Getting Started with REST

REST API

cybersource
A Visa Solution

Developer Guide

Cybersource Contact Information

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

Copyright

Restricted Rights Legends

For Government or defense agencies: Use,duplication, or disclosure by the Government or defense agencies is subject to restrictions as set forth the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and in similar clauses in the FAR and NASA FAR Supplement.

For civilian agencies: Use, reproduction, or disclosure is subject to restrictions set forth in suparagraphs (a) through (d) of the Commercial Computer Software Restricted Rights clause at 52.227-19 and the limitations set forth in Cybersource Corporation's standard commercial agreement for this software. Unpublished rights reserved under the copyright laws of the United States.

Trademarks

Confidentiality Notice

This document is furnished to you solely in your capacity as a client of Cybersource and as a participant in the Visa payments system.

By accepting this document, you acknowledge that the information contained herein (the "Information") is confidential and subject to the confidentiality restrictions contained in Visa's operating regulations and/or other confidentiality agreements, which limity our use of the Information. You agree to keep the Information confidential and not to use the Information for any purpose other than its intended purpose and in your capacity as a customer of Cybersource or as a participant in the Visa payments system. The Information may only be disseminated within your organization on a need-to-know basis to enable your participation in the Visa payments system. Please be advised that the Information may constitute material non-public information under U.S. federal securities laws and that purchasing or selling securities of Visa Inc. while being aware of material non-public information would constitute a violation of applicable U.S. federal securities laws.

Revision

Version: 25.04.01

Contents

Getting Started with REST Developer Guide **5**
Recent Revisions to This Document 5
Overview of REST **7**
Set Up Your Cybersource Account **10**
Set Up JSON Web Token Messaging **13**
Sign Up for a Sandbox Account 15
Create a P12 Certificate 16
Create a P12 File 16
Extract the Private Key from the P12 Certificate 22
Test the Shared Secret Key Pair 23
Test Endpoints 27
Construct Messages Using JSON Web Tokens 27
Elements of a JSON Web Token Message 27
Generate a Hash of the Message Body 28
Generate the Token Header 29
Generate a Hash of the Claim Set 30
Generate a Hash of the Token Header 30
Generate the Message Body 31
Generate a Token Signature 32
Generate a JSON Web Token 32
Create the Authorization Header 33
Enable Message-Level Encryption 33
Prerequisites for Message-Level Encryption 34
Message-Level Encryption Using JSON Web Tokens 35
Going Live 39
Create a Merchant ID 39
Activate your Merchant ID 40
Production Endpoints 41
Set Up an HTTP Signature Message **42**
Sign Up for a Sandbox Account 42
Shared Secret Key Pair 43

Create a Shared Secret Key Pair....44
Test the Shared Secret Key Pair....49
Test Endpoints....53
Construct Messages Using HTTP Signature Security....53
Elements of an HTTP Message....53
Generate a Hash of the Message Body....54
Generate the Signature Hash....55
Update Header Fields....56
Going Live....57
Create a Merchant ID....57
Activate your Merchant ID....58
Production Endpoints....59
VISA Platform Connect: Specifications and Conditions for Resellers/Partners....60

Getting Started with REST Developer Guide

This section describes how to use this developer guide and where to find further information.
Visit the *Cybersource documentation hub* to find additional technical documentation.

Audience and Purpose	This guide provides information about how to sign up for a sandbox account and set up the Cybersource REST API.
Customer Support	For support information about any service, visit the Support Center: *http://support.visaacceptance.com*

Recent Revisions to This Document

25.04.01

Updated the Header field from `Date` to `v-c-date` for HTTP Signature Method. See *Elements of an HTTP Message* on page 53 and *Update Header Fields* on page 56.

25.03.01

This revision contains only editorial changes and no technical updates.

25.02.01

Updated the link in Step 6 and added steps about using the JSON Web Token. See *Construct Messages Using JSON Web Tokens* on page 27.
Added the section *Create the Authorization Header* on page 33.
Updated the description of the signature keyid. See *Update Header Fields* on page 56.

25.01.02

Added testing information in an Important note in the Secure Communication Requirements section. See *Overview of REST* on page 7.
Added information about the purposes of a sandbox account for the JSON web token and HTTP signature messaging implementations. See *Sign Up for a Sandbox Account* on page 15

25.01.01

This revision contains only editorial changes and no technical updates.

Overview of REST

To get started using the Cybersource payment API, you must first set up your payment processing system to be REST compliant. Cybersource uses the REST, or REpresentational State Transfer architecture, for developing web services. REST enables communication between a client and server using HTTP protocols.
This guide explains how to set up secure communications between your client and server using one of these methods:

JSON Web Token	JSON Web Tokens (JWTs) are digitally signed JSON objects based on the open standard *RFC 7519*. These tokens provide a compact, self-contained method for securely transmitting information between parties. These tokens are signed with an RSA-encoded public/private key pair. The signature is calculated using the header and body, which enables the receiver to validate that the content has not been tampered with. Token-based messages are best for applications that use browser and mobile clients.
HTTP Signature	Each request is digitally signed, or the entire request is digitally hashed using a private key. Both the client and server have the same shared secret, which enables each request to be validated on either end. If the request transmission is compromised, the attacker cannot change the request or act as a user without the secret. HTTP signatures can be used only with API requests. They cannot be used in browser or mobile applications.

Secure Communication Requirements

REST-compliant machines communicate with each other using stateless messaging. Stateless messaging is a loosely coupled connection between a client and server, where each message is self-contained. This connection enables the client and server to communicate without first establishing a communication channel and without managing the state between systems.

To ensure secure communications between the client and server, you must provide these security measures:

- Sender Authentication: A receiver needs to know that a message comes from a trusted entity.
- Message Encryption: By encrypting the message before transmission and decrypting the message when received, you prevent man-in-the-middle attacks.

Important

When building your connection to the Cybersource payment gateway, ensure that you have implemented controls to prevent card testing or card enumeration attacks on your platform. For more information, see the *best practices guide*. When we detect suspicious transaction activity associated with your merchant ID, including a card testing or card enumeration attack, Cybersource reserves the right to enable fraud management tools on your behalf in order to mitigate the attack. The fraud team might also implement internal controls to mitigate attack activity. These controls block traffic that is perceived as fraudulent. Additionally, if you are using one of our fraud tools and experience a significant attack, our internal team might modify or add rules to your configuration to help prevent the attack and minimize the threat to our infrastructure. However, any actions taken by Cybersource would not replace the need for you to follow industry standard best practices to protect your systems, servers, and platforms.

Key Features of REST

These are the key features of REST:

- Client/Server model: Clients and servers are independent from each other, enabling portability and scalability.
- Stateless Communication: Each request is independent.
- Uniform Interface: Architecture is simplified through uniform standards.

Components of REST

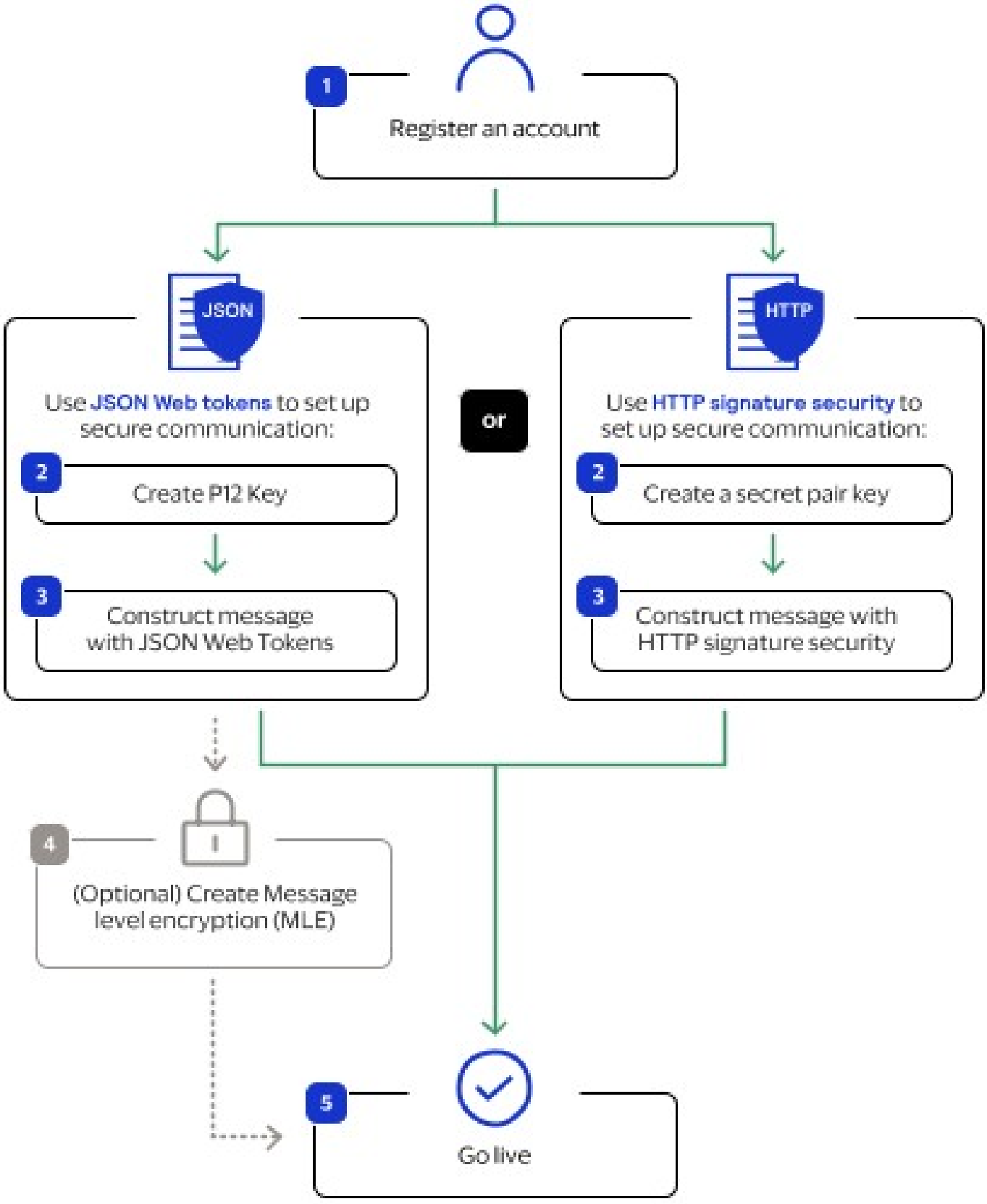
A REST message consists of these four components:

- Endpoint: The endpoint is a Uniform Resource Identifier (URI) that shows where and how to find the resource on the internet. For example, to test an authorization request, you can send the request to this endpoint: `https://apitest.cybersource.com/pts/v2/payments`.

- HTTP Method: The method is the action performed by the resource. There are four basic HTTP methods:
  - POST: Create a resource.
  - GET: Retrieve a resource.
  - PATCH: Modify a resource.
  - DELETE: Delete a resource.
- Headers: The header is a collection of fields and their associated values. It provides information about the message to the receiver. Think of it as metadata about the message. The header also contains authentication information that indicates that the message is legitimate.
- Body: The request in JSON format.

Set Up Your Cybersource Account

This overview lists the tasks you must to complete in order to set up your Cybersource account for sending and receiving REST API messages using either JSON Web Token messaging or HTTP signature messaging.



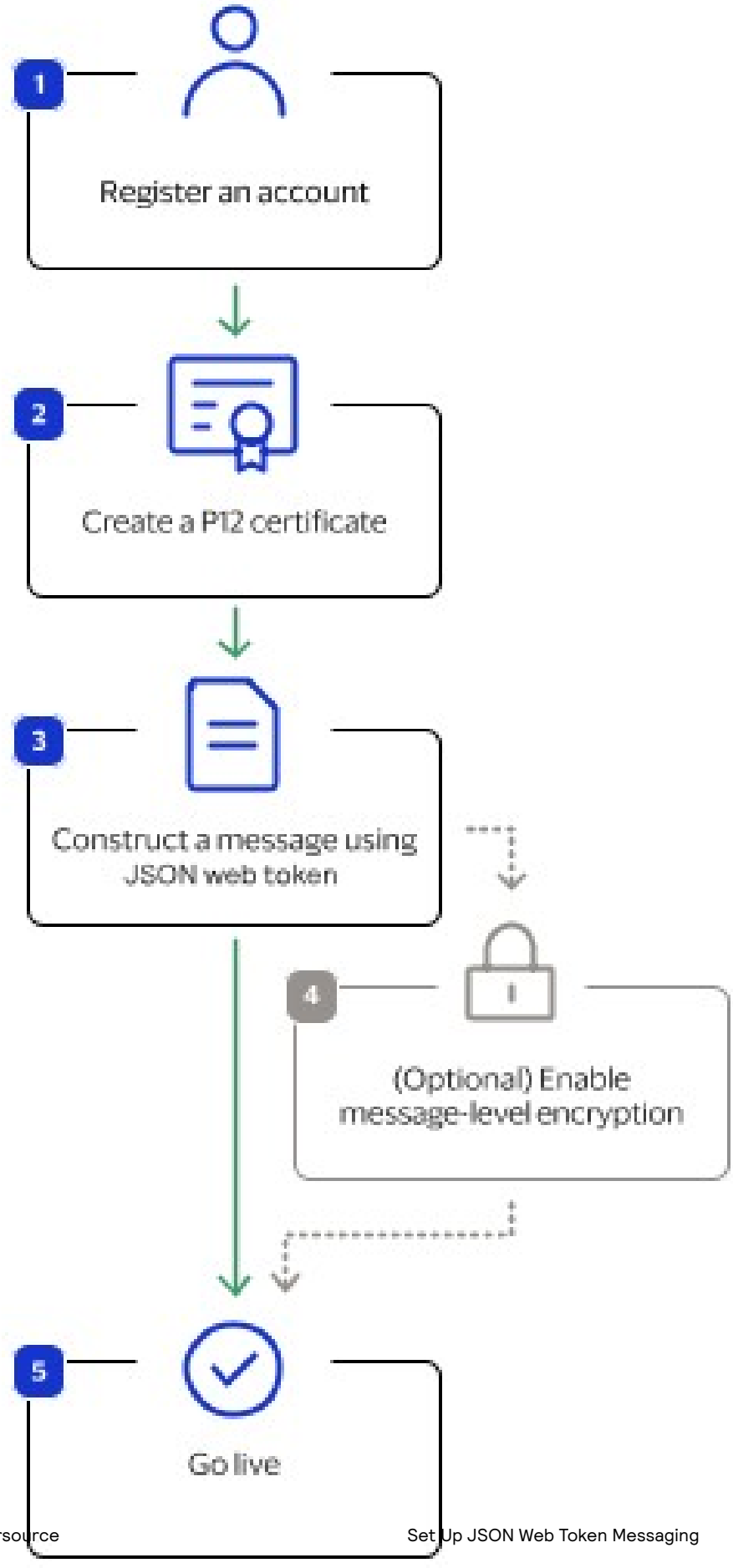
Enabling REST Workflow

To set up JSON web token messaging, see *Set Up JSON Web Token Messaging* on page 13.

To set up an HTTP signature messaging, see *Set Up an HTTP Signature Message* on page 42.

Set Up JSON Web Token Messaging

To set up JSON Web Token messaging, you will complete the tasks described in this section.


1
Register an account
2
Create a P12 certificate
3
Construct a message using JSON web token
4
(Optional) Enable message-level encryption
5
Go live

2. Create a P12 certificate. See *Create a P12 Certificate* on page 16.
3. Construct a message using a JSON web token. See *Construct Messages Using JSON Web Tokens* on page 27.
4. (Optional) Enable the optional message-level encryption (MLE) feature. See *Enable Message-Level Encryption* on page 33.
5. Go live by transitioning your sandbox account into a production account. See *Going Live* on page 39.

Sign Up for a Sandbox Account

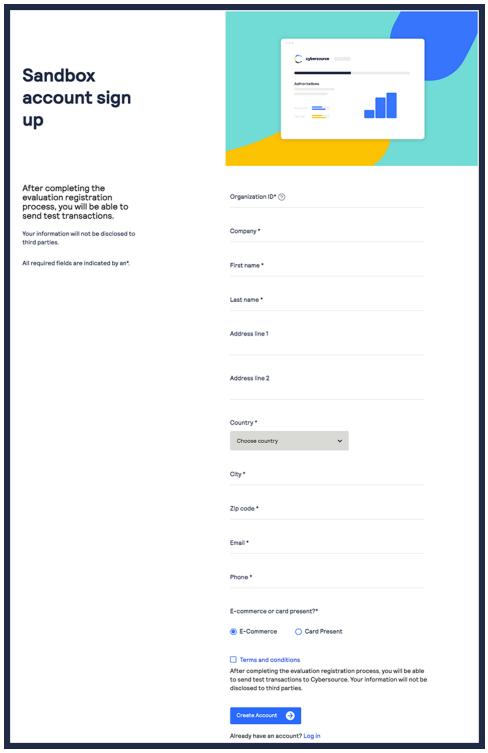
To begin setting up your account, you must first sign up for a sandbox account. A sandbox account enables you to obtain your security keys and test your implementation. After you verify that your system can send and receive REST messages, you can contact customer service to transition your sandbox account to a production account. Your production account is where you process live payments.

Important

A sandbox account cannot process live payments and is intended only for testing.

Follow these steps to sign up for a sandbox account:

1. Go to the Cybersource Developer Center sandbox account sign-up page: *https://developer.cybersource.com/hello-world/sandbox.html*
2. Enter your information into the sandbox account form, and click Create Account.



3. Go to your email and find a message titled: Merchant Registration Details. Click Set up your username and password now.
   Your browser opens the New User Sign Up wizard.
4. Enter the organization ID and contact email you supplied when you created your account. Follow the wizard pages to add your name, a username, and a password.
5. Log in to the Business Center.
   When you log in for the first time, you must verify your identity through a system-generated email sent to your email account.
6. Check your email for a message titled: Cybersource Identification Code. A passcode is included in the message.
7. Enter the passcode on the Verify your Identity page.
   You are directed to the Business Center home page.
   You have successfully signed up for a sandbox account.

> **Important**
>
> A sandbox account cannot process live payments. After you verify that your system can send and receive REST messages, you can contact customer service to transition your sandbox account to a production account.

Create a P12 Certificate

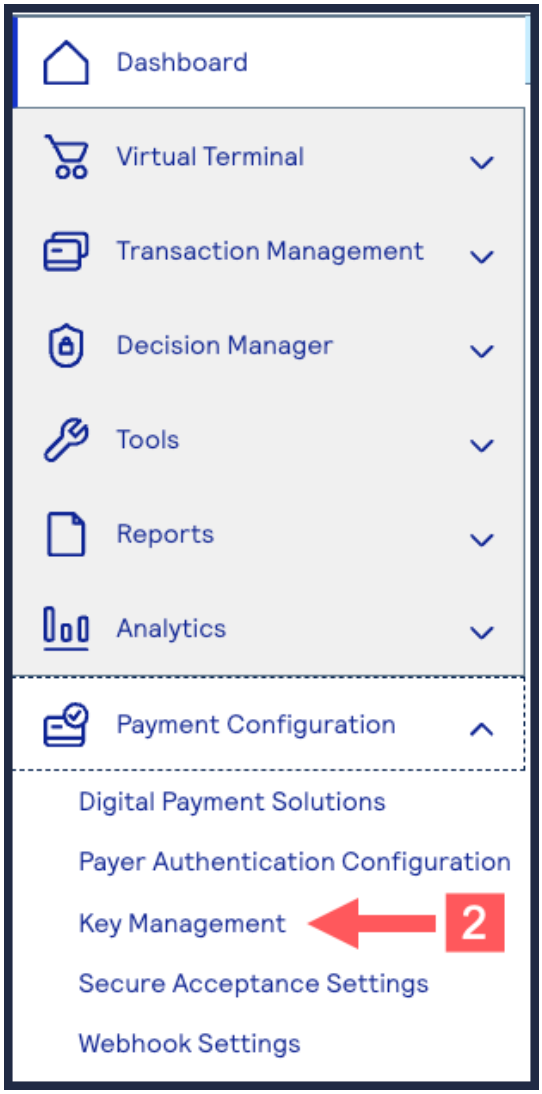
A P12 certificate and its private key are necessary for JSON Web Token message security. To create a P12 certificate, you must download a .p12 file from the Business Center and extract its private key.

Create a P12 File

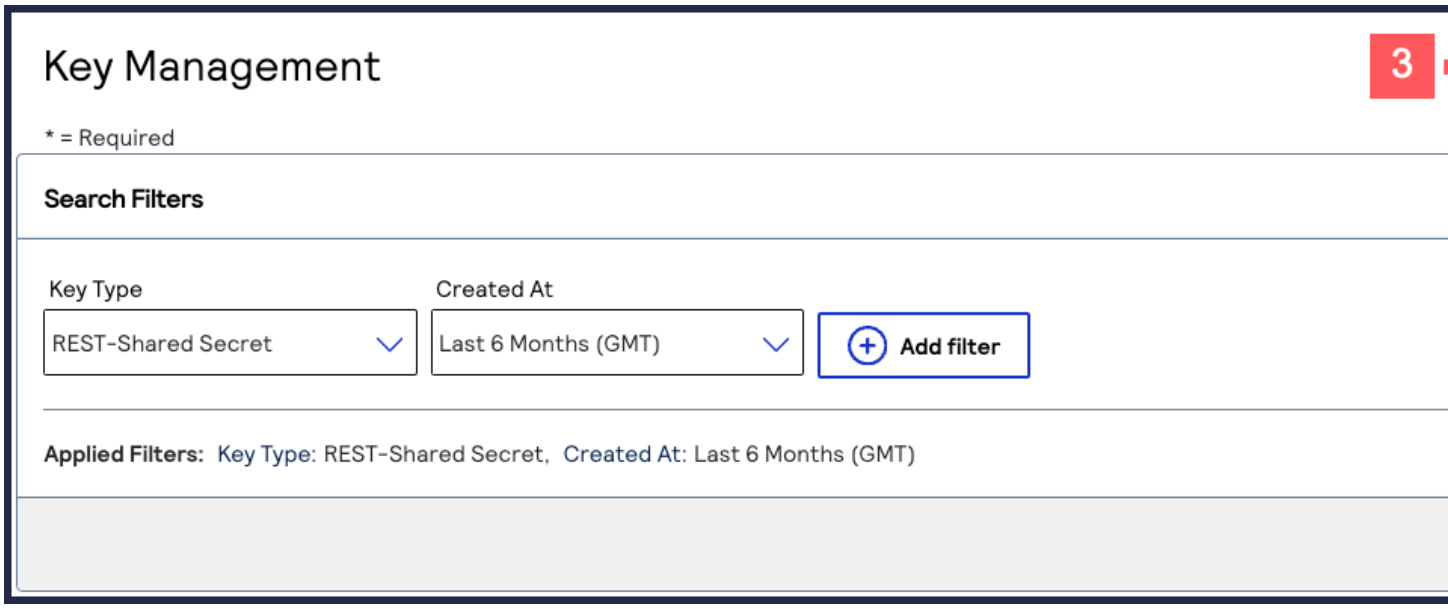
Follow these steps to create a .p12 file if you are using JSON Web Tokens to secure communication.

1. Log in to the Business Center:
   https://businesscentertest.cybersource.com

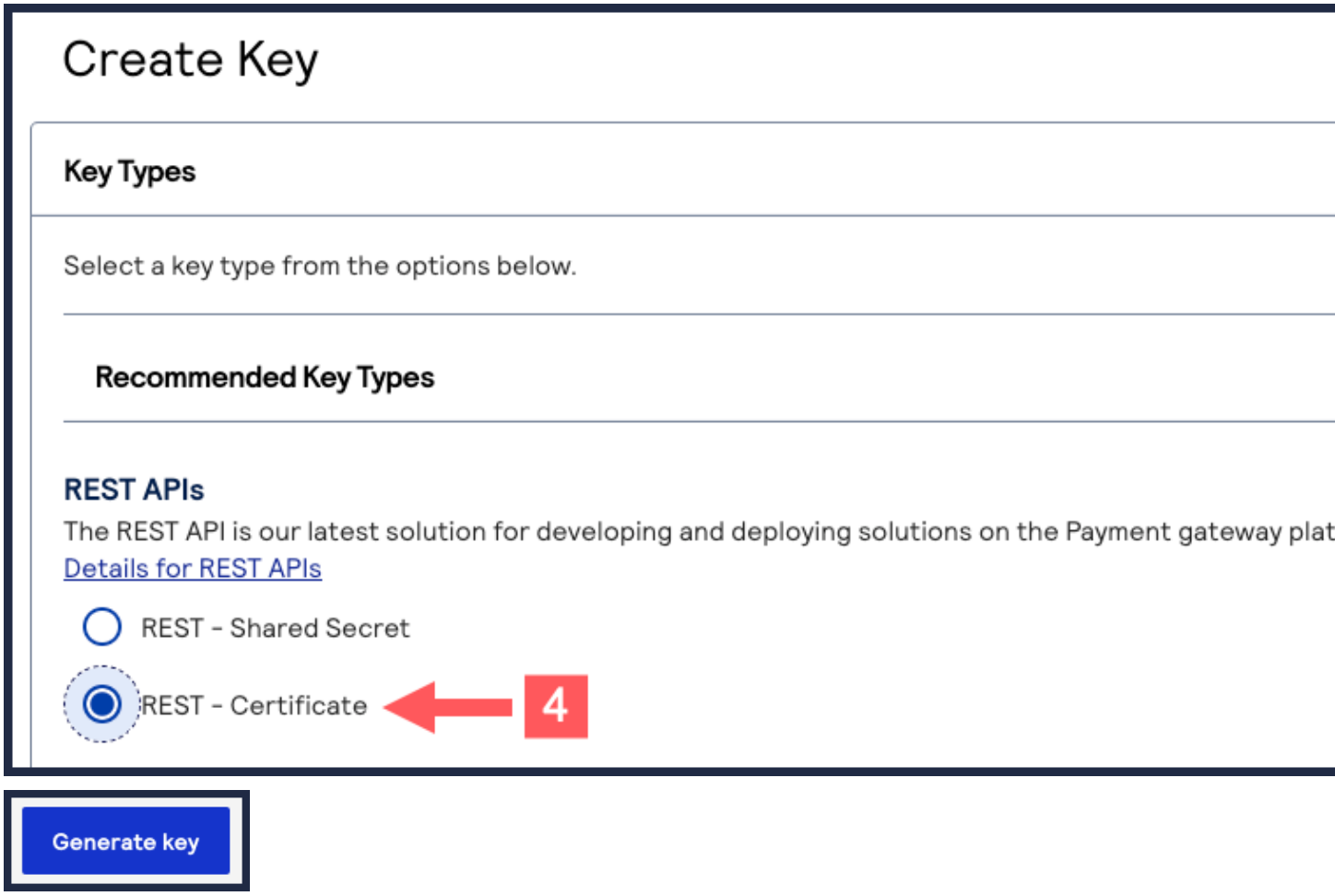
2. On the left navigation panel, navigate to



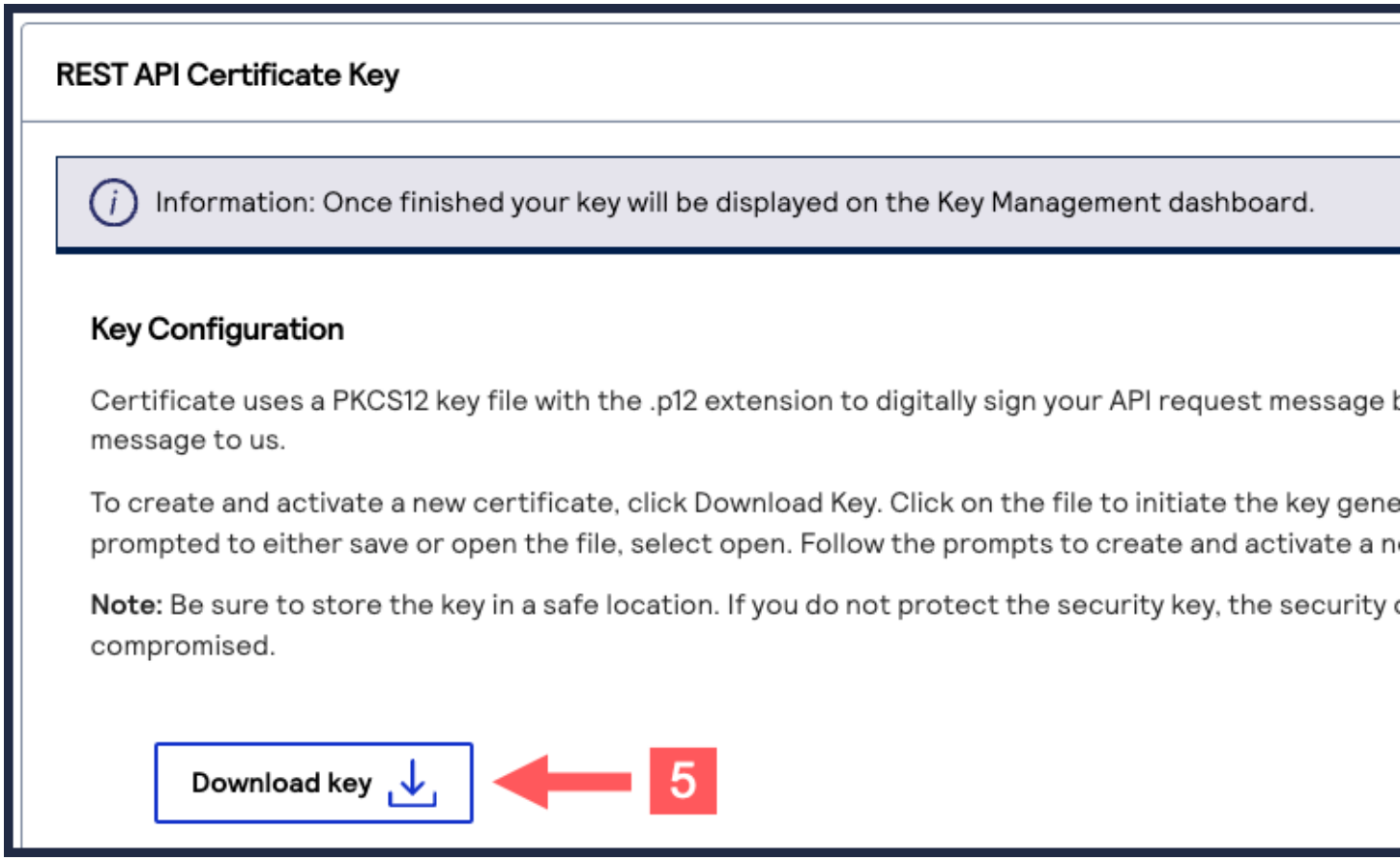
3. Click + Generate key.



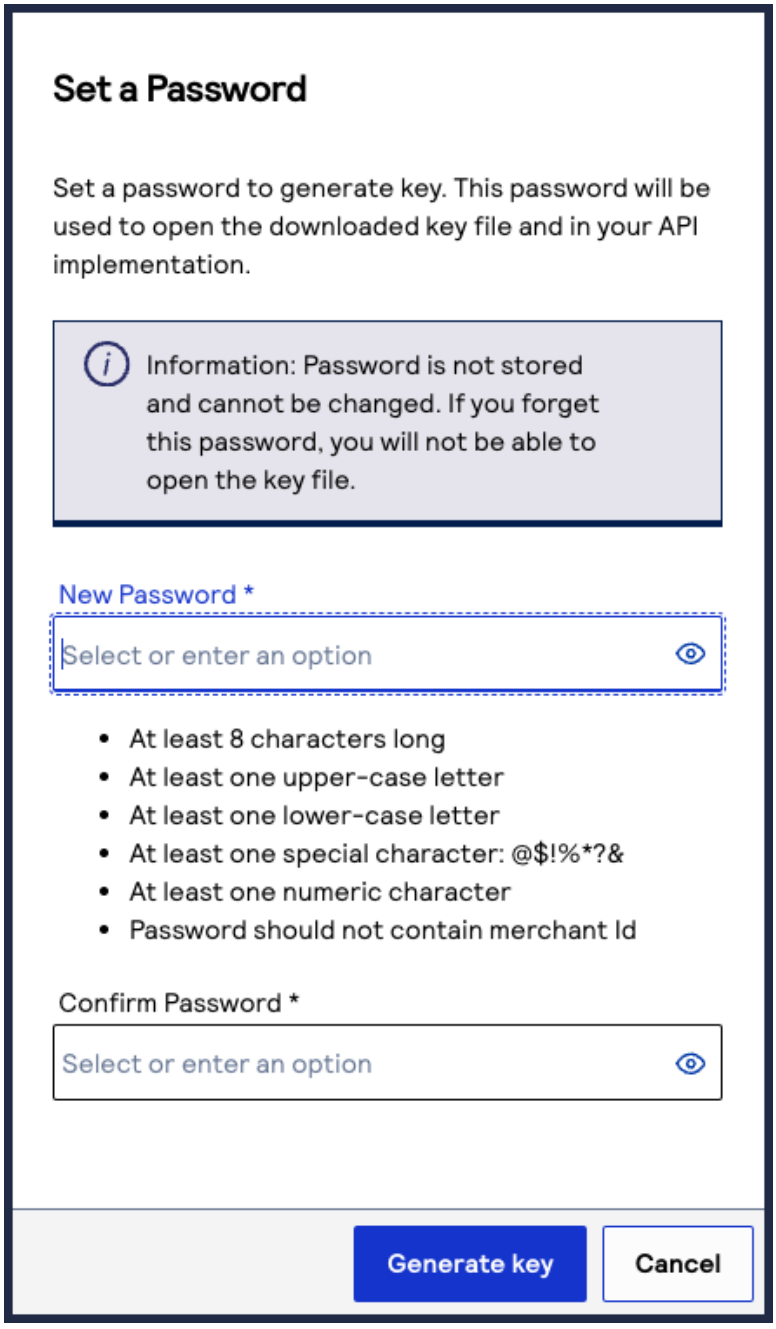
4. Under REST APIs, choose REST – Certificate, and then click Generate key.


Create Key
Key Types
Select a key type from the options below.
Recommended Key Types
REST APIs
The REST API is our latest solution for developing and deploying solutions on the Payment gateway plat
Details for REST APIs
REST - Shared Secret
REST - Certificate
4
Generate key

5. Click Download key



6. Create a password for the certificate by entering the password into the New Password and Confirm Password fields, and then click Generate key.



The .p12 file is downloaded to your desktop.

You generate multiple keys. View the keys on the Key Management page.

Extract the Private Key from the P12 Certificate

When you have your P12 certificate, extract the private key from the certificate. Use this key to sign your header when sending an API request.

Important

> If you are using the SDK to establish communication, you do not need to extract the private key from the P12 certificate.

Prerequisite

You must have a tool such as OpenSSL installed on your system.

Extract the Private Key

Follow these steps to extract the private key using OpenSSL:

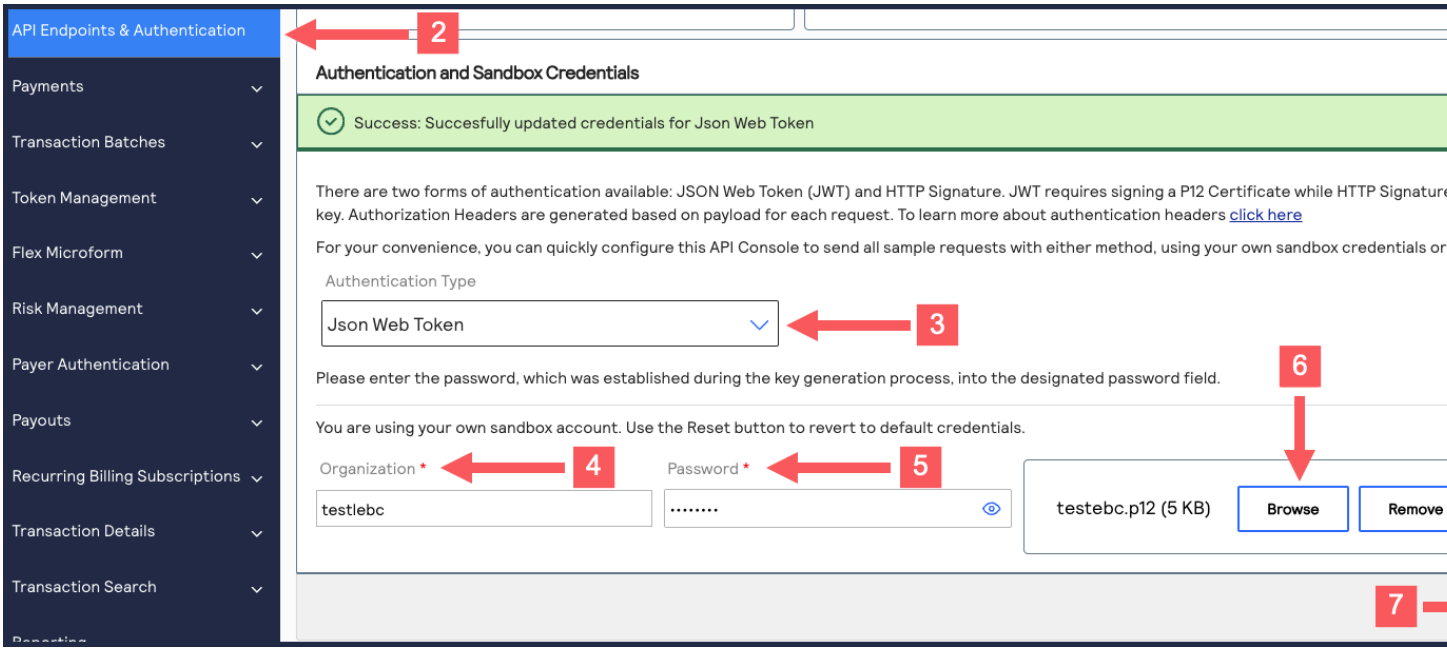
1. Open the command-line tool and navigate to the directory that contains the P12 certificate.
2. Enter this command:
   `openssl pkcs12 -in [certificate name] -nodes -nocerts -out [private key name]`
3. Enter the password for the certificate.
   You set this password when you created the P12 certificate in the Business Center.

The new certificate is added to the directory using the private key name you supplied in Step 2.

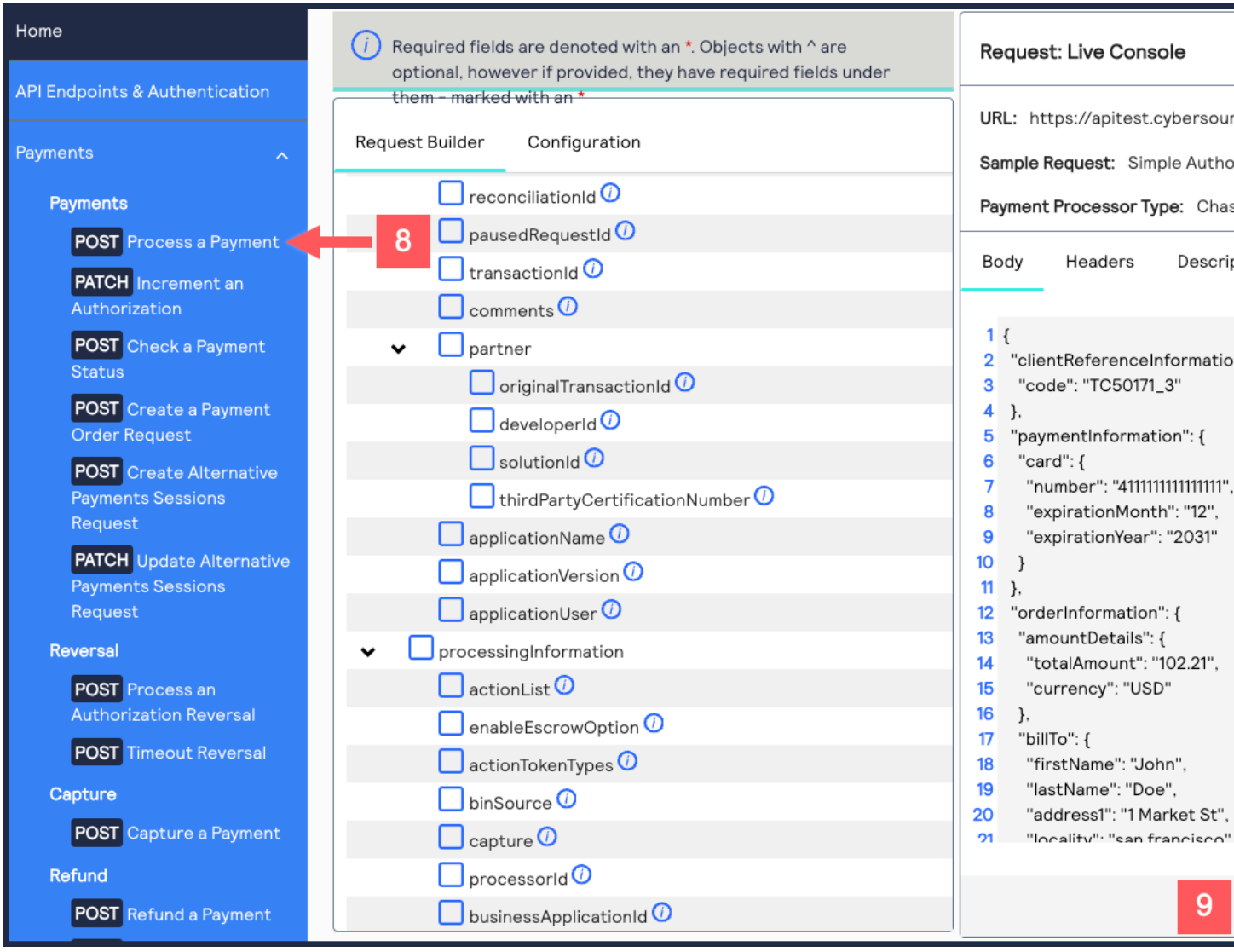
Test the Shared Secret Key Pair

After creating your key certificate, you must verify that it can successfully process API requests. This task explains how to test and validate your key pair in the Developer Center and the Business Center.

1. Go to the developer center's API Reference page:
   https://developer.cybersource.com/api-reference-assets/index.html#payments_payments_static-home-section
2. On the left navigation panel, click *API Endpoints & Authentication*.
3. Under Authentication and Sandbox Credentials, set the Authentication Type drop-down menu to JSON Web Token.
4. Enter your organization ID in the Organization field.
5. Enter your Password in the Password field.
6. Click Browse and upload your p12 certificate from your desktop.
7. Click Update Credentials.
   A confirmation message states that your credentials are successfully updated.



8. On the developer center's left navigation panel, navigate to Payments > POST Process a Payment.
9. Under Request: Live Console, click Send.

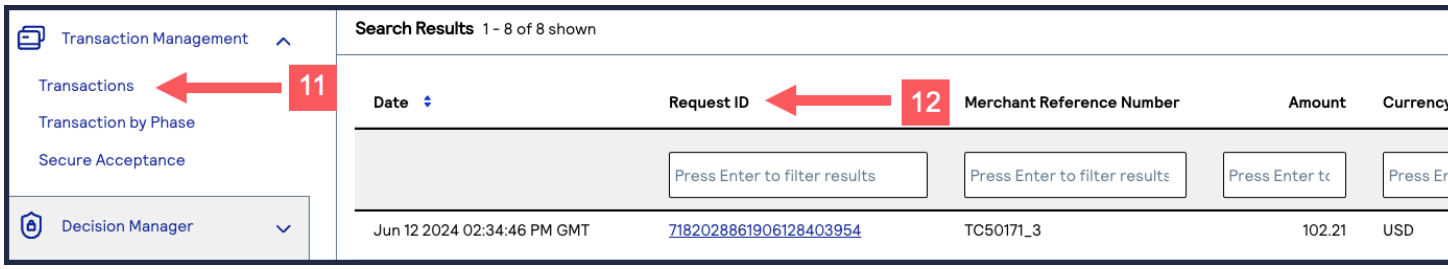


A message confirms that your request was successful with the status code 201.

10. Log in to the Business Center:
https://businesscentertest.cybersource.com

11. On the left navigation panel, navigate to

12. Under Search Results, verify that the request ID from the test authorization response is listed in the Request ID column.
    If the test authorization was successful, a success message is present in the corresponding Applications column.



Test Endpoints

When testing an API outside of the Developer Center's API Reference sandbox, send your test API requests to the test server:
https://apitest.cybersource.com
For example, to test an authorization request, you can send the request to this endpoint:
https://apitest.cybersource.com/pts/v2/payments

Construct Messages Using JSON Web Tokens

Follow these steps to construct messages using JWTs:

1. Generate a hash of the message body. See *Generate a Hash of the Message Body* on page 28.
   a. Optional: Encrypt the message. See *Message-Level Encryption Using JSON Web Tokens* on page 35.
2. Populate the header values. See *Generate the Token Header* on page 29.
3. Generate a hash of the claim set. See *Generate a Hash of the Claim Set* on page 30.
4. Generate a hash of the token header. See *Generate a Hash of the Token Header* on page 30.
5. Generate a token signature hash. See *Generate a Token Signature* on page 32.
6. Populate the **signature** header field. See *Generate a JSON Web Token* on page 32.
7. Create the JWT header. See *Create the Authorization Header* on page 33.

Elements of a JSON Web Token Message

A JWT Message is built with these elements:

Headers

Your message header must include these header fields:

Header Fields

Header Field	Description
v-c-merchant-id	Your Cybersource organization ID.
Date	The date of the transaction in the RFC1123 format. (Thu, 18 Jul 2019 00:18:03 GMT)
Content-Type	Also known as the Multipurpose Internet Mail Extension (MIME) type, this identifies the media or file type of the resource. (application/json)
kid	The ID of the key used to digitally sign the JWT. The Key ID (kid) must be registered with the authorizing server.
Host	The transaction endpoint. (api.cybersource.com)
alg	Algorithm used to sign the token header.

Body

The message body. For more information on setting up the body, see *Generate a Hash of the Message Body* on page 28.

Generate a Hash of the Message Body

Generate a Base64-encoded SHA-256 hash of the message, and place the hash in the header's **digest** field. This hash is used to validate the integrity of the message at the receiving end.
Follow these steps to generate the hash:

1. Generate the SHA-256 hash of the JSON payload (body of the message).
2. Encode the hashed string to Base64.
3. Add the message body hash to the **digest** payload field.
4. Add the hash algorithm used to the **digestAlgorithm** payload field.

Example: Digest Header Field

```
digest: RBNvo1WzZ4oRRq0W9+hknpT7T8If536DEMBg9hyq/4o=
```

Example: DigestAlgorithm Header Field

```
digestAlgorithm: SHA-256
```

Code Example: Creating a Message Hash Using C#

```
public static string GenerateDigest() {
    var digest = "";
    var bodyText = "{ your JSON payload }";
    using (var sha256hash = SHA256.Create()) {
```

```
        byte[] payloadBytes = sha256hash
            .ComputeHash(Encoding.UTF8.GetBytes(bodyText));
        digest = Convert.ToBase64String(payloadBytes);
        digest = digest;
    }
    return digest;
}
```

Code Example: Creating a Message Using Java

```
public static String GenerateDigest() throws NoSuchAlgorithmException {
    String bodyText = "{ your JSON payload }";
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    md.update(bodyText.getBytes(StandardCharsets.UTF_8));
    byte[] digest = md.digest();
    return Base64.getEncoder().encodeToString(digest);
}
```

Generate the Token Header

The token header is encrypted with a URL safe base64 algorithm. These three header fields must be included in the header.

Token Headers

Token Header Field	Description
kid	The ID of the key used to digitally sign the JWT.
alg	Algorithm used to sign the token header.
v-c-merchant-id	Merchant ID used in the request transaction. To obtain the merchant ID, see *Sign Up for a Sandbox Account* on page 15.

Token Header

```
eyJ2LWMtbWVyY2hhbnQtaWQiOiJtZXJjaGFudElEIiwiYWxnIjoiUlMyNTYiLCJraWQiOiI3MDc4NjMzMjg1MjUwMTc3MDQxNDk5In0
```

Generating the Token Header with Python

Encode the header data and then remove any padding added during the encryption process.

```
import base64

# open file in binary mode
data = b'{"v-c-merchant-id":"merchantID","alg":"RS256","kid":"70786332852501770414 99"}'
encoded = base64.urlsafe_b64encode(data)
stripped = encoded.decode('ascii').strip('=')

print(stripped)
```

Generate a Hash of the Claim Set

Generate a Base64-encoded SHA-256 hash of these header fields:

Headers

Header Field	Description
iat	The date and time of the message origin. Date formatting is defined by *RFC 7231, Section 7.1.1.1*.
digest	A Base64 encoded hash of the message payload. The digest field is not included in a GET request.
digestAlgorithm	The algorithm used to hash the message payload. The message payload should be hashed using the SHA-256 algorithm. The digestAlgorithm field is not included in a GET request.

Follow these steps to generate the hash:

1. Generate the SHA-256 hash of the fields in JSON format.
2. Encode the hashed string to Base64.
3. Add the message body hash to the **digest** header field.

Creating a Message Hash Using Command Line Tools

Generate the SHA-256 hash using the `shasum` tool.

```
echo -n "{"iat":"Thur, 15 June 2017 08:12:31 GMT","digest":"tP7hDajF4f6q0ysBQCHgef5K/PBq8iMASvlEARp8tl=",
    "digestAlgorithm":"SHA-256"}" | shasum -a 256
```

Base64 Encoding a Message Hash Using Command Line Tools

Generate the SHA-256 hash using the `base64` tool.

```
echo -n "5995a4f27b4b9256a94cf54489a9ef691d8dc8a590d322780d3b202cfa2f078f" | base64
```

Add the message body hash to the **digest** header field

```
NTk5NWE0ZjI3YjRiOTI1NmE5NGNmNTQ0ODlhOWVmNjkxZDhkYzhhNTkwZDMyMjc4MGQzYjIwMmNmYTJmMDc4Zg==
```

Generate a Hash of the Token Header

Generate a Base64-encoded SHA-256 hash of these header fields:

Token Headers

Token Header Field	Description
kid	The ID of the key used to digitally sign the JWT.
alg	Algorithm used to sign the token header.

Token Header Field	Description
v-c-merchant-id	Merchant ID used in the request transaction.

Follow these steps to generate the hash:

1. Generate the SHA-256 hash of the fields in JSON format.
2. Encode the hashed string to Base64.

Create a Message Hash Using the shasum Command Line Tool

```
echo -n "{"kid":"cc34c0a0-bd5a-4a3c-a50d-a2a7db7643df",
"alg":"RS256","v-c-merchant-id":"merchant_id"}"
| shasum -a 256
```

Create a Message Hash Using the base64 Command Line Tool

```
echo -n "a9953cdca19433ae5ec1c4eb0dafd41df6de4d20cd47cbace3c316a1ac6d2008" | base64
```

Example: Token Header Hash

```
NTc3N2RlOTAyZWEwNWU0NWM2YTBkNTI4Mjg0YTJmOTVlZGYxYWJlMzBjNzk5OTg1YzEzMjNiMDkzMzc0MWEwNA==
```

Generate the Message Body

Encode the message body (payload) using URL safe Base64 encryption. At a minimum, the body should include these fields:

Message Body Fields

Message Body Field	Description
digest	A base64 encoded SHA-256 has of the claim set.
digestAlgorithm	Algorithm used to sign the JWT.
iat	Time the JWT was issued.

Follow these steps to generate the hash:

1. Generate the SHA-256 hash of the JSON payload (body of the message).
2. Encode the hashed string to Base64.
3. Add the message body hash to the **digest** header field.
4. Add the hash algorithm used to the **digestAlgorithm** header field.

Encrypted Message Body

Line break added for readability.

```
digest: eyJkaWdlc3QiOiJSQk52bzFXelo0b1JScTBXOStoa25wVDdUOElmNTM2REVNQmc5aHlxLzRvPSIsImRpZ
2VzdEFsZ29yaXRobSI6IlNIQS0yNTYiLCJpYXQiOiIyMDI0LTA0LTA1VDE2OjI1OjE4LjI1OVoifQ
```

Encrypting Message Body Using Python

Generate the SHA-256 hash using the shasum tool. Line break on line three added for readability.

```
import base64

data = b'{"digest":"RBNvo1WzZ4oRRq0W9+hknpT7T8If536DEMBg9hyq/4o="","digestAlgorithm":"SHA-256",
   "iat":"2024-04-05T16:25:18.259Z"}'
encode = base64.urlsafe_b64encode(data)
stripped = encode.decode('ascii').strip('=')

print(stripped)
```

Generate a Token Signature

You can now build the JSON token signature. The token signature is made up of the JWT header and claim set hashes in the following format, and encrypted with the private key.
[Token Header].[Claim Set]
Follow these steps to generate the signature:

1. Concatenate the header and claim set hash strings with a period (.) separating the hashes:
   [Token Header].[Claim Set].
2. Generate an encoded version of the text file using your private key.
3. Base64 encode the signature output.

Example: Token Signature Hash

```
YjgwNGIxOTMxMzQ2NzhlYjdiMDdhMWZmYjZiYzUzNzliMTk5NzFmNjAzNWRmMThlNzk0N2NhY2U0YTEwNzYyYQ
```

Code Example: Encoding the Signature File Using OpenSSL

Encode the signature file using the openssl tool.

```
openssl rsautl -encrypt -inkey publickey.key -pubin -in [signature-text-file]
   > [signature-encoded-file]
```

Code Example: Base64 Encoding the Signature File Using the Command Line

Encode the signature file using the openssl tool and remove any padding.

```
base64 -i [signature-encoded-file]
```

Generate a JSON Web Token

You can now build the JWT. The JWT is made up of the token header Base64 encoded hash, the payload Base64 encoded hash, and the JWT signature in the following format:
[Token Header].[Payload].[Signature]

To generate the JWT, concatenate the header, payload, and signature strings with a period (.) separating the hashes:
[[Token Header].[Payload].[Signature].

Example: JSON Web Token

```
eyJ2LWMtbWVyY2hhbnQtaWQiOiJtZXJjaGFudElEIiwiYWxnIjoiUlMyNTYiLCJraWQiOiI3M
Dc4NjMzMjg1MjUwMTc3MDQxNDk5In0.eyJkaWdlc3QiOiJSQk52bzFXelo0b1JScTBXOStoa2
5wVDdUOElmNTM2REVNQmc5aHlxLzRvPSIsImRpZ2VzdEFsZ29yaXRobSI6IlNIQS0yNTYiLCJ
pYXQiOiIyMDI0LTA0LTA1VDE2OjI1OjE4LjI1OVoifQ.YjgwNGIxOTMxMzQ2NzhlYjdiMDdhM
WZmYjZiYzUzNzliMTk5NzFmNjAzNWRmMThlNzk0N2NhY2U0YTEwNzYyYQ
```

Create the Authorization Header

You can now build the transaction header using a JSON Web Token.
Your message header must include these header fields:

JWT Header Fields

JWT Header Field	Description
Authorization	The bearer token generated. (Bearer<JWT>)
Content-Type	Also known as the Multipurpose Internet Mail Extensi on (MIME) type, this identifies the me dia or file type of the resource. (appli cation/json)
Host	The transaction endpoint. (api.cybersource.com)

Example: JSON Web Token Header

```
Host: api.cybersource.com
Content-Type: application/json
Authorization: Bearer
eyJ2LWMtbWVyY2hhbnQtaWQiOiJtZXJjaGFudElEIiwiYWxnIjoiUlMyNTYiLCJraWQiOiI3M
Dc4NjMzMjg1MjUwMTc3MDQxNDk5In0.eyJkaWdlc3QiOiJSQk52bzFXelo0b1JScTBXOStoa2
5wVDdUOElmNTM2REVNQmc5aHlxLzRvPSIsImRpZ2VzdEFsZ29yaXRobSI6IlNIQS0yNTYiLCJ
pYXQiOiIyMDI0LTA0LTA1VDE2OjI1OjE4LjI1OVoifQ.YjgwNGIxOTMxMzQ2NzhlYjdiMDdhM
WZmYjZiYzUzNzliMTk5NzFmNjAzNWRmMThlNzk0N2NhY2U0YTEwNzYyYQ
```

Enable Message-Level Encryption

Important

This feature is in the pilot phase. To use message-level encryption, contact your sales representative.

> There are additional tasks you must complete before you can enable message-level encryption. For more information, see *Prerequisites for Message-Level Encryption* on page 34.

Message-Level Encryption (MLE) enables you to store information or communicate with other parties while helping to prevent uninvolved parties from understanding the stored information. MLE is optional and supported only for payments services.
MLE provides enhanced security for message payload by using an asymmetric encryption technique (public-key cryptography). The message encryption is implemented with symmetric encryption using Advanced Encryption Standard (AES), Galois Counter Mode (GCM) with 256-bit key size. The encryption of keys is supported using RSA Optimal Asymmetric Encryption Padding (OAEP) with 2048-bit key size. The encryption service is based on JSON Web Encryption (JWE), works on top of SSL and requires separate key-pairs for request and response legs of the transaction.
MLE is required for APIs that primarily deal with sensitive transaction data, both financial and non-financial. These are the types of sensitive transaction data:

- Personal identification information (PII)
- Personal account number (PAN)
- Personal account information (PAI)

MLE is supported when using JSON web tokens. For more information, see *Message-Level Encryption Using JSON Web Tokens* on page 35.
Each of these authentication schemes uses an encrypted payload, called the JWE. A JWE token has these five components, with each component separated by a period (.):

- JOSE header containing four elements:

```
"alg": "RSA-OAEP-256", //The algorithm used to encrypt the CEK
"enc": "A256GCM",  //The algorithm used to encrypt the message
"iat": "1702493653" //The current timestamp in milliseconds
"kid": "keyId" //The serial number of shared public cert for encryption of CEK
```

- JWE encrypted key
- JWE initialization vector
- JWE additional authentication data (AAD)
- JWE ciphertext and authentication tag

Prerequisites for Message-Level Encryption

Before enabling message-level encryption (MLE), you must complete these requirements:

1. Sign the pilot agreement for using MLE.
2. Confirm that the APIs you are integrating to support MLE.
3. Retrieve the Cybersource public key from either the Account Manager or Client Executive services in the Business Center.
4. Ensure that client-side systems are modified to read the public key and encrypt the API payload.

Message-Level Encryption Using JSON Web Tokens

To use message-level encryption (MLE) with JSON Web Tokens (JWT), you must generate the JWT and send it as part of the HTTP header. The payload is encrypted with the dynamic Content Encryption key (CEK) that is generated for each transaction. The serialized encrypted payload, the JWE, is passed as the request body.

1. Use the required Maven dependency:

```
<dependency>
  <groupId>com.nimbusds</groupId>
  <artifactId>nimbus-jose-jwt</artifactId>
  <version>9.0</version>
</dependency>
```

2. Prepare the API request payload. This example is hard-coded for demonstration.

```
String jsonMsg = "{\"clientReferenceInformation\":{\"code\":\"TC50171_3\"},\"processingInformation\":
{\"commerceIndicator\":\"internet\"},\"aggregatorInformation\":{\"subMerchant\":{\"cardAcceptorID
\":\"1234567890\",\"country\":\"US\",\"phoneNumber\":\"650-432-0000\",\"address1\":\"900MetroCenter
\",\"postalCode\":\"94404-2775\",\"locality\":\"FosterCity\",\"name\":\"VisaInc\",\"administrativeArea
\":\"CA\",\"region\":\"PEN\",\"email\":\test@cybs.com\},\"name\":\"V-Internatio\",\"aggregatorID\":
\"123456789\"},\"orderInformation\":{\"billTo\":{\"country\":\"US\",\"lastName\":\"VDP\",\"address2\":
\"Address2\",\"address1\":\"201S.DivisionSt.\",\"postalCode\":\"48104-2201\",\"locality\":\"AnnArbor
\",\"administrativeArea\":\"MI\",\"firstName\":\"RTS\",\"phoneNumber\":\"999999999\",\"district\":
\"MI\",\"buildingNumber\":\"123\",\"company\":\"Visa\",\"email\":\test@cybs.com\},\"amountDetails\":
{\"totalAmount\":\"102.21\",\"currency\":\"USD\"}},\"paymentInformation\":{\"card\":{\"expirationYear\":
\"2031\",\"number\":\"5555555555554444\",\"securityCode\":\"123\",\"expirationMonth\":\"12\",\"type\":
\"002\"}}}";
```

3. Read the merchant p12 file.
   The P12 file should have been created when you set up your test account. See *Create a P12 File* on page 16.

```
ClassLoader classLoader = Main.class.getClassLoader();
KeyStore merchantKeyStore = KeyStore.getInstance("PKCS12", new BouncyCastleProvider());
merchantKeyStore.load(classLoader.getResourceAsStream("test_merchant.p12"),
 "test_merchant".toCharArray());
String merchantKeyAlias = null;
Enumeration enumKeyStore = merchantKeyStore.aliases();
RSAPrivateKey rsaPrivateKey = null;
RSAPrivateKey rsaPrivateKey_SJC = null;
X509Certificate x509Certificate = null;
X509Certificate x509Certificate_SJC = null;
```

4. Loop through the Java KeyStore to extract the private key from merchant p12 file and extract the public key for Cybersource. You must use the Cybersource SJC (`CyberSource_SJC_US`) to encrypt the payload.

```
while (enumKeyStore.hasMoreElements()) {
        merchantKeyAlias = (String) enumKeyStore.nextElement();
        if (merchantKeyAlias.contains("test_merchant")) {
           KeyStore.PrivateKeyEntry keyEntry = (KeyStore.PrivateKeyEntry) merchantKeyStore.getEntry(
                merchantKeyAlias, new KeyStore.PasswordProtection(
```

```
                    "test_merchant".toCharArray()));
 //Extract the merchant certificate to sign the payload.
          x509Certificate = (X509Certificate) keyEntry.getCertificate();
          rsaPrivateKey = (RSAPrivateKey) keyEntry.getPrivateKey();

     //Extract the merchant certificate to encrypt the payload.
       } else if (merchantKeyAlias.contains("CyberSource_SJC_US")) {
          KeyStore.PrivateKeyEntry keyEntry = (KeyStore.PrivateKeyEntry) merchantKeyStore.getEntry(
               merchantKeyAlias, new KeyStore.PasswordProtection(
                    "test_merchant".toCharArray()));

 //Store the public key from the certificate.
          x509Certificate_SJC = (X509Certificate) keyEntry.getCertificate();
          //rsaPrivateKey_SJC = (RSAPrivateKey) keyEntry.getPrivateKey();
       }
    }
```

5. Update the custom headers to include `"iat"` with the current timestamp:

```
Map<String, Object> customHeaders = new HashMap<String, Object>();
customHeaders.put("iat", Instant.now().getEpochSecond());
```

6. Generate the JWE token (the encrypted payload) using the supported algorithm and the Cybersource public certificate. Include the JSON payload as the input.

```
String jweToken = encryptAttributeWithAlgo(jsonMsg, x509Certificate_SJC,
 JWEAlgorithm.RSA_OAEP_256, EncryptionMethod.256GCM, customHeaders);
```

```
public static String encryptAttributeWithAlgo(String content, X509Certificate x509Certificate,
 JWEAlgorithm algo, EncryptionMethod encryptionMethod, Map<String, Object> customHeaders) {
     if (isNullOrEmpty(content)) {
        System.out.println("empty or null content");
        return null;
     } else if ( x509Certificate == null) {
        System.out.println("public certificate is null");
        return null;
     }
     String serialNumber = extractSerialNumberFromDN(x509Certificate);
     JWEObject jweObject = new JWEObject(
          new JWEHeader.Builder(algo, encryptionMethod)
               .contentType("JWT") // required to signal nested JWT
               .keyID(serialNumber)
               .customParams(customHeaders)
               .build(),
          new Payload(content));

     jweObject = encrypt(jweObject, x509Certificate);
     return jweObject == null ? null : serializeToken(jweObject);
   }
public static boolean isNullOrEmpty(String string) {
     return (string == null || string.trim().length() == 0);
   }
```

7. Build the JSON request body for calling the Cybersource API.

```
String jsonBody = createJsonString(jweToken);
```

```
private static String createJsonString(String jweToken) {
      String message;
      JSONObject json = new JSONObject();
      json.put("encryptedRequest", jweToken);
      return json.toString();
   }
```

8. Generate the body digest to validate that the payload has not been compromised.

```
String bodyDigest = createBodyDigest(jsonBody);
```

```
public static String createBodyDigest(String jsonBody) {
      MessageDigest messageDigest = null;
      try {
         messageDigest = MessageDigest.getInstance(DEFAULT_HASH_ALG);
      } catch (NoSuchAlgorithmException e) {
         System.out.println("Couldn't instantiate SHA-256 digest " + e.getMessage());
         return null;
      }
      byte[] bodyDigestBytes = messageDigest.digest(jsonBody.getBytes());
      return java.util.Base64.getEncoder().encodeToString(bodyDigestBytes);
   }
```

9. Prepare the JWT payload for signature.

```
JWTPayload jwtPayload = createJWTPayloadClass(bodyDigest);
Map<String, Object> customHeader = new HashMap<String, Object>();
customHeader.put("v-c-merchant-id", "test_merchant");
```

```
private static JWTPayload createJWTPayloadClass(String bodyDigest) throws
 NoSuchAlgorithmException {
      JWTPayload jwtPayload = new JWTPayload();
      jwtPayload.setDigest(bodyDigest);
      jwtPayload.setDigestAlgorithm("SHA-256");
      jwtPayload.setIat(String.valueOf(System.currentTimeMillis()));
      return jwtPayload;
   }
```

10. Sign the payload and create the JWT token that is passed in the request header.

```
String jwsSignature = sign(Json.encode(jwtPayload), rsaPrivateKey, x509Certificate, customHeader);
```

```
public static String sign(String content, PrivateKey privateKey, X509Certificate x509Certificate,
 Map<String, ? extends
         Object>
         customHeaders) {
      return serializeToken(signPayload(content, privateKey, x509Certificate, customHeaders));
   }
protected static JOSEObject signPayload(String content, PrivateKey privateKey, X509Certificate
 x509Certificate,
```

```
                                Map<String, ? extends Object> customHeaders) {
    return signPayload(content, privateKey, x509Certificate, customHeaders, true);
  }
protected static JOSEObject signPayload(String content, PrivateKey privateKey, X509Certificate
 x509Certificate,
                                Map<String, ? extends Object> customHeaders, boolean includeKid) {
    if (isNullOrEmpty(content) || x509Certificate == null || privateKey == null) {
      System.out.println("empty or null content or Private key  or public certificate is null");
      return null;
    }

    String serialNumber = extractSerialNumberFromDN(x509Certificate);
    List<Base64> x5cBase64List = addCertificateToBase64List(x509Certificate);
    if (x5cBase64List.isEmpty()) return null;

    RSAPrivateKey rsaPrivateKey = (RSAPrivateKey) privateKey;
    Payload payload = new Payload(content);
    JWSHeader jwsHeader;
    if ( includeKid ) {
      jwsHeader = new JWSHeader.Builder(JWSAlgorithm.RS256)
          .customParams((Map<String, Object>) customHeaders)
          .keyID(serialNumber)
          .x509CertChain(x5cBase64List)
          .build();
    } else {
      jwsHeader = new JWSHeader.Builder(JWSAlgorithm.RS256)
          .customParams((Map<String, Object>) customHeaders)
          .x509CertChain(x5cBase64List)
          .build();
    }
    JWSObject jwsObject = new JWSObject(jwsHeader, payload);
    try {
      RSASSASigner signer = new RSASSASigner(rsaPrivateKey, true);
      jwsObject.sign(signer);
      if (!jwsObject.getState().equals(JWSObject.State.SIGNED)) {
        System.out.println("Payload signing failed.");
        return null;
      }
    } catch (JOSEException joseException) {
      System.out.println("ERROR_SIGN_AND_ENCRYPT_THE_PAYLOAD" + " " + joseException);
      return null;
    }
    return jwsObject;
  }
protected static String extractSerialNumberFromDN(X509Certificate x509Certificate) {
    String serialNumber = null;
    String serialNumberPrefix = "SERIALNUMBER=";
    String principal = x509Certificate.getSubjectDN().getName().toUpperCase();
    int beg = principal.indexOf(serialNumberPrefix);
    if (beg >= 0) {
      int end = principal.indexOf(",", beg);
      if (end == -1) end = principal.length();
      serialNumber = principal.substring(beg + serialNumberPrefix.length(), end);
    } else
      serialNumber = x509Certificate.getSerialNumber().toString();
    return serialNumber;
```

```
  }
```

11. Send the JWT as the Bearer token in the header and send the JWE as the body.

```
HTTP Request Header:
Content-Type: application/json
Authorization: Bearer
 eyJ2LWMtbWVyY2hhbnQtaWQiOiJtcG9zX3BheW1lbnRlY2giLCJhbGciOiJSUzI1NiIsImtpZCI6Im1wb3NfcGF5bWVudGVjaCIsI
jphx0dH6jzuJxoerkgz3VSMuDt9mDJtnlDnisSTf35NWh6u4TeJqGr3E8oOOJSX6N32r6XovCXyJyaDm4h2fJOeLZc8HcvfSC5SpM
_OwElAYZfOnZ9FphLQZWnwZ3mku0C6gysv6ISMrI9BlCpWaPbmDeuWBSLexC_U01cblg

HTTP Request Body:
{"encryptedRequest":"eyJ2LWMtbWVyY2hhbnQtaWQiOiJtcG9zX3BheW1lbnRlY2giLCJ4NWMiOlsiTUlJRE5UQ0NBaDJnQXdJQ
GCEaFs15Ul_BrtlhQTn9aKjX_-rbYxM-ZXJlbpg6CsyAqy63-
MkYPP2BNXjFfP3yUSxes76zHlMaJG0gp681QY85AqGq6mCSrDqWE7NUTWifseRtKMv5u9pMHMxddkz9Xvp6Q5TbiEjGZbvD-
xKhhgs0-IupvPDKhxdJSNVPaDiTnFVnYtyOuLZLOFO4Fq2bfj86iGHRjfh9zq91Gp4uN36kmRHzkLN4Wrr5R6D79Z-
FC5bLU4BUrilGQtVSWCWtcxYAIQOhz1w.tuv-9Xtl0uNoPxXV.RRGnkA1chplnGQf-SlXaXEntzGJrEF4EJU-
F6PEx6H3us1APoWAR-26aHdWctNFoGSalNt1ZzidRi3TA-iwpSFkEonSVbe7aVLJeAKgqCHnVXT-
eWb89gqTVkQFZiSZCHtIjDUtOMy95sU4MRcCvtrfAPDnIMudVVA5YtAsCZpta_AT1_iS6oLBMI57R0Ra7pO3MxFdLTrk-
FkLSd4JbGokm_JXpH8lI1V11vaMAtyEqGrzllrQv408zUGbvtvSirF31iiGITEF7QG5rbVn7oTWF4wWzKEkpSZ7J4LpIdjCG6sojeld4
vFVfa-
ua4uh4PNcVK0o3ke4TOqLnVcnaEtYW1AS2wIu_tHxW_hdkyPmDI8ceSBqmloRxV3q8xOS5u-2GNQ9p5pm2_NjkqVB8RYup9NFZW
RzJ0mOwPF2MZzQ318z78IyAGXotYT4QXGJZhnyDMNgHjyyGX7IZtGPRYDpxc10Kko9DLM_r6fWoDLemRhFbi8prnlJpQZbUh98TLF
```

Going Live

When you are ready to process payments in a live environment, you must transition your account to a live status with a valid configuration for your chosen payment processor. When live, your transaction data flows through the production Cybersource gateway, to your processor, and on to the appropriate payment network.
To transition your account:

1. Sign up for a merchant account.
2. *Contact sales* to establish a contract with Cybersource that enables you to process real transactions and receive support.
3. Submit a merchant ID (MID) activation request.

It may take up to three business days to complete a MID activation request.

Create a Merchant ID

The merchant ID (MID) is used to identify you and your transactions and is included in the header of each transaction request. When you signed up for a sandbox account, you received a merchant ID for testing purposes. If you choose, you can use that merchant ID as your production ID.
Follow these steps to sign up for a merchant account in order to create a production MID:

1. Navigate to the Business Center *Evaluation Account Sign-up page*, enter the required information, and click Create Account.

Choose your merchant ID name carefully. It cannot be changed. This name is not visible to your customers.

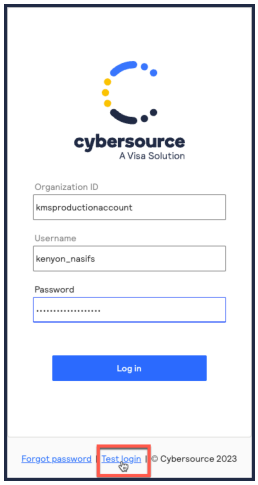
2. Review your information entered, especially your business email address. Your merchant ID registration information will be sent to the email entered on this form.
3. Check your email from customer support titled: Cybersource Merchant Evaluation Account.
   This email will include the Organization ID and contact email associated with your MID.
4. Go to your email and find a message titled: Merchant Registration Details. Click the Set up your username and password now link.
   Your browser opens the New User Sign Up wizard.
5. Enter the Organization ID and Contact email you supplied previously. Follow the wizard pages to add your name, a username, and password.
6. Log into the Business Center.
   When you log in for the first time, you will be asked to identify your identity through a system-generated email that is sent to your email account.
7. Check your email for a message titled: Cybersource Identification Code.
   Note the passcode.
8. Enter the passcode on the Verify your Identity page.
   You should be directed to the Business Center home page.

You have successfully created a merchant ID and merchant account.

Activate your Merchant ID

The activation process, also known as going live, transitions your MID and account from test status to live status, enabling you to process real transactions in production. It may take up to three business days to complete the MID activation request.
To transition your account complete these tasks:

1. Sign in to the *Support Center* as an administrator.
2. Enter your credentials and log in to your test environment.



   Enter your MID in the Organization ID text box.
3. Go to Support Cases > MID Configuration Request. The MID Configuration Request page should be open.

4. Select MID Activation.
5. In the Description field, enter the Merchant ID that you want to take live.
6. Select the processor configuration and enter the name of your processor.
   If you are unsure of your processor name, contact your merchant service provider or your merchant acquiring bank.
7. Select the environments that this change applies (test or production).
8. Select Service Enablement and list the products and services that you intend to use.
9. Select Submit.

Production Endpoints

When sending API request messages using your production account, send your requests to the production server:
`https://api.cybersource.com`
For example, to send a live authorization request, you can send the request to this endpoint:
`https://api.cybersource.com/pts/v2/payments`

Set Up an HTTP Signature Message

To set up your HTTP signature message requires you to follow these steps.

Set Up HTTP Signature Message Workflow

1. Sign up and register a sandbox account. See *Sign Up for a Sandbox Account* on page 15.
2. Create a shared secret key. See *Shared Secret Key Pair* on page 43.
3. Construct a message using HTTP signature security. See *Construct Messages Using HTTP Signature Security* on page 53.
4. Go live by transitioning your sandbox account into a production account. *Going Live* on page 39.

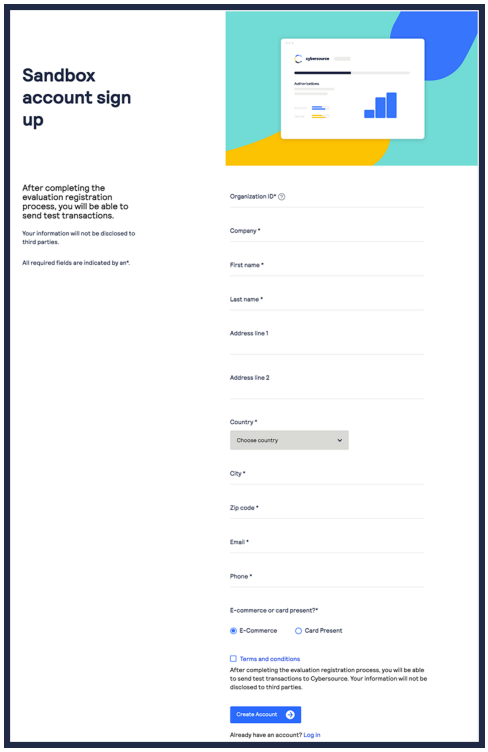
Sign Up for a Sandbox Account

To begin setting up your account, you must first sign up for a sandbox account. A sandbox account enables you to obtain your security keys and test your implementation. After you verify that your system can send and receive REST messages, you can contact customer service to transition your sandbox account to a production account. Your production account is where you process live payments.

> **Important**
>
> A sandbox account cannot process live payments and is intended only for testing.

Follow these steps to sign up for a sandbox account:

1. Go to the Cybersource Developer Center sandbox account sign-up page: *https://developer.cybersource.com/hello-world/sandbox.html*
2. Enter your information into the sandbox account form, and click Create Account.



3. Go to your email and find a message titled: Merchant Registration Details. Click Set up your username and password now.
   Your browser opens the New User Sign Up wizard.
4. Enter the organization ID and contact email you supplied when you created your account. Follow the wizard pages to add your name, a username, and a password.
5. Log in to the Business Center.
   When you log in for the first time, you must verify your identity through a system-generated email sent to your email account.
6. Check your email for a message titled: Cybersource Identification Code. A passcode is included in the message.
7. Enter the passcode on the Verify your Identity page.
   You are directed to the Business Center home page.
   You have successfully signed up for a sandbox account.

> **Important**
>
> A sandbox account cannot process live payments. After you verify that your system can send and receive REST messages, you can contact customer service to transition your sandbox account to a production account.

Shared Secret Key Pair

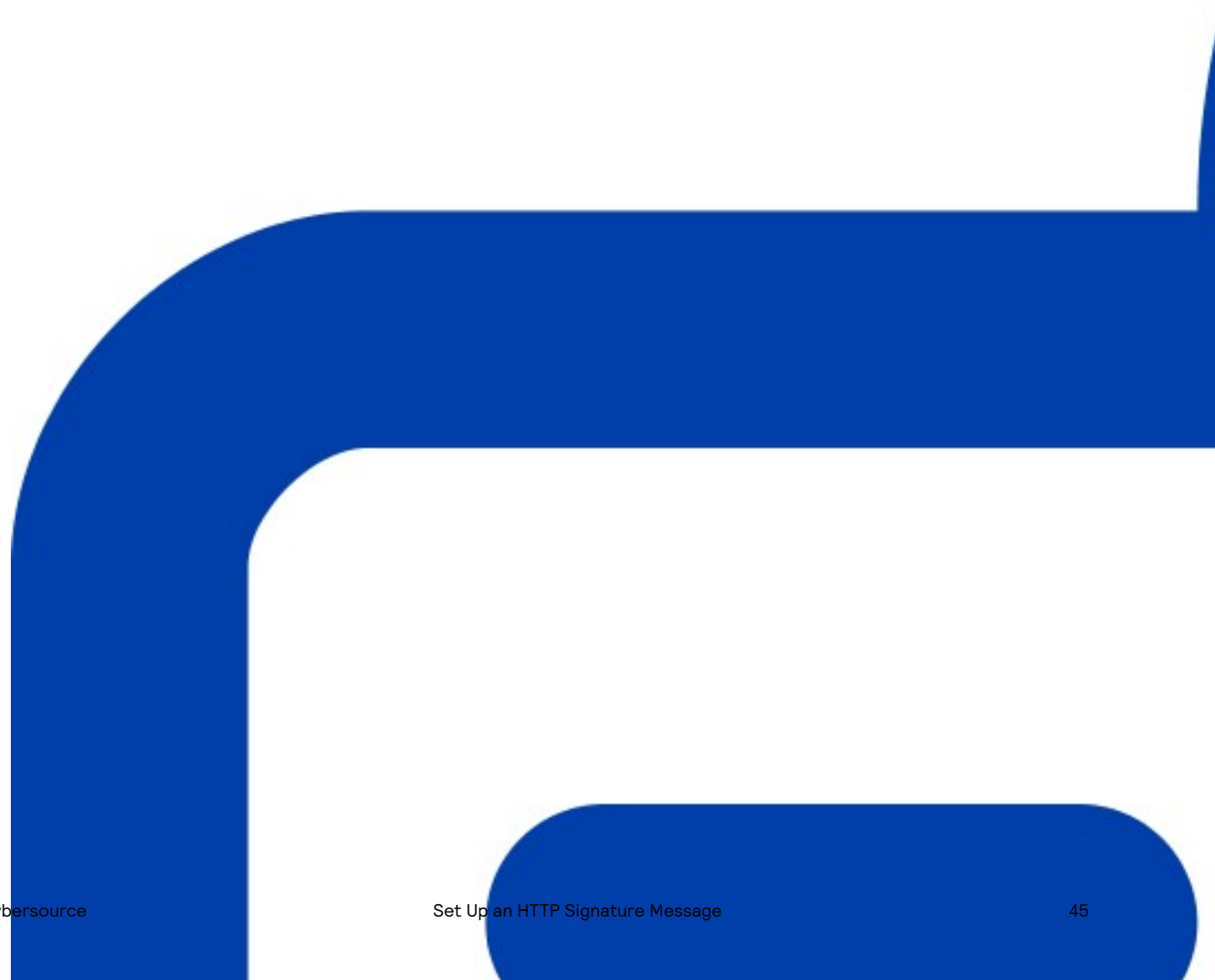
Key pairs are used with HTTP Signature message security.

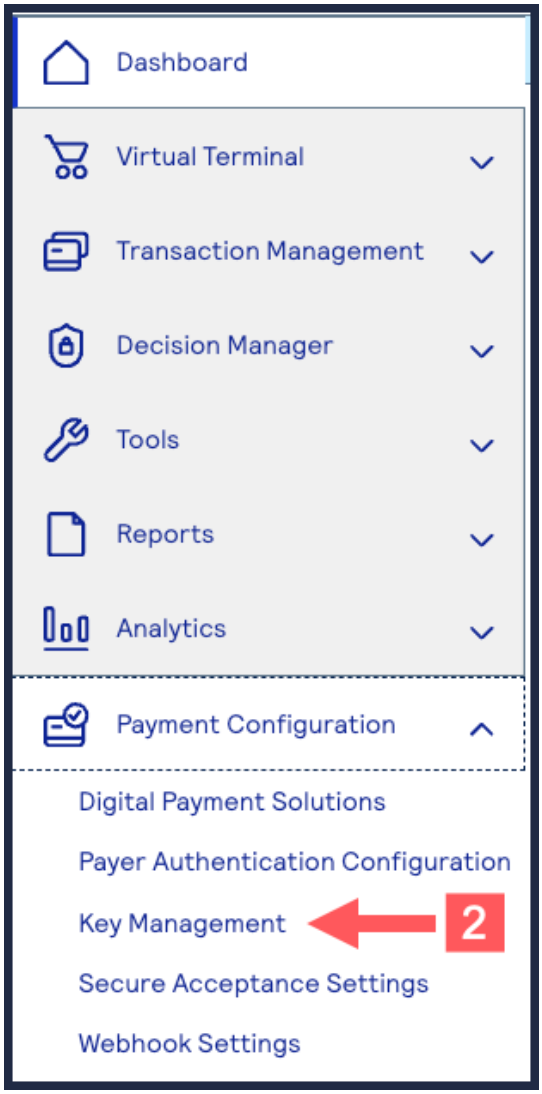
Create a Shared Secret Key Pair

Follow these steps to create a shared secret key pair.

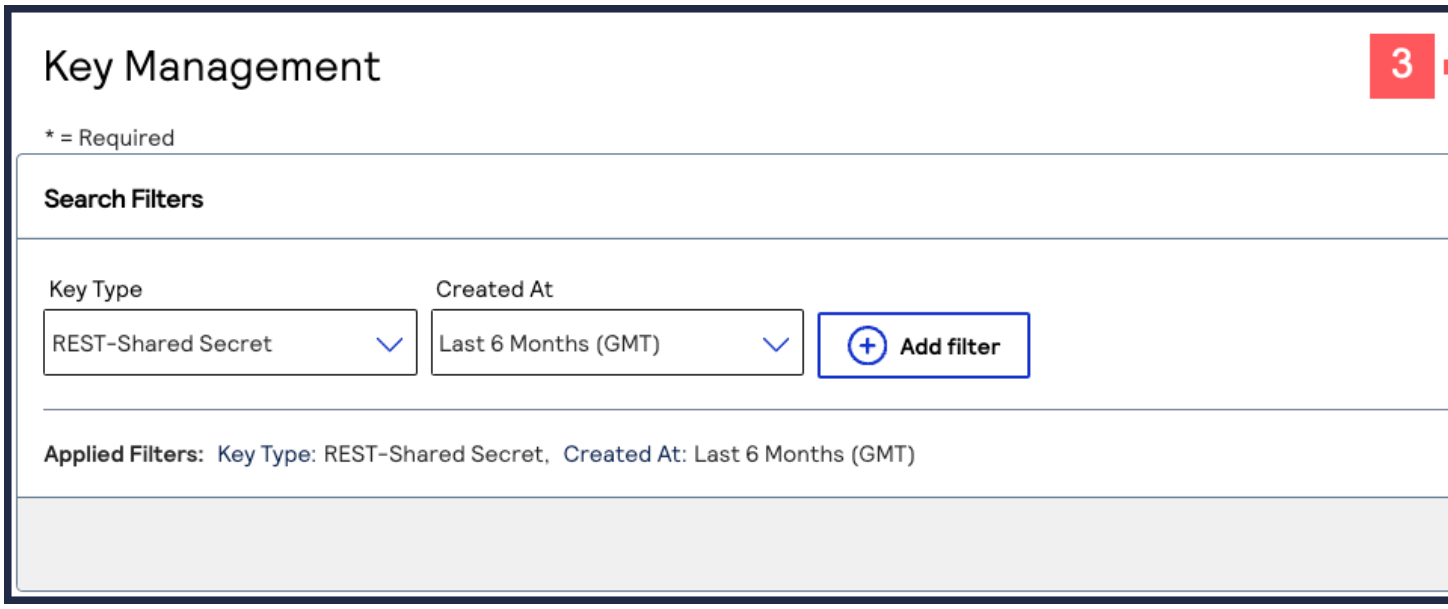
1. Log in to the Business Center:
   https://businesscentertest.cybersource.com

2. On the left navigation panel, navigate to

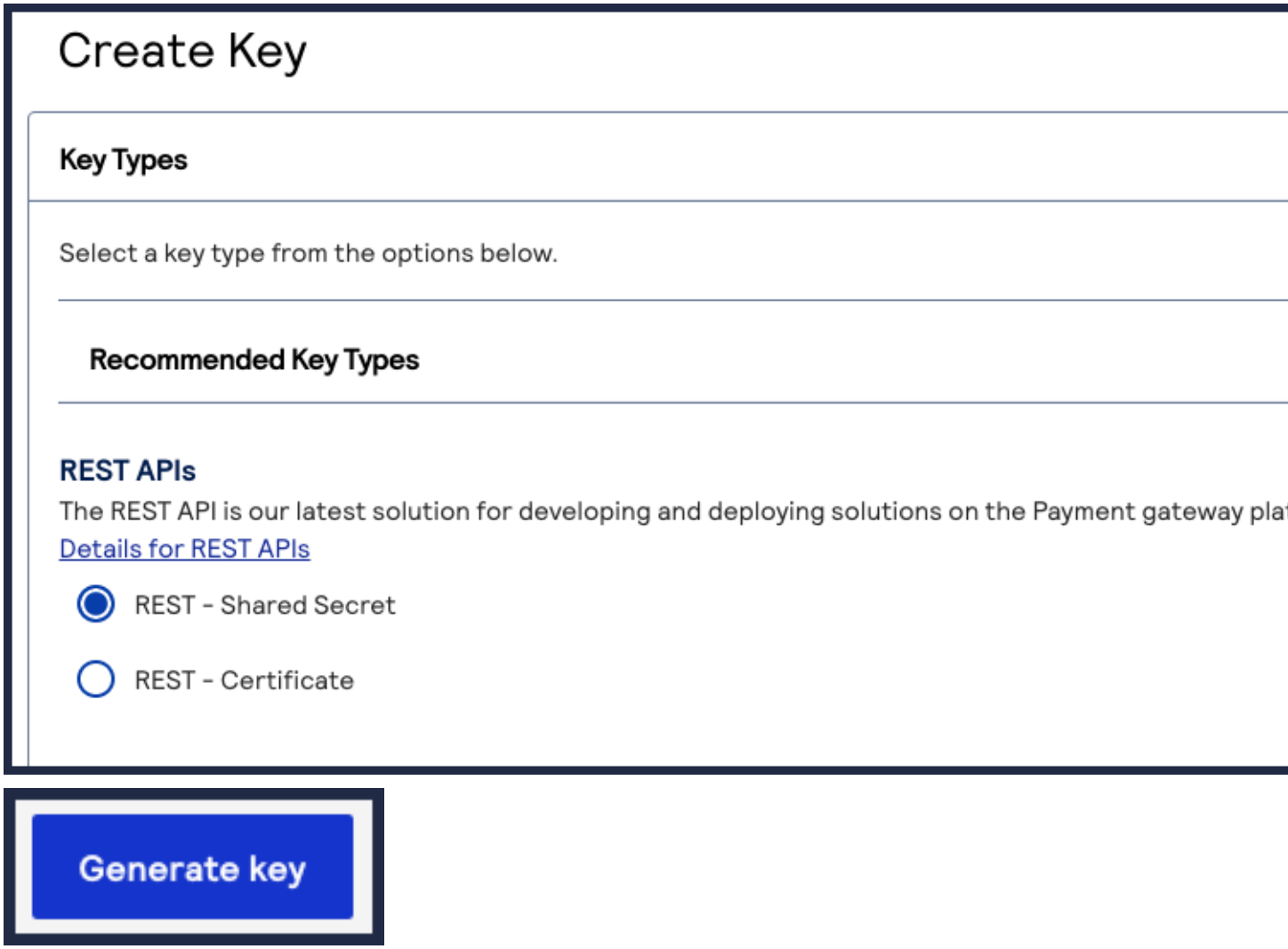




3. Click + Generate key.



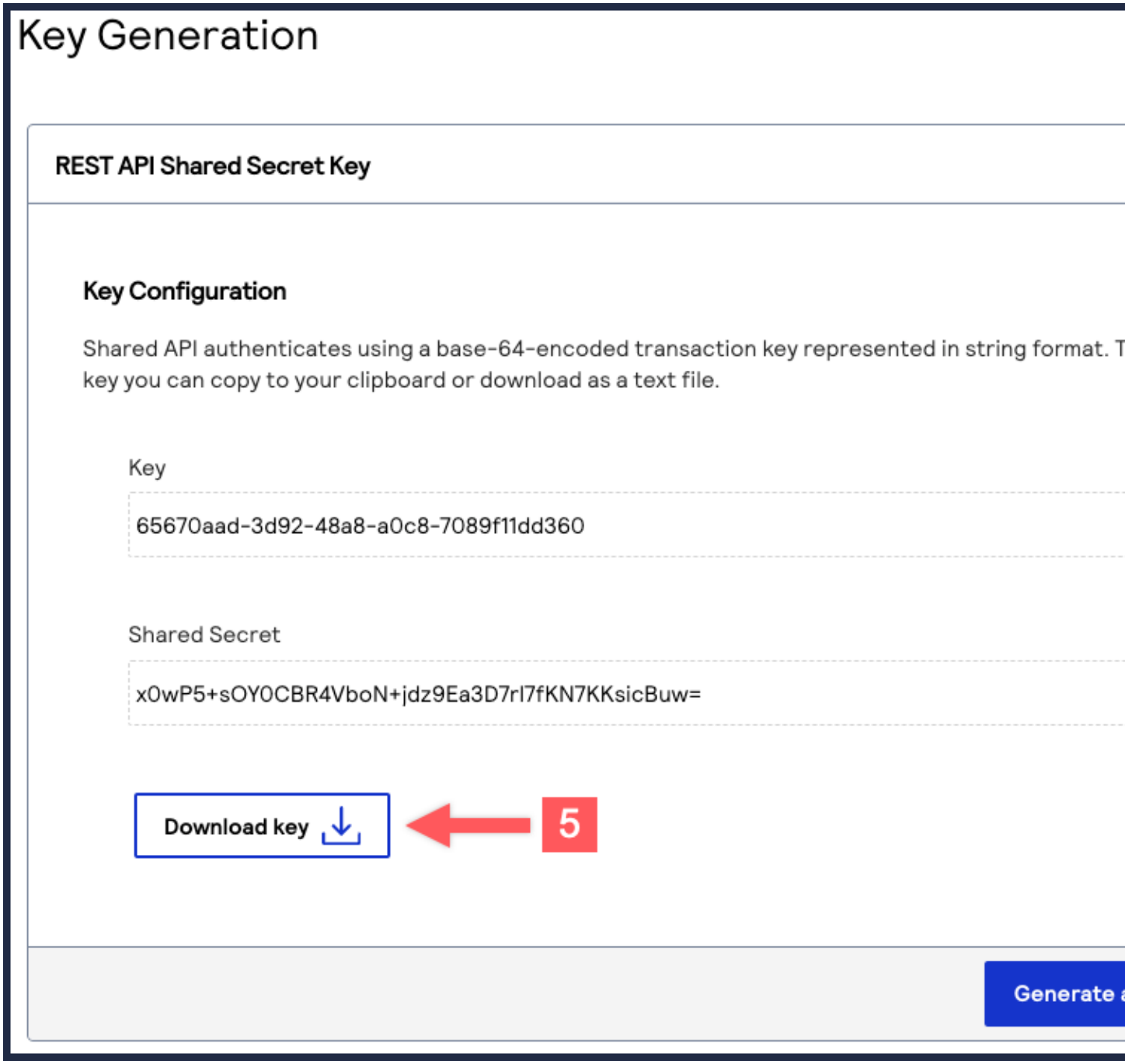
4. Under REST APIs, select REST – Shared Secret and then click Generate key.



The REST API Shared Secret Key page appears.

5. Click Download key

The .pem file is downloaded to your desktop.



You generate multiple keys. View the keys on the Key Management page.

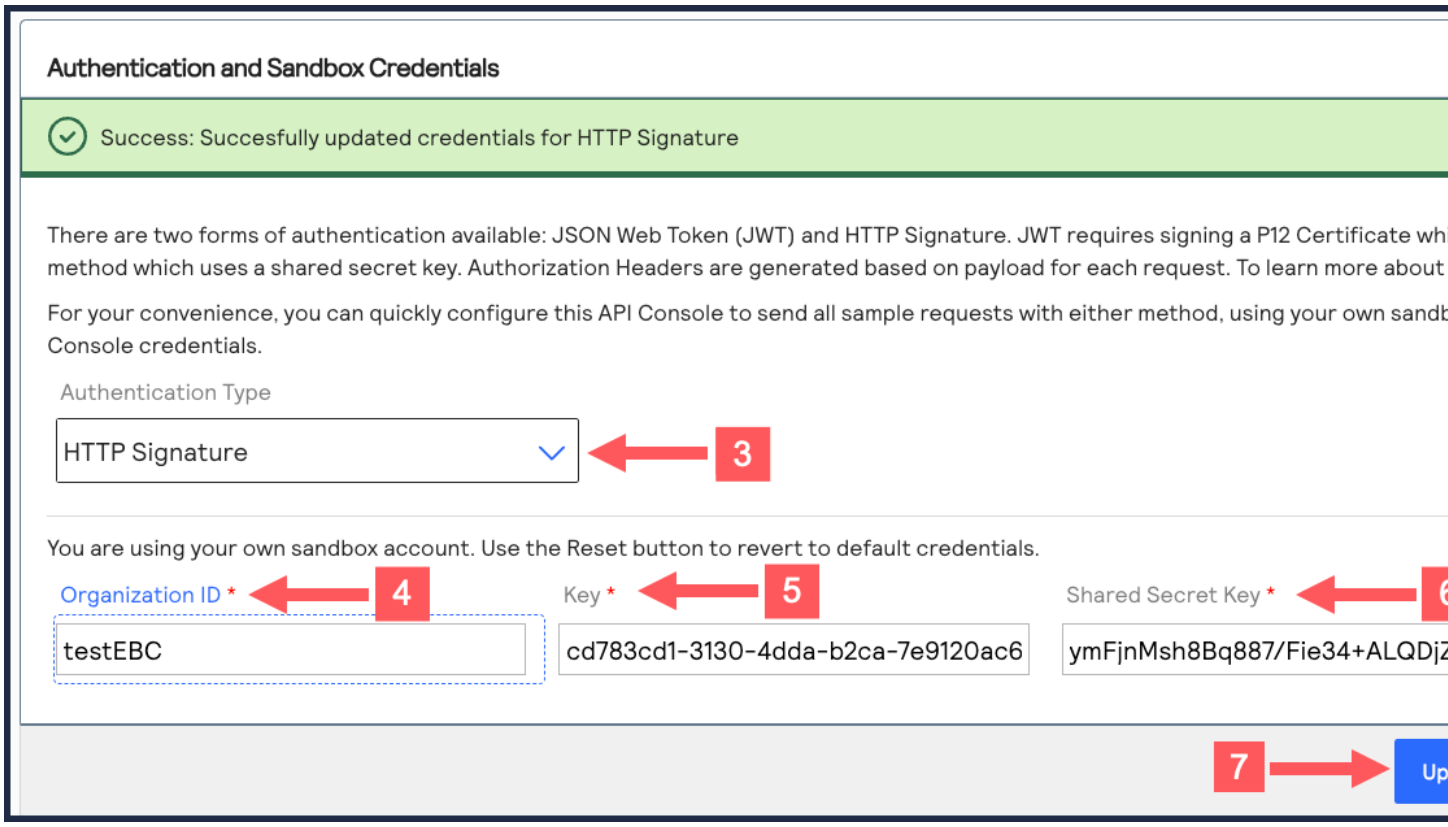
Test the Shared Secret Key Pair

After creating your key certificate, you must test and verify that your key can successfully process API requests. These tasks explain how to test and validate your key certificate using the developer center and the Business Center.

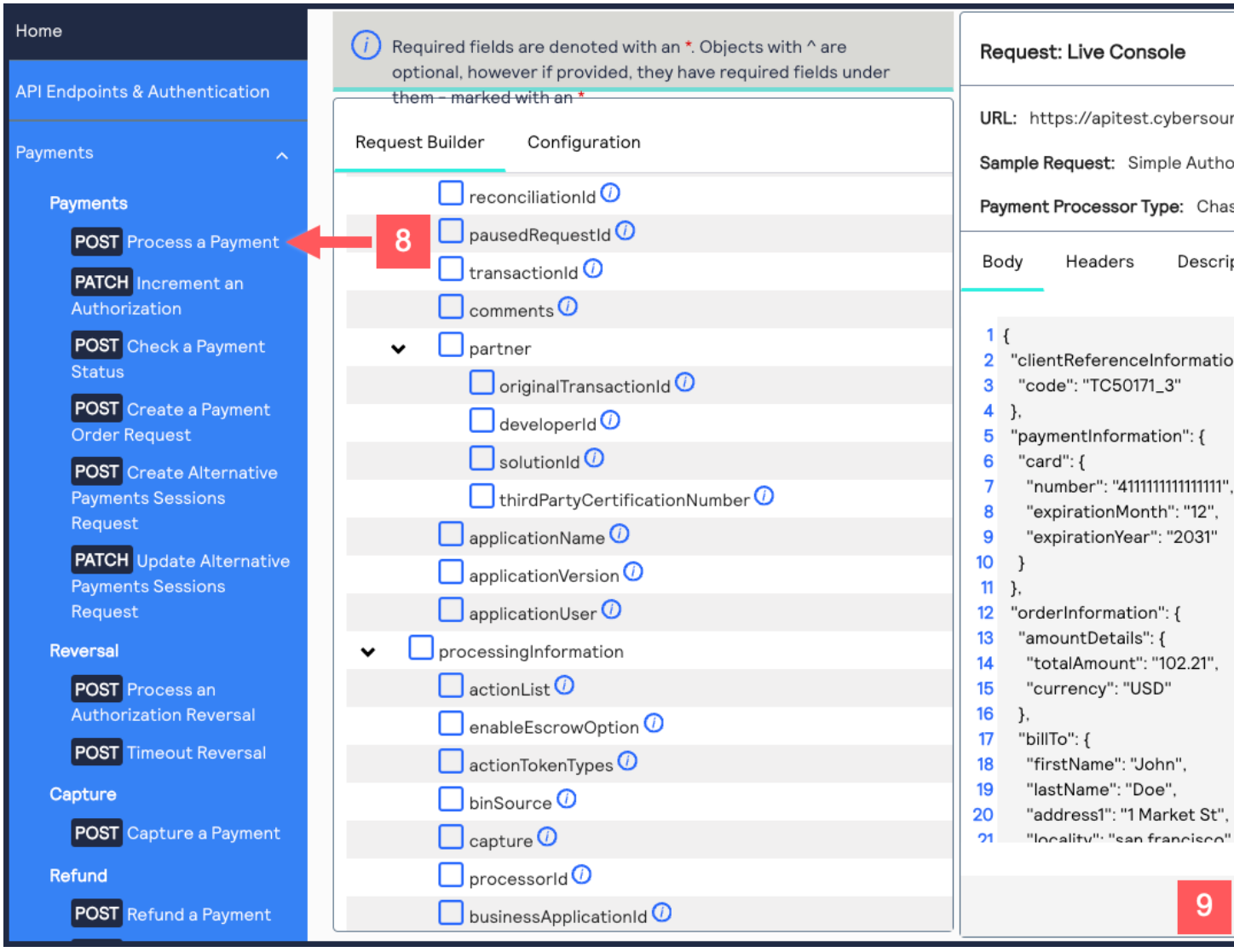
1. Go to the developer center's API Reference page:

https://developer.cybersource.com/api-reference-assets/index.html#payments_payments_static-home-section

2. On the left navigation panel, click *API Endpoints & Authentication*.
3. Under Authentication and Sandbox Credentials, set the Authentication Type drop-down menu to HTTP Signature.
4. Enter your organization ID in the Organization ID field.
5. Enter your key, also known as your private key, in the Key field.
6. Enter your secret key, also known as your public key, in the Shared Secret Key field.
7. Click Update Credentials.



8. On the developer center's left navigation panel, navigate to Payments > POST Process a Payment.
9. Under Request: Live Console, click Send.

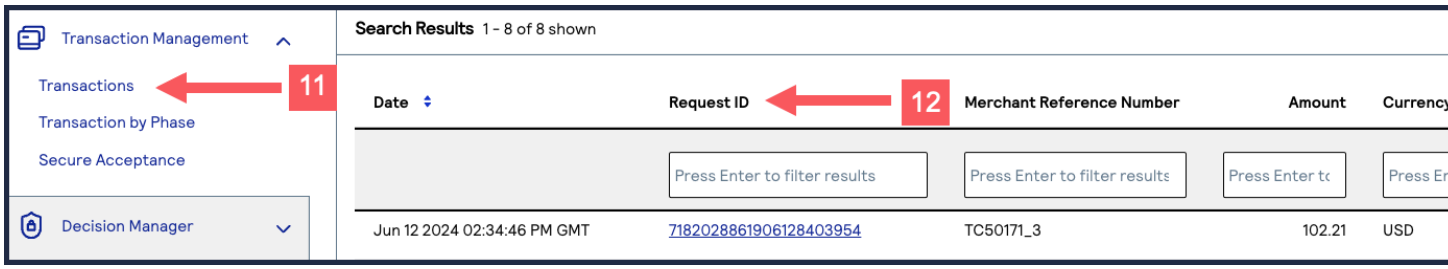


A message confirms that your request was successful with the status code 201.

10. Log in to the Business Center:
https://businesscentertest.cybersource.com

11. On the left navigation panel, navigate to

12. Under Search Results, verify that the request ID from the test authorization response is listed in the Request ID column.
    If the test authorization was successful, a success message is present in the corresponding Applications column.



Test Endpoints

When testing an API outside of the Developer Center's API Reference sandbox, send your test API requests to the test server:
https://apitest.cybersource.com
For example, to test an authorization request, you can send the request to this endpoint:
https://apitest.cybersource.com/pts/v2/payments

Construct Messages Using HTTP Signature Security

HTTP signatures use a digital signature to enable the receiver to validate the sender's authenticity and ensure that the message was not tampered with during transit. For more information about HTTP signatures, see the IETF Draft that is maintained by the IETF HTTP Working Group (*https://httpwg.org*).
Follow these steps to implement HTTP signatures:

1. Create the shared secret key pair. See *Create a Shared Secret Key Pair* on page 44.
2. Generate a hash of the message body. See *Generate a Hash of the Message Body* on page 54.
3. Generate a signature hash. See *Generate the Signature Hash* on page 55.
4. Populate the **signature** header field. See *Update Header Fields* on page 56.

Elements of an HTTP Message

A HTTP Message is built with the following elements:

Headers

Your message header must include these header fields:

HTTP Header Fields

HTTP Header Field	Description
v-c-merchant-id	Your Cybersource organization ID.
v-c-date	The date of the transaction in the RFC1123 format. (Thu, 18 Jul 2019 00:18:03 GMT)
Content-Type	Also known as the Multipurpose Internet Mail Extension (MIME) type, this identifies the media or file type of the resource. (application/json)
Host	The transaction endpoint. (api.cybersource.com)

Body

The message body. For more information on setting up the body, see *Generate a Hash of the Message Body* on page 54.

Generate a Hash of the Message Body

This hash is used to validate the integrity of the message at the receiving end. Follow these steps to generate the hash:

1. Generate the SHA-256 hash of the JSON payload (body of the message).
2. Encode the hashed string to Base64.
3. Prepend SHA-256= to the front of the hash.
4. Add the message body hash to the **digest** header field.

Creating a Message Hash Using the Command Line `shasum` Tool

```
echo -n "{"clientReferenceInformation":{"code":"TC50171_3"},"paymentInformation":{"card":{"number":
  "4111111111111111","expirationMonth":"12","expirationYear":"2031"}},"orderInformation":{"amountDetails":
  {"totalAmount":"102.21","currency":"USD"},"billTo":{"firstName":"John","lastName":"Doe","address1":
  "1MarketSt","locality":"sanfrancisco","administrativeArea":"CA","postalCode":"94105","country":"US",
  "email":"test@cybs.com","phoneNumber":"4158880000"}}}" | shasum -a 256
```

```
echo -n "6ae5459bc8a7d6a4b203e8a734d6a616725134088e13261f5bbcefc1424fc956" | base64
```

Creating a Message Hash Using the Command Line `base64` Tool

```
echo -n "6ae5459bc8a7d6a4b203e8a734d6a616725134088e13261f5bbcefc1424fc956" | base64
```

Creating a Message Hash Using C#

```
public static string GenerateDigest() {
  var digest = "";
  var bodyText = "{ your JSON payload }";
  using (var sha256hash = SHA256.Create()) {
  byte[] payloadBytes = sha256hash
  .ComputeHash(Encoding.UTF8.GetBytes(bodyText));
  digest = Convert.ToBase64String(payloadBytes);
  digest = "SHA-256=" + digest;
```

```
    }
    return digest;
    }
```

Creating a Message Using Java

```
public static String GenerateDigest() throws NoSuchAlgorithmException {
    String bodyText = "{ your JSON payload }";
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    md.update(bodyText.getBytes(StandardCharsets.UTF_8));
    byte[] digest = md.digest();
    return "SHA-256=" + Base64.getEncoder().encodeToString(digest);
    }
```

Digest Header Field

```
digest:
 SHA-256=NmFlNTQ1OWJjOGE3ZDZhNGIyMDNlOGE3MzRkNmE2MTY3MjUxMzQwODhlMTMyNjFmNWJiY2VmYzE0MjRmYzk1Ng==
```

Generate the Signature Hash

The signature hash is a Base64-encoded HMAC SHA-256 hash of the header fields and their values. The following information must be included in the signature hash:

Header Fields

Header Field	Description
Date	From the header, the date and time in the RFC1123 format . For example: Date: Thu, 18 Jul 2023, 22:18:03.
Digest	The Base64-encoded SHA-256 hash of the message body . For more information, see Generate a Hash of the Message Body. For example: Digest: SHA-256=gXWufV4Zc7VkN9Wkv9jh/JuAVclqDusx3vkyo3uJFWU=. Do not include the digest with GET requests.
Host	From the header, the endpoint host. For example: apitest.cybersource.com.
v-c-merchant-id	From the header, the merchant ID associated with the request. For example: v-c-merchant-id: mymerchantid.
request-target	The HTTP method and endpoint resource path. For example: request-target: post /pts/v2/payments/.

Follow these steps to generate the signature hash value:

1. Generate a byte array of the secret key generated previously. For more information, see *Create a Shared Secret Key Pair* on page 44.
2. Generate the HMAC SHA-256 key object using the byte array of the secret key.
3. Concatenate a string of the required information listed above.
   For more information, see Creating the Validation String below.

4. Generate a byte array of the validation string.
5. Use the HMAC SHA-256 key object to create the HMAC SHA-256 hash of the validation string byte array.
6. Base64 encode the HMAC SHA-256 hash.

Signature Hash

```
signature=”OuKeDxj+Mg2Bh9cBnZ/25IXJs5n+qj93FvPKYpnqtTE=”
```

Creating the Validation String

To create the validation string, concatenate the required information in the same order as listed in the signature header field parameter. Each item must be on a separate line, and each line should be terminated with a new line character `\n`.

Validation String Example

```
host: apitest.cybersource.com\n
date: Thu, 18 Jul 2019 00:18:03 GMT\n
request-target: post /pts/v2/payments/\n
digest: SHA-256=gXWufV4Zc7VkN9Wkv9jh/JuAVclqDusx3vkyo3uJFWU=\n
v-c-merchant-id: mymerchantid
```

Generating a Signature Hash in C#

```
private static string GenerateSignatureFromParams(string signatureParams, string secretKey) {
var sigBytes = Encoding.UTF8.GetBytes(signatureParams);
var decodedSecret = Convert.FromBase64String(secretKey);
var hmacSha256 = new HMACSHA256(decodedSecret);
var messageHash = hmacSha256.ComputeHash(sigBytes);
return Convert.ToBase64String(messageHash);
}
```

Generating a Signature Hash in Java

```
public static String GenerateSignatureFromParams(String keyString,
String signatureParams) throws InvalidKeyException, NoSuchAlgorithmException {
byte[] decodedKey = Base64.getDecoder().decode(keyString);
SecretKey originalKey = new SecretKeySpec(decodedKey, 0, decodedKey.length, "HmacSHA256");
Mac hmacSha256 = Mac.getInstance("HmacSHA256");
hmacSha256.init(originalKey);
hmacSha256.update(signatureParams.getBytes());
byte[] HmachSha256DigestBytes = hmacSha256.doFinal();
return Base64.getEncoder().encodeToString(HmachSha256DigestBytes);}
```

Update Header Fields

When the signature is generated, you can populate the **signature** header field. The **signature** header field includes these parameters:

Signatures

Signature Parameter	Description
keyid	The serial number of the signing certificate/key pair. Obtain this in the Business Center Key Management area. For more information, see *Create a Shared Secret Key Pair* on page 44.
algorithm	The HMAC SHA256 algorithm used to encrypt the signature. It should be formatted: HmacSHA256.
headers	This ordered list of the fields included in the signature: host v-c-date request-target digest v-c-merchant-id
signature	The signature hash.

Signature Header Field Format

Signature:"keyid:"[keyid]",algorithm="[encryption algoritm]",headers="field1" "field2" "field3" "etc.", signature="[signature hash]"

Signature Header Example

```
Signature:"keyid="123abcki-key1-key2-key3-keyid1234567",
algorithm="HmacSHA256", headers="host date request-target digest v-c-merchant-id",
signature="hrptKYTtn/VfwAdUqkrQ0HT7jqAbagAbFC6nRGXrNzE="
```

Going Live

When you are ready to process payments in a live environment, you must transition your account to a live status with a valid configuration for your chosen payment processor. When live, your transaction data flows through the production Cybersource gateway, to your processor, and on to the appropriate payment network.
To transition your account:

1. Sign up for a merchant account.
2. *Contact sales* to establish a contract with Cybersource that enables you to process real transactions and receive support.
3. Submit a merchant ID (MID) activation request.

It may take up to three business days to complete a MID activation request.

Create a Merchant ID

The merchant ID (MID) is used to identify you and your transactions and is included in the header of each transaction request. When you signed up for a sandbox account, you

received a merchant ID for testing purposes. If you choose, you can use that merchant ID as your production ID.
Follow these steps to sign up for a merchant account in order to create a production MID:

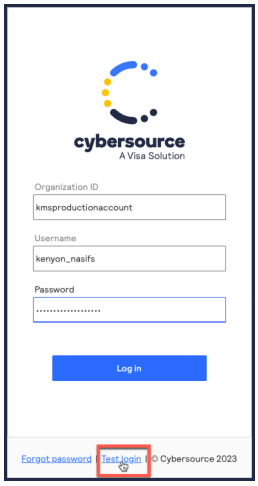
1. Navigate to the Business Center *Evaluation Account Sign-up page*, enter the required information, and click Create Account.
   Choose your merchant ID name carefully. It cannot be changed. This name is not visible to your customers.
2. Review your information entered, especially your business email address. Your merchant ID registration information will be sent to the email entered on this form.
3. Check your email from customer support titled: Cybersource Merchant Evaluation Account.
   This email will include the Organization ID and contact email associated with your MID.
4. Go to your email and find a message titled: Merchant Registration Details. Click the Set up your username and password now link.
   Your browser opens the New User Sign Up wizard.
5. Enter the Organization ID and Contact email you supplied previously. Follow the wizard pages to add your name, a username, and password.
6. Log into the Business Center.
   When you log in for the first time, you will be asked to identify your identity through a system-generated email that is sent to your email account.
7. Check your email for a message titled: Cybersource Identification Code.
   Note the passcode.
8. Enter the passcode on the Verify your Identity page.
   You should be directed to the Business Center home page.

You have successfully created a merchant ID and merchant account.

Activate your Merchant ID

The activation process, also known as going live, transitions your MID and account from test status to live status, enabling you to process real transactions in production. It may take up to three business days to complete the MID activation request.
To transition your account complete these tasks:

1. Sign in to the *Support Center* as an administrator.
2. Enter your credentials and log in to your test environment.



Enter your MID in the Organization ID text box.

3. Go to Support Cases > MID Configuration Request. The MID Configuration Request page should be open.
4. Select MID Activation.
5. In the Description field, enter the Merchant ID that you want to take live.
6. Select the processor configuration and enter the name of your processor.
   If you are unsure of your processor name, contact your merchant service provider or your merchant acquiring bank.
7. Select the environments that this change applies (test or production).
8. Select Service Enablement and list the products and services that you intend to use.
9. Select Submit.

Production Endpoints

When sending API request messages using your production account, send your requests to the production server:
`https://api.cybersource.com`
For example, to send a live authorization request, you can send the request to this endpoint:
`https://api.cybersource.com/pts/v2/payments`

VISA Platform Connect: Specifications and Conditions for Resellers/ Partners

The following are specifications and conditions that apply to a Reseller/Partner enabling its merchants through Cybersource for Visa Platform Connect ("VPC") processing. Failure to meet any of the specifications and conditions below is subject to the liability provisions and indemnification obligations under Reseller/Partner's contract with Visa/Cybersource.

1. Before boarding merchants for payment processing on a VPC acquirer's connection, Reseller/Partner and the VPC acquirer must have a contract or other legal agreement that permits Reseller/Partner to enable its merchants to process payments with the acquirer through the dedicated VPC connection and/or traditional connection with such VPC acquirer.
2. Reseller/Partner is responsible for boarding and enabling its merchants in accordance with the terms of the contract or other legal agreement with the relevant VPC acquirer.
3. Reseller/Partner acknowledges and agrees that all considerations and fees associated with chargebacks, interchange downgrades, settlement issues, funding delays, and other processing related activities are strictly between Reseller and the relevant VPC acquirer.
4. Reseller/Partner acknowledges and agrees that the relevant VPC acquirer is responsible for payment processing issues, including but not limited to, transaction declines by network/issuer, decline rates, and interchange qualification, as may be agreed to or outlined in the contract or other legal agreement between Reseller/ Partner and such VPC acquirer.

DISCLAIMER: NEITHER VISA NOR CYBERSOURCE WILL BE RESPONSIBLE OR LIABLE FOR ANY ERRORS OR OMISSIONS BY THE VISA PLATFORM CONNECT ACQUIRER IN PROCESSING TRANSACTIONS. NEITHER VISA NOR CYBERSOURCE WILL BE RESPONSIBLE OR LIABLE FOR RESELLER/PARTNER BOARDING MERCHANTS OR ENABLING MERCHANT PROCESSING IN VIOLATION OF THE TERMS AND CONDITIONS IMPOSED BY THE RELEVANT VISA PLATFORM CONNECT ACQUIRER.