Getting Started

REST API

February 2018

CyberSource®
the power of payment

CyberSource Corporation HQ | P.O. Box 8999 | San Francisco, CA 94128-8999 | Phone: 800-530-9095

CyberSource Contact Information

For general information about our company, products, and services, go to http://www.cybersource.com.

For sales questions about any CyberSource Service, email sales@cybersource.com or call 650-432-7350 or 888-330-2300 (toll free in the United States).

For support information about any CyberSource Service, visit the Support Center: http://www.cybersource.com/support

Copyright

Restricted Rights Legends

Trademarks

Contents

Recent Revisions to This Document

Release	Changes
February 2018	This release contains only formatting and editing updates.
September 2017	Added JSON Web Token (JWT) registration and authentication. For JWT registration, see "Creating a P12 Certificate for JSON Web Token," page 8. For JWT authentication, see "JSON Web Token Authentication," page 11.
August 2017	Initial release of document.

About This Guide

Audience and Purpose

This guide is written for application developers who want to use the CyberSource REST API to integrate CyberSource services into their order management system. It describes the basic steps you must complete in order to get started with the REST API.

Implementing the CyberSource services requires software development skills. You must use the REST API request and reply fields to integrate the services into your existing order management system.

Conventions

Note and Important Statements

Note

A *Note* contains helpful suggestions or references to material not contained in the document.

Important

An *Important* statement contains information essential to successfully completing a task or learning a concept.

Text and Command Conventions

Convention	Usage
Bold	Items that you are instructed to act upon; for example: Click **Save**.
`Screen text`	Code examples and samples.

Related Documents

REST API Documentation page

Refer to the Support Center for complete CyberSource technical documentation:

http://www.cybersource.com/support_center/support_documentation

Customer Support

For support information about any CyberSource service, visit the Support Center at:

http://www.cybersource.com/support

CHAPTER 1

Registration

Create an Evaluation Account

To create an evaluation account, see:

https://www.cybersource.com/register/

To complete the registration process, follow the email instructions you receive to activate your merchant account and log in to the new CyberSource Business Center.

Creating a Certificate or Shared Secret Key

There are two ways to create a key, depending on which authentication method you will use:

- JSON Web Token—P12 certificate
- HTTP Signature—shared secret key

Creating a P12 Certificate for JSON Web Token

The certificate contains both the public and private key.

To create a P12 certificate:

Step 1 Log in to the new Business Center.

Step 2 In the left navigation panel, choose **Key Management**.

Step 3 Expand the Transaction Processing menu.

Step 4 From the drop-down menu, choose **API Keys**.

Step 5 Click the **Add** icon at the bottom of the page.

Step 6 Select **Certificate**.

Step 7 Click **Download**.

Step 8 See the "JSON Web Token Authentication," page 11 for additional instructions for the P12 certificate.

Creating a Shared Secret Key for HTTP Signature

HTTP Signature authentication is provided by a base-64 encoded transaction key, represented in a string format. Before you can send requests for CyberSource REST API services that are authenticated using HTTP Signature, you must create a shared secret key for your CyberSource merchant account in the Business Center.

Important You must use separate keys for the test and production environments.

The shared secret key created in the Business Center lasts 3 years.

To create a shared secret key:

Step 1 Log in to the new Business Center.

Step 2 In the left navigation panel, choose **Key Management**.

Step 3 Expand the Transaction Processing menu.

Step 4 From the drop-down menu, choose **API Keys**.

Step 5 Click the **Add** icon at the bottom of the page.

Step 6 Select **Shared Secret Key**.

Step 7 Click **Generate New**.

Step 8 Copy the key from the user interface or download it to a .txt file.

Important

Take a note of the serial number that is generated for the key. This value is required in the header of each REST API call.

Deactivating Shared Secret Keys

When you deactivate a key it is immediately removed from active status.

To deactivate a shared secret key:

Step 1 Log in to the new Business Center.

Step 2 In the left navigation panel, choose **Key Management**.

Step 3 Expand the Transaction Processing menu.

Step 4 From the drop-down menu, choose **API Keys**.

Step 5 Select the key and click the **X** icon to deactivate the key.

The key is immediately deactivated.

CHAPTER 2

Authentication

You can use either HTTP Signature or JSON Web Token (JWT) to authenticate API requests.

Table 1 Authentication Methods

Authentication Method	Description
JSON Web Token (JWT)	■ Standard—IEFT draft; W3C Web Payment.
HTTP Signature	■ Signature Signing—supports shared symmetric keys and asymmetric keys. ■ Integration Scenario—additional HTTP headers only and the payload can not be modified. ■ Payload encryption—no.

JSON Web Token Authentication

After creating the P12 certificate (see "Creating a P12 Certificate for JSON Web Token," page 8), you must convert it to PEM format. First, you'll need to set up your environment.

Environment Setup

To encrypt using AES 256, Use the 'Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files'

-- local_policy.jar and US_export_policy.jar.

You can download these files from oracle website:

http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html

Once you have these jars, replace the *your_java_installation_directory/jre/lib/security* jar files with these.

Converting the Certificate to PEM Format using OpenSSL

When you run the OpenSSL tool, the tool prompts for your password, which is your Merchant ID. By default, your Merchant ID is also the name of the file.

The output of the tool will be text in PEM format. PEM defines each section of the file with “BEGIN” and “END” markers which denote the start and end of the objects within the file.

Example OpenSSL Conversion

```
bash-4.1$  openssl pkcs12 -in MID.p12 -out MID.pem
Enter Import Password:
MAC verified OK
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
bash-4.1$ cat MID.pem
Bag Attributes
    localKeyID: 01
    friendlyName: serialNumber=4654139977230172554798,CN=MID
subject=/CN=MID/serialNumber=4654139977230172554798
issuer=/CN=CyberSourceCertAuth
-----BEGIN CERTIFICATE-----
MIIB4TCCAUqgAwIBAgIWNDY1NDEzOTk3NzIzMDE3MjU1NDc5ODANBgkqhkiG9w0B
AQUFADAeMRwwGgYDVQQDDBNDeWJlclNvdXJjZUNlcnRBdXRoMB4XDTE2MDYwODE5
MjYzN1oXDTE5MDYwODE5MjYzN1owOjEXMBUGA1UEAwwOSmFzb25FYXRvbkNvcnAx
HzAdBgNVBAUTFjQ2NTQxMzk5NzcyMzAxNzI1NTQ3OTgwgZ8wDQYJKoZIhvcNAQEB
BQADgY0AMIGJAoGBALJKm1u6AcDNZQljcAtaG5II+FVefBtQF+xETFhCK0EJWfLh
XUNxTZIDHbZsf11IzRfs10w5sXviv5Z3vtCg8C1rJKoUuoJ5EJsWaEeBVKL6kZ4K
KlOm5559KTPYBfwCP73Hbu2qMGxfUu01ZUsOyKcSEFY3rxH6IQ6Z//qMZY5tAgMB
AAEwDQYJKoZIhvcNAQEFBQADgYEAj09Zlljg7pO+A0OUGZrNlPDDoSXjrGV/fyvn
fXXE598ldLZZ52nnEx+jJ4SQF0KguTP7OQWhV+8I6SKVePUDkOm4KBDEPmcRvjA9
0pgLVO7kwKX0r5QX0dTgR/GvDJyGi7znOu6Pr16WIU1Qh9jdqEF1wRxpS7SN6Uf1
u4KNO1c=
-----END CERTIFICATE-----
Bag Attributes
    friendlyName: serialNumber=4654139978710172554798,CN=CyberSource_SJC_
US
subject=/CN=CyberSource_SJC_US/serialNumber=4654139978710172554798
issuer=/CN=CyberSourceCertAuth
-----BEGIN CERTIFICATE-----
MIIB5TCCAU6gAwIBAgIWNDY1NDEzOTk3ODcxMDE3MjU1NDc5ODANBgkqhkiG9w0B
AQUFADAeMRwwGgYDVQQDDBNDeWJlclNvdXJjZUNlcnRBdXRoMB4XDTE2MDYwODE5
MjYzN1oXDTE5MDYwODE5MjYzN1owPjEbMBkGA1UEAwwSQ3liZXJTb3VyY2VfU0pD
X1VTMR8wHQYDVQQFExY0NjU0MTM5OTc4NzEwMTcyNTU0Nzk4MIGfMA0GCSqGSIb3
DQEBAQUAA4GNADCBiQKBgQCfQlT5TzBwYVDeqgpueMOas6Rc3hbzz/1IPWbqy2yi
```

```
8WCORXeQpLOHnBzz9JeefJgat2MzhEqC/yel8rsqIWFGBSP/4vShlIUgq+yIjlEA
8XPvniPb6EFs2Vyhqrz5o90+8CSf+mmJU9QH/l0HPjas7JevM6M4tjsCD3q7gNbU
SQIDAQABMA0GCSqGSIb3DQEBBQUAA4GBAJlAkv3Dw3u9o96W3f1JUC0Q+Z1NWDer
t2M+bvkXNBEiIi5dc4iRu4v/Axj+4OHBgt83eWXb2Htq5Sdj7XnF9aAS3q7ONUlS
59w/jPIm8e3ca9Zc/B9tAkyp8MNHrRTrGS3egDCroNwQnlkCD+gm6MYbvphNyIBg
3AnyvBvhr4Q0
-----END CERTIFICATE-----
Bag Attributes
    friendlyName: CN=CyberSourceCertAuth
subject=/CN=CyberSourceCertAuth
issuer=/CN=CyberSourceCertAuth
-----BEGIN CERTIFICATE-----
MIIBxTCCAS6gAwIBAgIWMzI2OTA0NjIyMzgwMDE2ODYyNjIyMDANBgkqhkiG9w0B
AQUFADAeMRwwGgYDVQQDDBNDeWJlclNvdXJjZUNlcnRBdXRoMB4XDTEyMDExODE2
MzcwMloXDTIyMDExODE2MzcwMlowHjEcMBoGA1UEAwwTQ3liZXJTb3VyY2VDZXJ0
QXV0aDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAuVMamKpEz+KaQk1ZujUY
ms6xO89gIsl/hm5hMf75Gc5GsOSGYD1UI34NZfrBW9mSoySPgXYBH0pq82X6pa7W
hKVZHzK/0AA7hnbKbrshyvHfv2lNjpHqRAmzwfA9gc9J3crTDXPq+tsQJjRpPhLd
M3+Paw6AxFtCeX4oX959mU8CAwEAATANBgkqhkiG9w0BAQUFAAOBgQA0Do9T1RqU
```

Converting the Certificate to PEM Format using a Java API Conversion

You can use Bouncy Castle JCE to perform a Java API conversion of the P12 certificate into the PEM format.

Example Java API Conversion

```
private static X509Certificate initializeCertificate(MerchantConfig
merchantConfig) throws CertificateException, NoSuchAlgorithmException,
IOException, KeyStoreException, UnrecoverableEntryException,
ConfigException {
        if(merchantConfig != null && merchantConfig.getKeyAlias() != null
&& merchantConfig.getKeyFile() != null) {
           KeyStore merchantKeyStore = KeyStore.getInstance("PKCS12", new
BouncyCastleProvider());
            merchantKeyStore.load(new
FileInputStream(merchantConfig.getKeyFile()),
merchantConfig.getKeyPassword().toCharArray());
            String merchantKeyAlias = null;
            Enumeration enumKeyStore = merchantKeyStore.aliases();

            while(enumKeyStore.hasMoreElements()) {
                merchantKeyAlias = (String)enumKeyStore.nextElement();
              if(merchantKeyAlias.contains(merchantConfig.getKeyAlias()))
{
                    break;
                }
```

```
            }

            PrivateKeyEntry keyEntry =
(PrivateKeyEntry)merchantKeyStore.getEntry(merchantKeyAlias, new
PasswordProtection(merchantConfig.getKeyPassword().toCharArray()));
            return (X509Certificate)keyEntry.getCertificate();
        } else {
           throw new ConfigException("merchant config fields missing: key
alias, key file");
        }
    }

    private static RSAPrivateKey initializePrivateKey(MerchantConfig
merchantConfig) throws CertificateException, NoSuchAlgorithmException,
IOException, KeyStoreException, UnrecoverableEntryException,
ConfigException {
        if(merchantConfig != null && merchantConfig.getKeyAlias() != null
&& merchantConfig.getKeyFile() != null) {
           KeyStore merchantKeyStore = KeyStore.getInstance("PKCS12", new
BouncyCastleProvider());
            merchantKeyStore.load(new
FileInputStream(merchantConfig.getKeyFile()),
merchantConfig.getKeyPassword().toCharArray());
            String merchantKeyAlias = null;
            Enumeration enumKeyStore = merchantKeyStore.aliases();

            while(enumKeyStore.hasMoreElements()) {
                merchantKeyAlias = (String)enumKeyStore.nextElement();
              if(merchantKeyAlias.contains(merchantConfig.getKeyAlias()))
{
                    break;
                }
            }

            PrivateKeyEntry keyEntry =
(PrivateKeyEntry)merchantKeyStore.getEntry(merchantKeyAlias, new
PasswordProtection(merchantConfig.getKeyPassword().toCharArray()));
            return (RSAPrivateKey)keyEntry.getPrivateKey();
        } else {
           throw new ConfigException("merchant config fields missing: key
alias, key file");
        }
    }
```

HTTP Headers

Table 2 HTTP Headers for JSON Web Token

Header Name	Description	Example
JWT header section		
kid	Serial Number or thumbprint for merchant's alias/CN name, corresponding to the key (.p12) used to digitally sign the JWS. Optional.	`501532503405017709696 5`
x5c	The "x5c" (X.509 certificate chain) Header Parameter contains the X.509 public key certificate or certificate chain corresponding to the key(.p12) used to digitally sign the JWS. Required.	`MIICZTCCAc6gAwIBAgIWNTAxNTMyNTAzND A1MDE3NzA5Njk2NTANBgkqhkiG9w0B AQsFADAeMRwwGgYDVQQDDBNDeWJlclNvdX JjZUNlcnRBdXRoMB4XDTE3MDczMTIw MjE0M1oXDTE5MDczMTIwMjE0M1owOjEXMB UGA1UEAwwOcmVzdG1lcmNoX2FjY3Qx HzAdBgNVBAUTFjUwMTUzMjUwMzQwNTAxNz cwOTY5NjUwggEiMA0GCSqGSIb3DQEB AQUAA4IBDwAwggEKAoIBAQC+BtdMcokaKw Tes9O+snqjWsehO0/Y4CJrwo4cdGCa fRO6ows3mC1xmpSYttTKIdeGR1qOYpTSeY 9uJh2cCfayMKXmhQf4d5v8pNDnmYBM GBwU1wWjlYsMLFbcPFWjdPtKlou2RIWZ56 6YWIbWCBf+G4dHkA0D7Csp3FT3lnUN 2me4lJklysfqp1X8q3JDqVIQFVAduamhWC YMRS5Kg8sKwMjldGQUby88S+MOwcCA 3HOHT64barvPId2ExqAqvfyBCPAR49V6/ ywDVsH9SyHewg121ckMcA9g55tIqdsy HGRZN9fs20kMyqEP78L4cxfisuVQC8knlf ZtdcqVAY7FAgMBAAEwDQYJKoZIhvcN AQELBQADgYEAZ9NWN0BNVEhtVLqldmEfMR qJCqfzFNq1UyQEu3CJOjPhN9dYq8Jl aD6d1mRpybeaclvHN/F/ 06RjZW9zujlDFE3X0qanU3Rkj7nStiDUEm j0F35Ew2ek 4VezUXnZ/ SMLvWEA6DG2sjSFCCuIot3mLJ3lI4AQSQS BSazhQec75Rk=`
alg	Identifier of the hashing algorithm. Required.	`RS256`
v-c-merchant-id	Merchant ID assigned in the CyberSource Business Center. Required.	`v-c-merchant-id: restmercht_id`
JWT payload section		

Table 2 HTTP Headers for JSON Web Token (Continued)

Header Name	Description	Example
iat	The date and time that the message was originated. Date can be in any for timezone. HTTP-date format is defined by RFC7231: `String gmtDateTime = DateTimeFormatter.RFC_1123_DATE_ TIME.format(ZonedDateTime.now(ZoneId.of("GMT")));` Required.	`iat: Thru, 15 June 2017 08:12:31 GMT`

Digest	SHA256 encryption of the payload that is Base64 encoded.
	Required.
		`example_payload:`
		`{`
		`  "profileId": "11111111-1111-1111-1111-111111111111",`
		`  "encryptionType": "RsaOaep256"`
		`}`
		
		`SHA256_hash_of_example_payload = 2b4fee10da8c5e1feaad32b014021e079fe4afcf06af223004af944011a7cb65c`
		
		`# The hash has Base64 encoded Digest header in RFC3230 defined format of "Digest: HASH-ALG=BASE64(SHA56_hash_of_example_payload)“ = tP7hDajF4f6q0ysBQCHgef5K/PBq8iMASvlEARp8tl=`
		
		`Digest: tP7hDajF4f6q0ysBQCHgef5K/PBq8iMASvlEARp8tl=`
		
		`Code Snippet:`
		`MessageDigest signatureString = MessageDigest.getInstance("SHA-256"); byte[] digestBytes = signatureString.digest(messageBody.getBytes()); String bluePrint = Base64.getEncoder().encodeToString(digestBytes);`

digestAlgorithm	Signature algorithm used. SHA256 if an asymmetric key is used. Required.	`digestAlgorithm: SHA-256`
Signature and token		

Table 2 HTTP Headers for JSON Web Token (Continued)

Header Name	Description	Example

Signature	The header and the payload created is BASE64 Encoded. Join the resulting encoded strings together with a period (.) in between them. In our pseudo code, this joined string is assigned to data. To get the JWT signature, the data string is hashed with RS256, with the secret key using the hashing algorithm specified in the JWT header.
	Required.
		`// signature algorithm`
		
		`data = base64urlEncode( header ) + “.” + base64urlEncode( payload )`
		`signature = RS256Hash( data, secret ) ;`

JWT Token	With All three components, We can create
	`JWT token = header.payload.signature`
	Combine the header and payload and signature with periods (.) separating them. We use the base64url encoded versions of the header and of the payload, and the signature.
		`JWT Token = base64url(header) + “.” + base64url(payload) + “.” + base64url(Signature)`
		
		`// header`
		`eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9`
		
		`// payload`
		`eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ`
		
		`// signature`
		`-xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzcM`
		
		`// JWT Token`
		`eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ4ZjItOGZhYi1jZWYzOTA0NjYwYmQifQ.-xN_h82PHVTCMA9vdoHrcZxH-x5mb11y1537t3rGzc`

Encoding and Hashing the Digest

```
if(requestBody != null && !requestBody.isEmpty()) {
     MessageDigest jwtBody = MessageDigest.getInstance("SHA-256");
     byte[] Headers = jwtBody.digest(requestBody.getBytes());
     e = Base64.getEncoder().encodeToString(Headers);
  }
```

Preparing the Payload

```
String jwtBody = "{\n            \"digest\":\"" + e + "\",\n
\"digestAlgorithm\":\"SHA-256\",\n          \"iat\":\"" +
DateTimeFormatter.RFC_1123_DATE_
TIME.format(ZonedDateTime.now(ZoneId.of("GMT"))) + "\"\n} \n\n";

HashMap customHeaders = new HashMap();
customHeaders.put(v-c-merchant-id, merchantConfig.getMerchantID());

String jwsSignatureValue = sign(jwtBody, rsaPrivateKey, x509Certificate,
customHeaders);
```

Generating the JWT Token – Header, Payload and Signature

```
private sign(String content, PrivateKey privateKey, X509Certificate
x509Certificate, Map<String, ? extends Object> customHeaders) {
    if(!this.isNullOrEmpty(content) && x509Certificate != null &&
privateKey != null) {
        String serialNumber = null;
        String serialNumberPrefix = "SERIALNUMBER=";
        String principal =
x509Certificate.getSubjectDN().getName().toUpperCase();
        int beg = principal.indexOf(serialNumberPrefix);
        if(beg >= 0) {
            int x5cBase64List = principal.indexOf(",", beg);
            if(x5cBase64List == -1) {
                x5cBase64List = principal.length();
            }

            serialNumber = principal.substring(beg +
serialNumberPrefix.length(), x5cBase64List);
        } else {
            serialNumber = x509Certificate.getSerialNumber().toString();
        }

        ArrayList x5cBase64List1 = new ArrayList();

        try {

x5cBase64List1.add(Base64.encode(x509Certificate.getEncoded()));
        } catch (CertificateEncodingException var16) {
            logger.error("can\'t signAndEncrypt the payload", var16);
            return null;
        }

        RSAPrivateKey rsaPrivateKey = (RSAPrivateKey)privateKey;
        Payload payload = new Payload(content);
```

```
        JWSHeader jwsHeader = (new
com.nimbusds.jose.JWSHeader.Builder(JWSAlgorithm.RS256)).customParams(cus
tomHeaders).keyID(serialNumber).x509CertChain(x5cBase64List1).build();
        JWSObject jwsObject = new JWSObject(jwsHeader, payload);

        try {
            RSASSASigner joseException = new RSASSASigner(rsaPrivateKey);
            jwsObject.sign(joseException);

if(!jwsObject.getState().equals(com.nimbusds.jose.JWSObject.State.SIGNED)
) {
                logger.error("Payload signing failed.");
                return null;
            } else {
                return jwsObject;
            }
        } catch (JOSEException var15) {
            logger.error("can\'t signAndEncrypt the payload", var15);
            return null;
        }
    } else {
        logger.error("empty or null content or Private key  or public
certificate is null");
        return null;
    }
}
```

HTTP Signature Authentication

All requests to the CyberSource REST API must be authenticated. This section explains how to use headers in the request. Before you begin, you will need three pieces of information from the CyberSource Business Center.

Table 3 Required Information from the Business Center

Information	Description
Serial number/ key ID	The serial number of the signing certificate/key pair. Obtain this in the Business Center's Key Management area. For more information, see Creating a Shared Secret Key for HTTP Signature.
Shared Secret Key	Obtain this in the Business Center's Key Management area. For more information, see Creating a Shared Secret Key for HTTP Signature.
Merchant ID	This is the ID of the merchant account that generated the shared secret key in the Business Center.

Signing a POST Request

Let's start by taking a look at the end result, and then we'll break it down piece by piece. Below is an example of a payment request. Headers are in **bold** font and signature parameters are in ***bold and italic*** font. Note that the `Signature` heading contains a `signature` parameter. Extra spaces have been added for readability.

Example POST Request

```
POST https://api.cybersource.com/pts/v2/payments
HTTP/1.1
Content-Type: application/json

v-c-merchant-id: merchant123
Host: api.cybersource.com
Date: Wed, 25 Dec 2017 00:23:05 GMT
Digest: SHA-256=bena9bhB3Jy4uPvfu1tAC0uN8AuzzM+xjqmDwR5//EA=

Signature: keyid="4853015631630181645627",
algorithm="SHA256withRSA", headers="host date (request-target)
digest v-c-merchant-id",
signature="q+j2M0fIQ1lIJJnjVtSISR0IZMTnury5m+XqZDNvCsHeF4WxhdOotqI
IqqzEUoXMDcwPXVOUNJYXBBn1TaWLSisCO9ZIvVZghuJV3VwW1xqg9Rtyu6rxDZ1fL
v+GMo4gUEV9y3sJq7cfJ0HZRN05ud/FRZYTeMlVWC5hXE2WRqhbtPg="
```

v-c-merchant-id

Required. Contains the merchant ID for the account that generated the shared secret key in the Business Center. See "Creating a Shared Secret Key for HTTP Signature," page 9.

Example:

```
v-c-merchant-id: merchant123
```

Host

The CyberSource environment server to which the request is sent. Use the Sandbox environment for testing and Production environment for processing real transactions.

Sandbox Host Name

```
apitest.cybersource.com
```

Production Host Name

```
api.cybersource.com
```

Date

Date and time the message originated, using GMT format, as defined by RFC7231.

Example:

```
Date: Wed, 25 Dec 2017 00:23:05 GMT
```

Code Snippet

```
String gmtDateTime = DateTimeFormatter.RFC_1123_DATE_
TIME.format(ZonedDateTime.now(ZoneId.of("GMT")));
```

Digest

HMAC-SHA256 hash of the HTTP request's `payload(messageBody),` which is Base64- encoded and normalized as a Digest HTTP header. You must concatenate SHA-256=" with the digest string, as shown below.

Example

```
messageBody =
  {
    "paymentInformation": {
      "card": {
        "expirationYear": "2031",
        "number": "4111111111111111",
        "securityCode": "123",
        "expirationMonth": "12",
        "type": "002"
    }
  }
digestString = BASE64( HMAC-SHA256 ( messageBody));
Digest: "SHA-256=" + digestString;
```

Code Snippet

```
#Creating the SHA256 String
MessageDigest hashString = MessageDigest.getInstance("SHA-256");
byte[] digestBytes = hashString.digest(messageBody.getBytes());

#Creating the BASE64 String
String digetString =
Base64.getEncoder().encodeToString(digestBytes);

#Creating the Digest
String Digest = "SHA-256="+ digestString;
```

Signature

The `Signature` header contains various parameters that must be submitted in order. Note that one of the parameters is called `signature`. Be careful not to confuse the header with its parameters. Let's take a look at the full signature header before examining its parameters.

```
Signature: keyid="485301563163018164562 7", algorithm="HmacSHA256",
headers="host date (request-target) digest v-c-merchant-id",
signature="q+j2M0fIQ1lIJJnjVtSISR0IZMTnury5m+XqZDNvCsHeF4WxhdOotqI
IqqzEUoXMDcwPXVOUNJYXBBn1TaWLSisCO9ZIvVZghuJV3VwW1xqg9Rtyu6rxDZ1fL
v+GMo4gUEV9y3sJq7cfJ0HZRN05ud/FRZYTeMlVWC5hXE2WRqhbtPg="
```

keyid

The serial number of the signing certificate/key pair. Obtain this in the Business Center's Key Management area. For more information, see Creating a Shared Secret Key for HTTP Signature.

algorithm

The signature algorithm used. The value should be `HmacSHA256`.

headers

Contains the headers that are submitted in the HTTP request, separated by spaces. Headers include:

- host
- date
- request-target—an HTTP method followed by a resource.

```
(request-target): method path
```

For example:

```
(request-target): post /pts/v2/payments
```

- digest
- v-c-merchant-id

Signature

The Base64-encoded signature of the headers listed above. The example below contains line breaks for readability.

```
signature = BASE64 ( host: apitest.cybersource.com
          date: Sat, 8 Jul 2017 01:08:05 GMT
         (request-target): post /pts/v2/payments
          digest: SHA-256=/P58nNyRHCicPrDhJW8niEDMSnYCMRsITm17INuaq6E=
          v-c-merchant-id: restmerch_acct )
```

Signing a GET Request

Let's start by taking a look at the end result, and then we'll break it down piece by piece. Below is an example of a request for the details of a single payment. Headers are in **bold** font and signature components are in ***bold and italic*** font. Note that the `Signature` heading contains a `signature` parameter. Extra spaces have been added for readability.

Example GET Request

```
GET https://api.cybersource.com/pts/v2/payments/98347878329748239
HTTP/1.1
Content-Type: application/json

v-c-merchant-id: merchant123
Host: api.cybersource.com
Date: Wed, 25 Dec 2017 00:23:05 GMT

Signature: keyid="485301563163018164562 7",
algorithm="SHA256withRSA", headers="host date (request-target)  v-
```

```
c-merchant-id",
signature="q+j2M0fIQ1lIJJnjVtSISR0IZMTnury5m+XqZDNvCsHeF4WxhdOotqI
IqqzEUoXMDcwPXVOUNJYXBBn2F333F51TaWLSisCO9ZIvVZghuJV3VwW1xqg9Rtyu6
rxDZ1fLv+GMo4gUEV9y3sJq7cfJ0HZRN05ud/FRZYTeMlVWC5hXE2WRqhbtPg="
```

Code Snippet

The Shared Secret Key should be Base64-encoded and used to sign the signature parameter.

```
SecretKeySpec secretKey = new
SecretKeySpec(Base64.getDecoder().decode(“YOUR SHARED SECRET KEY”),
"HmacSHA256");
HttpSignatureHeader httpSignatureHeader =
HttpSignature.createHttpSignatureHeaders(messageBody, map, secretKey);

Note
‘map’ above is the HashMap of all the five headers discussed below.
```

v-c-merchant-id

Required. Contains the merchant ID for the account that generated the shared secret key in the Business Center. See "Creating a Shared Secret Key for HTTP Signature," page 9.

Example:

```
v-c-merchant-id: merchant123
```

Date

Date and time the message originated, using GMT format, as defined by RFC7231.

Example:

```
Date: Wed, 25 Dec 2017 00:23:05 GMT
```

Code Snippet

```
String gmtDateTime = DateTimeFormatter.RFC_1123_DATE_
TIME.format(ZonedDateTime.now(ZoneId.of("GMT")));
```

Host

The CyberSource environment server to which the request is sent. Use the Sandbox environment for testing and Production environment for processing real transactions.

Sandbox Host Name

`apitest.cybersource.com`

Production Host Name

`api.cybersource.com`

Code Snippet

```
#Creating the SHA256 String
MessageDigest hashString = MessageDigest.getInstance("SHA-256");
byte[] digestBytes = hashString.digest(messageBody.getBytes());

#Creating the BASE64 String
String digetString =
Base64.getEncoder().encodeToString(digestBytes);

#Creating the Digest
String Digest = "SHA-256="+ digestString;
```

Signature

The `Signature` header contains various parameters that must be submitted in order. Note that one of the parameters is called `signature`. Be careful not to confuse the header with its parameter. Let's take a look at the full signature header before examining its parameters.

```
Signature: keyid="485301563163018164562 7", algorithm="HmacSHA256",
headers="host date (request-target) digest v-c-merchant-id",
signature="q+j2M0fIQ1lIJJnjVtSISR0IZMTnury5m+XqZDNvCsHeF4WxhdOotqI
IqqzEUoXMDcwPXVOUNJYXBBn1TaWLSisCO9ZIvVZghuJV3VwW1xqg9Rtyu6rxDZ1fL
v+GMo4gUEV9y3sJq7cfJ0HZRN05ud/FRZYTeMlVWC5hXE2WRqhbtPg="
```

keyid

The serial number of the signing certificate/key pair. Obtain this in the Business Center's Key Management area. For more information, see Creating a Shared Secret Key for HTTP Signature.

algorithm

The signature algorithm used. The value should be `HmacSHA256`.

headers

Contains the headers that are submitted in the HTTP request, separated by spaces. Headers include:

- host
- date
- request-target—Request-target is an HTTP method followed by a resource.

```
(request-target): method path
```

For example:

```
(request-target): post /pts/v2/payments
```

- digest
- v-c-merchant-id

Signature

The Base64-encoded signature of the headers listed above. The example below contains line breaks for readability.

```
signature = BASE64 ( host: apitest.cybersource.com
          date: Sat, 8 Jul 2017 01:08:05 GMT
         (request-target): get /pts/v2/payments/98347878329748239
         v-c-merchant-id: restmerch_acct )
```

Code Examples

Adding Headers

Example 1 Adding HTTP Headers

```
HashMap map = new HashMap();
map.put("keyid", merchantConfig.getKeyId());
map.put("host", merchantConfig.getRequestHost());
map.put("v-c-merchant-id", merchantId);
map.put("(request-target)", merchantConfig.getRequestTarget());
String hdrRequestDigest = " digest";
map.put("headers", "host date (request-target)" + hdrRequestDigest + " " + "v-c-
merchant-id");
SecretKeySpec secretKey = new
SecretKeySpec(Base64.getDecoder().decode(merchantConfig.getSecretKey()),
"HmacSHA256");
HttpSignatureHeader httpSignatureHeader =
HttpSignature.createHttpSignatureHeaders(requestBody, map, secretKey);
Connection connection = Connection.getInstance(merchantConfig, logger);
connection.setHeader("Content-Type", "application/json");
connection.setHeader("date", (String)httpSignatureHeader.getHeaders().get("date"));
connection.setHeader("digest",
(String)httpSignatureHeader.getHeaders().get("digest"));

connection.setHeader("host", (String)httpSignatureHeader.getHeaders().get("host"));
connection.setHeader("v-c-merchant-id",
(String)httpSignatureHeader.getHeaders().get("v-c-merchant-id"));
connection.setHeader("signature", httpSignatureHeader.getSignatureHeader());
logger.log("REQUEST  ", "Signed headers... " + connection.getHeaderMap().toString());
return connection.post(requestBody);
```

Creating a Signature Header

Example 2 Creating a Signature Header

```
public static HttpSignatureHeader createHttpSignatureHeaders(String messageBody,
Map<String, String> headers, SecretKey secretKey) {
        HashMap parsed = new HashMap();
                 headers.forEach((k, v) -> {
                 parsed.put(k.toLowerCase().trim(), v);
        });
        String signatureString = prepareSignatureCreationString(messageBody, parsed);
        Mac aKeyId = Mac.getInstance("HmacSHA256");
        aKeyId.init(secretKey);
        aKeyId.update(signatureString.getBytes());
        byte[] aHeaders = aKeyId.doFinal();
        String base64EncodedSignature = Base64.getEncoder().encodeToString(aHeaders);

        parsed.put("algorithm", "HmacSHA256");
        String aKeyId1 = (String)parsed.get("keyid");
        String aHeaders1 = (String)parsed.get("headers");
        StringBuilder signatureHeaderValue = new StringBuilder();
        signatureHeaderValue.append("keyid=\"" + aKeyId1 + "\"");
        signatureHeaderValue.append(", algorithm=\"HmacSHA256\"");
        signatureHeaderValue.append(", headers=\"" + aHeaders1 + "\"");
       signatureHeaderValue.append(", signature=\"" + base64EncodedSignature + "\"");
   return new HttpSignatureHeader(parsed, signatureHeaderValue.toString());
   } else {
       LOGGER.error("Null or empty signature String value");
       return null;
}

private static String prepareSignatureCreationString(String messageBody, Map<String,
String> parsed) {

String keyid = (String)parsed.get("keyid");
String headers = (String)parsed.get("headers");
String host = (String)parsed.get("host");
    if(keyid != null && !keyid.isEmpty() && headers != null && !headers.isEmpty() &&
host != null && !host.isEmpty()) {
    gmtDateTime = DateTimeFormatter.RFC_1123_DATE_
TIME.format(ZonedDateTime.now(ZoneId.of("GMT")));
    parsed.put("date", gmtDateTime);
```

```
    if(isNotEmpty(messageBody)) {
        String bluePrint;
        try {
            MessageDigest signatureString = MessageDigest.getInstance("SHA-256");
            byte[] digestBytes = signatureString.digest(messageBody.getBytes());
            bluePrint = Base64.getEncoder().encodeToString(digestBytes);
        } catch (NoSuchAlgorithmException var12) {
            LOGGER.error("HttpSignature:createHttpSignatureHeaders - Cannot create
digest");
            return null;
        }

        parsed.put("digest", "SHA-256=" + bluePrint);
    }
}
String[] splitHeader = headers.split(" ");
            signatureStringer signatureString = new signatureStringer();
            String[] splitHeader2 = splitHeader;

          for(int headerLoop = 0; headerLoop < splitHeader.length; ++headerLoop) {
                String s = splitHeader2[headerLoop];
                if(!parsed.containsKey(s)) {
                    LOGGER.error("HttpSignature:createHttpSignatureHeaders - Missing
passed or calculated headers");
                    return null;
                }

                signatureString.append('\n');
                signatureString.append(s);
                signatureString.append(": ");
                signatureString.append((String)parsed.get(s));
            }

            signatureString.delete(0, 1);
            return signatureString.toString();
}
```

CHAPTER 3

Testing

CyberSource recommends testing your REST API integration by sending transactions to the CyberSource test server:

apitest.cybersource.com


Note

The test server responses simulates processor responses.

To manage your evaluation account, log into the Test Business Center to:

- view test transactions.
- access administer users and access privileges.
- create roles with predefined access permissions.
- view reports.

Go-Live


Important

CyberSource recommends that you submit all banking information and required integration services in advance of going live. Doing so will speed up your merchant account configuration.

When you are ready to implement the REST API in your live environment, you must contact CyberSource Customer Support and request Go-Live. When all the banking information has been received by CyberSource the Go-Live procedure may require three days to complete. No Go-Live implementations take place on a Friday.