Using OAuth 1.0a to Access Mastercard APIs

Reading Time: *15 minutes*

CopiedOverview

Authentication is a key process when integrating with Mastercard APIs.

Mastercard uses one-legged OAuth 1.0aopens in a new tab for authenticating and authorizing client applications. It means every request sent to us must be digitally signed, and only requests with *valid signatures* created by *authorized clients* are granted access to our services.

In addition to that, requests with a body must be signed using the Google Request Body Hashopens in a new tab extension for OAuth.

Your ApplicationMastercard APISends signed request1Validates and grants access2Sends response3Your ApplicationMastercard API

In the next sections, you will learn a bit more about the OAuth1 scheme and how to set up keys for your project. You will also find links to our OAuth client libraries, some development tips and answers to frequently asked questions.

Tip:

Want to see some code? Browse our client library projects on **GitHub**.

CopiedOAuth 1.0a Basics

OAuth is a protocol originally published as RFC 5849opens in a new tab and used for securing access to APIs.

Mastercard uses OAuth 1.0aopens in a new tab in its simplest form, also known as "*One Leg*". This implementation involves one single step, in which we rely on OAuth signatures for server-to-server authentication.

The next few sections give a brief overview of:

- What it means to digitally *sign* requests in this context
- The format of the expected Authorization header
- What the OAuth credentials for your client application are.

Note:

This page sticks to what is necessary for consuming Mastercard APIs. To dive into the details and get a comprehensive definition of the protocol, we suggest you browse the links in the **further reading** section.

CopiedSigned Requests

As explained previously, HTTP requests are signed by client applications. Digital signatures are particularly useful as they guarantee integrity, authenticity and allow for non-repudiation of incoming API calls.

The OAuth signature process:

1. Normalizes requests (URL, body, ...) and OAuth parameters into a *single* string to sign

The Signature Base String is a consistent reproducible concatenation of the request elements into a single string (…) used as an input in hashing or signing algorithms

2. Signs this string using a client RSA private key and, in our case, the RSA-SHA256 method
3. Adds an Authorization header to the requests (for sending OAuth parameters and the signature along with the request)

Upon reception of the request, the signature base string is recomputed and the RSA signature is verified using the client public key. The signature must be valid and the signed base string and recomputed base string must be identical.

CopiedThe Authorization Header

OAuth 1.0a uses the standard HTTP Authorizationopens in a new tab header with the authentication scheme set to OAuth. The header value contains signed OAuth parameters and the request signature value only your application can produce.

Let's break a real-life example down:

Authorization: OAuth

oauth_body_hash="94cOcstEzvTvyBcNV94PCbo1b5IA35XgPf5dWR4OamU=",

oauth_nonce="32lqGrl0f0nQEW85",

oauth_signature="MhfaStcHU0vlloeaBLuP14(...)qqd99lI56XuCk8RM5dDA%3D%3D",

oauth_consumer_key="aXqayIybNdwMnzGIZMAkQYSq(...)139a87746d5b00000000000000",

oauth_signature_method="RSA-SHA256",

oauth_timestamp="1558370962",

oauth_version="1.0"

Copiedoauth_body_hash

This element is an extension to OAuth (also known as Google Body Hashopens in a new tab) in order to systematically check the integrity of request bodies. It contains a base64-encoded SHA-256 digest of the body.

Copiedoauth_nonce and oauth_timestamp

The OAuth nonce is a random string, uniquely generated for each request and also unique within a given time window.
The request timestamp is a positive integer representing the number of *seconds* since 1970. It must match the current time. In this example, 1558370962 is equivalent to 05/20/2019 @ 4:49pm (UTC).
The timestamp and nonce pair must be unique, or the request will be treated as a replay attack.

Copiedoauth_signature

This is the base64-encoded RSA signature of the request, produced using your private signing key. More details in the signed requests section.

Copiedoauth_consumer_key

The consumer key is a string identifing your application. More details in the consumer key section.

Copiedoauth_signature_method

This parameter indicates the signature method used to compute the oauth_signature value. It must be RSA-SHA256.

Copiedoauth_version

The OAuth version used. It must be 1.0.

CopiedThe Consumer Key

Consumer keys are 97-character identifiers for client applications consuming Mastercard APIs. The consumer key is not a secret (unlike the signing key), but rather a kind of username which appears in the OAuth header value sent with each request (see oauth_consumer_key).

CopiedThe Signing Key

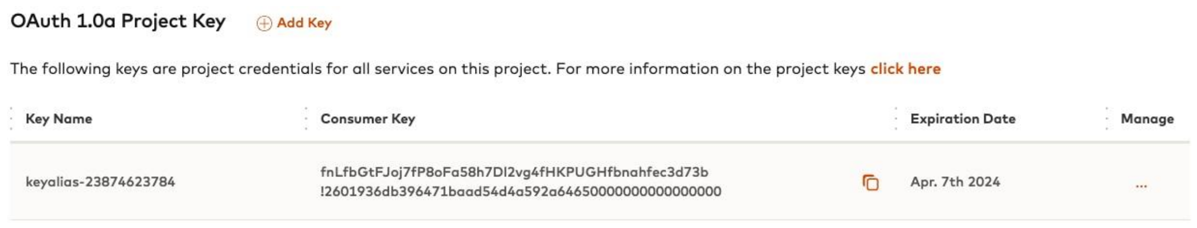
The signing key is a 2048-bit *private* RSA key used to generate request signatures and confirms ownership of the consumer key. This private key is part of a public/private key pair generated by you (or your web browser) and whose public key is certified by Mastercard.

Example of unencrypted RSA key (PKCS#1 PEM):

```
-----BEGIN RSA PRIVATE KEY-----

MIIEowIBAAKCAQEAx8k7j56VpkzxJT3qb23cYioMZ8VtGSTglwhXjyvags4VsSIH

oCCRd0Ich4UgNfat2adhwoPEw/bER/yaKhfFxiNYyjWruhIbvmzzgcz/w679JTuD

…

TTJVAoGBAMHkR/0+VAR8vvSEEENov43R7jnuggoQej5r9R/gG8UngoJkQ8b3aKGD

fRKHqz7nK1VgT8mh+M63gzL/UW8jSn2c1CbbojK/wU3FuaaRT5eY

-----END RSA PRIVATE KEY-----
```

CopiedGetting Keys for Your Application

As seen above, the OAuth protocol involves a consumer key and a private request signing key. OAuth keys for your projects are set up through your Developer Dashboard and you can manage the keys' lifecycle from there:



OAuth 1.0a Project Key ⊕ Add Key

The following keys are project credentials for all services on this project. For more information on the project keys click here

Key Name	Consumer Key	Expiration Date	Manage
keyalias-23874623784	fnLfbGtFJoj7fP8oFa58h7Dl2vg4fHKPUGHfbnahfec3d73b !2601936db396471baad54d4a592a6465000000000000000000	Apr. 7th 2024	...

CopiedKeys and Environments

There are two different types of keys, depending on the stage of your project:

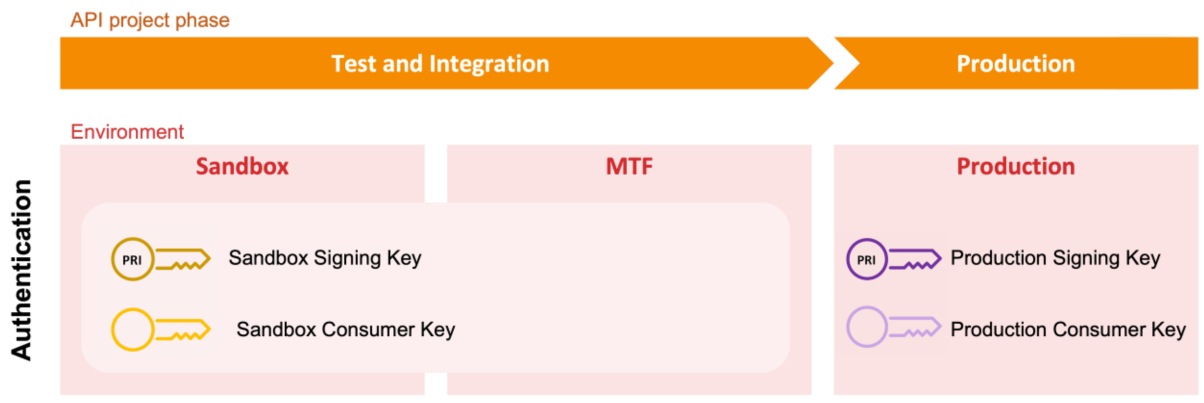
1. Sandbox keys, which give access to an API sandbox that mimics a live production environment (sandbox.api.mastercard.com)
2. Production keys, which allow an application to access the production environment (api.mastercard.com)

Domain/Server URL may vary depending on the API. Please check the API Reference section of the service documentation for the correct Server URL.

A pre-production environment called MTF (for "*Mastercard Test Facility*") is also made available for certain APIs. MTF access will follow the models below depending on the API.

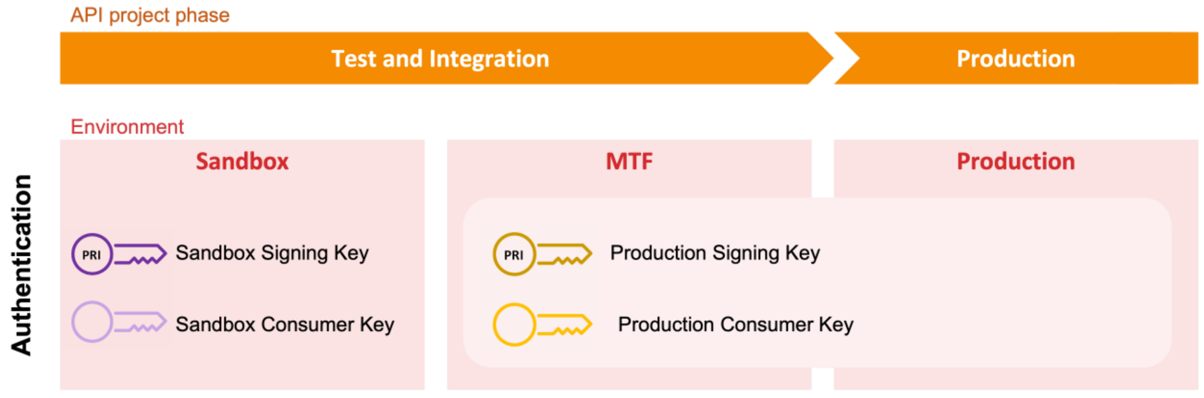
CopiedModel 1 - Single API for All Environments

Products using this model have one API for all environments and MTF is accessed using sandbox keys.



CopiedModel 2 - Separate MTF and Production APIs

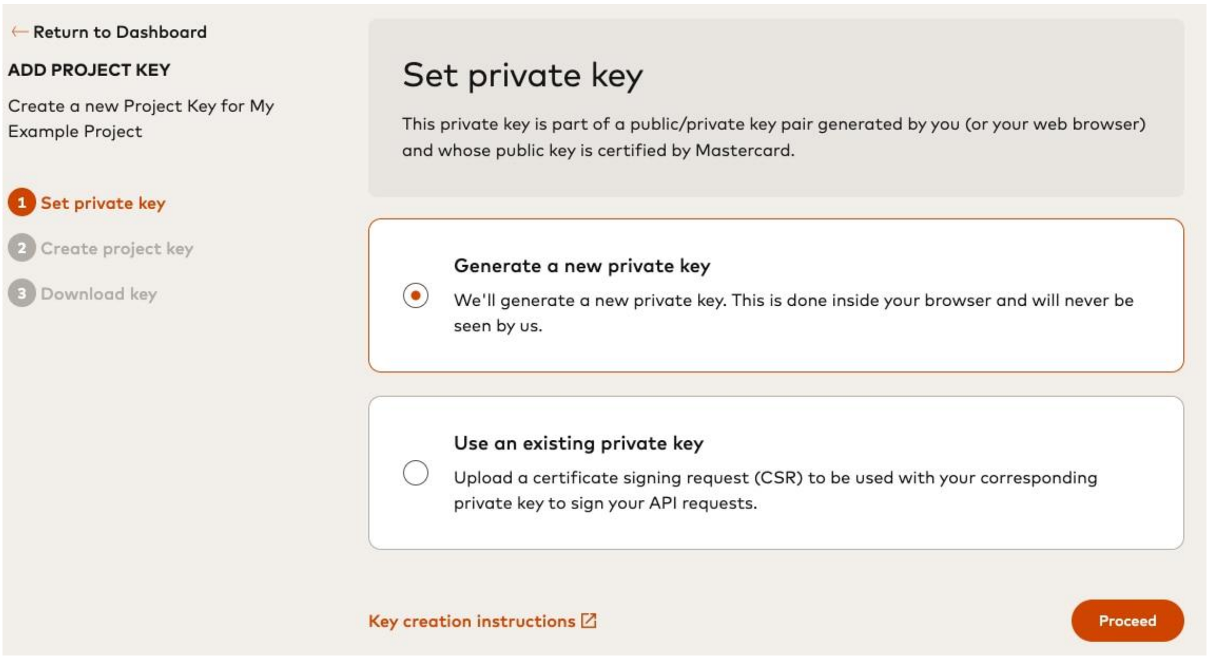
Products using this model have separate APIs for MTF and Production. They will have the same name, except the MTF API will be suffixed with "MTF." For example: *MDES Customer Service* and *MDES Customer Service MTF*. The APIs use the same keys and MTF is accessed using "production" keys.



CopiedCreating Keys

Sandbox keys are created as part of the Create Project wizard. You can add additional keys to an existing project by clicking the "*Add Key*" button on the Sandbox Credentials page. Production keys are created as part of the Request Production Access wizard. You can add additional keys to an existing project by clicking the "*Add Key*" button on the Production Credentials page.

Two methods are available when creating keys:



← Return to Dashboard

ADD PROJECT KEY

Create a new Project Key for My Example Project

1 Set private key
2 Create project key
3 Download key

Set private key

This private key is part of a public/private key pair generated by you (or your web browser) and whose public key is certified by Mastercard.

[✓ Generate a new private key]
We'll generate a new private key. This is done inside your browser and will never be seen by us.

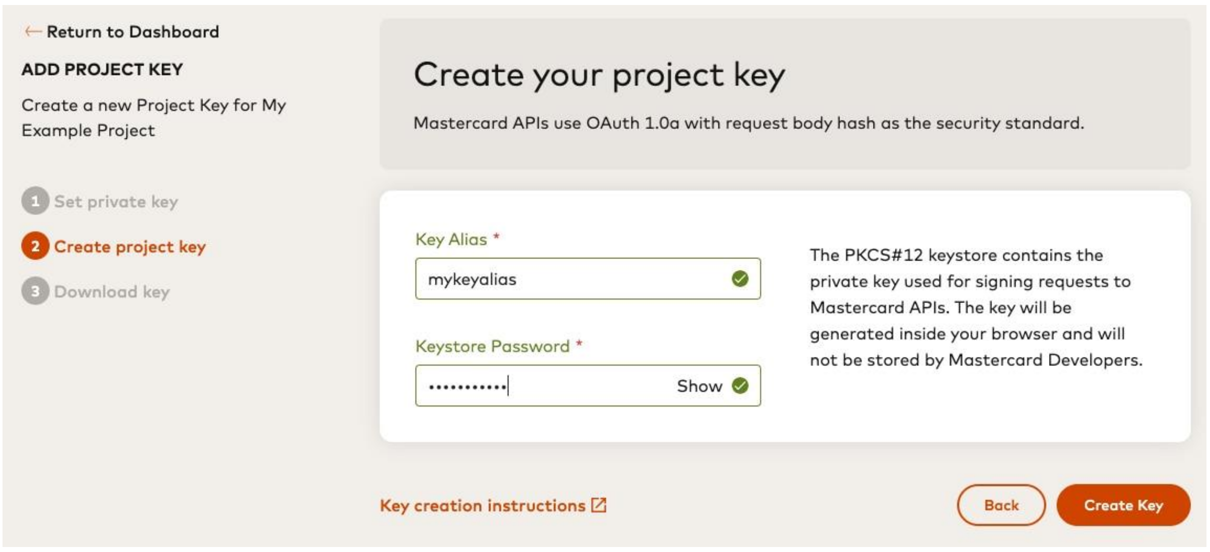
Use an existing private key
Upload a certificate signing request (CSR) to be used with your corresponding private key to sign your API requests.

Key creation instructions

Proceed

CopiedMethod 1

- A key pair and a certificate signing request (CSR)opens in a new tab are generated inside your web browser.
- Mastercard signs and sends your browser back an X509 certificateopens in a new tab matching your private key.
- Your browser generates a PKCS#12opens in a new tab key store file securing your private key with the provided passphrase.
- Mastercard generates a new consumer key.



Create a new Project Key for My Example Project

Create your project key

Mastercard APIs use OAuth 1.0a with request body hash as the security standard.

Key Alias *

mykeyalias

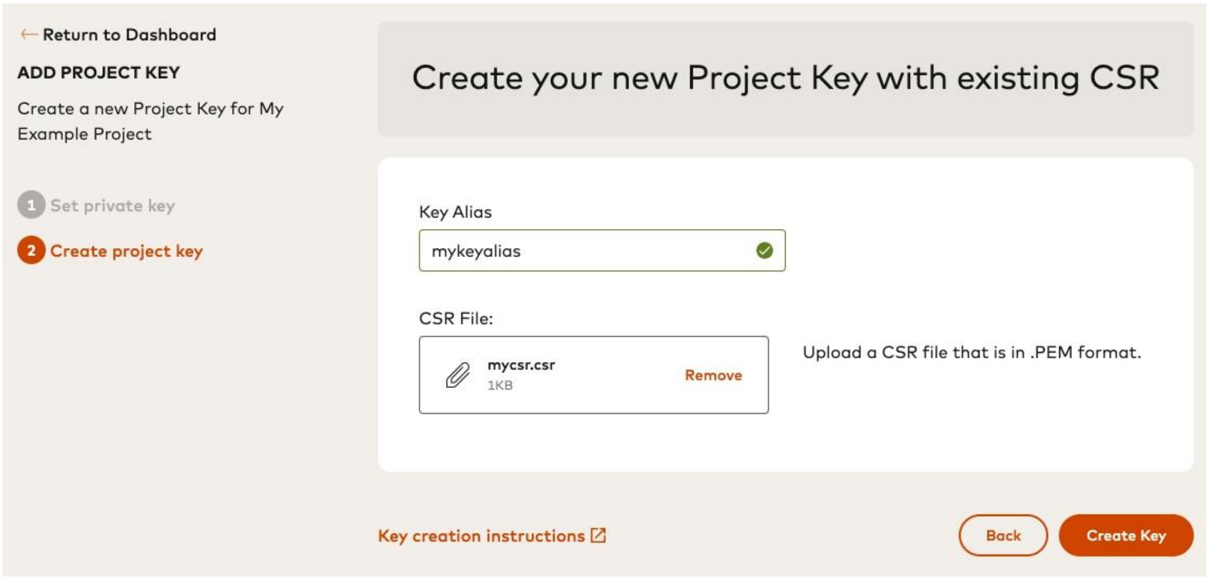
Keystore Password *

•••••••••••

The PKCS#12 keystore contains the private key used for signing requests to Mastercard APIs. The key will be generated inside your browser and will not be stored by Mastercard Developers.

CopiedMethod 2

- You manually upload a certificate signing request (CSR)opens in a new tab generated in your own system.
- Mastercard issues an X509 certificateopens in a new tab matching your private key.
- Mastercard generates a new consumer key.
- If needed, you can request the X509 certificateopens in a new tab and/or the Mastercard root and intermediate certificates in the Support section.



Create a new Project Key for My Example Project

Create your new Project Key with existing CSR

Key Alias

mykeyalias

CSR File:

mycsr.csr
1KB

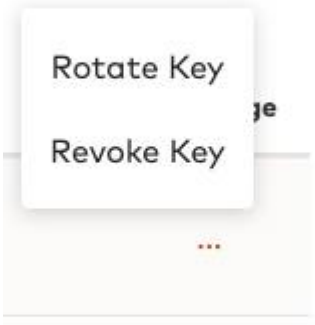
Upload a CSR file that is in .PEM format.

CopiedRenewing Keys

Please reference our *Renewing Your Keys* guides.

CopiedRevoking Keys

To revoke a key, select "*Revoke Key*" in the *Manage* menu:



Warning:

Mastercard doesn’t know and never stores your private keys. It means *only you* can sign your requests, but it also means there is no way to recover a key you lost!

CopiedClient Libraries

Mastercard provides client authentication librariesopens in a new tab in several languages you can integrate to your project or use as reference OAuth 1.0a implementations:

Download/install	opens in a new tab	opens in a new tab	opens in a new tab	ope tab
View on GitHub	opens in a new tab	opens in a new tab	opens in a new tab	ope tab

To get started, simply add to your project the package matching your application development language. You can also refer to the different README.mdopens in a new tab files for detailed how-to information.

Tip:

OAuth 1.0a presents many edge cases that are easy to miss. We strongly encourage you to use existing OAuth libraries rather than implementing the specification yourself.

CopiedInsomnia Plugin

While integrating with Mastercard APIs, you can also try our plugin for Insomniaopens in a new tab. Click the links below to browse the project.

Download/install					

	opens in a new tab				
View on GitHub	opens in a new tab				

CopiedUsing Postman

You can use Postman application to call Mastercard APIs. Refer to our Postman Tutorial, which provides detailed how-to information for configuring Postman with the needed authentication.

CopiedDevelopment Tips

CopiedError Troubleshooting

In case your request is rejected, you will receive a response with the following fields:

```
{
  "Source": "{Source}",
  "ReasonCode": "{ReasonCode}",
  "Description": "{Description}",
  "Recoverable": false
}
```

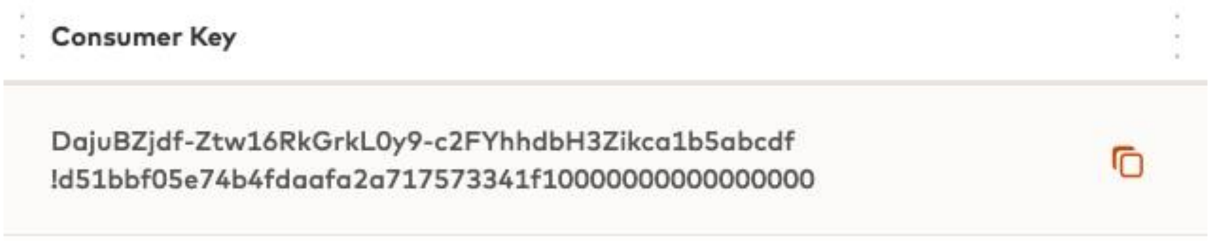
Here's a list of common error reason codes and how to fix your request.

CopiedINVALID_OAUTH_SBS

The Authorization header is missing from the request or OAuth parameters are missing from the header. See also: The Authorization Header.

CopiedINVALID_OAUTH_CONSUMER_KEY

The value for oauth_consumer_key doesn't conform to the length/format requirements (97-character length, with an exclamation mark in the middle). Make sure you copied the key from your Developer Dashboard by clicking the copy icon next to your consumer key.



Consumer Key

DajuBZjdf-Ztw16RkGrkL0y9-c2FYhhdbH3Zikca1b5abcdf
!d51bbf05e74b4fdaafa2a717573341f10000000000000000

CopiedINVALID_CLIENT_ID, INVALID_KEY_ID

The value for oauth_consumer_key doesn’t match an existing key or doesn’t have access to the requested API.

CopiedINVALID_OAUTH_SIGNATURE_METHOD

The value for oauth_signature_method is invalid or not supported.

CopiedINVALID_OAUTH_TIMESTAMP

The value for oauth_timestamp was rejected. Check your clock is accurate (you can use public NTP poolsopens in a new tab for that). See also: oauth_nonce and oauth_timestamp.

CopiedOAUTH_NONCE_USED

The value for oauth_nonce was already used. Check you actually send a different nonce with each new request and that your system is generating enough entropy. See also: oauth_nonce and oauth_timestamp.

CopiedINVALID_BODY_HASH

The value for oauth_body_hash is incorrect. You can check:

- The digest algorithm used for computing the oauth_body_hash value matches the one used in oauth_signature_method
- There is no network proxy or equipment updating the request body after the request is sent
- oauth_body_hash contains the digest of an empty string when the request does not have an entity body
- In case the service requires encryption, make sure the payload is encrypted *before* the request is signed.

CopiedAUTHENTICATION_FAILED

The oauth_signature value can’t be verified (wrong signing key, wrong signing base string, …). See also: Signed Requests.

CopiedINVALID_KEY

Renew your signing key or create a new one. See also: Getting Keys for Your Application.

CopiedExporting Your Signing Key

Your private signing key is encrypted in a PKCS#12 password-protected file you can directly use with Mastercard client libraries. Here are some commands for exporting RSA keys to alternative formats, for instance in case you want to use them with other libraries.

Warning:

Always keep your private signing keys safe, in a password-protected or hardware key store!

CopiedPKCS#1 PEM

To export your key to a PKCS#1 PEM file (BEGIN RSA PRIVATE KEY), run:

```
openssl pkcs12 -in key.p12 -nodes | openssl rsa -outform pem -out pkcs1-key.pem
```

CopiedPKCS#8 PEM

To export your key to a PKCS#8 PEM file (BEGIN PRIVATE KEY), run:

```
openssl pkcs12 -in key.p12 -nodes | openssl pkcs8 -topk8 -nocrypt -out pkcs8-key.pem
```

CopiedBinary DER-encoded PKCS#8

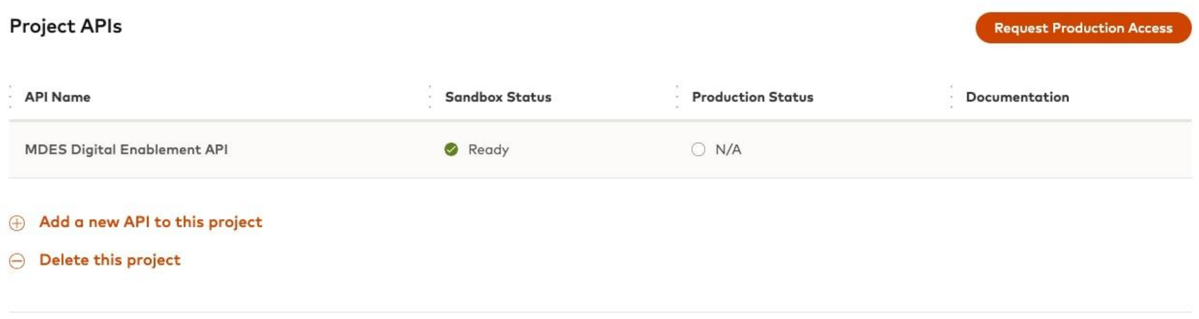
To export your key to a raw PKCS#8 file (binary), run:

```
openssl pkcs12 -in key.p12 -nodes | openssl rsa -outform der -out pkcs8-key.der
```

CopiedFrequently Asked Questions

CopiedCan the Same Keys Be Used for Consuming Multiple APIs?

Yes. The consumer and signing keys are created for a project and projects can reference more than one API. You can manage APIs for your project in the *Project APIs'* section of the Summary page.



Project APIs

API Name	Sandbox Status	Production Status	Documentation
MDES Digital Enablement API	✓ Ready	○ N/A	

CopiedWill My Consumer Key Change After I Renewed My Signing Key?

The consumer key and signing key are linked together, as you use the signing key to prove you own the consumer key.

- If you simply renew your keys before they expire, your consumer key will stay the same

- If you revoke the keys, then you will have to create new keys and a new consumer key will be generated for you

See also: Getting Keys for Your Application

CopiedHow Can I Validate My OAuth Implementation?

OAuth 1.0a presents many edge cases that are easy to miss. We strongly encourage you to use existing OAuth libraries rather than implementing the specification yourself. For more information, see the client libraries section above.