大批量资金结算产品

接入手册

V1.0

上海快钱信息服务有限公司

目录

1 文档说明

1.1 文档目标

本文档的目的是为快钱大批量资金结算产品定义一个接入规范，帮助商户技术人掌握该产品相关功能，并顺利完成技术接入.

1.2 阅读对象

快钱商户及合作伙伴的开发人员、维护人员和管理人员。

他们应具备以下基本知识：

- 了解 MICROSOFT WINDOWS/NT、WINDOWS9X、WINDOWS 2000、HP-UX、AIX、SUN SOLARIS、LINUX、BSD 等操作系统的其中一种；
- 了解上述系统上的网站设置和网页制作方法；
- 熟悉 CGI、ASP、PHP、.NET、JAVA 以及 HTML、XML、WEB SERVICE 等开发语言及技术；
- 了解信息安全的基本概念。

1.3 相关约定

- 商户：使用该接口完成大批量资金结算的用户。
- PKI-ASAP：PKI-Application Security Authentication Platform，PKI-应用安全认证平台 。
- WSDL：Web service Description Language(web service 描述语言)。

- SFTP：SFTP 是 Secure File Transfer Protocol 的缩写，安全文件传送协议。可以为传输文件提供一种安全的加密方法。sftp 与 ftp 有着几乎一样的语法和功能。
- HTTP：超文本传输协议(HTTP，HyperText Transfer Protocol)是互联网上应用最为广泛的一种网络协议。所有的 WWW 文件都必须遵守这个标准。设计 HTTP 最初的目的是为了提供一种发布和接收 HTML 页面的方法。
- GZIP：压缩数据流

1.4 技术支持

如果您有任何技术上的疑问，可按如下方式寻求帮助：

技术支持热线：86 -21 - 58777299 / 58777399 - 8163 / 8161

技术支持邮箱：support@99bill.com

技术支持时间：周一到周五 9:00-18:00

2. 接口开发

2.1 功能说明

大批量资金结算产品，是让签约用户可以无需登录快钱平台,使用系统对接的方式完成付款指令的提交.

2.2 开发准备

商户开发人员应该仔细阅读本接口规范，并准备好如下资料：

- 商户在快钱的商户编号
- 商户授权的批量付款的产品功能代号
- 商户策略配置文件
- 参考 ASAP 应用程序开发包操作手册_v1.1.doc

2.3 参数说明

参考【大批量资金结算产品】接口.doc　中的参数说明

2.4 SFTP 目录说明

根目录：/home

用户目录：根目录+/用户名，比如快钱目录为：/home/99bill/

商户操作目录：

发送文件目录：用户目录+/to99bill，比如快钱目录为：/home/99bill/to99bill

接收文件目录：用户目录+/from99bill，比如快钱目录为：/home/99bill /from99bill

临时文件目录：用户目录+/temp，比如快钱目录为：/home/99bill/temp

2.5 文件名说明

送盘文件名：INBOUND_MEMBERCODE_YYYYMMDDHHMISS_批次号.PKI

送盘回执文件：INBOUND_RESP_MEMBERCODE_YYYYMMDDHHMISS_批次号.PKI

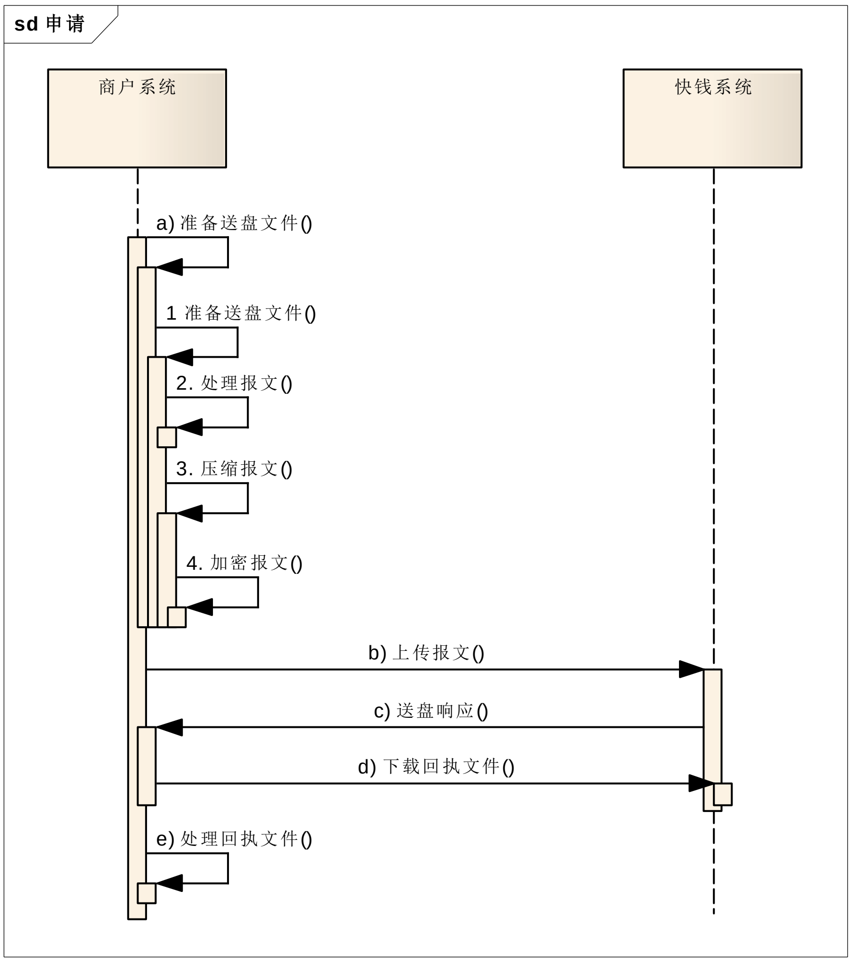
返盘文件名：OUTBOUND_MEMBERCODE_YYYYMMDDHHMISS_批次号.PKI

编码方式：统一使用 UTF-8，包括文件，Webservice，http.

2.6 处理流程

2.6.1 FTP 送盘

一、　大批量结算申请业务处理时序图



二、　申请业务时序图说明

a) 商户系统准备送盘文件；

1) 参考【大批量资金结算产品】接口.docx　并准备报文；

2) 商户报文进行 GZIP 格式压缩

3) 商户系统调用快钱提供的 PKI SDK 按照商户跟快钱签订协议中的加密策略

进行加密；

4) 商户系统对加密的结果进行 base64 的 encoding ,然后将结果输出到文件；

b) 商户系统上传送盘文件至快钱 SFTP 服务器；

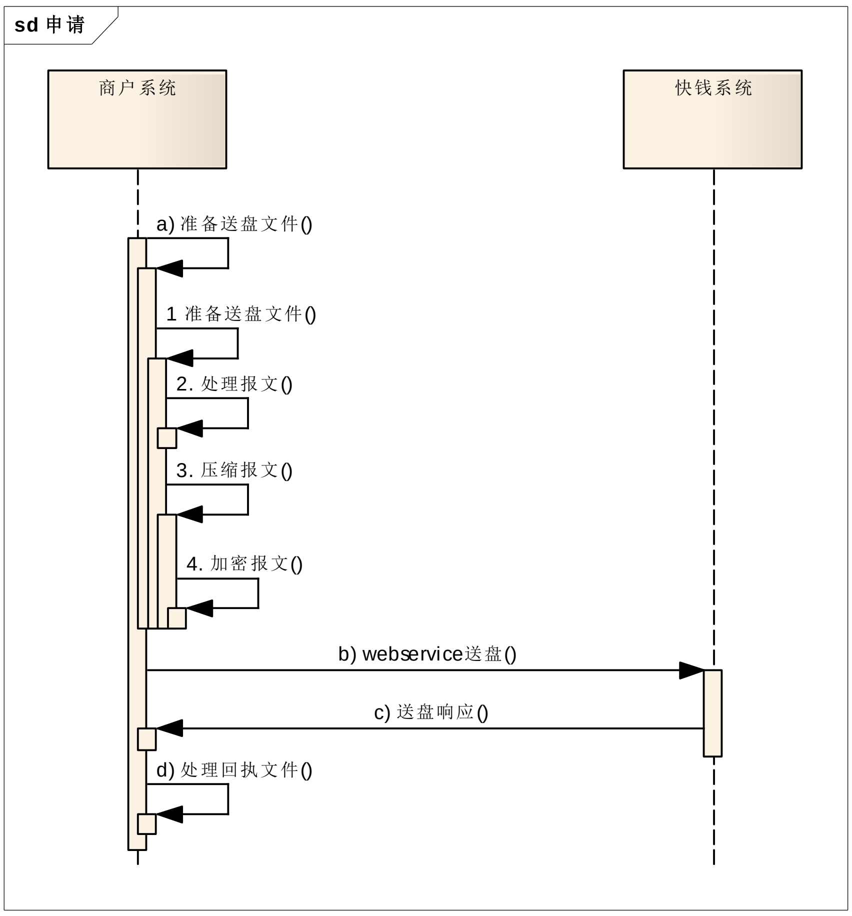
c) 快钱处理完后，通过 http 通知商户处理结果；

d) 商户系统收到快钱系统的批次处理 http 通知后，通过 SFTP 服务器指定的回执接收目录，下载送盘结果；

e) 商户系统根据自身系统，处理送盘结果。

2.6.2 Webservice 送盘

一、 大批量结算申请业务处理时序图



二、 申请业务时序图说明

a) 商户系统准备送盘文件；

   1) 商户参考【大批量资金结算产品】接口.docx　并准备报文；
   2) 商户报文进行 GZIP 格式压缩
   3) 商户调用快钱提供的 PKI SDK 按照商户跟快钱签订协议中的加密策略进行加密；
   4) 商户对加密的结果进行 base64 的 encoding ,然后将结果输出到文件；

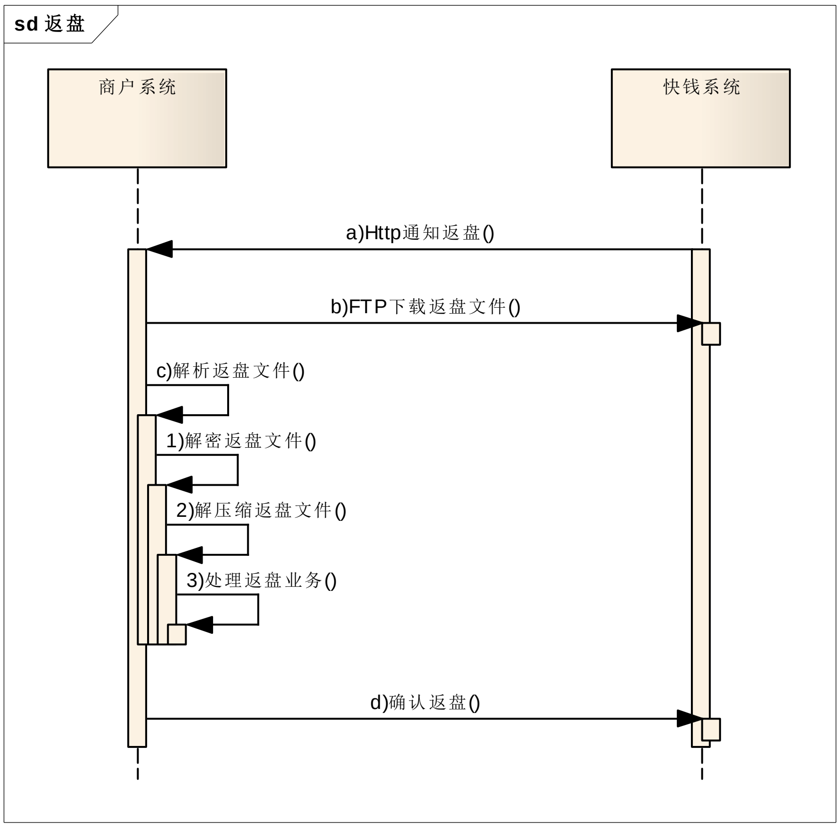
b) 商户通过 webservice 服务，发送送盘文件到快钱系统；

c) 快钱处理 webservice 送盘内容后，通过 webservice 响应返回处理结果；

d) 商户系统根据自身系统，处理送盘结果。

2.6.3 FTP 返盘

一、　返盘业务时序图



二、　返盘业务时序图说明

a) 快钱系统产生返盘文件后，通知商户接收返盘文件并告知返盘的文件名；

b) 商户根据通知指定的文件到快钱的 SFTP 服务器上下载返盘文件；

c) 商户解析下载的返盘文件

   1) 商户对返盘文件中的相关字段进行 base64 的 decoding；

   2) 商户对解压缩后的返盘文件进行解密；

   3) 商户对返盘文件进行 GZIP 格式解压缩；

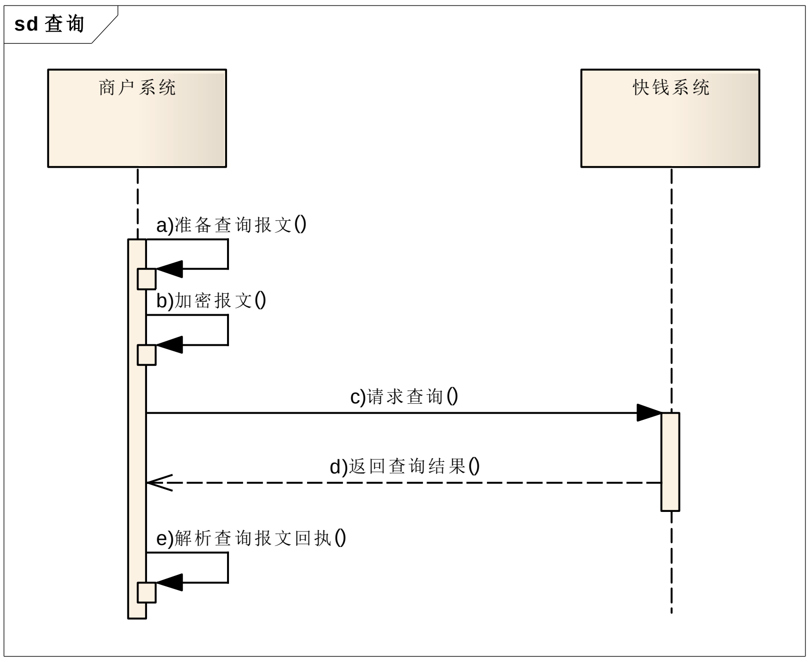
   4) 商户对解压缩后的结果做后续的业务处理；

d) 商户反馈返盘结果。

b 和 e 商户可选。当商户未选中快钱主动通知，则需要商户定时到目录中按照文件名规则来读取返盘文件。

2.6.4 Webservice 查询

一、　大批量结算查询业务时序图



二、 大批量结算查询业务时序图说明

a) 商户系统准备查询报文；
b) 商户系统调用快钱提供的 PKI SDK 按照商户跟快钱签订协议中的加密策略进行加密；
c) 商户系统发送查询请求，请求参数具体参考《【大批量资金结算产品】接口.doc》中的查询接口
d) 快钱大批量结算系统执行查询指令并生成查询回执；回执具体请参照请求参数具体参考《【大批量资金结算产品】接口.doc》查询接口的回执
e) 商户接收查询回执；

PKI 解密

GZIP 解压缩后做后续业务处理；

2.7 商户应用整合开发范例

2.7.1 送盘（FTPS 上传）

Step1:假设我们有如下数据准备：具体可参考【大批量资金结算产品】接口.docx 2.3 参数说明

根据【大批量资金结算产品】接口.doc 描述，生成 request 对象

```
BatchSettlementApplyRequest request = new BatchSettlementApplyRequest();
RequestHeader head = new RequestHeader();
Version version = new Version();
head.setTime(DateUtil.formatDateTime("yyyyMMddHHmmss", new Date()));
version.setService("fo.apipki.pay");
version.setVersion("1.0");
head.setVersion(version);
request.setRequestHeader(head);
ApplyRequestType body = new ApplyRequestType();
body.setApplyDate(uploadResult.getApplyDate());
body.setAutoRefund(uploadResult.getAutoRefund());
body.setBatchFail(uploadResult.getBatchFail());
body.setBatchNo(uploadResult.getBatchId());
body.setCheckAmtCnt(uploadResult.getCheckAmtCnt());
body.setCur(uploadResult.getCur());
body.setFeeType(uploadResult.getFeeType());
body.setMerchantMemo1(uploadResult.getMerchantMemo1());
body.setMerchantMemo2(uploadResult.getMerchantMemo2());
body.setMerchantMemo3(uploadResult.getMerchantMemo3());
body.setName(uploadResult.getName());
body.setPayerAcctCode(uploadResult.getPayerAcctCode());
body.setPhoneNoteFlag(uploadResult.getPhoneNoteFlag());
body.setRechargeType(uploadResult.getRechargeType());
body.setTotalAmt(uploadResult.getTotalAmt());
body.setTotalCnt(uploadResult.getTotalCnt());
body.setPay2bankLists(getPay2BankList(uploadResult));
request.setRequestBody(body);
return request;
```

Step2：调用 JiBX 对付款请求信息进行 XML 绑定。

```
    public String objectToXml(BatchSettlementApplyRequest request) {
        String result = "";
        try {

            IBindingFactory bfact = BindingDirectory
                    .getFactory(BatchSettlementApplyRequest.class);
            IMarshallingContext mctx = bfact.createMarshallingContext();
            mctx.setIndent(2);
            StringWriter sw = new StringWriter();
            mctx.setOutput(sw);
            mctx.marshalDocument(request);
            result = sw.toString();
            // System.out.println(result);
            return result;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
}
```

Step3: 用 Gzip 压缩 xml 原数据

```
/**
 * gzip压缩字符串
 * @param str
 * @return
 */
public static byte[] gzip(byte[] b1) {
    byte[] b = null;
    ByteArrayOutputStream bo = null;
    GZIPOutputStream gzipo = null;
    try {
        bo = new ByteArrayOutputStream();
        gzipo = new GZIPOutputStream(bo);
        gzipo.write(b1);
        gzipo.finish();
        b = bo.toByteArray();
    } catch (Exception e) {
        logger.error(e);
    } finally {
        try {
            if (gzipo != null)
                gzipo.close();
        } catch (Exception e) {
            logger.warn(e);
        }
        try {
            if (bo != null)
                bo.close();
        } catch (Exception e) {
            logger.warn(e);
        }
    }
    return b;
}
```

Step4: 使用快钱 PKI-SDK 的范例：对数据进行加密，并用 base64encode 转码加密后的文件

```
   Mpf mpf = new Mpf();
   mpf.setMemberCode(""); //填入会员编号：即使用该功能商户在快钱的标识
   mpf.setFeatureCode(""); //填入功能号：即使用本功能快钱唯一编号
ICryptoService service = CryptoServiceFactory.createCryptoService();
   sealedData = service.seal(mpf,
                 GzipUtil.gzip(originalData.getBytes(ENCODING)););
          byte[] nullbyte = {};
          byte[] byteOri = sealedData.getOriginalData() == null ? nullbyte
                 : sealedData.getOriginalData();
          byte[] byteEnc = sealedData.getEncryptedData() == null ? nullbyte
                 : sealedData.getEncryptedData();

          byte[] byteOri2 = Base64Util.encode(byteOri);
          byte[] byteEnc2 = Base64Util.encode(byteEnc);
          byte[] byteEnv2 = Base64Util.encode(byteEnv);
          byte[] byteSig2 = Base64Util.encode(byteSig);
          sealedData.setOriginalData(byteOri2);
          sealedData.setSignedData(byteSig2);
          sealedData.setEncryptedData(byteEnc2);
          sealedData.setDigitalEnvelope(byteEnv2);
```

Step5 加密文输出到文件准备上传 SFTP

```
        SettlementPkiApiRequest request = new SettlementPkiApiRequest();
        RequestHeader head = new RequestHeader();
        Version version = new Version();
        head.setTime(DateUtil.formatDateTime("yyyyMMddHHmmss", new Date()));
        version.setService("fo.apipki.pay");
        version.setVersion("1.0");
        head.setVersion(version);
        request.setRequestHeader(head);
        body.setMemberCode(uploadResult.getMemberCode());
        SealDataType sealdata = new SealDataType();
        byte[] byteOri = sealedData.getOriginalData();
        byte[] byteEnc = sealedData.getEncryptedData();
        byte[] byteEnv = sealedData.getDigitalEnvelope();
        byte[] byteSig = sealedData.getSignedData();
        sealdata.setOriginalData(new String(byteOri, ENCODING));
        sealdata.setEncryptedData(new String(byteEnc, ENCODING));
        sealdata.setDigitalEnvelope(new String(byteEnv, ENCODING));
        sealdata.setSignedData(new String(byteSig, ENCODING));
        body.setData(sealdata);
        request.setRequestBody(body);
        BufferedOutputStream io = null;
        String filename = "TO99BILL_" + uploadResult.getMemberCode() + "_"
            + DateUtil.formatDateTime("yyyyMMddHHmmss", new Date()) + "_"
            + uploadResult.getBatchId() + ".PKI";
        try {
            File name = new File(filename);
            if (name.exists()) {
                name.delete();
                name.createNewFile();
            } else {
                name.createNewFile();
            }
            filename = name.getAbsolutePath();
            String org = objectToXml(request);
            io = new BufferedOutputStream(new FileOutputStream(name));
            io.write(org.getBytes());
            return filename;
```

将 filename 文件上传至快钱 SFTP 服务器

```
/**
 * 连接sftp服务器
 * @param host 主机
 * @param port 端口
 * @param username 用户名
 * @param password 密码
 * @return
 */
public ChannelSftp connect(String host, int port,
           String username, String password) {
    ChannelSftp sftp = null;
    try {
        JSch jsch = new JSch();
        jsch.getSession(username, host, port);
        Session sshSession = jsch.getSession(username, host, port);
        System.out.println("Session created.");
        sshSession.setPassword(password);
        Properties sshConfig = new Properties();
        sshConfig.put("StrictHostKeyChecking", "no");
        sshSession.setConfig(sshConfig);
        sshSession.connect();
        System.out.println("Session connected.");
        System.out.println("Opening Channel.");
        Channel channel = sshSession.openChannel("sftp");
        channel.connect();
        sftp = (ChannelSftp) channel;
        System.out.println("Connected to " + host + ".");
    } catch (Exception e) {
    }
    return sftp;
}
```

```
/**
* 上传文件
* @param directory 上传的目录
* @param uploadFile 要上传的文件
* @param sftp
*/
public void upload(String directory, String uploadFile,
           ChannelSftp sftp) {
    try {
        sftp.cd(directory);
        File file = new File(uploadFile);
        sftp.put(new FileInputStream(file), file.getName());
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
String host = "192.168.52.71";
int port = 22;
String username = "root";
String password = "111111";
String directory = "/home/httpd/test/";
String uploadFile = filename;
connect(host, port, username, password);
upload(directory, uploadFile, sftp);
```

2.7.2 送盘（Webservice 请求）

Step1:假设我们有如下数据准备：具体可参考【大批量资金结算产品】接口.docx 2.3 参数说明

根据【大批量资金结算产品】接口.doc 描述，生成 request 对象

```
    BatchSettlementApplyRequest request = new BatchSettlementApplyRequest();
      RequestHeader head = new RequestHeader();
      Version version = new Version();
      head.setTime(DateUtil.formatDateTime("yyyyMMddHHmmss", new Date()));
      version.setService("fo.apipki.pay");
      version.setVersion("1.0");
      head.setVersion(version);
      request.setRequestHeader(head);
      ApplyRequestType body = new ApplyRequestType();
      body.setApplyDate(uploadResult.getApplyDate());
      body.setAutoRefund(uploadResult.getAutoRefund());
      body.setBatchFail(uploadResult.getBatchFail());
      body.setBatchNo(uploadResult.getBatchId());
      body.setCheckAmtCnt(uploadResult.getCheckAmtCnt());
      body.setCur(uploadResult.getCur());
      body.setFeeType(uploadResult.getFeeType());
      body.setMerchantMemo1(uploadResult.getMerchantMemo1());
      body.setMerchantMemo2(uploadResult.getMerchantMemo2());
      body.setMerchantMemo3(uploadResult.getMerchantMemo3());
      body.setName(uploadResult.getName());
      body.setPayerAcctCode(uploadResult.getPayerAcctCode());
      body.setPhoneNoteFlag(uploadResult.getPhoneNoteFlag());
      body.setRechargeType(uploadResult.getRechargeType());
      body.setTotalAmt(uploadResult.getTotalAmt());
      body.setTotalCnt(uploadResult.getTotalCnt());
      body.setPay2bankLists(getPay2BankList(uploadResult));
      request.setRequestBody(body);
      return request;
```

Step2：调用 JiBX 对付款请求信息进行 XML 绑定。

```java
    public String objectToXml(BatchSettlementApplyRequest request) {
        String result = "";
        try {

            IBindingFactory bfact = BindingDirectory
                    .getFactory(BatchSettlementApplyRequest.class);
            IMarshallingContext mctx = bfact.createMarshallingContext();
            mctx.setIndent(2);
            StringWriter sw = new StringWriter();
            mctx.setOutput(sw);
            mctx.marshalDocument(request);
            result = sw.toString();
            // System.out.println(result);
            return result;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
}
```

Step3: 用 Gzip 压缩 xml 原数据

```java
public static byte[] gzip(byte[] b1) {
    byte[] b = null;
    ByteArrayOutputStream bo = null;
    GZIPOutputStream gzipo = null;
    try {
        bo = new ByteArrayOutputStream();
        gzipo = new GZIPOutputStream(bo);
        gzipo.write(b1);
        gzipo.finish();
        b = bo.toByteArray();
    } catch (Exception e) {
        logger.error(e);
    } finally {
        try {
            if (gzipo != null)
                gzipo.close();
        } catch (Exception e) {
            logger.warn(e);
        }
        try {
            if (bo != null)
                bo.close();
        } catch (Exception e) {
            logger.warn(e);
        }
    }
    return b;
}
```

Step4: 使用快钱 PKI-SDK 的范例：对数据进行加密，并用 base64encode 转码加密后的文件

```
  Mpf mpf = new Mpf();
  mpf.setMemberCode(""); //填入会员编号：即使用该功能商户在快钱的标识
  mpf.setFeatureCode(""); //填入功能号：即使用本功能快钱唯一编号
ICryptoService service = CryptoServiceFactory.createCryptoService();
  sealedData = service.seal(mpf,
                 GzipUtil.gzip(originalData.getBytes(ENCODING)););
         byte[] nullbyte = {};
         byte[] byteOri = sealedData.getOriginalData() == null ? nullbyte
                 : sealedData.getOriginalData();
         byte[] byteEnc = sealedData.getEncryptedData() == null ? nullbyte
                 : sealedData.getEncryptedData();

         byte[] byteOri2 = Base64Util.encode(byteOri);
         byte[] byteEnc2 = Base64Util.encode(byteEnc);
         byte[] byteEnv2 = Base64Util.encode(byteEnv);
         byte[] byteSig2 = Base64Util.encode(byteSig);
         sealedData.setOriginalData(byteOri2);
         sealedData.setSignedData(byteSig2);
         sealedData.setEncryptedData(byteEnc2);
         sealedData.setDigitalEnvelope(byteEnv2);
```

step4: 对加密信息进行压缩打包并构建 PKI 申请

```
         String reslut = null;
        SettlementPkiApiResponse response = null;
        SettlementPkiApiRequest request = new SettlementPkiApiRequest();
        RequestHeader head = new RequestHeader();
        Version version = new Version();
        head.setTime(DateUtil.formatDateTime("yyyyMMddHHmmss", new Date()));
        version.setService("fo.apipki.query");
        version.setVersion("1.0.1");
        head.setVersion(version);
        request.setRequestHeader(head);

        SettlementPkiRequestType body = new SettlementPkiRequestType();
        try {
            body.setMemberCode(uploadResult.getMemberCode());
            SealDataType sealdata = new SealDataType();
            byte[] byteOri = sealedData.getOriginalData();
            byte[] byteEnc = sealedData.getEncryptedData();
            byte[] byteEnv = sealedData.getDigitalEnvelope();
            byte[] byteSig = sealedData.getSignedData();

            sealdata.setOriginalData(new String(byteOri, ENCODING));
            sealdata.setEncryptedData(new String(byteEnc, ENCODING));
            sealdata.setDigitalEnvelope(new String(byteEnv, ENCODING));
            sealdata.setSignedData(new String(byteSig, ENCODING));

            body.setData(sealdata);
            request.setRequestBody(body);
            response = foApiPkiClient.doit(request);
```

step5：调用服务客户端处理请求并返回处理结果

```
    public SettlementPkiApiResponse doit(SettlementPkiApiRequest request) {
        try {
            Object obj=sealWebServiceTemplate.marshalSendAndReceive(request);
            SettlementPkiApiResponse response = (SettlementPkiApiResponse) obj;
            return response;
        } catch (Throwable t) {
            logger.error(null, t);
            return null;
        }
```

step6: 对返回结果进行 base64decode

```
    originalData = this.objectToXml(this.genRequest(uploadResult));
    Mpf mpf = new Mpf();
    mpf.setMemberCode(uploadResult.getMemberCode());
    mpf.setFeatureCode(uploadResult.getFeatureCode());
    ICryptoService service = CryptoServiceFactory.createCryptoService();
    sealedData=service.seal(mpf,GzipUtil.gzip(originalData.getBytes(ENCODING
));
            byte[] nullbyte = {};
            byte[] byteOri = sealedData.getOriginalData() == null ? nullbyte
                    : sealedData.getOriginalData();
            byte[] byteEnc = sealedData.getEncryptedData() == null ? nullbyte
                    : sealedData.getEncryptedData();
            byte[] byteEnv = sealedData.getDigitalEnvelope() == null ? nullbyte
                    : sealedData.getDigitalEnvelope();
            byte[] byteSig = sealedData.getSignedData() == null ? nullbyte
                    : sealedData.getSignedData();
            byte[] byteOri2 = Base64Util.encode(byteOri);
            byte[] byteEnc2 = Base64Util.encode(byteEnc);
            byte[] byteEnv2 = Base64Util.encode(byteEnv);
            byte[] byteSig2 = Base64Util.encode(byteSig);
            sealedData.setOriginalData(byteOri2);
            sealedData.setSignedData(byteSig2);
            sealedData.setEncryptedData(byteEnc2);
            sealedData.setDigitalEnvelope(byteEnv2);
```

step7:对解压结果进行解密验签并解压缩

```
    SealedData sealedData = new SealedData();
    sealedData.setSignedData(resDecodeSigData);
    sealedData.setOriginalData(resDecodeOriData);
    sealedData.setEncryptedData(resDecodeEncData);
    sealedData.setDigitalEnvelope(resDecodeEnvData);
```

```
    Mpf mpf = new Mpf();
    mpf.setMemberCode(uploadResult.getMemberCode());
    mpf.setFeatureCode(uploadResult.getFeatureCode());
    ICryptoService service = CryptoServiceFactory.createCryptoService();
    UnsealedData unsealedData = service.unseal(mpf, sealedData);

     if (unsealedData.getVerifySignResult()) {
        byte[] DecryptedData = unsealedData.getDecryptedData();
                    if (DecryptedData == null)
                        unsealedResult = new String(new String(GzipUtil
                                .unBGzip(resDecodeOriData), ENCODING));
                    else {
        byte[] unsealedResultbyte = GzipUtil.unBGzip(DecryptedData);
}
}
```

2.7.3 按批次查询申请

Step 1: 构建批量查询申请

```java
    public BatchidQueryRequest getQueryRequest(MassOutUploadResult uploadResult)
{
        BatchidQueryRequest request = new BatchidQueryRequest();
        RequestHeader head = new RequestHeader();
        Version version = new Version();
        head.setTime(DateUtil.formatDateTime("yyyyMMddHHmmss", new Date()));
        version.setService("fo.apipki.query");
        version.setVersion("1.0");
        head.setVersion(version);
        request.setRequestHeader(head);
        ApiQueryRequestType body = new ApiQueryRequestType();
        body.setBatchNo(uploadResult.getBatchId());
        body.setListFlag(uploadResult.getListFlag());
        body.setPage(uploadResult.getPage());
        body.setPageSize(uploadResult.getPageSize());
        request.setRequestBody(body);
        return request;
    }
```

Step2：调用 JiBX 对付款请求信息进行 XML 绑定。

```
    public String queryobjectToXml(BatchidQueryRequest request) {
        String result = "";
        try {

            IBindingFactory bfact = BindingDirectory
                    .getFactory(BatchidQueryRequest.class);
            IMarshallingContext mctx = bfact.createMarshallingContext();
            mctx.setIndent(2);
            StringWriter sw = new StringWriter();
            mctx.setOutput(sw);
            mctx.marshalDocument(request);
            result = sw.toString();
            return result;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
```

step3: 对原文压缩并调用 PKI 对报文加密同时进行 base64encode 转码

```
    originalData = this.objectToXml(this.genRequest(uploadResult));
    Mpf mpf = new Mpf();
    mpf.setMemberCode(uploadResult.getMemberCode());
    mpf.setFeatureCode(uploadResult.getFeatureCode());
    ICryptoService service = CryptoServiceFactory.createCryptoService();
    sealedData=service.seal(mpf,GzipUtil.gzip(originalData.getBytes(ENCODING
));
            byte[] nullbyte = {};
            byte[] byteOri = sealedData.getOriginalData() == null ? nullbyte
                    : sealedData.getOriginalData();
            byte[] byteEnc = sealedData.getEncryptedData() == null ? nullbyte
                    : sealedData.getEncryptedData();
            byte[] byteEnv = sealedData.getDigitalEnvelope() == null ? nullbyte
                    : sealedData.getDigitalEnvelope();
            byte[] byteSig = sealedData.getSignedData() == null ? nullbyte
                    : sealedData.getSignedData();
            byte[] byteOri2 = Base64Util.encode(byteOri);
            byte[] byteEnc2 = Base64Util.encode(byteEnc);
            byte[] byteEnv2 = Base64Util.encode(byteEnv);
            byte[] byteSig2 = Base64Util.encode(byteSig);
            sealedData.setOriginalData(byteOri2);
            sealedData.setSignedData(byteSig2);
            sealedData.setEncryptedData(byteEnc2);
            sealedData.setDigitalEnvelope(byteEnv2);
```

step4: 对加密信息进行压缩打包并构建批量付款查询

```
         String reslut = null;
        SettlementPkiApiResponse response = null;
        SettlementPkiApiRequest request = new SettlementPkiApiRequest();
        RequestHeader head = new RequestHeader();
        Version version = new Version();
        head.setTime(DateUtil.formatDateTime("yyyyMMddHHmmss", new Date()));
        version.setService("fo.apipki.query");
        version.setVersion("1.0.1");
        head.setVersion(version);
        request.setRequestHeader(head);

        SettlementPkiRequestType body = new SettlementPkiRequestType();
        try {
            body.setMemberCode(uploadResult.getMemberCode());
            SealDataType sealdata = new SealDataType();
            byte[] byteOri = sealedData.getOriginalData();
            byte[] byteEnc = sealedData.getEncryptedData();
            byte[] byteEnv = sealedData.getDigitalEnvelope();
            byte[] byteSig = sealedData.getSignedData();

            sealdata.setOriginalData(new String(byteOri, ENCODING));
            sealdata.setEncryptedData(new String(byteEnc, ENCODING));
            sealdata.setDigitalEnvelope(new String(byteEnv, ENCODING));
            sealdata.setSignedData(new String(byteSig, ENCODING));

            body.setData(sealdata);
            request.setRequestBody(body);

           response = foApiPkiClient.doit(request);
```

step5: 调用批量付款 API 服务客户端处理请求并返回处理结果

```
    public SettlementPkiApiResponse doit(SettlementPkiApiRequest request) {
        try {
            Object obj=sealWebServiceTemplate.marshalSendAndReceive(request);
            SettlementPkiApiResponse response = (SettlementPkiApiResponse) obj;
            return response;
        } catch (Throwable t) {
            logger.error(null, t);
            return null;
        }
```

step6: 对返回结果进行 base64decode

```
     SettlementPkiRequestType responsebody = response.getRequestBody();
    SealDataType responseSealedData = responsebody.getData();
    result=responsebody.getErrorMsg()+":"+responsebody.getErrorCode();
    byte[] resOriData =
            responseSealedData.getOriginalData().getBytes(ENCODING);
    byte[] resSigData =
            responseSealedData.getSignedData().getBytes(ENCODING);
    byte[] resEnvData =
            responseSealedData.getDigitalEnvelope().getBytes(ENCODING);
    byte[] resEncData =
            responseSealedData.getEncryptedData().getBytes(ENCODING);

          // decode
                byte[] resDecodeOriData = Base64Util.decode(resOriData);
                byte[] resDecodeSigData = Base64Util.decode(resSigData);
                byte[] resDecodeEnvData = Base64Util.decode(resEnvData);
                byte[] resDecodeEncData = Base64Util.decode(resEncData);
```

step7:对解压结果进行解密验签并解压缩

```
    SealedData sealedData = new SealedData();
    sealedData.setSignedData(resDecodeSigData);
    sealedData.setOriginalData(resDecodeOriData);
    sealedData.setEncryptedData(resDecodeEncData);
    sealedData.setDigitalEnvelope(resDecodeEnvData);
    Mpf mpf = new Mpf();
    mpf.setMemberCode(uploadResult.getMemberCode());
    mpf.setFeatureCode(uploadResult.getFeatureCode());
    ICryptoService service = CryptoServiceFactory.createCryptoService();
    UnsealedData unsealedData = service.unseal(mpf, sealedData);

     if (unsealedData.getVerifySignResult()) {
        byte[] DecryptedData = unsealedData.getDecryptedData();
                    if (DecryptedData == null)
                        unsealedResult = new String(new String(GzipUtil
                                .unBGzip(resDecodeOriData), ENCODING));
                    else {
        byte[] unsealedResultbyte = GzipUtil.unBGzip(DecryptedData);
}
}
```

2.7.4 组合查询申请

Step 1: 构建批量查询申请

```
    public ComplexQueryRequest getTradeRequest(MassOutTrade uploadResult) {
        ComplexQueryRequest request = new ComplexQueryRequest();
        RequestHeader head = new RequestHeader();
        Version version = new Version();
        head.setTime(DateUtil.formatDateTime("yyyyMMddHHmmss", new Date()));
        version.setService("fo.apipki.trade");
        version.setVersion("1.0");
        head.setVersion(version);
        request.setRequestHeader(head);
        ComplexQueryRequestType body = new ComplexQueryRequestType();
        body.setBankCardNo(uploadResult.getBankCardNo());
        body.setBeginApplyTime(uploadResult.getBeginApplyDate());
        body.setBranchBank(uploadResult.getBranchBank());
        body.setCity(uploadResult.getCity());
        body.setBank(uploadResult.getBankName());
        body.setEndApplyTime(uploadResult.getEndApplyDate());
        body.setMerchantId(uploadResult.getMemberCode());
        body.setProvince(uploadResult.getProvince());
        body.setName(uploadResult.getName());
        body.setOrderBankErrorCode(uploadResult.getOrderBankErrorCode());
        body.setOrderErrorCode(uploadResult.getOrderErrorCode());
        body.setOrderStatus(uploadResult.getOrderStatus());
        body.setPayeeType(uploadResult.getPayeeType());
        body.setPage(uploadResult.getPage());
        body.setPageSize(uploadResult.getPageSize());
        request.setRequestBody(body);
        return request;
    }
}
}
```

Step2：调用 JiBX 对付款请求信息进行 XML 绑定。

```
    /**
     * ComplexQueryRequest object to xml string
     *
     * @param request
     * @return
     */
    public String tradeobjectToXml(ComplexQueryRequest request) {
        String result = "";
        try {

            IBindingFactory bfact = BindingDirectory
                    .getFactory(ComplexQueryRequest.class);
            IMarshallingContext mctx = bfact.createMarshallingContext();
            mctx.setIndent(2);
            StringWriter sw = new StringWriter();
            mctx.setOutput(sw);
            mctx.marshalDocument(request);
            result = sw.toString();
            // System.out.println(result);
            return result;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
```

step3: 对原文压缩并调用 PKI 对报文加密同时进行 base64encode 转码

```
    originalData = this.tradeobjectToXml(this
                   .getTradeRequest(uploadResult));
    Mpf mpf = new Mpf();
    mpf.setMemberCode(uploadResult.getMemberCode());
    mpf.setFeatureCode(uploadResult.getFeatureCode());
    ICryptoService service = CryptoServiceFactory.createCryptoService();
    sealedData=service.seal(mpf,GzipUtil.gzip(originalData.getBytes(ENCODING
));
           byte[] nullbyte = {};
           byte[] byteOri = sealedData.getOriginalData() == null ? nullbyte
                   : sealedData.getOriginalData();
           byte[] byteEnc = sealedData.getEncryptedData() == null ? nullbyte
                   : sealedData.getEncryptedData();
           byte[] byteEnv = sealedData.getDigitalEnvelope() == null ? nullbyte
                   : sealedData.getDigitalEnvelope();
           byte[] byteSig = sealedData.getSignedData() == null ? nullbyte
                   : sealedData.getSignedData();
           byte[] byteOri2 = Base64Util.encode(byteOri);
           byte[] byteEnc2 = Base64Util.encode(byteEnc);
           byte[] byteEnv2 = Base64Util.encode(byteEnv);
           byte[] byteSig2 = Base64Util.encode(byteSig);
           sealedData.setOriginalData(byteOri2);
           sealedData.setSignedData(byteSig2);
           sealedData.setEncryptedData(byteEnc2);
           sealedData.setDigitalEnvelope(byteEnv2);
```

step4: 对加密信息进行压缩打包并构建组合付款查询

```
         String reslut = null;
        SettlementPkiApiResponse response = null;
        SettlementPkiApiRequest request = new SettlementPkiApiRequest();
        RequestHeader head = new RequestHeader();
        Version version = new Version();
        head.setTime(DateUtil.formatDateTime("yyyyMMddHHmmss", new Date()));
        version.setService("fo.apipki.query");
        version.setVersion("1.0.1");
        head.setVersion(version);
        request.setRequestHeader(head);

        SettlementPkiRequestType body = new SettlementPkiRequestType();
        try {
            body.setMemberCode(uploadResult.getMemberCode());
            SealDataType sealdata = new SealDataType();
            byte[] byteOri = sealedData.getOriginalData();
            byte[] byteEnc = sealedData.getEncryptedData();
            byte[] byteEnv = sealedData.getDigitalEnvelope();
            byte[] byteSig = sealedData.getSignedData();

            sealdata.setOriginalData(new String(byteOri, ENCODING));
            sealdata.setEncryptedData(new String(byteEnc, ENCODING));
            sealdata.setDigitalEnvelope(new String(byteEnv, ENCODING));
            sealdata.setSignedData(new String(byteSig, ENCODING));

            body.setData(sealdata);
            request.setRequestBody(body);
            response = foApiPkiClient.doit(request);
```

step5: 调用组合付款 API 服务客户端处理请求并返回处理结果

```
    public SettlementPkiApiResponse doit(SettlementPkiApiRequest request) {
        try {
            Object obj=sealWebServiceTemplate.marshalSendAndReceive(request);
            SettlementPkiApiResponse response = (SettlementPkiApiResponse) obj;
            return response;
        } catch (Throwable t) {
            logger.error(null, t);
            return null;
        }
```

step6: 对返回结果进行 base64decode

```
     SettlementPkiRequestType responsebody = response.getRequestBody();
    SealDataType responseSealedData = responsebody.getData();
    result=responsebody.getErrorMsg()+":"+responsebody.getErrorCode();
    byte[] resOriData =
            responseSealedData.getOriginalData().getBytes(ENCODING);
    byte[] resSigData =
            responseSealedData.getSignedData().getBytes(ENCODING);
    byte[] resEnvData =
            responseSealedData.getDigitalEnvelope().getBytes(ENCODING);
    byte[] resEncData =
            responseSealedData.getEncryptedData().getBytes(ENCODING);

          // decode
              byte[] resDecodeOriData = Base64Util.decode(resOriData);
              byte[] resDecodeSigData = Base64Util.decode(resSigData);
              byte[] resDecodeEnvData = Base64Util.decode(resEnvData);
              byte[] resDecodeEncData = Base64Util.decode(resEncData);
```

step7:对解压结果进行解密验签并解压缩

```
    SealedData sealedData = new SealedData();
    sealedData.setSignedData(resDecodeSigData);
    sealedData.setOriginalData(resDecodeOriData);
    sealedData.setEncryptedData(resDecodeEncData);
    sealedData.setDigitalEnvelope(resDecodeEnvData);
```

```
    Mpf mpf = new Mpf();
    mpf.setMemberCode(uploadResult.getMemberCode());
    mpf.setFeatureCode(uploadResult.getFeatureCode());
    ICryptoService service = CryptoServiceFactory.createCryptoService();
    UnsealedData unsealedData = service.unseal(mpf, sealedData);

     if (unsealedData.getVerifySignResult()) {
        byte[] DecryptedData = unsealedData.getDecryptedData();
                    if (DecryptedData == null)
                        unsealedResult = new String(new String(GzipUtil
                                .unBGzip(resDecodeOriData), ENCODING));
                    else {
        byte[] unsealedResultbyte = GzipUtil.unBGzip(DecryptedData);
}
}
```

2.7.5 返盘

Step1 :商户接收到快钱通知。

```
    String version = request.getParameter("Version");
    String fileName = request.getParameter("fileName");
```

Step 2:商户从快钱通知中给定的文件名称到快钱 SFTP 上下载文件

```
public InputStream downloadFile(final String remoteDir,
            final String remoteFile, final String localfile) throws Exception {
        this.execute(new JschClientCallBack() {
            public void processFTP(Channel channel) throws Exception {
                ChannelSftp sftp = (ChannelSftp) channel;
                sftp.cd(remoteDir);
                sftp.get(localfile, new BufferedOutputStream(
                        new FileOutputStream(remoteFile)));
            }
        });
        return new BufferedInputStream(new FileInputStream(localfile));
    }
```

Step3 ： 商 户 拿 到 文 件 ， 读 取 文 件 流 准 备 对 文 件 进 行 解 密

```
public static SettlementPkiApiResponse queryXmlToObject(InputStream input) {
        try {

            IBindingFactory bfact = BindingDirectory
                    .getFactory(SettlementPkiApiResponse.class);
            IUnmarshallingContext uctx = bfact.createUnmarshallingContext();
            SettlementPkiApiResponse response = (SettlementPkiApiResponse) uctx
                    .unmarshalDocument(input, null);
            return response;
        } catch (Exception e) {
            return null;
        }
    }
```

Step4: 商户对解密结果进行 base64decode 转码，解密验签并解压，具体参考 2.5.2 step6 ，step7

Step 6: 商户将 membercode 加密

```java
public SealedData sealMembercode(String memberCode, String featureCode) {
        SealedData sealedData = null;
        Mpf mpf = new Mpf();
        mpf.setMemberCode(memberCode);
        mpf.setFeatureCode(featureCode);
        ICryptoService service;
        try {
            service = CryptoServiceFactory.createCryptoService();
            sealedData = service.seal(mpf, memberCode.getBytes());
            byte[] nullbyte = {};
            byte[] byteOri = sealedData.getOriginalData() == null ? nullbyte
                    : sealedData.getOriginalData();
            byte[] byteEnc = sealedData.getEncryptedData() == null ? nullbyte
                    : sealedData.getEncryptedData();
            byte[] byteEnv = sealedData.getDigitalEnvelope() == null ? nullbyte
                    : sealedData.getDigitalEnvelope();
            byte[] byteSig = sealedData.getSignedData() == null ? nullbyte
                    : sealedData.getSignedData();
            byte[] byteOri2 = Base64Util.encode(byteOri);
            byte[] byteEnc2 = Base64Util.encode(byteEnc);
            byte[] byteEnv2 = Base64Util.encode(byteEnv);
            byte[] byteSig2 = Base64Util.encode(byteSig);
            sealedData.setOriginalData(byteOri2);
            sealedData.setSignedData(byteSig2);
            sealedData.setEncryptedData(byteEnc2);
            sealedData.setDigitalEnvelope(byteEnv2);
            return sealedData;
        } catch (CryptoException e) {
            e.printStackTrace();
            return  null;
        }
    }
```

Step 6: 商户将结果反馈给快钱；

```
public void SendMessage(SealedData sealedData ,String file ,String batchNo,String
vesion, String status , String memberCode) {
        try {
            HttpClient client = new HttpClient();
            PostMethod authpost = new PostMethod("/servlet/applyPaymentServlet");
            NameValuePair s = new NameValuePair("status",status);
            NameValuePair v = new NameValuePair("vesion",vesion);
            NameValuePair batchid = new NameValuePair("batchNo",batchNo);
            NameValuePair filename = new NameValuePair("filename ", file);
            NameValuePair mc= new NameValuePair("memberCode",
                    memberCode);
            NameValuePair signedMemberCode = new NameValuePair(
                    "signedMemberCode ", new String(Base64Util
                            .decode(sealedData.getSignedData()),"utf-8"));
            NameValuePair encryptedMemberCode = new NameValuePair(
                    "encryptedMemberCode ", new String(Base64Util
                            .decode(sealedData.getEncryptedData()),"utf-8"));
            NameValuePair digitalEnvelope = new NameValuePair(
                    "digitalEnvelope", new String(Base64Util.decode(sealedData
                            .getDigitalEnvelope()),"utf-8"));
            authpost.setRequestBody(new NameValuePair[] {s,v,batchid,mc ,filename,
                    signedMemberCode, encryptedMemberCode, digitalEnvelope});
            client.executeMethod(authpost);
        } catch (HttpException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

2.7.4 依赖列表

商户必选的包（快钱提供）			
名称	文件名/压缩	版本	发布日期
ASAP Crypto	if-crypto-sdk.jar	3.3.2	

SDK	if-jibx-schema-asap.jar	1.0	
	if-jibx-schema-commons.jar	1.0.1	
大批量资金解决产品报文相关	fo-jibx-api.jar	1.0.1	
	fo-jibx-commons.jar	1.0.1	
商户使用 Spring+JIBX Webservice 的第三方包（可根据商户端情况变更版本号）			
Commons	commons-beanutils.jar	1.7.0	
	commons-codec.jar	1.3	
	commons-collections.jar	3.2.1	
	commons-io.jar	1.2	
	commons-lang.jar	2.4	
	commons-logging.jar	1.1.1	
	commons-codes	1.3	
	commons-httpclient	1.3	
Log4j	log4j.jar	1.2.13	
Jsch	Jsch.jar	0.1.42	
Slf4j	slf4j-api.jar	1.4.3	
	slf4j-jcl.jar	1.4.3	
JIBX	jibx-run.jar	1.2	
Spring	spring-oxm.jar	1.5.6	
	spring.jar	2.5.5	
	spring-oxm-tiger.jar	1.5.6	

	spring-ws-core.jar	1.5.6	
	spring-ws-core-tiger.jar	1.5.6	
	spring-xml.jar	1.5.6	
Others	Servlet-api.jar	2.4	
	jaxbapi.jar	2.1.7	
	jxl.jar	2.6.2	
	saaj-api.jar	1.3	
	saaj-impl.jar	1.3.2	
	standard.jar	1.1	
	stax-api.jar	1.0.1	
	wstx-asl.jar	3.2.0	

注：若商户使用其他 webservice 客户端，则可选择其他的第三方包

2.8 开发提示

2.8.1 PKI 加密，解密，验签

目前快钱可支持包括签名[Key]，签名[证书]，签名-加密[证书]在内的加密策略和支持多做加密算法，以求最大限度保证商户提交数据的安全性。

2.8.2 通知付款结果

在本产品中，商户提交请求时即与快钱服务器端建立会话，服务器查询到符合条件的记录后，会即时将付款结果生成返盘文件并放置在快钱 SFTP 服务器中，

返盘文件名会即时返回到商户。商户可以在接收到回应后从回应中指定的文件名去 SFTP 服务器上获取返盘文件。

2.8.3 商户对批量付款订单的提交

a) 目前快钱仅支持批量付款到银行交易请求。查询仅支持基于批次号，交易号，保单号，批次号，起讫时间的查询。

3. 参考资料

3.1 常见问题

3.1.1 http 404

请确认 web service 的 URL 是否配置正确，或者通过访问 https://www.99bill.com/fo-batch-settlement/services/batchSettlement.wsdl 确认快钱的批量付款服务是否可用。

3.1.2 http 500

报文在加密之前需转化为 byte[]格式，因此请注意一定使用 UTF-8 encoding.如 originalData.getBytes("utf-8")

3.1.3 content not allowed in prolog exception

报文压缩后出现了在网络传输过程未可见的字符，因此建议传输之前用基于base64做encoding 动作。如 **byte**[] bytes01 = GzipUtil.*gzip*(b);

```
byte[] bytes02 = Base64Util.encode(bytes01);
```

4 附录

4.1 版权说明

此文档的版权归上海快钱信息服务有限公司所有，作为本系统的最终用户，可以拥有该份文档的使用权，但未征得上海快钱信息服务有限公司的书面批准，不得修改、公布本文档，不得向第三方借阅、出让、出版本文档。

4.2 参考资料

1. ASAP 应用程序开发包操作手册_v1.1.pdf
2. ASAP-OpenSSL 证书生成手册. pdf
3. 99Bill-PMD-L463-Application Security Authentication Platform. pdf
4. 【大批量资金结算产品】错误代码对照表. pdf
5. 【大批量资金结算产品】接口. pdf
6. 【大批量资金结算产品】省份城市列表. pdf
7. 【大批量资金结算产品】银行列表. pdf

4.3 快钱资源

快钱网站：http://www.99bill.com

快钱帮助中心：http://help.99bill.com

****

如果您对本文档及快钱有任何意见或建议，请发送邮件至 support@99bill.com

快钱衷心感谢您的支持！