 消息体，经过加密、签名及 base64 编码处理

响应负载示例：

200 - ACCP (Accepted)

```
{
  "referenceId": "b7e0fdac-d8bd-4950-bd3a-38fac5cc25df",
  "profileId": "PC12345678",
  "statusCode": "ACCP",
  "statusDesc": "Accepted Instruction Validation (L2) - Description: Preceding check of instruction validation was successful. Customer profile check was also successful.",
  "responseBase64": "+DQogIDxTdGF0dXNDb2RlPlNVQk1JVFRFRDwvU3RhZHVzQ29kZT4gIA0KICA8U3RhZHVzRZhm9"
}
```

此消息是 L2 级确认表示该支付操作指令已通过验证并已被接受。下表定义了可能的状态代码和说明。

支付状态码和描述

状态码	描述
ACTC	Accepted Technical Validation (L1) – 认证、语法和语义校验成功
ACCP	Accepted Instruction Validation (L2) - 指令有效性的预先检查通过，客户信息检查通过
ACWC	Accepted with condition – 指令被接受但将会发生变化，例如起息日展期

PART	Partially Processed – 一些交易被接受，同时另外一部分交易还没有变成“accepted”状态
ACSP	Accepted Settlement In Process – 所有的预先检查如校验、客户信息检查都通过，支付操作指令被接受且准备执行
ACSC	Accepted Settlement Completed – 账号结算完成，支付被受益行接受并将记入受益人的账号
RJCT	Rejected – 支付操作被拒绝或包含在支付报文中的个人交易被拒绝
PDNG	Time-Out – 表示延迟、超时，你的支付操作被挂起，请使用支付查询接口查询你的账号的支付状态

如何处理响应负载：

- 从响应的 json 消息中提取出 `responseBase64` 值
- 使用 base64 解码出负载主体
- 使用你的私钥解密出负载并验签

实际返回的负载主体内容遵循 ISO 20022 pain.002.001.03 格式

文档链接: [HSBC pain 002 001 03 PaymentStatusReportCombined Core.pdf](#)

支付状态查询返回示例 - pain.002.001.03

```
<?xml version="1.0" encoding="UTF-8"?>
<Document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:iso:std:iso:20022:tech:xsd:pain.002.001.03">
  <CstmrPmtStsRpt>
    <GrpHdr>
      <MsgId>93a8f0703f0911eb974d39b880a13651</MsgId>
      <CreDtTm>2020-12-15T19:13:12</CreDtTm>
      <InitgPty>
        <Id>
          <OrgId>
            <BICOrBEI>MIDLGB22ABC</BICOrBEI>
          </OrgId>
        </Id>
      </InitgPty>
    </GrpHdr>
    <OrgnlGrpInfAndSts>
      <OrgnlMsgId>MSGREFPAPI003</OrgnlMsgId>
      <OrgnlMsgNmId>pain.001.001.03</OrgnlMsgNmId>
      <OrgnlNbOfTx>2</OrgnlNbOfTx>
      <OrgnlCtrlSum>0.03</OrgnlCtrlSum>
      <GrpSts>ACCP</GrpSts>
    </OrgnlGrpInfAndSts>
    <OrgnlPmtInfAndSts>
      <OrgnlPmtInfId>PYMTREFINAPI003</OrgnlPmtInfId>
      <OrgnlNbOfTx>1</OrgnlNbOfTx>
      <TxInfAndSts>
```

```

<StsId>D350000400850001</StsId>
<OrgnlInstrId>INSRTIDREFAPI002</OrgnlInstrId>
<OrgnlEndToEndId>E2EIDREFAPI002</OrgnlEndToEndId>
<TxSts>ACCP</TxSts>
<StsRsnInf>
  <AddtlInf>/UETR/8e195783-db5a-4f78-bb73-e08589c7b921</AddtlInf>
</StsRsnInf>
<OrgnlTxRef>
  <Amt>
    <InstdAmt Ccy="USD">0.01</InstdAmt>
  </Amt>
  <ReqdExctnDt>2020-12-15</ReqdExctnDt>
  <Dbtr>
    <Nm>DEBTOR NAME</Nm>
  </Dbtr>
  <DbtrAcct>
    <Id>
      <Othr>
        <Id>123456789</Id>
      </Othr>
    </Id>
  </DbtrAcct>
  <DbtrAgt>
    <FinInstnId>
      <BIC>MRMDUS33</BIC>
    </FinInstnId>
  </DbtrAgt>
  <CdtrAgt>
    <FinInstnId>
      <BIC>MRMDUS33</BIC>
    </FinInstnId>
  </CdtrAgt>
  <Cdtr>
    <Nm>John Doe</Nm>
  </Cdtr>
  <CdtrAcct>
    <Id>
      <Othr>
        <Id>987654321</Id>
      </Othr>
    </Id>
  </CdtrAcct>
</OrgnlTxRef>
</TxInfAndSts>
</OrgnlPmtInfAndSts>
<OrgnlPmtInfAndSts>
  <OrgnlPmtInfId>PYMTREFINAPI005</OrgnlPmtInfId>
  <OrgnlNbOfTx>1</OrgnlNbOfTx>

```

```

<TxInfAndSts>
  <StsId>D350000400860001</StsId>
  <OrgnlInstrId>INSRTIDREFAPI005</OrgnlInstrId>
  <OrgnlEndToEndId>E2EIDREFAPI005</OrgnlEndToEndId>
  <TxSts>ACCP</TxSts>
  <StsRsnInf>
    <AddtlInf>/UETR/c39057ec-df84-4a35-8953-8435d3f237e0</AddtlInf>
  </StsRsnInf>
  <OrgnlTxRef>
    <Amt>
      <InstdAmt Ccy="USD">0.02</InstdAmt>
    </Amt>
    <ReqdExctnDt>2020-12-15</ReqdExctnDt>
    <Dbtr>
      <Nm>DEBTOR NAME</Nm>
    </Dbtr>
    <DbtrAcct>
      <Id>
        <Othr>
          <Id>123456789</Id>
        </Othr>
      </Id>
    </DbtrAcct>
    <DbtrAgt>
      <FinInstnId>
        <BIC>MRMDUS33</BIC>
      </FinInstnId>
    </DbtrAgt>
    <CdtrAgt>
      <FinInstnId>
        <BIC>MRMDUS33</BIC>
      </FinInstnId>
    </CdtrAgt>
    <Cdtr>
      <Nm>John Doe</Nm>
    </Cdtr>
    <CdtrAcct>
      <Id>
        <Othr>
          <Id>987654321</Id>
        </Othr>
      </Id>
    </CdtrAcct>
  </OrgnlTxRef>
</TxInfAndSts>
</OrgnlPmtInfAndSts>
</CstmrPmtStsRpt>
</Document>

```

1.9 Swagger

[Swagger 文档](#)

1.10 Treasury API 沙盒

开始

要尝试使用“企业沙盒”，请先创建一个项目。这将生成在调用沙盒 API 时进行身份验证所需的客户端凭据和加密密钥。

要创建第一个项目，请按照主导航中屏幕右上角的“开发中心”链接进行操作。

开发人员中心是管理沙盒项目的中心位置，允许管理员在开发人员门户内向组织添加和删除成员。

参见：

[Detailed User Guide for Sandbox projects](#)

[Download the Sandbox Toolkit](#)

[Detailed Guide to Sandbox Data Management](#)

沙盒请求响应

我们在企业沙箱中支持 2 种类型的响应：

模拟沙盒向请求的端点返回基于文件的静态模拟响应。在您准备将加密模块用于智能沙盒时，可以将这些端点用于快速测试。

模拟沙盒只需要请求和响应负载的 Base64 编码。

它使用沙盒项目中提供的客户端凭据。

请使用 200 作为 x-hsbc-profile-id 来触发成功的模拟响应。

不需要 PGP 密钥。

模拟沙盒 URL: `https://sandbox.corporate-api.hsbc.com/mock/{version}/{treasury api endpoint}`

cURL 示例

```
curl --location --request POST 'https://sandbox.corporate-api.hsbc.com/mock/v2/balances' \
--header 'Content-Type: application/json' \
--header 'x-hsbc-client-id: mock-clientId' \
--header 'x-hsbc-client-secret: mock-secret' \
--header 'x-report-type: JSON' \
--header 'x-hsbc-profile-id: 200';
```

智能沙盒是一个完整的测试环境，它基于请求的内容从数据库返回相应的响应。

- 智能沙盒维护数据的状态，如新的付款输入等。
- 它要求对请求和响应负载消息进行签名、加密和 Base64 编码。
- 它使用沙盒中提供的客户端凭据、配置文件 ID 和加密密钥。
- 在下载加密密钥 zip 文件中，可以在 client-pgp-keys.info 文件找到客户端私钥的密码。

智能沙盒 URL: <https://sandbox.corporate-api.hsbc.com/{version}/{treasury api endpoint}>

Postman 集合文件可从以下链接下载，可帮助你开始使用智能沙盒：

[下载沙盒工具包](#)

智能沙盒账号信息

以下测试帐户已添加到智能沙盒默认数据集，请在准备测试负载时使用这些帐户。

目前，每个国家/地区有两个沙盒帐户。

账号	账号国家/地区	机构代码	账号类型	Bic
10000001	GB	HBEU	CA	MIDLGB22XXX
10000002	GB	HRFB	CA	HBUKGB4BXXX
339200000003	HK	HBAP	CA	HSBCHKHHHKH
339200000004	HK	HBAP	CA	HSBCHKHHHKH
12340000000000000005	CN	HBAP	CA	HSBCCNSHXXX
12340000000000000006	CN	HBAP	CA	HSBCCNSHXXX
146000000007	SG	HBAP	CA	HSBCSGSGXXX
146000000008	SG	HBAP	CA	HSBCSGSGXXX
146000000009	IN	HBAP	CA	HSBCINBBXXX
146000000010	IN	HBAP	CA	HSBCINBBXXX
1100000011	US	HBUS	CA	MRMDUS33XXX
1100000012	US	HBUS	CA	MRMDUS33XXX
0215800000000000130	MX	HBMI	CA	BIMEMXMMXXX
0215800000000000143	MX	HBMI	CA	BIMEMXMMXXX
220000000015	AE	HBME	CA	BBMEAEADXXX
220000000016	AE	HBME	CA	BBMEAEADXXX

2. Treasury – 账户信息 API

2.1 API 信息

- 版本: v3
- 协议: HTTPS
- 沙盒环境 URL: <https://sandbox.corporate-api.hsbc.com/>
- 生产环境 URL: **生产环境 – 请联系您的客户经理**
- ISO 20022 camt.052 响应规范: [HSBC camt.052.001.02 BankToCustomerAccountReport](#)
- ISO 20022 camt.053 响应规范: [HSBC camt.053.001.02 BankToCustomerStatement](#)

- JSON 响应规范: [HSBC JSON format Balance and Transaction response](#)

2.1.1 描述

账户信息 API 可以为账户持有者提供某一天或者某一天给定时间段内，该账户的余额以及已计入账户的交易记录明细。

客户发起账户信息 API 请求时，可以指定账户余额以及交易记录明细响应的格式。它可以是 ISO 20022 的现金管理 (CAMT) 标准格式，也可以是汇丰银行专有的 JSON 格式。具体如下：

此 API 可用的付款类型和终结点包括：

- camt.052.001.02 - 银行致客户账户报告 – 用于查询账户当天交易的余额及日内交易报告；
- camt.053.001.02 - 银行致客户对账单 – 用于查询账户历史日期的余额及交易记录；
- HSBC JSON – 用于查询账户当天，或者历史日期的余额及交易记录。

此文档旨在给客户提在其系统和流程中集成账户信息 API 服务所需的指南。

2.1.2 版本历史

版本	发布日期	状态	描述
V1	2018 年 12 月	弃用	账户余额 API
V2	2019 年 6 月	启用	漏洞修复，及账户交易记录明细功能发布。
V2	2021 年 9 月	启用	增强余额查询功能，使客户可以指定查询某一天的账户余额信息。
V3	2023 年 9 月	启用	API 使用规格说明书更新：增加了 OpenAM 的安全认证方式。

2.2 API 安全与认证

本节内容将帮助你：

- 了解保护和加密与 HSBC 的连接所需的步骤
- 查看 JWT 头部和负载中要提供的字段
- 了解如何生成所需 JSON Web 令牌 (JWT) 的代码示例

2.2.1 安全概况

要在汇丰银行和 API 消费者之间建立安全连接，需要采取及使用以下措施和组件：

类别	定义	组件
传输层安全协议(TLS)	HTTPS 连接	数字证书(SSL)
消息级别安全	数据加密	PGP 私钥和公钥

API 认证	对 API 请求进行认证	签名和 Base64 编码的 JWT (JWS) - 由客户创建
客户身份认证	客户的身份信息	客户 Profile ID – 由汇丰银行方提供

2.2.2 传输层安全

为了通信的安全，信息传输会采用加密方式进行，即客户和汇丰银行 API 之间的连接需要使用 HTTPS 协议和传输层安全协议 (TLS)。 TLS 是一种加密协议，目的是在计算机数据传输层面提供通信安全。

当客户请求与汇丰银行建立 HTTPS 连接时，需要使用数字证书。该证书包含开启安全套接字层 (SSL) 握手所需的加密技术，其中涉及生成共享密钥以在客户和汇丰银行之间建立特定的安全连接。

在尝试连接到我们的 API 之前，请确保您的组织拥有来自公认的证书颁发机构颁发的有效数字证书。

2.2.3 消息级别的加密

除了传输层安全性之外，汇丰银行还在会话层的消息级别采用了额外的安全性措施。消息级加密使用了基于公钥的非对称加密方案，为企业、组织和汇丰银行之间提供 API 消息负载级别的安全性增强。

公钥加密涉及一对称为公钥和私钥的密钥。这些密钥分配给需要以电子方式进行身份验证、或要对数据进行签名或加密的企业实体或组织。每个公钥都是公开的，而相应的私钥是保密的。使用公钥加密的数据只能使用相应的私钥解密。

公钥加密可实现加密、解密和不可否认性。汇丰银行的 API 支持非对称加密，即使用公钥加密数据，使用私钥解密数据。公钥在受信任的证书中可用，而私钥是保密的且不可以共享的。

出于消息加密的目的，您的企业、组织和汇丰银行都需要交换公钥，以便双方可以通过 API 加密和解密消息。

PGP 加密

PGP 是一种加密方案，为数据通信提供加密隐私和身份验证。 PGP 用于对文本、电子邮件、文件、目录和整个磁盘分区进行签名、加密和解密，并提高电子邮件通信的安全性。

1. PGP 加密的目的是什么？

出于 API 消息加密的目的，必须在 API 消费者 (客户端) 和 API 提供商 (HSBC) 之间交换公共 PGP 密钥。

2. 怎样获取 PGP 密钥对？

客户可以使用 GnuPG 或 GPG 等密钥生成工具生成自己的私钥/公钥对。

公共 PGP 密钥规范：

- 签名算法：2048 位
- RSA 哈希算法：SHA-256

您的证书准备就绪后，请联系您的 HSBC 客户端集成联系人以触发密钥交换程序。

3. 怎么使用 PGP 密钥对？

客户端每次向汇丰银行发送消息时都使用汇丰银行的公钥来加密消息。每次从汇丰银行收到消息时，客户都会验证汇丰银行的数字签名。汇丰银行使用客户的公钥对从客户发送和接收的消息执行相同的操作。

出于安全目的，汇丰银行的公钥每年都会更新，并且会触发证书更新流程。

4. PGP 密钥的保存和有效期

组件	存储	有效期
客户方私钥	应使用尽可能最安全的方法维护和处理私钥。最常见且安全的方法是：密钥密码 - 不要以纯文本或硬编码的形式将密码保存在应用程序中。我们建议使用密码加密工具/密钥存储对其进行加密 - 存储在受密码保护的密钥存储库中，例如 JKS 或 PKCS12 密钥存储库。密钥库密码也应该加密。	没有有效期。然而，如果客户怀疑密钥可能被泄露，或任何其他安全原因，则应生成新的私钥及其关联的公钥。
客户方公钥	由于公钥是公开分发的，因此中等安全的存储方法是可以接受的。客户端可以将物理文件存储在任何计算机的文件系统中，或者用于集中密钥管理 - 将所有密钥和证书存储在一个密钥存储库中。	对于自签名 PGP 公钥，适用与上述相同的条件。但是，CA 签名的证书的有效期限取决于颁发 CA 的购买计划。最常见的标准是 1 到 2 年。
汇丰银行方公钥	同上	有效期通常为 1 年，另加 1 至 2 个月。额外的时间是一个缓冲区，使客户端能够在 PGP 密钥更新过程中将“即将过期”的证书切换到新的证书。

2.2.4 API 身份验证 - JWT 验证

API 身份验证是使用签名的 JSON Web 令牌 (JWT) 执行的。每次调用汇丰 API 服务时，都必须在 HTTP 请求标头中提供令牌。这确保只有经过授权和预先批准的组织才能访问和使用我们的 API 服务。JWT 由客户生成，而不是由汇丰银行提供。有许多实用程序可以为所有常用的编程语言提供 JWT 生成和验证逻辑。以下示例使用“io.jsonwebtoken”java 库。

创建 JWT 你需要：

- 客户私钥
- 客户公钥
- 客户 Profile ID

客户 Profile ID 是 HSBC 在创建新用户过程中提供给您标识符。

私钥是加密密钥对的一部分需妥善保管避免泄露，公钥需与汇丰银行共享，是客户创建流程的一部分。

对于沙盒访问，用到的加密密钥和 Profile ID 将在创建沙盒项目过程中提供。

代码样例

```
//1. Build the JWT structure
Map<String, Object> claims = new HashMap<>();

long nowSeconds = System.currentTimeMillis() / 1000;
claims.put("iat", nowSeconds);
claims.put("jti", UUID.randomUUID().toString());
claims.put("sub", "BS0000111111"); //your profile id
claims.put("aud", "baas");

//2 Generate hash code for request payload and put into JWT Token
claims.put("payload_hash", "038c3d8437c49840591baf7eb2a2f292c4f959acfe048444da48fc376df36b87");
//your payload hash code
claims.put("payload_hash_alg", "RSASHA256");

//3 Set JWT header
Map<String, Object> header = new HashMap<String, Object>();
header.put("kid", "71F28C2FB8886368"); //key id
header.put("typ", "JWT");
header.put("ver", "1.0");
header.put("key_typ", "PGP");
header.put("alg", "PS256");

//4 Generate and Sign the JWT (JWS)
PrivateKey rsaKey = pgpHelper.convertToRsaKey(getPrivateKey());
String token = Jwts.builder().setHeader(header).setClaims(claims).signWith(rsaKey,
SignatureAlgorithm.PS256).compact();
String[] result = token.split("\\.");
Assertions.assertEquals(3, result.length);
```

2.2.5 JWT Header 中的字段

下表定义了 JWT Header 中要用到的字段

键名	名称	是否必填	描述	示例
typ	令牌类型	是	必须设置为 JWT	JWT
kid	PGP Key ID	是	HSBC 用于验证 JWT 签名的公钥 ID。 汇丰银行从客户提供的 PGP 密钥文件中通过此密钥 ID 读取公钥,以验证 JWT 签名。	900864F8C11EB743
alg	签名算法	是	用于验证令牌上的签名的算法。支持以下算法: RS256、RS384、RS512、PS256、PS384、PS512。	PS256

ver	版本	是	汇丰私有声明。目前客户须将其设置为“1.0”。	1.0
-----	----	---	-------------------------	-----

注：我们推荐 PS256、PS384 和 PS512 算法，而不是其他 RSA 算法。PKCS#1.5 填充方案（RS256、RS384 和 RS512）没有提供所需的安全级别，通常应由 PSS 填充方案取代。

示例：

```
{
  "typ": "JWT",
  "kid": "A9248D0E4A463426",
  "alg": "PS256"
  "ver": "1.0"
}
```

2.2.6 JWT Payload 中的字段

下表定义了 JWT Payload 中要用到的字段

键名	名称	是否必填	描述	示例
jti	JWT ID	是	令牌是区分大小写的唯一标识符，即使在不同发行者之间也是如此。将相同的值意外分配给不同数据对象的概率可以忽略不计；如果应用程序使用多个发布者，则必须在不同发布者生成的值之间避免冲突。“jti”声明可用于防止 JWT 被重播。“jti”值是区分大小写的字符串。	74760410-f963-11e8-b2a3-1bb26e1e5b69
iat	签发时间	是	标识 JWT 签发的时间，值必须是日期的数值形式	1565150776
sub	主体	是	标识 JWT 的主体，必须是客户的 Profile ID	PC000012345
aud	受众	是	标识 JWT 的接收方，一直设置成 baas	baas
payload_hash	Payload 哈希码	是	HTTP 请求正文的 hash 码，用于 Mule Policy 做校验	1dab559f1f4bf4f3ca...
payload_hash_alg	Payload 哈希算法	是	Payload 的哈希算法，支持以下算法： - SHA-256 (RSASHA256) - SHA-384 (RSASHA384)	RSASHA256

			● SHA-512 (RSASHA512)	
--	--	--	-----------------------	--

示例:

```
{
  "jti": "5ccfd3a0-36a1-11ea-b780-eeee0af2723c",
  "iat": 1578987726,
  "sub": "PC000012345",
  "aud": "baas",
  "payload_hash": "a00ccb...",
  "payload_hash_alg": "RSASHA256",
}
```

2.3 实施指南

本节内容将帮助你:

- 准备 API 请求有效负载并执行有效负载加密
- 生成并签署客户端令牌 (JWT)
- 处理 API 响应所需的步骤

对于每个请求, 请准备以下内容:

6. 准备 JSON 结构中的普通 HTTP 请求正文。
7. 使用 HSBC 的公钥加密 HTTP 请求正文。
8. 生成客户端令牌(JWT)。
9. 在相关的授权请求标头中添加已签名 JWT(JWS)。
10. 添加关联 ID 以唯一标识系统中的请求并提交。

以下部分包含带有示例代码的实施指南。

2.3.1 步骤 1-准备请求正文

使用 API 参考文档根据特定的服务的 API 数据格式准备 payload 数据。

API 参考文档中的每个 API 服务地址都提供的请求正文供参考 — 请参见端点菜单。

2.3.2 步骤 2-加密 HTTP 请求正文

组件	描述	备注
明文	HTTP 请求正文 (JSON 格式)	参考 API 断点页
密钥	HSBC 提供的公钥	密钥交换时由 HSBC 提供的公钥

消息体加密步骤:

- 使用第一步准备的请求正文

- 使用 HSBC 提供的公钥加密消息体
- 使用 Base64 编码加密的消息体

代码示例：

为了了解如何构造加密消息，让我们以下面的 Java 程序为例来描述所有必需的组件

```
/**
 * Returns the encrypted and signed message
 *
 * @param plainText message which need to be encrypted
 * @param encryptKey the public key used for encryption provided by HSBC
 * @param signKey your own private key for signing
 * @return Encrypted message encoded by base64
 * @throws Exception
 */
public String encryptAndSign(String plainText, PGPPublicKey encryptKey, PGPPrivateKey signKey, String
signer) throws Exception {
    if (plainText == null || plainText.length() <= 0) {
        throw new IllegalArgumentException("Plain text is empty.");
    }
    ByteArrayInputStream in = new ByteArrayInputStream(plainText.getBytes());
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    encryptAndSign(out, in, encryptKey, signKey, signer);
    byte[] encrypted = out.toByteArray();
    String encryptedBody = new String(Base64.getEncoder().encode(encrypted), Charset.defaultCharset());
    in.close();
    out.close();
    return encryptedBody;
}

/**
 * Returns the encrypted and signed message
 * @param outputStream Encrypted and signed output stream
 * @param inputStream Input stream that needs to be encrypted
 * @param encryptKey the public key used for encryption provided by HSBC
 * @param signKey your own private key for signing
 * @param signer the name of signer
 * @throws Exception
 */
public void encryptAndSign(OutputStream outputStream, InputStream inputStream, PGPPublicKey
encryptKey, PGPPrivateKey signKey, String signer) throws Exception {
    OutputStream out = new ArmoredOutputStream(outputStream);
    try {
        //generate encrypted data packets
        PGPEncryptedDataGenerator encryptedDataGenerator = new PGPEncryptedDataGenerator(new
JcePGPDataEncryptorBuilder(PGPEncryptedData.AES_256)
            .setWithIntegrityPacket(true)
            .setSecureRandom(new SecureRandom())
```

```

        .setProvider(bcp));
        encryptedDataGenerator.addMethod(new
JcePublicKeyKeyEncryptionMethodGenerator(encryptKey).setProvider(bcp));
        OutputStream encryptedOutputStream = encryptedDataGenerator.open(out, new
byte[DEFAULT_BUFFER_SIZE]);

        PGPCompressedDataGenerator compressedDataGenerator = new
PGPCompressedDataGenerator(PGPCompressedData.ZIP);
        OutputStream compressedData = compressedDataGenerator.open(encryptedOutputStream);

        //generate complete OpenPGP signed packets
        PGPSignatureGenerator signatureGenerator = new PGPSignatureGenerator(new
JcaPGPContentSignerBuilder(signKey.getPublicKeyPacket().getAlgorithm(),
PGPUtil.SHA512).setProvider(bcp));
        signatureGenerator.init(PGPSignature.BINARY_DOCUMENT, signKey);

        //generate signed sub packets, such as signer, sign time, sign key id etc
        PGPSignatureSubpacketGenerator subpacketGenerator = new
PGPSignatureSubpacketGenerator();
        subpacketGenerator.setSignatureCreationTime(false, new Date());
        subpacketGenerator.setSignerUserID(false, signer);
        subpacketGenerator.setIssuerKeyID(false, signKey.getKeyID());

        signatureGenerator.setHashedSubpackets(subpacketGenerator.generate());
        signatureGenerator.generateOnePassVersion(false).encode(compressedData);

        //generate literal data, specify metadata such as data type, date, and file name
        PGPLiteralDataGenerator literalDataGenerator = new PGPLiteralDataGenerator();
        OutputStream lOut = literalDataGenerator.open(compressedData, PGPLiteralData.BINARY,
"Sample-Data", new Date(), new byte[DEFAULT_BUFFER_SIZE]); //bOut

        // convert input stream into byte array, then write into output stream wrapped by encryption,
compress algorithm
        byte[] data = IOUtils.toByteArray(inputStream);
        lOut.write(data);
        signatureGenerator.update(data);

        lOut.close();
        literalDataGenerator.close();

        signatureGenerator.generate().encode(compressedData);

        compressedDataGenerator.close();
        compressedData.close();

        encryptedOutputStream.close();
        encryptedDataGenerator.close();

```

```

        out.close();

        logger.info("encryption key id: {}, sign key id: {}", encryptKey.getKeyID(), signKey.getKeyID());
    } catch (PGPException e) {
        logger.error("PGP Exception", e);
        if (e.getUnderlyingException() != null) {
            logger.error("PGP Exception ", e.getUnderlyingException());
        }
    } catch (Throwable e) {
        logger.error("unknown PGP Error", e);

        if (inputStream != null) {
            inputStream.close();
        }
    }
}

@Test
public void testEncryptionAndSign() throws Exception {
    String plainText = "Trouble will find you no matter where you go.";
    String r = pgpHelper.encryptAndSign(plainText, getPublicKey(), getPrivateKey(), "Customer A");
    Assertions.assertTrue(r.length() > 0);
}

```

代码说明：

- 5) 将你需要加密的负载明文转换为字符串
- 6) 使用 IOUtils.toInputStream 将负载转换为输入流
- 7) 使用 HSBC 公钥作为加密器对明文负载进行加密
- 8) 使用 base64 对加密结果进行编码

下一步，生成 JWS

2.3.3 步骤 3-生成和签名客户令牌

JWT 组件

下表定义了 JWT Payload 中要用到的字段，所有字段都必填。字段的定义请参考“[1.2.6 JWT Payload](#)”部分

编码	名字	位置	示例
typ	Token 类型	JWT 头	JWT – 必须设置为“JWT”
kid	PGP Key Id	JWT 头	900864F8C11EB743
alg	签名算法	JWT 头	PS256
ver	版本	JWT 头	1.0 – 必须设置为“1.0”

jti	JWT ID	JWT 负载主体	74760410-f963-11e8-b2a3-1bb26e1e5b69
iat	发布时间	JWT 负载主体	1565150776
sub	主体	JWT 负载主体	PC000012345
aud	受众	JWT 负载主体	baas
payload_hash	负载主体 hash 值	JWT 负载主体	1dab559f1f4bf4f3ca...
payload_hash_alg	负载主体 hash 算法	JWT 负载主体	RSASHA256

JWT 准备步骤:

- 5) 使用上表提供的步骤构建 JWT 结构
- 6) 为用于步骤 2 中的加密请求负载生成 hash 码
- 7) 构建 JWT 头
- 8) 生成和签名 JWT(JWS)

代码示例:

```
//1. Build the JWT structure
Map<String, Object> claims = new HashMap<>();

long nowSeconds = System.currentTimeMillis() / 1000;
claims.put("iat", nowSeconds);
claims.put("jti", UUID.randomUUID().toString());
claims.put("sub", "BS0000111111"); //your profile id
claims.put("aud", "baas");

//2 Generate hash code for request payload and put into JWT Token
claims.put("payload_hash", "038c3d8437c49840591baf7eb2a2f292c4f959acfe048444da48fc376df36b87");
//your payload hash code
claims.put("payload_hash_alg", "RSASHA256");

//3 Set JWT header
Map<String, Object> header = new HashMap<String, Object>();
header.put("kid", "71F28C2FB8886368"); //key id
header.put("typ", "JWT");
header.put("ver", "1.0");
header.put("key_typ", "PGP");
header.put("alg", "PS256");

//4 Generate and Sign the JWT (JWS)
PrivateKey rsaKey = pgpHelper.convertToRsaKey(getPrivateKey());
String token = Jwts.builder().setHeader(header).setClaims(claims).signWith(rsaKey,
SignatureAlgorithm.PS256).compact();
```

代码说明:

- 5) 构建 JWT 结构
- 6) 为加密的请求负载主体生成 hash 码

- 7) 构建 JWT 头
- 8) 生成和签名 JWT(JWS)

下一步，添加 JWS 到 HTTP 头

2.3.4 步骤 4-添加签名的 JWT(JWS)到 HTTP 头

添加 JWS 到 HTTP 头部 Authorization 字段

```
Authorization=JWS eyJ2ZXIiOiIxLjAiLCJraWQiOiJBNjI4RjhBRDY3NERCQz..
```

下一步，根据每个服务 API URL 的规范和要求添加其它字段

2.3.5 步骤 5-添加保留的头部并提交

根据每个具体的 API URL 规范确定是否要添加其它必要的 header 字段

关联 ID 是一个唯一 ID，它由你的系统生成用来唯一标识每一个请求

2.3.6 处理响应

响应由 HTTP 响应码和加密的响应消息组成

响应处理步骤：

- 5) 一个被成功处理的请求的响应消息是一个加密的且用 base64 进行编码了的负载
- 6) 使用 base64 解码响应消息负载
- 7) 使用你的私钥解密解码后的消息负载
- 8) 解密后的负载主体每个 API URL 可能有所不同，具体请参考你调用的 API URL 文档

一个典型的加密和编码的响应消息：

```
{
  "profileId": "PC12345678",
  "referenceId": "b7e0fdacd8bd",
  "statusCode": "ACCP",
  "statusDesc": "Accepted Technical Validation (L2)",
  "reportBase64": "PFJlc3BvbnNlPg0KICA8UGF5bWVudElkPml3ZTBmZGFjLWQ4YmQtNDk1MC1iZDNGljSWQ+UEMxMjM+"
}
```

代码示例：

```
/**
 * Decrypt and verify encrypted message with your private key and HSBC public key
 *
 * @param ciphertext encrypted message encoded with Base64
```

```

* @param privateKeys your own private key used for decryption
* @param publicKeys public key provided by HSBC
* @return Decrypted message
* @throws Exception
*/
public String decryptAndVerify(String ciphertext, List<PGPPrivateKey> privateKeys, List<PGPPublicKey>
publicKeys) throws Exception {
    ByteArrayInputStream inputStream = new
ByteArrayInputStream(Base64.getDecoder().decode(ciphertext));

    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();

    decryptAndVerify(inputStream, outputStream, privateKeys, publicKeys);

    return outputStream.toString("UTF-8");
}

/**
* Decrypt and verify encrypted message with your private key and HSBC public key
* @param in encrypted input stream
* @param out decrypted output stream
* @param privateKeys your own private key used for decryption
* @param publicKeys public key provided by HSBC
* @throws Exception
*/
public void decryptAndVerify(InputStream in, OutputStream out, List<PGPPrivateKey> privateKeys,
List<PGPPublicKey> publicKeys) throws Exception {
    if (in == null) {
        throw new IllegalArgumentException("The encrypted data is null");
    }

    PGPEncryptedDataList encryptedDataList;
    PublicKeyDataDecryptorFactory dataDecryptorFactory = null;
    PGPPublicKeyEncryptedData encryptedData = null;
    try {
        // create PGPObjFactory obj, parse and handle input stream, get encrypted data list obj
        PGPObjFactory pgpObjFactory = new PGPObjFactory(PGUtil.getDecoderStream(in),
keyFingerPrintCalculator);

        ///#1 get data obj from PGPObjFactory check if it's PGPEncryptedDataList
        Object obj = pgpObjFactory.nextObject();
        if (obj instanceof PGPEncryptedDataList) {

```

```

        encryptedDataList = (PGPEncryptedDataList) obj;
    } else {
        // the first object might be a PGP marker packet.
        encryptedDataList = (PGPEncryptedDataList) pgpObjectFactory.nextObject();
    }

    // data list should only include one encrypted data obj
    if (encryptedDataList.size() > 1) {
        logger.warn("There are {} encrypted data, only decrypt the last one",
encryptedDataList.size());
    }

    ##2 get the encrypted data, only process the last one
    Iterator<PGPPublicKeyEncryptedData> it = encryptedDataList.getEncryptedDataObjects();
    while (it.hasNext()) {
        encryptedData = it.next();
    }

    ##3 Find the matching private key to create PublicKeyDataDecryptorFactory obj
    for (PGPPrivateKey privateKey : privateKeys) {
        if (encryptedData.getKeyID() == privateKey.getKeyID()) {
            dataDecryptorFactory = new
JcePublicKeyDataDecryptorFactoryBuilder().setProvider(bcp).setContentProvider(bcp).build(privateKey);

            break;
        }
    }
} catch (Throwable e) {
    logger.error("Parse and handle input stream failed", e);
}

if (dataDecryptorFactory == null) {
    throw new PGPEException("Provided key id does not match the key client used on this payload");
}

PGPCompressedData compressedData;
PGPOnePassSignatureList onePassSignatureList = null;
PGPSignatureList signatureList = null;
ByteArrayOutputStream plainOutput = new ByteArrayOutputStream();
InputStream decryptedDataStream = null;
byte[] output = null;
try {

```

```

// #4 Get the plain text by using your private key to decrypt the encrypted text
decryptedDataStream = encryptedData.getDataStream(dataDecryptorFactory);

// parse and handle decrypted data stream, get compressed data obj
PGPObjectFactory pgpObjectFactory2 = new PGPObjectFactory(decryptedDataStream,
keyFingerPrintCalculator);
Object dataObj = pgpObjectFactory2.nextObject();

while (dataObj != null) {
    if (dataObj instanceof PGPCompressedData) {
        compressedData = (PGPCompressedData) dataObj;

        // parse out compressed data
        pgpObjectFactory2 = new PGPObjectFactory(compressedData.getDataStream(),
keyFingerPrintCalculator);
        dataObj = pgpObjectFactory2.nextObject();
    }

    if (dataObj instanceof PGPLiteralData) {
        // have to read it and keep it somewhere.
        Streams.pipeAll(((PGPLiteralData) dataObj).getInputStream(), plainOutput);
    } else if (dataObj instanceof PGPOnePassSignatureList) {
        onePassSignatureList = (PGPOnePassSignatureList) dataObj;
    } else if (dataObj instanceof PGPSignatureList) {
        signatureList = (PGPSignatureList) dataObj;
    } else {
        throw new GPException("Unknown message type.");
    }

    // get and handle the next data obj from factory
    dataObj = pgpObjectFactory2.nextObject();
}

output = plainOutput.toByteArray();
} catch (Throwable e) {
    logger.error("Parse and handle decrypted data stream failed", e);
} finally {
    plainOutput.close();
    if (decryptedDataStream != null) {
        decryptedDataStream.close();
    }
}
}

```

```

if (onePassSignatureList == null || signatureList == null) {
    throw new SignatureException("Poor PGP. Signatures not found.");
}

boolean signatureVerified = false;
try {
    // #5 Verify the signature using HSBC's Public Key, should only include one signature in encrypted
data
    for (int i = 0; i < onePassSignatureList.size(); i++) {
        PGPOnePassSignature ops = onePassSignatureList.get(i);
        if (publicKeys != null) {
            for (PGPPublicKey publicKey : publicKeys) {
                ops.init(new JcaPGPContentVerifierBuilderProvider().setProvider(bcp), publicKey);
                ops.update(output);

                PGPSignature signature = signatureList.get(i);
                if (ops.verify(signature)) {
                    signatureVerified = true;
                }
            }
        }
    }
} catch (Throwable e) {
    logger.error("verification failed", e);
}

if (!signatureVerified) {
    throw new SignatureException("Signature verification failed");
}

if (encryptedData.isIntegrityProtected() && !encryptedData.verify()) {
    throw new PGPDDataValidationException("Data is integrity protected but integrity is lost.");
}

try {
    // write decrypted byte array into output stream
    out.write(output);
    out.flush();
} catch (Throwable e) {
    logger.error("Write output stream failed", e);
} finally {

```

```

        if (out != null) {
            out.close();
        }
        in.close();
    }
}

@Test
public void decryptAndVerifyTest() throws Exception {
    String expected = "Trouble will find you no matter where you go.";
    String ciphertext = "LS0tLS1CRUdJTiBQR1AgTUVTU0FHRS0tLS0tDQpWZXJzaW9uOiBCQ1BHI...";
    String actual = pgpHelper.decryptAndVerify(ciphertext, getBankPrivateKeys(), getClientPublicKeys());
    Assertions.assertEquals(expected, actual);
}

@Test
public void decryptAndVerifyTest2() throws Exception {
    String expected = "Trouble will find you no matter where you go.";
    String ciphertext = "LS0tLS1CRUdJTiBQR1AgTUVTU0FHRS0tLS0tDQpWZXJzaW9uOiBCQ1BHI...";
    byte[] decodedData = Base64.getDecoder().decode(ciphertext);
    InputStream inputStream = new ByteArrayInputStream(decodedData);
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    pgpHelper.decryptAndVerify(inputStream, outputStream, getBankPrivateKeys(),
getClientPublicKeys());
    String actual = outputStream.toString();
    Assertions.assertEquals(expected, actual);
}

```

代码说明：

- 4) 解码 base64 编码的响应
- 5) 使用自己的私钥解密响应消息，注意该私钥是你与 HSBC 交换的密钥对中的私钥
- 6) 使用 HSBC 的公钥验签消息

2.4 POST /balances

账户余额查询 API。

2.4.1 描述

- 账户余额查询是一个同步查询 API，用于查询当天或历史日期账户的实时余额信息；
- balanceDate 字段可用于过去 365 天的历史日期的余额信息查询；

- 一次请求可以支持最多 50 个相同或不同地区的账户余额信息查询；
- 支持的请求明文格式为 JSON，而响应的明文格式可以是 JSON 或者是 CAMT (ISO 20022 XML)；
- 对于 camt.052 XML 的响应，将会返回账户当天的余额信息报告；
- 对于 camt.053 XML 的响应，将会返回账户历史日期的余额信息报告；
- JSON 格式的响应将返回账户当天以及历史日期的余额信息报告；
- 账户余额查询 API 不提供该账户的交易记录信息。

API 响应中包含 6 种类型的余额信息
Opening Available Balance (OPAV)
Opening Booked Balance (OPBD)
Interim (Current Day) Available Balance (ITAV)
Interim (Current Day) Booked Balance (ITBD)
Closed (Historical Date) Available Balance (CLAV)
Closed (Historical Date) Booked Balance (CLBD)

2.4.2 API 请求正文

账户余额信息查询可以支持一个或者多个账户的查询，单次查询最多可支持 50 个账户。具体的 API 请求标头请参考 API 文档规范。

字段名	描述	字段类型	最大长度	必填	示例
accountBalances	要查询的汇丰银行账户列表	Array	50	Yes	
accountNumber	汇丰银行账户	String	34	Yes	40000012345678
accountCountry	汇丰银行账户所在的国家代码	String	2	Yes	GB
institutionCode	汇丰银行账户所在的机构代码	String		No	HBEU
accountType	账户类型标识。例如，CA 表示现金账户（Current Account）	String		No	CA
balanceDate	要查询余额的日期	Date		No	yyyy-mm-dd

请求正文示例:

1. 单一账户查询:

```
{
```

```

"balanceDate": "2021-09-06",
"accountBalances": [
  {
    "accountNumber": "12345678",
    "accountCountry": "GB",
    "accountType": "CA",
    "institutionCode": "HBEU"
  }
]
}

```

2. 多账户查询:

```

{
  "balanceDate": "2021-09-06",
  "accountBalances": [
    {
      "accountNumber": "12345678",
      "accountCountry": "GB",
      "accountType": "CA",
      "institutionCode": "HBEU"
    },
    {
      "accountNumber": "12345678",
      "accountCountry": "US",
      "accountType": "CA",
      "institutionCode": "HBUS"
    }
  ]
}

```

准备 API 请求正文:

- 按照约定格式准备请求正文明文，多账户查询时最大不超过 50 个账户；
- 使用您的客户私钥对请求正文明文签名，并在此基础上使用汇丰银行公钥对其加密；
- 对加密好的密文做 Base64 编码；
- 将 Base64 编码的密文填入请求正文的 JSON 字段 - balancesRequestBase64

Base64 编码的请求正文示例:

```

{
  "balancesRequestBase64": "PD94bWwgdmVyc2lvdj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCjxEb2N1bWVudCB4bWxucz0idXJuOmlzbz="
}

```

2.4.3 API 响应正文

API 的成功响应正文包含以下信息：

字段名	描述
referenceId	由 API 生成的对应当前请求的唯一标识码。最大长度 12 位。
profileId	使用汇丰银行 API 服务的 Profile ID
statusCode	响应状态码。比如，ACCP
statusDesc	对应响应状态码的描述。
reportBase64	已被签名，加密以及 Base64 编码的 XML 或者 JSON 格式的响应正文明文。

成功响应正文的示例：

```
{
  "referenceId": "b7e0fdacd8bd",
  "profileId": "PC12345678",
  "statusCode": "ACCP",
  "statusDesc": "Accounts balances report generated successfully.",
  "reportBase64": "PFJlc3BvbmlPg0KICA8UGF5bWVudElkPmI3ZTBmZGFjLWQ4YmQtNDk1MC1iZDNG1jS
WQ+"
}
```

响应正文的处理：

- 从响应正文中获取 JSON 字段 reportBase64 的值；
- 对获取的值进行 Base64 解码；
- 使用您客户端的私钥对解码后的内容做解密得到响应正文明文；
- 您查询的账户余额信息就在解密后的明文里，基于您做 API 请求时指定的报表返回格式，它可能是 CAMT52，CAMT53 或者 JSON 的格式。

响应正文明文的示例

以下文档规范及示例提供了被加密，编码的响应正文明文内各个字段的定义和详细信息。这些信息也可以在“Endpoints”页面下载的 API 文档规范（Swagger 文件）里找到。

CAMT52 格式（当天）

详情请参考 SWIFT MyStandards 规范：

camt.052 规范：[HSBC.camt.052.001.02 BankToCustomerAccountReport](#)

camt.052 Excel 格式映射：[HSBC.camt.052.001.02 BankToCustomerAccountReport Excel](#)

camt.052 标准账户示例：[balances camt.052 standard account sample](#)

camt.052 IBAN 示例：[balances camt.052 IBAN sample](#)

CAMT53 格式（历史日期）

详情请参考 SWIFT MyStandards 规范:

camt.053 规范: [HSBC.camt.053.001.02 BankToCustomerStatement](#)

camt.053 Excel 格式映射: [HSBC.camt_053_001_02_BankToCustomerStatement_Excel](#)

camt.053 文件示例: [balances camt.053 sample](#)

JSON 格式

详情请参考 API 文档规范中 JSON 字段的定义。

```
{
  "balances": [
    {
      "identification": "USHBUSCA123456789",
      "balance": [
        {
          "type": "OPAV",
          "creditDebitIndicator": "Credit",
          "dateTime": "2019-09-26T",
          "amount": {
            "amount": "9.63",
            "currency": "USD"
          },
          "creditLine": [
            {
              "included": false,
              "amount": {
                "amount": "0.00",
                "currency": "USD"
              }
            }
          ]
        },
        {
          "type": "OPBD",
          "creditDebitIndicator": "Credit",
          "dateTime": "2019-09-26T",
          "amount": {
            "amount": "9.63",
            "currency": "USD"
          },
          "creditLine": [
            {
              "included": false,
              "amount": {
                "amount": "0.00",
                "currency": "USD"
              }
            }
          ]
        }
      ]
    },
    {
      "type": "ITAV",
      "creditDebitIndicator": "Credit",
      "dateTime": "2019-09-26T00:00:00",
      "amount": {
        "amount": "9.63",
        "currency": "USD"
      }
    }
  ]
}
```


API 响应中交易记录明细的格式由查询日期决定
查询日期为当天，则返回 camt.052 格式的日内交易记录明细
查询日期为历史日期，则返回 camt.053 格式的历史交易记录明细
JSON 格式的响应则可以支持当天的日内交易记录，或者历史日期的交易记录明细查询

2.5.2 API 请求正文

账户交易记录明细查询可以支持单一账户的当天或者历史日期的查询。具体的 API 请求标头请参考 API 文档规范。

字段名	描述	字段类型	必填	示例
transactionDate	要查询交易记录的日期	Date	Yes	yyyy-mm-dd
transactionTimeFrom	交易记录时间起点（24 小时制格式）。仅适用于当天的查询。	Time	No	hh:mm
transactionTimeTo	交易记录时间终点（24 小时制格式）。仅适用于当天的查询。	Time	No	hh:mm
accountNumber	汇丰银行账户	String	Yes	40000012345678
accountCountry	汇丰银行账户所在的国家代码	String	Yes	GB
institutionCode	汇丰银行账户所在的机构代码	String	No	HBEU
accountType	账户类型标识。例如，CA 表示现金账户（Current Account）	String	No	CA
bankTransactionType	银行交易类型代码，缺省为 SWIFT，该输入决定了响应返回里的银行交易代码。	String	No	SWIFT 或者 BAI

请求正文示例:

1. 当天查询:

```
{
  "transactionDate": "YYYY-MM-DD",
  "transactionTimeFrom": "12:56",
  "transactionTimeTo": "18:56",
```

```

"accountNumber": "123456789",
"accountCountry": "GB",
"institutionCode": "HBEU",
"accountType": "CA",
"bankTransactionType": "SWIFT"
}

```

2. 历史日期查询:

```

{
"transactionDate": "YYYY-MM-DD",
"accountNumber": "123456789",
"accountCountry": "GB",
"institutionCode": "HBEU",
"accountType": "CA",
"bankTransactionType": "BAI"
}

```

准备 API 请求正文:

- 按照约定格式准备请求正文明文;
- 使用您的客户私钥对请求正文明文签名, 并在此基础上使用汇丰银行公钥对其加密;
- 对加密好的密文做 Base64 编码;
- 将 Base64 编码的密文填入请求正文的 JSON 字段 - transactionsRequestBase64

Base64 编码的请求正文示例:

```

{
"transactionsRequestBase64": "PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluc2Z0iVVRGLTgiPz4NCjxlb2N1bWVudCB4bWxucz0idXJuOmlzbnZ="
}

```

2.5.3 API 响应标头

使用以下响应标头来确认是否有更多的交易记录数据 (超过 1000 条时) 需要获取。

标头名	可能的值	描述
x-next-batch	Y	当 x-next-batch = Y 时, 表示对于查询的日期/时间, 还有更多的交易记录待返回。
x-batch-token	wEJihgdikdimnADMwyLsHGD	当 x-next-batch = Y 时, 提取 the x-batch-token 以继续获取当前查询日期/时间的下一批交易记录。

备注: x-next-batch 和 x-batch-token 仅在查询的交易记录超过 1000 条时返回。

2.5.4 API 响应正文

API 的成功响应正文包含以下信息：

字段名	描述
referenceId	由 API 生成的对应当前请求的唯一标识码。最大长度 12 位
profileId	使用汇丰银行 API 服务的 Profile ID
statusCode	响应状态码。比如，ACCP
statusDesc	对应响应状态码的描述。
reportBase64	已被签名，加密以及 Base64 编码的 XML 或者 JSON 格式的响应正文明文。

成功响应正文的示例：

```
{
  "referenceId": "b7e0fdacd8bd",
  "profileId": "PC12345678",
  "statusCode": "ACCP",
  "statusDesc": "Account Transaction Report generated successfully.",
  "reportBase64": "PFJlc3BvbmlPg0KICA8UGF5bWVudElkPmI3ZTBmZGFjLWQ4YmQtNDk1MC1iZDNGljS
WQ+"
}
```

响应正文的处理：

- 从响应正文中获取 JSON 字段 reportBase64 的值；
- 对获取的值进行 Base64 解码；
- 使用您客户端的私钥对解码后的内容做解密得到响应正文明文；

您查询的账户交易记录信息就在解密后的明文里，基于您做 API 请求时指定的报表返回格式，它可能是 CAMT52，CAMT53 或者 JSON 的格式。

响应正文明文的示例

以下文档规范及示例提供了被加密，编码的响应正文明文内各个字段的定义和详细信息。这些信息也可以在“Endpoints”页面下载的 API 文档规范（Swagger 文件）里找到。

CAMT52 格式（当天）

详情请参考 SWIFT MyStandards 规范：

camt.052 规范：[HSBC.camt.052.001.02 BankToCustomerAccountReport](#)

camt.052 Excel 格式映射：[HSBC.camt.052.001.02 BankToCustomerAccountReport Excel](#)

camt.052 示例：[transactions camt.052 sample](#)

CAMT53 格式（历史日期）

详情请参考 SWIFT MyStandards 规范：

camt.053 规范：[HSBC.camt.053.001.02 BankToCustomerStatement](#)

camt.053 Excel 格式映射：[HSBC.camt_053_001_02 BankToCustomerStatement Excel](#)

JSON 格式

详情请参考 API 文档规范中 JSON 字段的定义。

```
{
  "transactions": {
    "scheme": {
      "identificaton": "USHBUSCA123456789"
    },
    "requestorAccountId": "123456789",
    "balance": [
      {
        "type": "OPAV",
        "creditDebitIndicator": "CREDIT",
        "dateTime": "2019-09-23T",
        "amount": {
          "amount": "9.68",
          "currency": "USD"
        },
        "creditLine": [
          {
            "included": false,
            "amount": {
              "amount": "0.00",
              "currency": "USD"
            }
          }
        ]
      },
      {
        "type": "OPBD",
        "creditDebitIndicator": "CREDIT",
        "dateTime": "2019-09-23T",
        "amount": {
          "amount": "9.68",
          "currency": "USD"
        },
        "creditLine": [
          {
            "included": false,
            "amount": {
              "amount": "0.00",
              "currency": "USD"
            }
          }
        ]
      }
    ]
  }
}
```

```

    },
    {
      "type": "CLBD",
      "creditDebitIndicator": "CREDIT",
      "dateTime": "2019-09-23TCLOSE",
      "amount": {
        "amount": "9.61",
        "currency": "USD"
      },
      "creditLine": [
        {
          "included": false,
          "amount": {
            "amount": "0.00",
            "currency": "USD"
          }
        }
      ]
    },
    {
      "type": "CLAV",
      "creditDebitIndicator": "CREDIT",
      "dateTime": "2019-09-23TCLOSE",
      "amount": {
        "amount": "9.61",
        "currency": "USD"
      },
      "creditLine": [
        {
          "included": false,
          "amount": {
            "amount": "0.00",
            "currency": "USD"
          }
        }
      ]
    }
  ],
  "statementCount": "5",
  "transaction": [
    {
      "items": {
        "transactionReference": "NONREF",
        "statementReference": "REGJ2",
        "creditDebitIndicator": "D",
        "reversalIndicator": "N",

```

```

    "transactionStatus": "S",
    "bookingDateTime": "2019-09-23",
    "valueDateTime": "2019-09-23",
    "transactionInformation": "MISCELLANEOUS DEBIT ACH CHARGE FOR GENERATED OFF
SET",
    "transactionAmount": {
      "amount": "0.07",
      "currency": "USD"
    },
    "bankTransactionCode": {
      "code": "MDOP",
      "subcode": "OTHR"
    },
    "proprietaryBankTransactionCode": {
      "code": "NMSC",
      "issuer": "SWIFT"
    }
  },
  {
    "items": {
      "transactionReference": "266490729",
      "statementReference": "E2EIDREFPVT066",
      "creditDebitIndicator": "D",
      "reversalIndicator": "N",
      "transactionStatus": "S",
      "bookingDateTime": "2019-09-23",
      "valueDateTime": "2019-09-23",
      "transactionInformation": "BOOK TRANSFER DEBIT 45BOOK DEBIT HSBC BANK USA",
      "transactionAmount": {
        "amount": "0.01",
        "currency": "USD"
      },
      "bankTransactionCode": {
        "code": "ICDT",
        "subcode": "BOOK"
      },
      "proprietaryBankTransactionCode": {
        "code": "NTRF",
        "issuer": "SWIFT"
      }
    }
  },
  {
    "items": {
      "transactionReference": "NONREF",

```

```

        "statementReference": "NONREF",
        "creditDebitIndicator": "D",
            "reversalIndicator": "N",
        "transactionStatus": "S",
        "bookingDateTime": "2019-09-23",
        "valueDateTime": "2019-09-23",
        "transactionInformation": "ACH PAYMENT HEXAGON PAYMENT PVT TESTI PAYMENT 13
2774727",
        "transactionAmount": {
            "amount": "0.07",
            "currency": "USD"
        },
        "bankTransactionCode": {
            "code": "ICDT",
            "subcode": "ATXN"
        },
        "proprietaryBankTransactionCode": {
            "code": "NTRF",
            "issuer": "SWIFT"
        }
    },
    {
        "items": {
            "transactionReference": "NONREF",
            "statementReference": "NONREF",
            "creditDebitIndicator": "C",
                "reversalIndicator": "N",
            "transactionStatus": "S",
            "bookingDateTime": "2019-09-23",
            "valueDateTime": "2019-09-23",
            "transactionInformation": "ACH CONCENTRATION CREDIT CORPORATE DEPOSIT FROM
HEXAGON TESTING",
            "transactionAmount": {
                "amount": "0.04",
                "currency": "USD"
            },
            "bankTransactionCode": {
                "code": "RCCN",
                "subcode": "ACON"
            },
            "proprietaryBankTransactionCode": {
                "code": "NTRF",
                "issuer": "SWIFT"
            }
        }
    }
}

```

```

    },
    {
      "items": {
        "transactionReference": "NONREF",
        "statementReference": "NONREF",
        "creditDebitIndicator": "C",
        "reversalIndicator": "N",
        "transactionStatus": "S",
        "bookingDateTime": "2019-09-23",
        "valueDateTime": "2019-09-23",
        "transactionInformation": "ACH CREDIT DEPOSIT FROM PVT TESTING IV H",
        "transactionAmount": {
          "amount": "0.04",
          "currency": "USD"
        },
        "bankTransactionCode": {
          "code": "MCOP",
          "subcode": "OTHR"
        },
        "proprietaryBankTransactionCode": {
          "code": "NTRF",
          "issuer": "SWIFT"
        }
      }
    }
  ],
  "pagination": {
    "pageNumber": "1",
    "lastpageInd": "true"
  }
}

```

2.6 Swagger

[Swagger 文档](#)

2.7 Treasury API 沙盒

2.7.1 使用指南

试验企业沙盒之前，您需要创建一个项目。项目创建过程中会生成用于沙盒 API 调用时身份认证所需的客

户凭证以及加密密钥。

要创建您的第一个沙盒项目，请点击 Developer Portal 主页面右上角的 Dev Hub 链接。

Dev Hub 是集中管理您沙盒项目的地方。在这里，项目管理员可以为某个沙盒项目增加/移除您组织里的成员。

更多参考：

[沙盒项目的详细用户指南](#)

[沙盒工具箱下载](#)

[沙盒数据管理详细指南](#)

2.7.2 沙盒的响应

我们的企业沙盒支持对此 API 请求的智能响应。

智能沙盒是一个完整的测试环境，它可以根据 API 请求的输入从数据中返回经过验证的响应。

- 智能沙盒维护着数据的状态，如新的支付请求等；
- 它跟我们生产环境的 API 一样，采用了基于 JWT 的安全策略以及 PGP 的加密机制；
- 请求和响应的正文明文都需要被签名，加密以及做 Base64 编码；
- 加密密钥（客户端跟银行端）会在沙盒项目中提供；
- 客户私钥的密码可以在您下载的加密密钥压缩包内的 client-pgp-keys.info 文件中找到。

智能沙盒 URL: <https://sandbox.corporate-api.hsbc.com/{version}/{api endpoint name}>

2.7.3 智能沙盒支持的测试账户

以下测试帐户已添加到智能沙箱默认数据集中。准备沙盒测试负载时请使用这些帐户。

沙盒测试账户：

目前，为您的沙盒项目创建的默认数据中定义了两个沙盒客户和帐户。

Customer ID	Treasury ID	Cards ID	Account Number	
USCIN:1111111111	USTRSY:PC000000001	USCARD:PC000000000001	1100000011	
USCIN:2222222222	USTRSY:PC000000002	USCARD:PC000000000002	1100000012	
accountNumber	accountCountry	institutionCode	accountType	bic
1100000011	US	HBUS	CA	MRMDUS33XXX
1100000012	US	HBUS	CA	MRMDUS33XXX

成功完成引入流程创建的客户将被添加到沙盒数据集中，且可以使用 OBO 查询沙盒管理 API 进行查询。

https://sandbox.corporate-api.hsbc.com/v1/form/customer_onboarding/obo_profiles

2.7.4 Postman Collection

通过以下链接您可以下载一个可用的 Postman Collection 来开启您的智能沙盒之旅。

Run in Postman

Please either:

1. 在 HSBC 工作区里使用只读模式查看 Postman Collection, 此工作区下 HSBC 的改动对您都是可见的;
2. 创建属于您自己使用 Collection 的分支 – 注意, 此分支下 HSBC 工作区的改动是不可见的;
3. 下载 Postman Collection 以在您的桌面 Postman 应用中使用它。

请注意, Postman 的注册是必须的。

重要声明

本文件由汇丰银行(中国)有限公司(“汇丰”)提供。汇丰对本文件内容的准确性、充分性或与收件人意图的相关性不作保证，且汇丰不承诺亦无义务向收件人提供任何额外信息、或更新本文件全部或部分内容、或更正任何可能显见的不准确处。收取本文件全部或部分并不构成要约、邀请或业务招徕。收件人应自行负责对本文件所涉产品、服务和其他内容做出独立评估。本文件应作为整体被完整阅读，且未经相关汇丰集团成员事先书面同意，不得被全文或部分复制、翻印、派发或向他人披露。

© 版权所有。汇丰集团 2024