payby-java SDK

# Description

This guide is written for application developers who want to integrate Payby payment solution

# 2 Pre-Condition

## 2.1 JDK

JDK 8+

Executing validation commands on the command line java -version, return similar content

java version "1.8.0\_191"

Java(TM) SE Runtime Environment (build 1.8.0\_191-b12)

Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)

## 2.2 Maven

Maven 3.0

Executing validation commands on the command line mvn –version, return similar content

Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5ffb20918da4f719f3; 2018-10-25T02:41:47+08:00)

Maven home: D:\apache-maven-3.6.0\bin\..

Java version: 1.8.0\_191, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0\_191\jre

Default locale: zh\_CN, platform encoding: GBK

OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"

## Import maven repository

Import local library or remote library, choose by yourself according to the user's R & D environment

### 2.3.1 Download dependency

git clone <https://github.com/PayBy/PayBy-java.git>

Open download directory: PayBy-java/dependency

### 2.3.2 Install local repository

Move to ‘PayBy-java/dependency’ subdirectory

mvn install:install-file -Dfile=payby-api-1.1.jar -DpomFile=payby-api-1.1.pom

### 2.3.3 Deploy remote repository

mvn deploy:deploy-file -Durl=company maven repository url path -DrepositoryId=repository name -Dfile=payby-api-1.1.jar -DpomFile=payby-api-1.1.pom

## 2.4 Project import maven dependency

<dependency>

<groupId>com.payby</groupId>

<artifactId>payby-api</artifactId>

<version>1.1</version>

</dependency>

### 2.4.1 Sdk cascade depends on specific content

The specific content varies according to the actual version, execute at project root

mvn dependency:tree

Get results:

com.payby:payby-api:jar:1.1

+- commons-io:commons-io:jar:2.4:compile

+- commons-codec:commons-codec:jar:1.13:compile

+- org.projectlombok:lombok:jar:1.18.8:provided

+- org.bouncycastle:bcprov-jdk15on:jar:1.64:compile

+- org.bouncycastle:bcpkix-jdk15on:jar:1.64:compile

+- com.alibaba:fastjson:jar:1.2.58:compile

+- org.slf4j:slf4j-api:jar:1.7.26:compile

+- org.apache.commons:commons-lang3:jar:3.9:compile

+- com.squareup.okhttp3:okhttp:jar:3.11.0:compile

| \- com.squareup.okio:okio:jar:1.14.0:compile

+- org.apache.httpcomponents:httpclient:jar:4.5.6:compile

| +- org.apache.httpcomponents:httpcore:jar:4.4.10:compile

| \- commons-logging:commons-logging:jar:1.2:compile

\- com.madgag.spongycastle:bcpkix-jdk15on:jar:1.58.0.0:compile

+- com.madgag.spongycastle:core:jar:1.58.0.0:compile

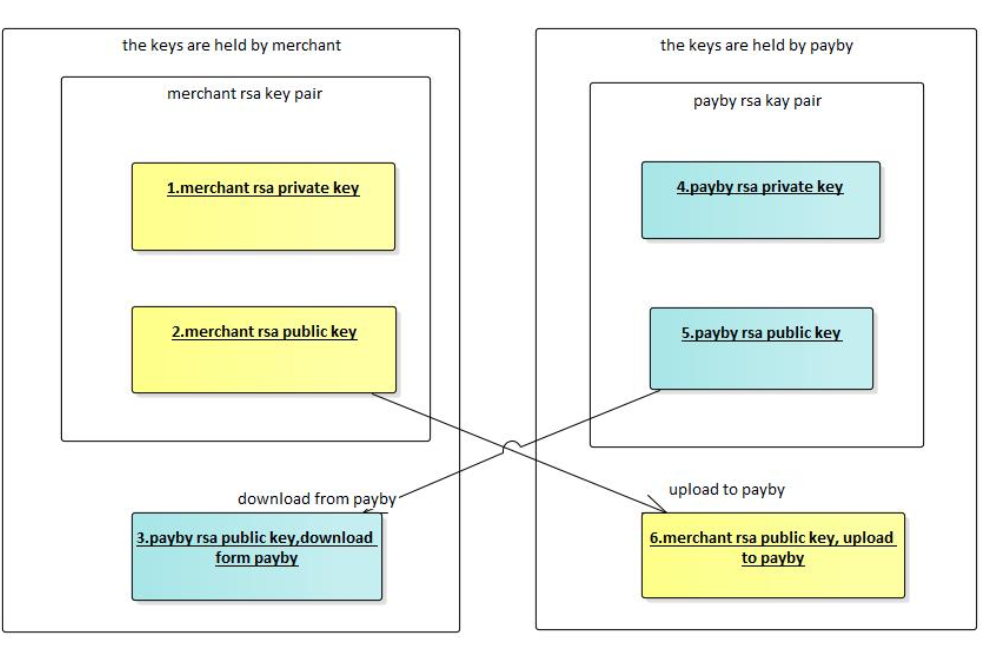
\- com.madgag.spongycastle:prov:jar:1.58.0.0:compile

\- junit:junit:jar:4.12:compile

\- org.hamcrest:hamcrest-core:jar:1.3:compile

# Digital signature

## Preparing keys



Recommended operation for private key generation

### Generate private key  
# PayBy\_key.pem Private key file name  
# 2048 Private key size, at least 2048  
openssl genrsa -out PayBy\_key.pem 2048  
​  
### Export public key  
# PayBy\_key.pem Private key generated in the previous step # PayBy\_key\_public.pem Exported public key name  
openssl rsa -in PayBy\_key.pem -out PayBy\_key\_public.pem -pubout  
​  
### Export private key for Java  
# PayBy\_key.pem Private key generated in the 1st step  
# PayBy\_key\_Private.pem  
openssl pkcs8 -in PayBy\_key.pem -topk8 -nocrypt -out PayBy\_key\_private.pem  
​

## 3.2 Signature algorithm

1. Using SHA256WithRSA for signature algorithm. The private key is issued by the merchant itself.
2. Original signature rule: Requesting the original content of the body.
3. Using UTF-8 to encode the original content.
4. Using Base64 to encode the resulting signature.

## 3.3 Encryption Algorithm

1. The encryption algorithm uses RSA public-key encryption, and the public-key is issued by PayBy.
2. The encrypted field should not be too large, generally any more than 200 bytes.
3. Encryption rules: RSA (encrypted original encryption).
4. UTF-8 encoding is used for plaintext.
5. The encryption result is encoded with Base64.

## Verify Signature Algorithm

1. Using SHA256WithRSA to verify the signature algorithm. The RSA public-key is downloaded from the payby merchant console.
2. Using Base64 to decode the signature, ie. decoded\_sign\_data.
3. Using UTF-8 to decode the original content, ie. decoded\_content\_data
4. verify signature using parameters as listed (rsa\_public\_key, decoded\_sign\_data, decoded\_content\_data).

# API description

## Function description

### 4.1.1 HttpClient

ApiConfig apiConfig = **new** ApiConfig();

// setting interface url

apiConfig.setDomain("https://sim.test2pay.com/sgs/api");

// setting merchant pkcs8 privateKey path

String merchantPrivateKey = **new** String(Files.*readAllBytes*(

Paths.*get*(PayByDemo.**class**.getClassLoader().getResource("merchant\_demo\_private.pem").toURI())));

// setting payby publicKey path

String payByPubKey = **new** String(Files

.*readAllBytes*(Paths.*get*(PayByDemo.**class**.getClassLoader().getResource("payby\_public\_key.pem").toURI())));

apiConfig.setCert(**new** KeyCert(merchantPrivateKey, payByPubKey));

// setting http header params

apiConfig.setFixHeaders(getFixHeaders());

ClientConfig config = **new** OkHttpClientConfig.Builder()

.interceptor(**new** OkHttpClientConfig.SignInterceptor(apiConfig.getCert())).apiConfig(apiConfig).build();

HttpClient client = **new** HttpClient(config);

List<Pair<String, String>> getFixHeaders() {

List<Pair<String, String>> pairs = **new** ArrayList<>();

// setting Partner-Id Required

pairs.add(**new** ImmutablePair<>("partner-id", "200000042607"));

// setting group-name Optional

pairs.add(**new** ImmutablePair<>("group-name", ""));

// setting branch-name Optional

pairs.add(**new** ImmutablePair<>("branch-name", ""));

**return** pairs;

}

### Order creation

Map<String, Object> wrap = **new** HashMap<>();

// Request time Required

wrap.put("requestTime", System.*currentTimeMillis*());

Map<String, Object> req = **new** HashMap<>();

// Merchant order number Required

req.put("merchantOrderNo", "M202005120001");

// Product name Required

req.put("subject", "Ipad");

Map<String, Object> amount = **new** HashMap<>();

// Order currency Required

amount.put("currency", "AED");

// Order amount Required

amount.put("amount", **new** BigDecimal("0.1"));

req.put("totalAmount", amount);

// Payment scenario code Required

req.put("paySceneCode", "DYNQR");

// Notification URL Optional

req.put("notifyUrl", "http://yoursite.com/api/notification");

// Accessory content Optional

Map<String, Object> accessoryContent = **new** HashMap<>();

// Amount detail Optional

Map<String, Object> amountDetail = **new** HashMap<>();

Map<String, Object> vatAmount = **new** HashMap<>();

vatAmount.put("currency", "AED");

vatAmount.put("amount", **new** BigDecimal("0.1"));

// Vat amount Optional

amountDetail.put("vatAmount", vatAmount);

// Goods detail Optional

Map<String, Object> goodsDetail = **new** HashMap<>();

goodsDetail.put("body", "Gifts");

goodsDetail.put("goodsName", "candy flower");

goodsDetail.put("goodsId", "GI1005");

// Terminal detail Optional

Map<String, Object> terminalDetail = **new** HashMap<>();

terminalDetail.put("merchantName", "candy home");

accessoryContent.put("amountDetail", amountDetail);

accessoryContent.put("goodsDetail", goodsDetail);

accessoryContent.put("terminalDetail", terminalDetail);

req.put("accessoryContent", accessoryContent);

wrap.put("bizContent", req);

System.***out***.println(JSON.*toJSONString*(wrap));

HttpRequest request = **new** HttpRequest.Builder().api("/acquire2/placeOrder").body(JSON.*toJSONBytes*(wrap)).build();

String response = client.execute(request);

System.***out***.println(response);

### Order cancellation

Map<String, Object> wrap = **new** HashMap<>();

// Request time Required

wrap.put("requestTime", System.*currentTimeMillis*());

Map<String, Object> req = **new** HashMap<>();

// Merchant order number Required

req.put("merchantOrderNo", "M202005120001");

wrap.put("bizContent", req);

System.***out***.println(JSON.*toJSONString*(wrap));

HttpRequest request =

**new** HttpRequest.Builder().api("/acquire2/cancelOrder").body(JSON.*toJSONBytes*(wrap)).build();

String response = client.execute(request);

System.***out***.println(response);

### Order query

Map<String, Object> wrap = **new** HashMap<>();

// Request time Required

wrap.put("requestTime", System.*currentTimeMillis*());

Map<String, Object> req = **new** HashMap<>();

// Merchant order number Required

req.put("merchantOrderNo", "M202005120001");

wrap.put("bizContent", req);

System.***out***.println(JSON.*toJSONString*(wrap));

HttpRequest request =

**new** HttpRequest.Builder().api("/acquire2/getOrder").body(JSON.*toJSONBytes*(wrap)).build();

String response = client.execute(request);

System.***out***.println(response);

### Order refund

Map<String, Object> wrap = **new** HashMap<>();

// Request time Required

wrap.put("requestTime", System.*currentTimeMillis*());

Map<String, Object> req = **new** HashMap<>();

// Merchant order number Required

req.put("refundMerchantOrderNo", "M220000000001");

// Original merchant order number

req.put("originMerchantOrderNo", "M80000000001");

Map<String, Object> amount = **new** HashMap<>();

// Refund order currency Required

amount.put("currency", "AED");

// Refund order amount Required

amount.put("amount", **new** BigDecimal("0.1"));

req.put("amount", amount);

// Refund operator name Optional

req.put("operatorName", "JACKMA");

// Refund reason name Optional

req.put("reason", "reason123");

// Notification URL Optional

req.put("notifyUrl", "http://yoursite.com/api/notification");

wrap.put("bizContent", req);

System.***out***.println(JSON.*toJSONString*(wrap));

HttpRequest request =

**new** HttpRequest.Builder().api("/acquire2/refund/placeOrder").body(JSON.*toJSONBytes*(wrap)).build();

String response = client.execute(request);

System.***out***.println(response);

### Order refund query

Map<String, Object> wrap = **new** HashMap<>();

// Request time Required

wrap.put("requestTime", System.*currentTimeMillis*());

Map<String, Object> req = **new** HashMap<>();

//Refund merchant order number Required

req.put("refundMerchantOrderNo", "M50000000001");

wrap.put("bizContent", req);

System.***out***.println(JSON.*toJSONString*(wrap));

HttpRequest request =

**new** HttpRequest.Builder().api("/acquire2/refund/getOrder").body(JSON.*toJSONBytes*(wrap)).build();

String response = client.execute(request);

System.***out***.println(response);

### Transfer

Map<String, Object> wrap = **new** HashMap<>();

// Request time Required

wrap.put("requestTime", System.*currentTimeMillis*());

Map<String, Object> req = **new** HashMap<>();

// Merchant order number Required

req.put("merchantOrderNo", UUID.*randomUUID*().toString());

// Beneficiary Identity Type Required

req.put("beneficiaryIdentityType", "PHONE\_NO");

String payByPubKey = **new** String(Files

.*readAllBytes*(Paths.*get*(PayByDemo.**class**.getClassLoader().getResource("payby\_public\_key.pem").toURI())));

// Beneficiary Identity

req.put("beneficiaryIdentity", RsaUtil.*encrypt*("971-585812341", Charset.*forName*("UTF-8"), payByPubKey, 2048));

req.put("beneficiaryFullName", RsaUtil.*encrypt*("JACKMA", Charset.*forName*("UTF-8"), payByPubKey, 2048));

Map<String, Object> amount = **new** HashMap<>();

// Transfer order currency Required

amount.put("currency", "AED");

// Transfer order amount Required

amount.put("amount", **new** BigDecimal("0.1"));

req.put("amount", amount);

// memo Required

req.put("memo", "Bonus");

// Notification URL Optional

req.put("notifyUrl", "http://yoursite.com/api/notification");

wrap.put("bizContent", req);

System.***out***.println(JSON.*toJSONString*(wrap));

HttpRequest request =

**new** HttpRequest.Builder().api("/transfer/placeTransferOrder").body(JSON.*toJSONBytes*(wrap)).build();

String response = client.execute(request);

System.***out***.println(response);

Map<String, Object> wrap = **new** HashMap<>();

// Request time Required

wrap.put("requestTime", System.*currentTimeMillis*());

Map<String, Object> req = **new** HashMap<>();

// Merchant order number Required

req.put("merchantOrderNo", "M50000000002");

// Beneficiary Identity Type Required

req.put("beneficiaryIdentityType", "PHONE\_NO");

// Beneficiary Identity

req.put("beneficiaryIdentity", "971-585812341");

req.put("beneficiaryFullName", "JACKMA");

### Transfer query

Map<String, Object> wrap = **new** HashMap<>();

// Request time Required

wrap.put("requestTime", System.*currentTimeMillis*());

Map<String, Object> req = **new** HashMap<>();

// Merchant order number Required

req.put("merchantOrderNo", "M50000000002");

wrap.put("bizContent", req);

System.***out***.println(JSON.*toJSONString*(wrap));

HttpRequest request = **new** HttpRequest.Builder().api("/transfer/getTransferOrder").body(JSON.*toJSONBytes*(wrap)).build();

String response = client.execute(request);

System.***out***.println(response);

### Transfer to bank

Map<String, Object> wrap = **new** HashMap<>();

// Request time Required

wrap.put("requestTime", System.*currentTimeMillis*());

Map<String, Object> req = **new** HashMap<>();

// Merchant order number Required

req.put("merchantOrderNo", UUID.*randomUUID*().toString());

String payByPubKey = **new** String(Files

.*readAllBytes*(Paths.*get*(PayByDemo.**class**.getClassLoader().getResource("payby\_public\_key.pem").toURI())));

// Holder Name Required

req.put("holderName", RsaUtil.*encrypt*("JACKMA", Charset.*forName*("UTF-8"), payByPubKey, 2048));

// Iban Required

req.put("Iban", RsaUtil.*encrypt*("5000312313111", Charset.*forName*("UTF-8"), payByPubKey, 2048));

// SwiftCode Optional

req.put("swiftCode", "ARABAEADDER");

Map<String, Object> amount = **new** HashMap<>();

// Transfer order currency Required

amount.put("currency", "AED");

// Transfer order amount Required

amount.put("amount", **new** BigDecimal("0.1"));

req.put("amount", amount);

// memo Required

req.put("memo", "Bonus");

// Notification URL Optional

req.put("notifyUrl", "http://yoursite.com/api/notification");

wrap.put("bizContent", req);

System.***out***.println(JSON.*toJSONString*(wrap));

HttpRequest request =

**new** HttpRequest.Builder().api("/transfer/placeTransferToBankOrder").body(JSON.*toJSONBytes*(wrap)).build();

String response = client.execute(request);

System.*out*.println(response);

### Transfer to bank query

Map<String, Object> wrap = **new** HashMap<>();

// Request time Required

wrap.put("requestTime", System.*currentTimeMillis*());

Map<String, Object> req = **new** HashMap<>();

// Merchant order number Required

req.put("merchantOrderNo", UUID.*randomUUID*().toString());

wrap.put("bizContent", req);

System.***out***.println(JSON.*toJSONString*(wrap));

HttpRequest request = **new** HttpRequest.Builder().api("/transfer/getTransferToBankOrder").body(JSON.*toJSONBytes*(wrap)).build();

String response = client.execute(request);

System.***out***.println(response);

## Result notification

### Verify

// setting payby publicKey path

String payByPubKey = **new** String(Files

.*readAllBytes*(Paths.*get*(PayByDemo.**class**.getClassLoader().getResource("payby\_public\_key.pem").toURI())));

String plain =

"{\"notify\_time\":\"20200428133706\",\"acquireOrder\":{\"product\":\"Basic Payment Gateway\",\"orderNo\":\"911588066370006619\",\"paySceneCode\":\"DYNQR\",\"subject\":\"123456\",\"accessoryContent\":{\"amountDetail\":{\"vatAmount\":{\"amount\":0.10,\"currency\":\"AED\"}},\"terminalDetail\":{\"merchantName\":\"binge test merchant\"},\"goodsDetail\":{\"goodsId\":\"GI1005\",\"body\":\"Gifts\",\"goodsName\":\"candy flower\"}},\"merchantOrderNo\":\"eaa16681-070b-4ae6-9b13-809b0db6eb89\",\"expiredTime\":1588073568745,\"requestTime\":1588066368745,\"totalAmount\":{\"amount\":0.10,\"currency\":\"AED\"},\"payeeMid\":\"200000042607\",\"notifyUrl\":\"http://yoursite.com/api/notification\",\"paymentInfo\":{\"payChannel\":\"BALANCE\",\"paidTime\":1588066471000,\"payeeFeeAmount\":{\"amount\":0.01,\"currency\":\"AED\"},\"payerFeeAmount\":{\"amount\":0.00,\"currency\":\"AED\"},\"paidAmount\":{\"amount\":0.10,\"currency\":\"AED\"},\"payerMid\":\"100000001104\"},\"status\":\"PAID\_SUCCESS\"},\"\_input\_charset\":\"UTF-8\",\"notify\_timestamp\":1588066626060,\"notify\_id\":\"202004280007581901\"}";

String sign =

"dPVyhhidZioH00QCVglgDfXNcDXHuMXmtzYF4WCEvnvmL3nCyqP5r9DmQ2bQYOf30tLEpqx1vmJpcT85f8voual7+sKPAehGRbyL9m30BF1KHwOFOhZSnOsBO8NQDJ1WMkG34mRmndfKRWpzi6RzWwfS/twJRXTt7maY2yPt93xhqeb2JbG2hktDFx8tnk3oxXil3oZLFq75X2Gbpd1SkrsnvxTLq5Bo98i4K3Kl4jWySU/vu8nX0M2JPWF6uC3OlOMq32Wo3mDPqM0DzH9t9WBwz2X6MQVnc/aFA5GfJbMKMYFFYwCh9CPtaWbrGmAFXAC9u/sz8bt3IAyiAGdd4w==";

System.***out***.println("verify result=>" + RsaUtil.*verifySign*(plain, Charset.forName("UTF-8"), sign, payByPubKey));

### Servlet receive messages（springmvc）

@Override

**protected** **void** doPost(HttpServletRequest req, HttpServletResponse httpResponse) **throws** ServletException, IOException {

String sign = req.getHeader("sign");

String body = IOUtils.*toString*(req.getInputStream(), "UTF-8");

// setting payby publicKey path

String payByPubKey = **new** String(Files.*readAllBytes*(Paths.*get*(PayByDemo.**class**.getClassLoader().getResource("payby\_public\_key.pem").toURI())));

System.***out***.println("verify result=>" + RsaUtil.*verifySign*(body, Charset.*forName*("UTF-8"), sign, payByPubKey));

//Business logic

httpResponse.setContentType("application/json;charset=UTF-8");

httpResponse.getOutputStream().write("SUCCESS".getBytes("UTF-8"));

httpResponse.flushBuffer();

}