

Getting Started with REST

REST API



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

© 2020. Cybersource Corporation. All rights reserved. Cybersource Corporation ("Cybersource") furnishes this document and the software described in this document under the applicable agreement between the reader of this document ("You") and Cybersource ("Agreement"). You may use this document and/or software only in accordance with the terms of the Agreement. Except as expressly set forth in the Agreement, the information contained in this document is subject to change without notice and therefore should not be interpreted in any way as a guarantee or warranty by Cybersource. Cybersource assumes no responsibility or liability for any errors that may appear in this document. The copyrighted software that accompanies this document is licensed to You for use only in strict accordance with the Agreement. You should read the Agreement carefully before using the software. Except as permitted by the Agreement, You may not reproduce any part of this document, store this document in a retrieval system, or transmit this document, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written consent of Cybersource.

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 in 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 subparagraphs (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 Unpublished rights reserved under the copyright laws of the United States.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

Authorize.Net, eCheck.Net, and The Power of Payment are registered trademarks of Cybersource Corporation. Cybersource, Cybersource Payment Manager, Cybersource Risk Manager, Cybersource Decision Manager, and Cybersource Connect are trademarks and/or service marks of Cybersource Corporation. Visa, Visa International, Cybersource, the Visa logo, and the Cybersource logo are the registered trademarks of Visa International in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners.

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 limit 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: 24.10

Contents

Getting Started with REST	5
Recent Revisions to This Document.....	5
Overview of Getting Started with REST	7
Set Up Your Cybersource Account	9
Set Up a JSON Web Token Message	12
Sign Up for a Sandbox Account.....	14
Create a P12 Certificate.....	15
Create a P12 File.....	15
Extract the Private Key from the P12 Certificate.....	20
Test the Shared Secret Key Pair.....	21
Test Endpoints.....	24
Construct Messages Using JSON Web Tokens.....	25
Elements of a JSON Web Token Message.....	25
Generate a Hash of the Message Body.....	26
Generate the Token Header.....	26
Generate a Hash of the Claim Set.....	27
Generate a Hash of the Token Header.....	28
Generate the Message Body.....	29
Generate a Token Signature.....	29
Generate a JSON Web Token.....	30
Enable Message-Level Encryption.....	30
Prerequisites for Message-Level Encryption.....	31
Message-Level Encryption Using JSON Web Tokens.....	32
Going Live.....	36
Create a Merchant ID.....	36
Activate your Merchant ID.....	37
Production Endpoints.....	38
Set Up HTTP Signature Message	39
Sign Up for a Sandbox Account.....	41
Create a Shared Secret Key Pair.....	42
Create a Shared Secret Key Pair.....	42

Test the Shared Secret Key Pair.....	47
Test Endpoints.....	49
Construct Messages Using HTTP Signature Security.....	50
Elements of an HTTP Message.....	50
Generate a Hash of the Message Body.....	50
Generate the Signature Hash.....	51
Update Header Fields.....	53
Going Live.....	54
Create a Merchant ID.....	54
Activate your Merchant ID.....	55
Production Endpoints.....	55
VISA Platform Connect: Specifications and Conditions for Resellers/Partners.....	56

Getting Started with REST

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

24.10

Fixed error in the JWE token for message level encryption (MLE). See [Message-Level Encryption Using JSON Web Tokens](#) on page 32.

24.09

Fixed signature header parameter "keyid" typo. See [Update Header Fields](#) on page 53.

24.08

Removed support for message level encryption (MLE) when setting up an HTTP signature message. See [Set Up HTTP Signature Message](#) on page 39.

24.07

This update contains editorial changes.

24.06

The guide has undergone a major reorganization.

24.05

Corrected typo.

Updated the creating validation string example for generating an HTTP signature hash using the `\n` newline switch. For more information, see [Construct Messages Using HTTP Signature Security](#) on page 50.

24.04

Updated JSON Web Token Construction

Updated the Constructing Messages Using JSON Web Tokens Section.

Minor update to Message Level Encryption

Added note to Enabling MLE Encryption about the P12 certificate.

24.03

Update token signature to use the Key ID (kid).

Replaced x5c signature with kid when creating a token signature.

Added Message Level Encryption

Message Level Encryption (MLE) was added to the guide. See [Construct Messages Using HTTP Signature Security](#) on page 50

24.02

Updated the link to the IETF HTTP Working Group website. See [Construct Messages Using HTTP Signature Security](#) on page 50.

24.01

Message Elements

Added the message elements required to send a successful message. See [Elements of a JSON Web Token Message](#) on page 25

Overview of Getting Started with 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 applications 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 will 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 because they do not have 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 came 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.

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.

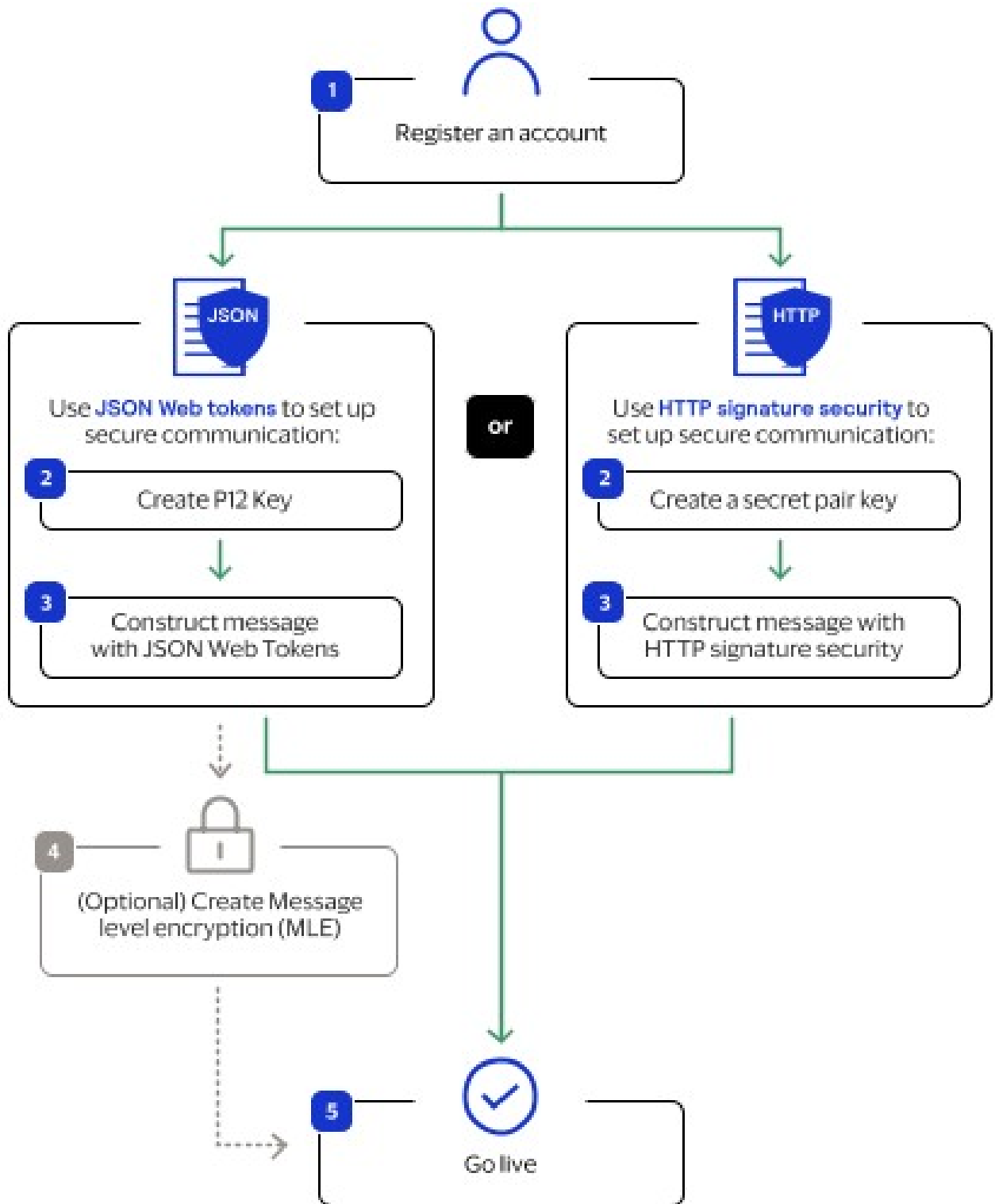
Components of REST

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 will need 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.



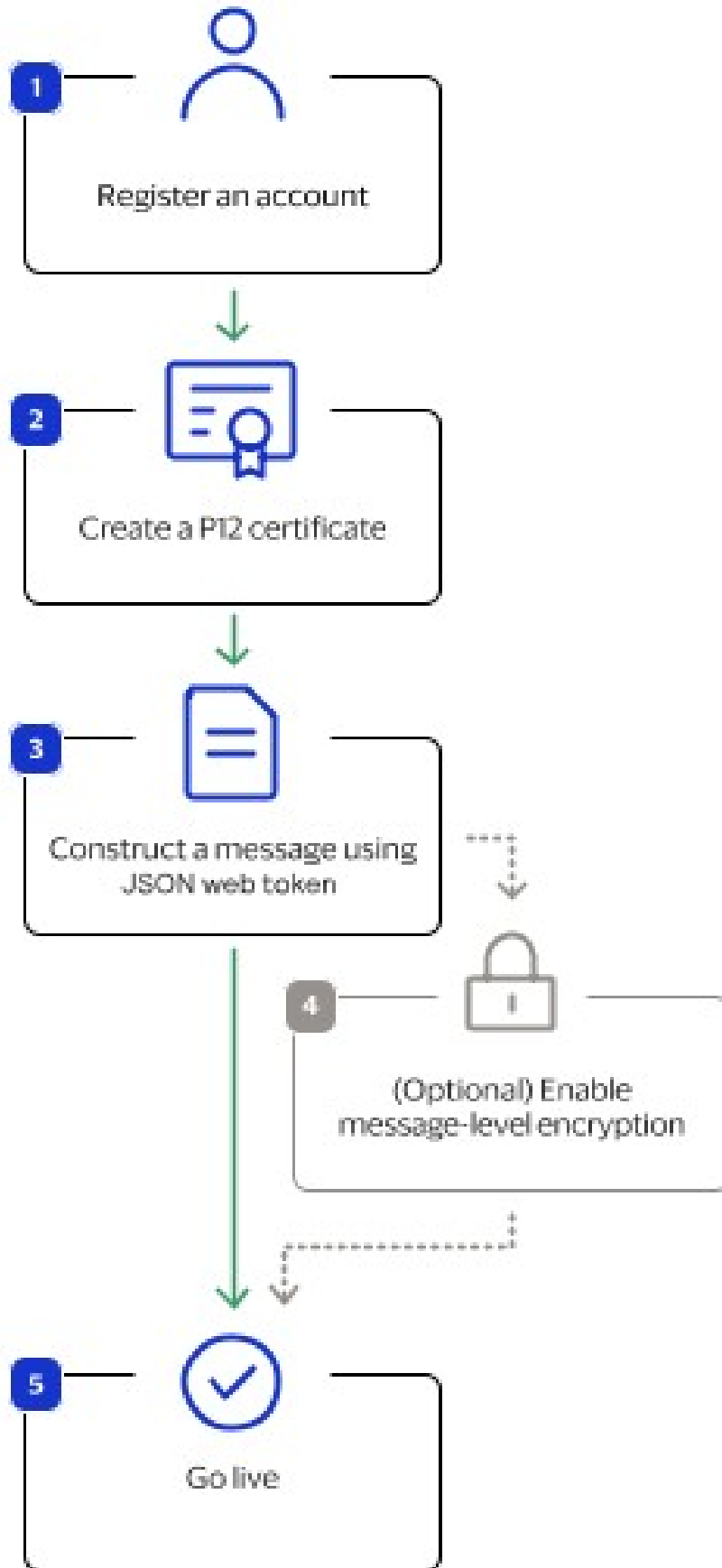
Enabling REST Workflow

1. Sign up and register a sandbox account.
2. Obtain security keys for either:
 - JWT message security uses a P12 certificate. See [Set Up a JSON Web Token Message](#) on page 12.

- HTTP Signature message security uses a key pair. See [Set Up HTTP Signature Message](#) on page 39.
3. Configure your messaging.
 4. (Optional) Enable message-level encryption (MLE) for JWT message security.
 5. Go live by signing up for a production account.

Set Up a JSON Web Token Message

Setting up your JSON web token message requires you to complete these tasks:



How to Set Up JSON Web Token Messaging

1. Sign up and register a Cybersource Business Center sandbox account. See [Sign Up for a Sandbox Account](#) on page 41.

2. Create a P12 certificate. See [Create a P12 Certificate](#) on page 15.
3. Construct a message using a JSON web token. See [Construct Messages Using JSON Web Tokens](#) on page 25.
4. (Optional) Enable the optional message-level encryption (MLE) feature. See [Enable Message-Level Encryption](#) on page 30.
5. Go live. See [Going Live](#) on page 54.

Sign Up for a Sandbox Account

The first step to set up your account is to sign up for a sandbox account. From this account you can obtain your security keys and test your implementation. 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.

Sandbox account sign up

After completing the evaluation registration process, you will be able to send test transactions.
Your information will not be disclosed to third parties.
All required fields are indicated by an*.

Organization ID *

Company *

First name *

Last name *

Address line 1

Address line 2

Country *

Choose country

City *

Zip code *

Email *

Phone *

E-commerce or card present? *

E-Commerce Card Present

Terms and conditions
After completing the evaluation registration process, you will be able to send test transactions to Cybersource. Your information will not be disclosed to third parties.

Create Account

Already have an account? [Log in](#)

3. 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.
4. Enter the Organization ID and Contact email you supplied previously. 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 will be asked to verify your identity through a system-generated email 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 should be directed to the Business Center home page.
You have successfully signed up for a sandbox account.

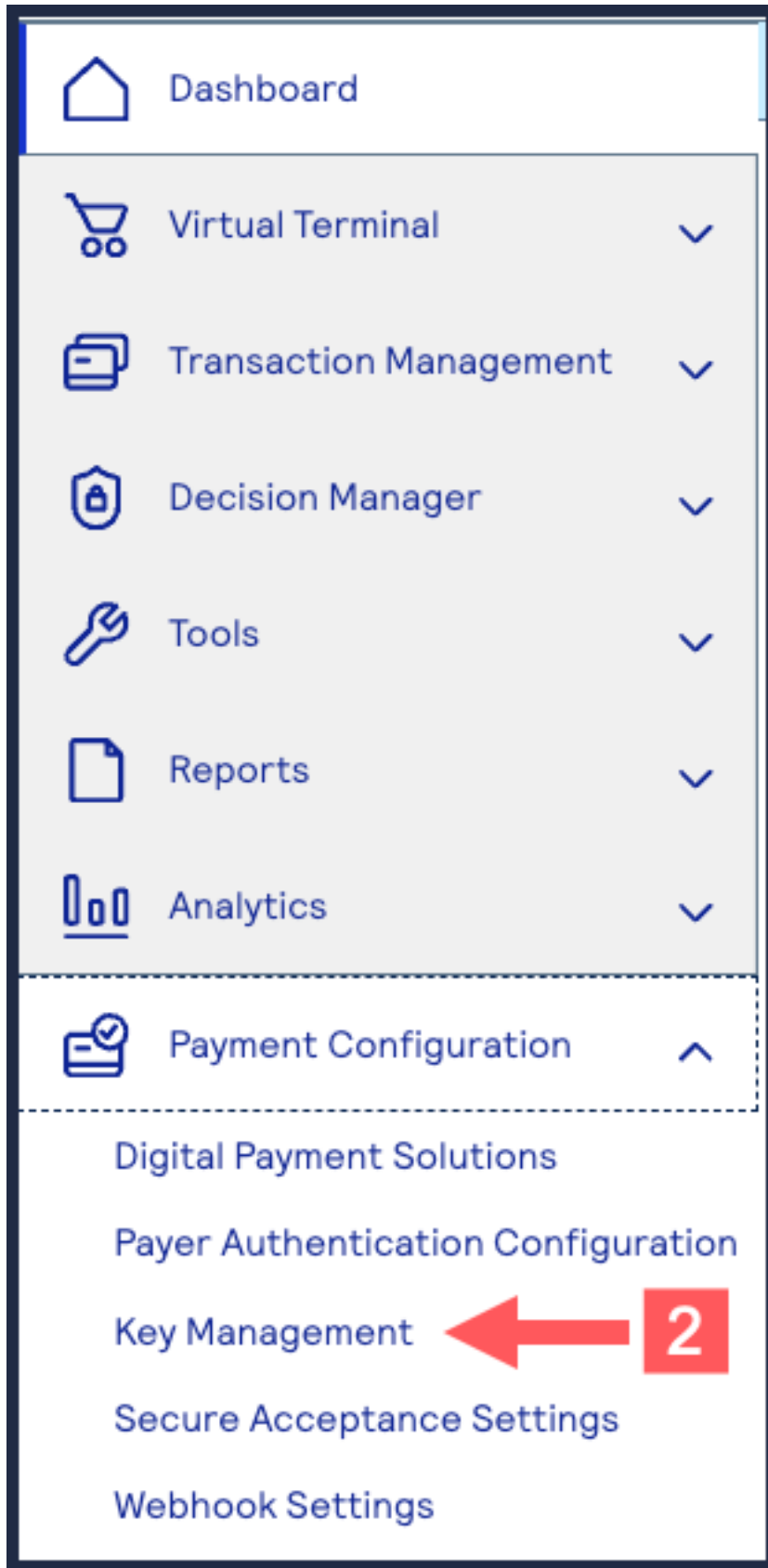
Create a P12 Certificate

A P12 certificate and its private key are used with 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  Payment Configuration > Key Management.





3. Click + Generate key.

Key Management

* = Required

Search Filters

Key Type	Created At
REST-Shared Secret 	Last 6 Months (GMT) 

Applied Filters: Key Type: REST-Shared Secret, Created At: Last 6 Months

4. Under REST APIs, select 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.

[Details for REST APIs](#)


REST - Shared Secret

REST - Certificate  



5. Click Download key .

REST API Certificate Key



 Information: Once finished your key will be displayed on the Key M

Key Configuration

Certificate uses a PKCS12 key file with the .p12 extension to digitally sign message to us.

To create and activate a new certificate, click Download Key. Click on t prompted to either save or open the file, select open. Follow the prom


Note: Be sure to store the key in a safe location. If you do not protect t compromised.

[Download key !\[\]\(f9ccf36cb8f1dba8b11feb5692e99a8b_img.jpg\)](#)  


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

Set a Password

Set a password to generate key. This password will be used to open the downloaded key file and in your API implementation.


 Information: Password is not stored and cannot be changed. If you forget this password, you will not be able to open the key file.

New Password *

Select or enter an option 

- At least 8 characters long
- At least one upper-case letter
- At least one lower-case letter
- At least one special character: @\$!%*?&
- At least one numeric character
- Password should not contain merchant Id

Confirm Password *

Select or enter an option 

Generate key

Cancel

The .p12 file is downloaded to your desktop.

When you generate one or more keys, you can view the keys on the Key Management page.

Extract the Private Key from the P12 Certificate

When you have your P12 certificate, you can extract the private key from the certificate. Use this key to sign your header when sending an API message. Follow these steps to automate the extraction of your private key.



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:

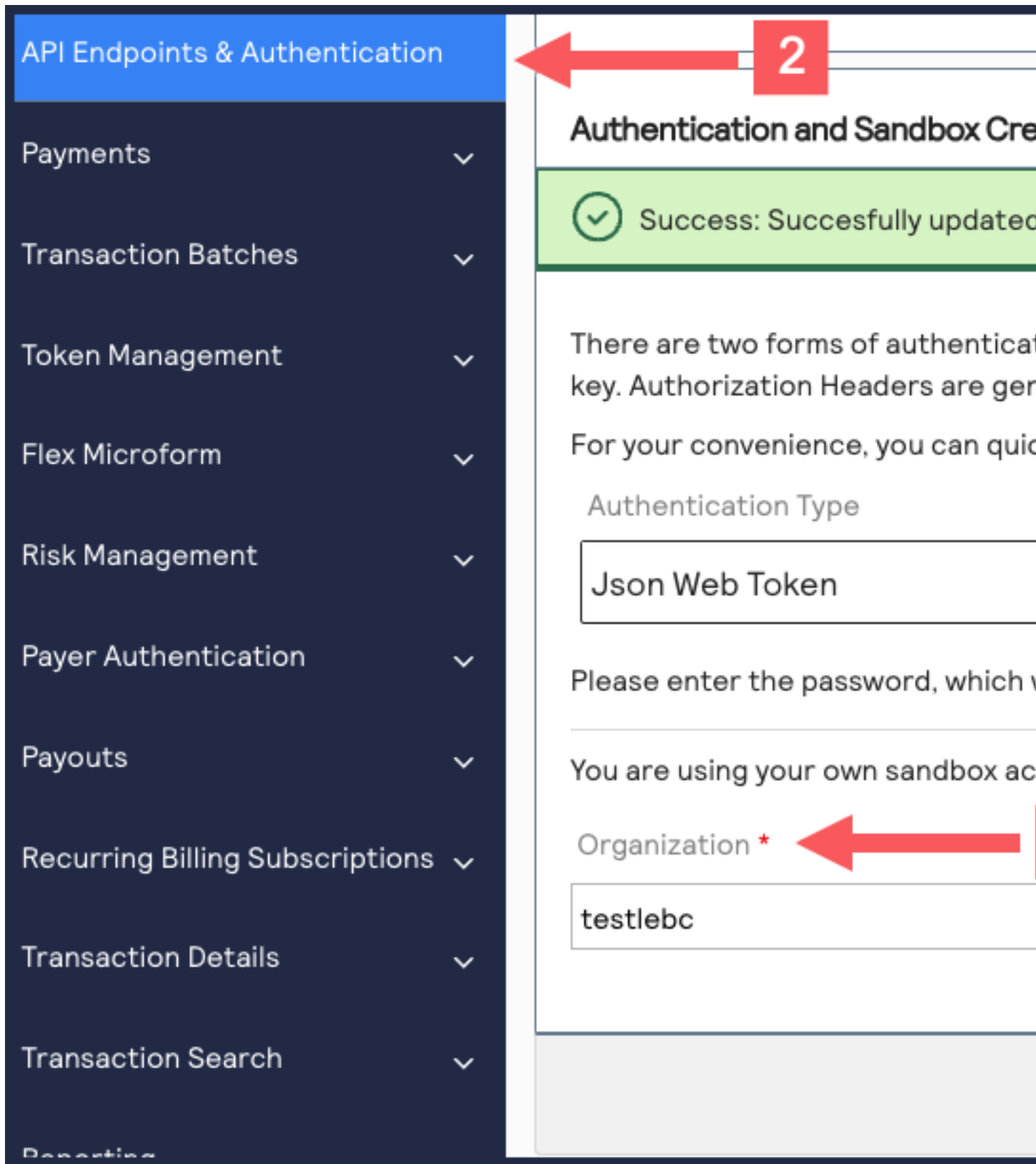
1. Open the command-line tool and navigate to the directory that contains the P12 certificate.
2. Enter this command:
`openssl pkcs12 -in [certificate name] -nodes -nocerts -out [private key name]`
3. Enter the password for the certificate.
The password is set when creating the P12 certificate in the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

The new certificate will be 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 test it to verify that your key can successfully process API requests. This task explains how to test and validate your key pair using the developer center and the Business Center.


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

Home

API Endpoints & Authentication

Payments 

Payments



- POST** Process a Payment 
- PATCH** Increment an Authorization
- POST** Check a Payment Status
- POST** Create a Payment Order Request
- POST** Create Alternative Payments Sessions Request
- PATCH** Update Alternative Payments Sessions Request

Reversal



















- POST** Process an Authorization Reversal
- POST** Timeout Reversal

Capture

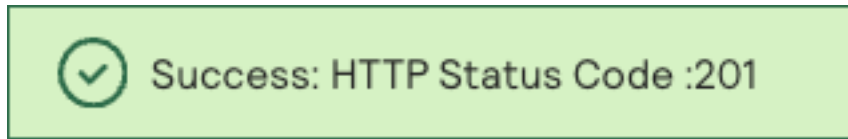
- POST** Capture a Payment

 Required fields are denoted with an asterisk * optional, however if provided them – marked with an 

Request Builder Configuration

- reconciliationId 
- 8** pausedRequestId 
- transactionId 
- comments 
-  partner
- originalTransactionId 
- developerId 
- solutionId 
- thirdPartyCertificateId 
- applicationName 
- applicationVersion 
- applicationUser 
-  processingInformation 
- actionList 
- enableEscrowOptions 
- actionTokenTypes 
- binSource 
- capture

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



10. Log in to the Business Center:

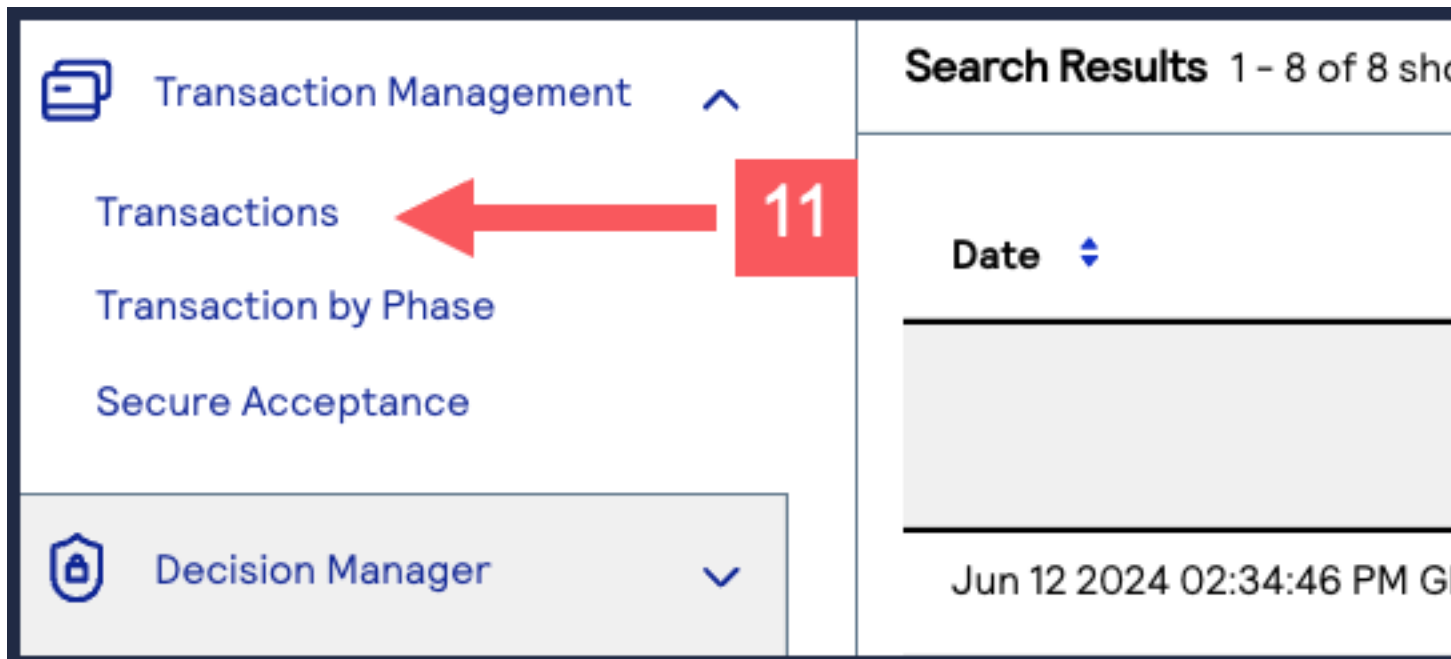
<https://businesscentertest.cybersource.com>

11.

On the left navigation panel, navigate to  Transaction Management > Transactions.

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 request messages 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 26.
2. Populate the header values. See [Generate the Token Header](#) on page 26.
3. Generate a hash of the claim set. See [Generate a Hash of the Claim Set](#) on page 27.
4. Generate a hash of the token header. See [Generate a Hash of the Token Header](#) on page 28.
5. Generate a token signature hash. See [Generate a Token Signature](#) on page 29.
6. Populate the **signature** header field. See [Update Header Fields](#) on page 53.

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. (https://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 26.

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: RBNvo1WzZ4oRRq@W9+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 41.

Token Header

```
eyJ2LWMtbWVyY2hhbnQtaWQiOiJtZXJjaGFudEElIiwiaWF0IjoiI3MDc4NjMzMjg1MjUwMTc3MDQxNDk5In0
```

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":"7078633285250177041499"}'
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/PBq8iMASv1EARp8t1=",
"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

```
NTk5NWE0ZjI3YjRiOTI1NmE5NGNmNTQ0OD1hOWVmNjRkZDhkYzhhNTkwZDMyMjc4MGQzYjIwMmNmYTJmMDc4Zg==
```

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.
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: eyJkaWd1c3QiOiJSQk52bzFXe1o0b1JScTBXOStoa25wVDdUOEImNTM2REVNQmc5aHlxLzRvPSIsImRpZ2VzdEFsZ29yaXRobSI6I1NIQS0yNTYiLCJpYXQiOiIyMDI0LTA0LTA1VDE2OjI1OjE4LjI1OVoifQ
```

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.

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 31.

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 32.

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 15.

```
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);
    JWEObjct jweObjct = new JWEObjct(
        new JWEHeader.Builder(algo, encryptionMethod)
            .contentType("JWT") // required to signal nested JWT
            .keyID(serialNumber)
            .customParams(customHeaders)
            .build(),
        new Payload(content));

    jweObjct = encrypt(jweObjct, x509Certificate);
    return jweObjct == null ? null : serializeToken(jweObjct);
}

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 (isEmptyOrNull(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();
    }
    JOSEObject jwsObject = new JOSEObject(jwsHeader, payload);
    try {
        RSASSASigner signer = new RSASSASigner(rsaPrivateKey, true);
        jwsObject.sign(signer);
        if (!jwsObject.getState().equals(JOSEObject.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

```
eyJ2LWMTbWVvY2hhbnQtaWQiOiJtcG9zX3BheW11bnRlY2giLCJhbGciOiJSUzI1NiIsImtpZCI6Im1wb3NfcGF5bWVudGVjaCI6Ijx0dH6jzuJxoerkgz3VSMuDt9mDjtn1DnisSTf35NWh6u4TeJqGr3E8oOOJSX6N32r6XovCXyJyaDm4h2fJOeLZc8HcvfSC55SpM
_OwE1AYZfOnZ9FphLQZWnwZ3mku0C6gysv6ISMRI9B1CpWaPbmDeuWBSLexC_U01cblg
```

HTTP Request Body:

```
{"encryptedRequest": "eyJ2LWMTbWVvY2hhbnQtaWQiOiJtcG9zX3BheW11bnRlY2giLCJ4NWMiOiJ01siTU1JRE5UQ0NBaDnQXdxJGZbvd-
GCEaFs15U1_Brt1hQTn9aKjX_-rbYxM-ZXJlbp6CsyAqy63-
MkYYP2BNXjFfP3yUSxes76zH1MaJG0gp681QY85AqGq6mCSrDqWE7NUTWifseRtKMv5u9pMHMxddkz9Xvp6Q5TbiEjGZbvD-
xKhhgs0-IupvPDKhxdJSNVPaDiTnFvNtyOuLZLOFO4Fq2bfj86iGHRjfh9zq91Gp4uN36kmRHZkLN4Wrr5R6D79Z-
FC5bLU4BUriLQQtVSWCWtcxYAIQOhz1w.tuv-9Xt10uNoPxXV.RRGnKA1chpLnGQf-S1XaXEntzGJrEF4EJU-
F6PEX6H3us1APoWAR-26aHdWctNFoGSa1Nt1ZzidRi3TA-iwpSFkEonSVbe7aVLJeAKgqChnVXT-
eWb89gqTVkQZiSIZCHtIjDUtOMy95sU4MRcCvtrfAPDnIMudVVA5YtAsCZpta_AT1_iS6oLBMi57R0Ra7pO3MxFdLTrk-
FkLSd4JbGokm_JXpH81I1V11vaMAtyEqGrz1lrQv408zUGbvtvSirF31iiGITEF7QG5rbVn7oTWF4wWzKEkpSZ7J4LpLdjCG6soje1d4
vFVfa-
ua4uh4PNcVK0o3ke4TOqLnVcnaEtYW1AS2wIu_tHxW_hdkyPmDI8ceSBqmIoRxV3q8xOS5u-2GNQ9p5pm2_NjkqVB8RYup9NFZW
RzJ0mOwPF2MzZQ318z78IyAGXotYT4QXGJZhnYDMNgHjyyGX7IZtGPRYDpxc10Kko9DLM_r6fWoDLemRhFbi8prn1JpQZbUh98TLF
```

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:

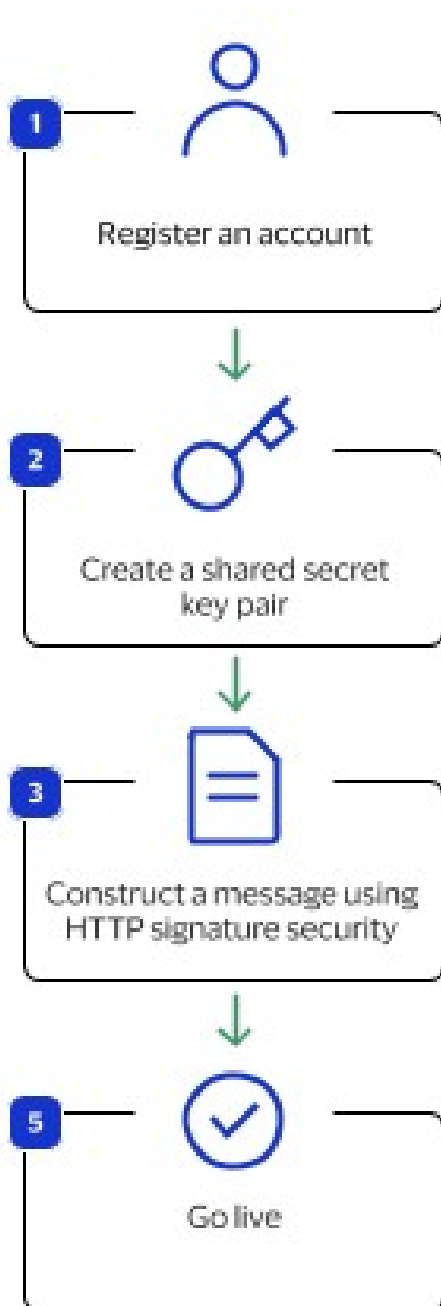
<https://api.cybersource.com>

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 HTTP Signature Message

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



Set Up HTTP Signature Message Workflow

1. Sign up and register a Cybersource Business Center sandbox account. See [Sign Up for a Sandbox Account](#) on page 41.
2. Create a shared secret key. See [Create a Shared Secret Key Pair](#).
3. Construct a message using HTTP signature security. See [Construct Messages Using HTTP Signature Security](#) on page 50.
4. Go live by signing up and registering a Cybersource Business Center production account. [Going Live](#) on page 54.

Sign Up for a Sandbox Account

The first step to set up your account is to sign up for a sandbox account. From this account you can obtain your security keys and test your implementation. 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.

Sandbox account sign up

After completing the evaluation registration process, you will be able to send test transactions.
Your information will not be disclosed to third parties.
All required fields are indicated by an*.

Organization ID *

Company *

First name *

Last name *

Address line 1

Address line 2

Country *

Choose country

City *

Zip code *

Email *

Phone *

E-commerce or card present? *

E-Commerce Card Present

Terms and conditions
After completing the evaluation registration process, you will be able to send test transactions to Cybersource. Your information will not be disclosed to third parties.

Create Account

Already have an account? Log in

3. 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.
4. Enter the Organization ID and Contact email you supplied previously. 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 will be asked to verify your identity through a system-generated email 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 should be directed to the Business Center home page.
You have successfully signed up for a sandbox account.

Create a Shared Secret Key Pair

Key pairs are used with HTTP Signature message security.

Create a Shared Secret Key Pair

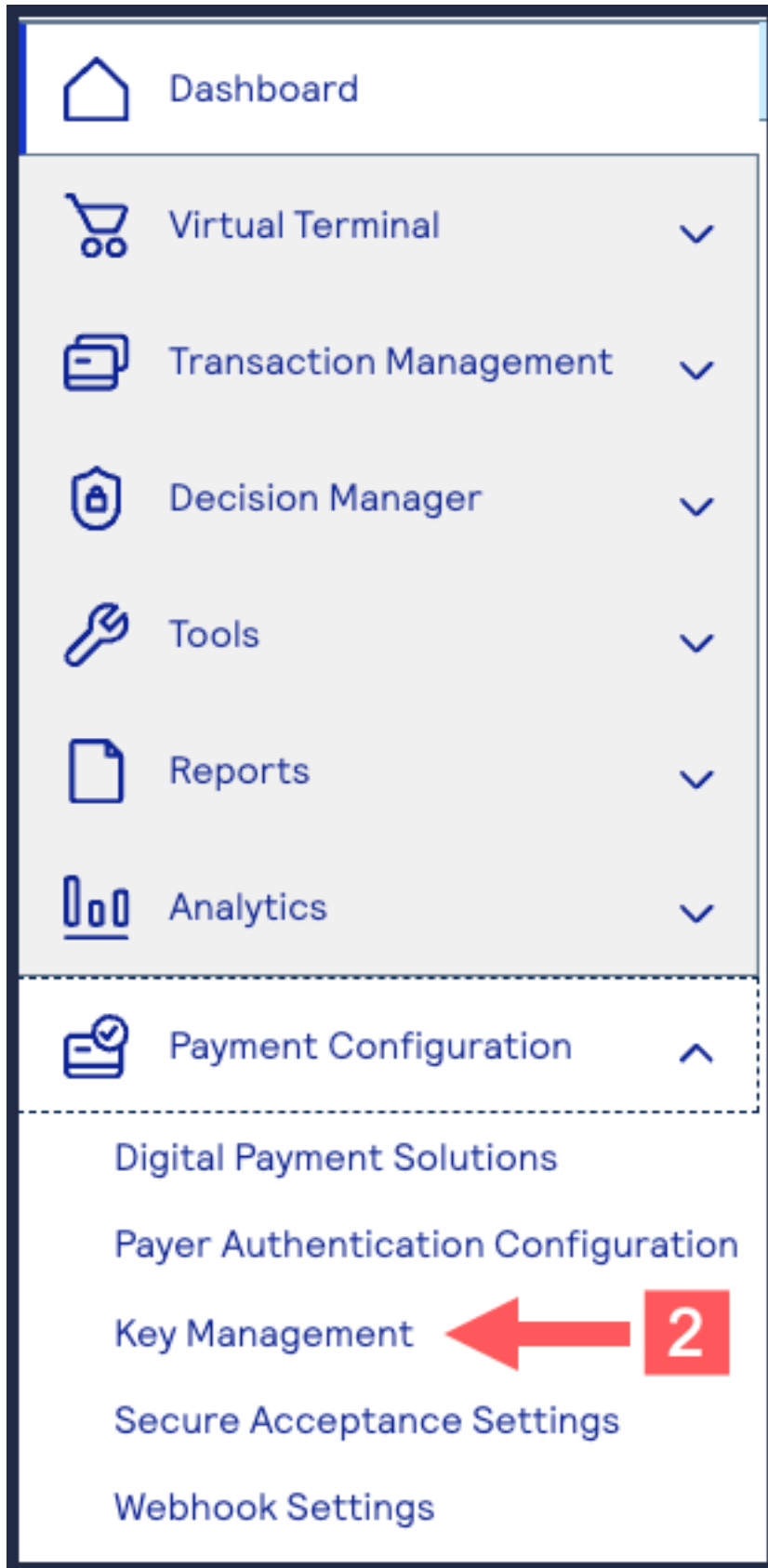
Follow these steps to create a shared secret key pair.

1. Log in to the Business Center:

<https://businesscentertest.cybersource.com>

- 2.

On the left navigation panel, navigate to  Payment Configuration > Key Management.





3. Click + Generate key.

Key Management

* = Required

Search Filters

Key Type	Created At
REST-Shared Secret 	Last 6 Months (GMT) 

Applied Filters: Key Type: REST-Shared Secret, Created At: Last 6 Months

4. Under REST APIs, select REST – Shared Secret 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 solution

[Details for REST APIs](#)

REST - Shared Secret

REST - Certificate



The REST API Shared Secret Key page appears.

5. Click Download key .
The .pem file is downloaded to your desktop.

Key Generation

REST API Shared Secret Key

Key Configuration

Shared API authenticates using a base-64-encoded transaction key. The key you can copy to your clipboard or download as a text file.

Key

65670aad-3d92-48a8-a0c8-7089f11dd360

Shared Secret

x0wP5+sOY0CBR4VboN+jdz9Ea3D7rI7fKN7KKsicBuw=

Download key 



When you generate one or more keys, you can 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:
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.

Authentication and Sandbox Credentials

Success: Successfully updated credentials for HTTP Signature

There are two forms of authentication available: JSON Web Token (JWT) and HTTP Signature. JWT requires signing a P12 Certificate with a method which uses a shared secret key. Authorization Headers are generated based on payload for each request. To learn more about this, see [JSON Web Tokens](#).

For your convenience, you can quickly configure this API Console to send all sample requests with either method, using your own sandbox Console credentials.

Authentication Type

HTTP Signature ▼ ← 3

You are using your own sandbox account. Use the Reset button to revert to default credentials.

Organization ID * ← 4

Key * ← 5

Shared Secret Key * ← 6


← 7

Authenticate Key and Shared Secret Key

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

Home

API Endpoints & Authentication

Payments 

Payments



- POST** Process a Payment
- PATCH** Increment an Authorization
- POST** Check a Payment Status
- POST** Create a Payment Order Request
- POST** Create Alternative Payments Sessions Request
- PATCH** Update Alternative Payments Sessions Request

Reversal




















- POST** Process an Authorization Reversal
- POST** Timeout Reversal

Capture

- POST** Capture a Payment

 Required fields are denoted with an asterisk * optional, however if provided them – marked with an 

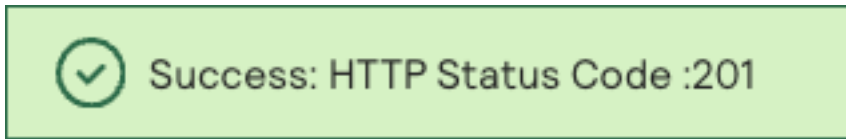
Request Builder Configuration

- reconciliationId 
- 8** pausedRequestId 
- transactionId 
- comments 
-  partner
- originalTransactionId 
- developerId 
- solutionId 
- thirdPartyCertificateId 
- applicationName 
- applicationVersion 
- applicationUser 
-  processingInformation 
- actionList 
- enableEscrowOptions 
- actionTokenTypes 
- binSource 
- capture 



8

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



10. Log in to the Business Center:

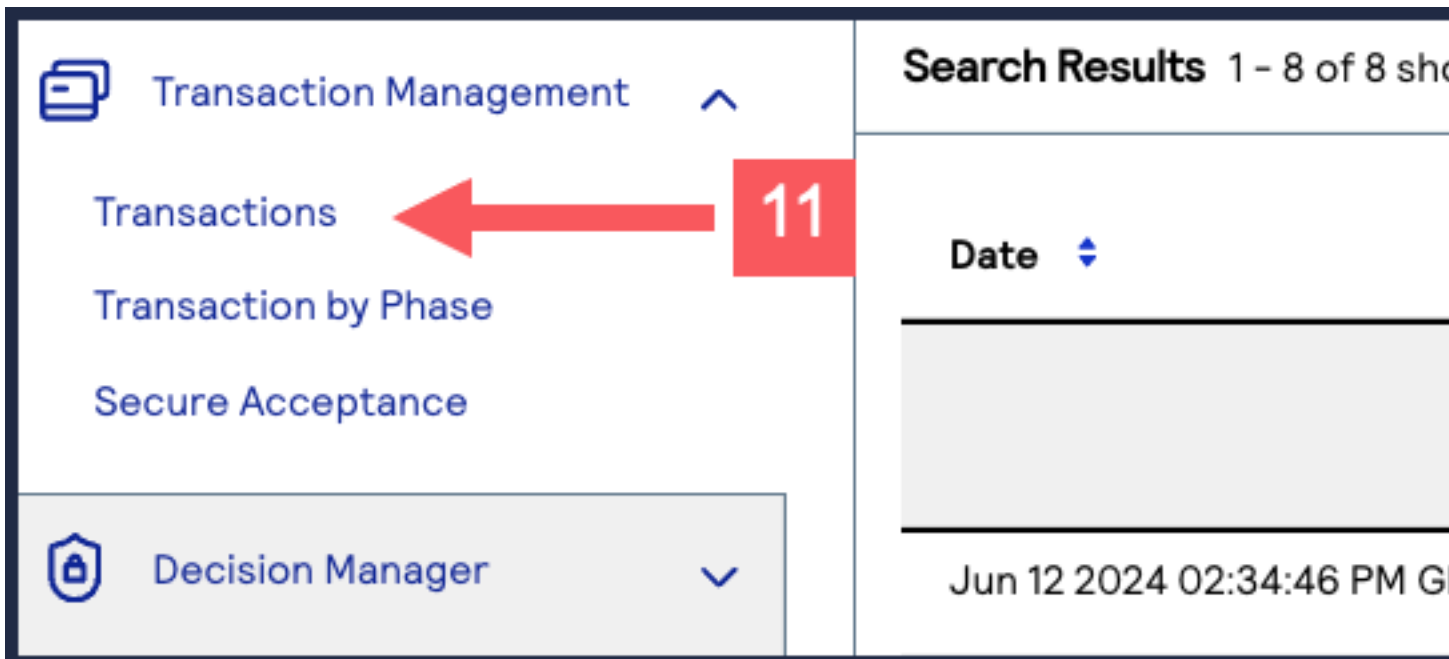
<https://businesscentertest.cybersource.com>

11.

On the left navigation panel, navigate to  Transaction Management > Transactions.

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 request messages 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 42.
2. Generate a hash of the message body. See [Generate a Hash of the Message Body](#) on page 50.
3. Generate a signature hash. See [Generate the Signature Hash](#) on page 51.
4. Populate the **signature** header field. See [Update Header Fields](#) on page 53.

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.
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. (https://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 50.

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).

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 42.
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[] HmacSha256DigestBytes = hmacSha256.doFinal();
    return Base64.getEncoder().encodeToString(HmacSha256DigestBytes);}
```

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 shared secret key used to encrypt the signature.
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 date request-target digest v-c-merchant-id
signature	The signature hash.

Signature Header Field Format

Signature:"keyid:"[shared secret key]",algorithm="[encryption algorithm]",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.