Developer Portal

Login

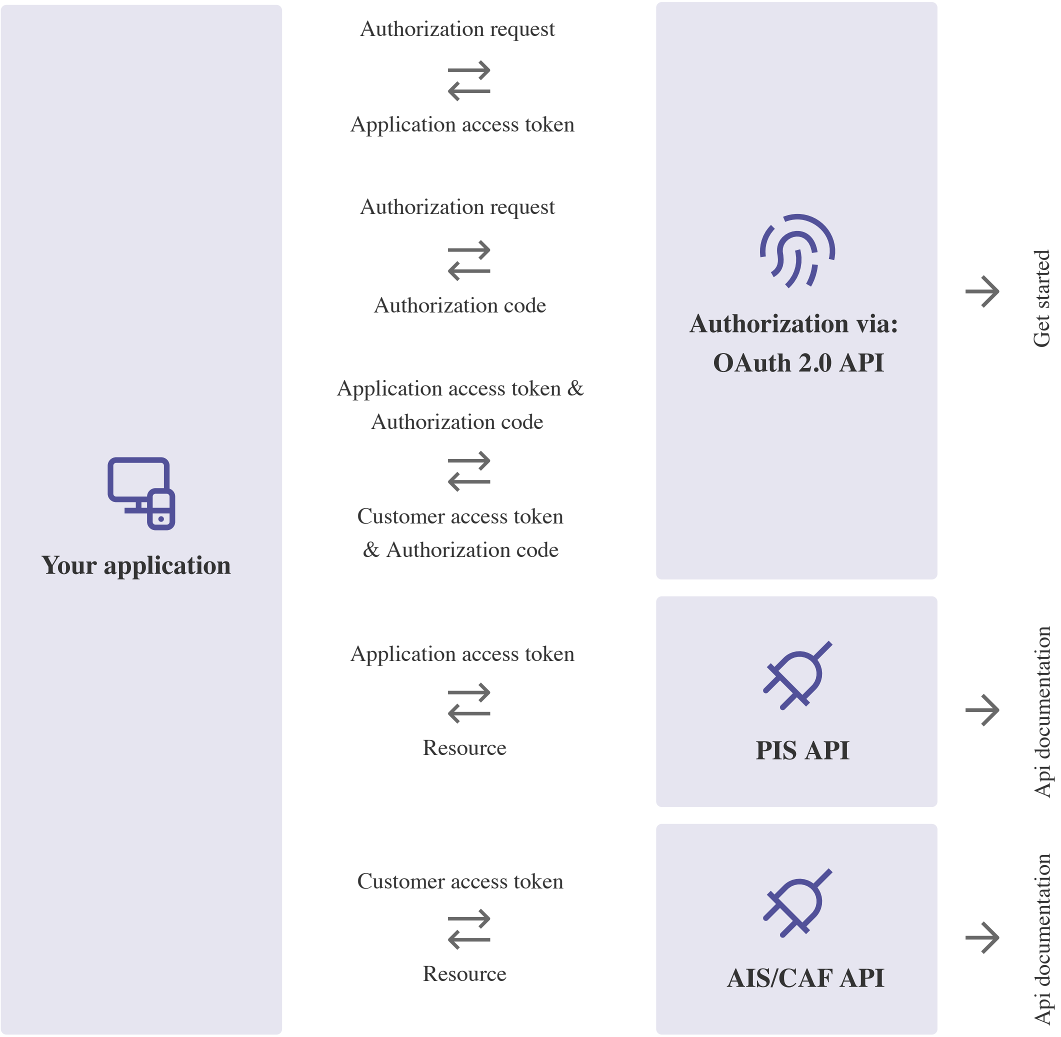
Home Products Resources Inspiration Status Contact

Resources / Psd2

PSD2 Get Started guide for new TPPs to ING

Our Payment Services Directive (PSD2) APIs provide secure access to the Payment Initiation Service (PIS), Account Information Service (AIS) and the Confirmation Availability of Funds (CAF) Service. On this page you can find information on how to test connectivity to PSD2 APIs in ING's sandbox environment and how to connect to them in the production environment.

Provide feedback



Sandbox

Our sandbox environment enables you to test the connectivity to our APIs. The sandbox contains a simulator that simulates API responses. This section explains how to consume PSD2 APIs (PIS, AIS or CAF) in the sandbox environment from your application.

💡 The simulator only supports the exact request explained in the documentation tab of the PSD2 API. In order to get a successful response, a request should match the specific header, path, body and query parameters as per the documentation or else you will receive an HTTP status code 404. If you use an incorrect application access token or the signed signature is incorrect you will receive an HTTP status code 401.

Pre-requisite to consume PSD2 APIs in the sandbox environment

In order to test the connectivity to our PSD2 APIs in the sandbox, you are not required to register with the National Competence Authority (NCA or regulator).

Step 1: Register your application

For the Sandbox environment, we have pre-registered an application and provided a set of example certificates that you can use to access PSD2 APIs (PIS, AIS and CAF) in the sandbox environment. These certificates will give you the scopes of all the PSD2 APIs. For the production environment, you will have to registerd your application, explained below under the Production section.

Download the certificates and the private keys from the below links:

- example_client_signing.cer
- example_client_signing.key
- example_client_tls.cer
- example_client_tls.key

The example certificates are valid for two years from the date of issue. New certificates are published in the week before the expiration date.

The serial number of the example certificate for http signature signing is

"SN=49BA5C71". This is required once in the next step when you request an application access token.

For the remainder of this tutorial, we will assume that you have downloaded these private keys and certificates to /certs sub-directory as shown in the example below.

```
johnd@johns-machine:~/certs$ ls -l
total 4
johnd@johns-machine:~/certs$ ls -l
-rw-rw-r-- 1 johnd johnd 1409 Mar 6 22:40
example_client_signing.cer
-rw-rw-r-- 1 johnd johnd 1679 Mar 4 14:59
example_client_signing.key
-rw-rw-r-- 1 johnd johnd 1432 Mar 4 14:59
example_client_tls.cer
-rw-rw-r-- 1 johnd johnd 1675 Mar 4 14:59
example_client_tls.key
```

While registering a new application is not possible, we are providing a script for testing the request against our stubbed environment. Please be aware that a succesful call will not register a new application, but will give you the opportunity to test the request.

```
#!/bin/bash

# Note: current script was tested with the
following combinations:
# curl 8.6.0 and OpenSSL 3.2.1 and bash version
GNU bash, version 3.2.57(1)-release (arm64-apple-
darwin23)
# curl 8.7.1 and OpenSSL 3.1.4 and bash version
GNU bash, version 3.2.57(1)-release (arm64-apple-
darwin23)
# curl 8.7.1 and LibreSSL 3.3.6 and bash version
GNU bash, version 3.2.57(1)-release (x86_64-apple-
darwin23)
```

```
# curl 8.9.1 and OpenSSL 3.2.2 and bash version
GNU bash, version 5.2.26(1)-release (x86_64-pc-
msys) windows 10

# In order to use the script for production
environment, replace:
# httpHost with "https://api.ing.com"
# signCertificate and signKey with your own
signing certificates
# tlsCertificate and tlsKey with your own PSD2 tls
certificates
# payload with your request body

httpHost="https://api.sandbox.ing.com"

## THE SCRIPT USES THE DOWNLOADED EXAMPLE EIDAS
CERTIFICATES
signCertificate="./example_client_signing.cer"
signKey="./example_client_signing.key"
tlsCertificate="./example_client_tls.cer"
tlsKey="./example_client_tls.key"


##################################################
#############################
#                        1. GENERATE JWS SIGNATURE
#
# Issues might appear for RSA-PSS with older
OpenSSL versions.                #
##################################################
#############################

httpMethod="POST"
reqPath="/oauth2/applications"

payload="{ \"contact\": \"test@ingsandbox.com\",
\"redirect_uris\":[ \"https://example.com\",
\"https://another-example.com\"]}"
```

```
echo Creating x-jws-signature for calling
"$httpMethod" "$reqPath" with payload "$payload"

## Step 1: Create JWS Protected Header
# Description: Produce JWS header parameters which
define how the signature is created.

# generate the sha-256, Base64url, for the signing
certificate
base64UrlFingerprint=$(openssl x509 -noout -
fingerprint -sha256 -inform pem -in
$signCertificate | cut -d'=' -f2 | sed s/://g  |
xxd -r -p | base64 | tr -d '=' | tr '/+' '_-' | tr
-d '\n')
echo Base64url for the signing certificate:
"$base64UrlFingerprint"

# generate the current signing time, encoded using
RFC 3339 Internet time format for UTC without
fractional seconds (e.g. "2019-11-19T17:28:15Z")
sigT=$(date -u +"%Y-%m-%dT%H:%M:%SZ")
echo Current signing time: "$sigT"

# create the JWS Protected Header with the sigD
parameters containing the mandatory headers
jwsHeader='{"b64":false,"x5t#S256":"'"$base64UrlFi
ngerprint"'","crit":[ "sigT", "sigD",
"b64"],"sigT":"'"$sigT"'","sigD":{ "pars":[ "
(request-target)", "digest", "content-type" ],
"mId":"http://uri.etsi.org/19182/HttpHeaders"},"al
g":"PS256"}'
echo JWS Protected Header: "$jwsHeader"

## Step 2: Encode JWS Protected Header into
Base64url
# Description: Convert JWS Protected Header
(without line breaks or extra spaces) into a
Base64url encoded string.
```

```
jwsHeaderBase64URL=$(echo -n "$jwsHeader" |
openssl base64 -e -A | tr -d '=' | tr '/+' '_-' |
tr -d '\n')
echo Encoded JWS Protected Header:
"$jwsHeaderBase64URL"

## Step 3a: Compute Digest of HTTP Body
# Description: Calculate hash of the HTTP Body
(payload without HTTP header and following empty
line).
digest="SHA-256="$(echo -n "$payload" | openssl
dgst -binary -sha256 | openssl base64)
echo Digest of HTTP Body: "$digest"

## Step 3b: Collect HTTP Headers to be signed
# Description: Create HTTP header string, as
selected using the JWS header parameter sigD,
including Digest (base64 encoded).
# After each line, a line feed must be present.
lowerCaseHttpMethod=`echo $httpMethod | tr
[:upper:] [:lower:]`

signingString="(request-target):
$lowerCaseHttpMethod $reqPath
digest: $digest
content-type: application/json"
echo HTTP Headers to be signed: "$signingString"

## Step 4: Prepare input for Signature Value
Computation
# Description: Combine Base64url encoded JWS
Protected Header with HTTP Header to be signed,
separated by ".", ready for computation of
signature value.
inputForSignatureValueComputation="$jwsHeaderBase6
4URL.$signingString"

## Step 5: Compute JWS Signature Value
```

```
# Description: Compute the digital signature
cryptographic value calculated over a sequence of
octets derived from the JWS Protected Header and
HTTP Header Data to be Signed.
# This is created using the signing key associated
with the certificate identified in the JWS
Protected Header "x5t#S256" and using the
signature algorithm identified by "alg".
jwsSignatureValue=$(printf %s
"$inputForSignatureValueComputation"| openssl dgst
-sha256 -sign $signKey -sigopt
rsa_padding_mode:pss | openssl base64 -A  | tr -d
'=' | tr '/+' '_-' | tr -d '\n')
echo JWS Signature Value: "$jwsSignatureValue"

## Step 6: Build JSON Web Signature
# Description: Create JSON Web Signature
containing the Base64url encoded JWS Protected
header and ".." and the JWS Signature Value.
# This is encoded using JWS compact serialisation
with the HTTP Header Data to be Signed detached
from the signature.
jwsSignature=$jwsHeaderBase64URL..$jwsSignatureVal
ue
echo x-jws-signature: "$jwsSignature"

##################################################
#############################
#                                2. ONBOARDING
#
##################################################
#############################

response=$(curl -k -vv -X "$httpMethod"
"${httpHost}${reqPath}" \
           -H 'Content-Type: application/json' \
           -H "Digest: $digest" \
           -H "x-jws-signature: $jwsSignature" \
           -H "TPP-Signature-Certificate: $(tr -d
"\n\r" < $signCertificate)" \
           -d "$payload" \
           --cert $tlsCertificate \
           --key $tlsKey)

echo "$response" | jq '.'
```

Step 2: Retrieve your application's registered metadata

2.1. Pre-requisite - Request an mTLS only application access token

When using the **/oauth2/applications/** endpoint, the way you request an access token from ING is different than for the other interactions. In broad lines, you will not have to sign this request, but rely on mutual TLS. We have described this for the Production environment.

2.2. Retrieve your application's registered metadata via GET /oauth2/applications/me or /oauth2/applications/{id}

In the Sandbox environment we are providing a pre-registered application and a set of example certificates. In order to test this step, we are providing a script for retrieving the pre-registered applications metadata. Please be aware that this is being tested against a stubbed environment and PUT updates (step 2.b.) will not be reflected and this request will always return the pre-registered application metadata.

```
#!/bin/bash

# Note: current script was tested with the
following combinations:
# curl 8.6.0 and OpenSSL 3.2.1 and bash version
GNU bash, version 3.2.57(1)-release (arm64-apple-
darwin23)
# curl 8.7.1 and OpenSSL 3.1.4 and bash version
GNU bash, version 3.2.57(1)-release (arm64-apple-
darwin23)
```

```
# curl 8.7.1 and LibreSSL 3.3.6 and bash version
GNU bash, version 3.2.57(1)-release (x86_64-apple-
darwin23)
# curl 8.9.1 and OpenSSL 3.2.2 and bash version
GNU bash, version 5.2.26(1)-release (x86_64-pc-
msys) windows 10

# In order to use the script for production
environment, replace:
# httpHost with "https://api.ing.com"
# client_id with your own client id
# tlsCertificate and tlsKey with your own PSD2 tls
certificates

httpHost="https://api.sandbox.ing.com"

# client_id as provided in the documentation
client_id="5ca1ab1e-c0ca-c01a-cafe-154deadbea75"

## THE SCRIPT USES THE DOWNLOADED EXAMPLE EIDAS
CERTIFICATES
tlsCertificate="./example_client_tls.cer"
tlsKey="./example_client_tls.key"

##################################################
#############################
#                 1. REQUEST APPLICATION ACCESS
TOKEN  (MTLS)                  #
##################################################
#############################

httpMethod="POST"
reqPath="/oauth2/token"

# client_id path param required for mtls flow
# You can also provide scope parameter in the body
E.g.
"grant_type=client_credentials&scope=granting"
```

```
# scope is an optional parameter. The downloaded
certificate contains all available scopes.
# If you don't provide a scope, the accessToken is
returned for all scopes available in certificate
payload="grant_type=client_credentials&client_id=$
{client_id}"

response=$(curl -k -vv -X  "$httpMethod"
"${httpHost}${reqPath}" \
            -H 'Accept: application/json' \
            -H 'Content-Type: application/x-www-
form-urlencoded' \
            -d "${payload}" \
            --cert $tlsCertificate \
            --key $tlsKey)

echo "$response" | jq '.'
access_token=`echo $response | jq -r
'.access_token'`


##################################################
#############################
#                        2. GET APPLICATIONS'
METADATA                        #
##################################################
#############################
httpMethod="GET"
reqPath="/oauth2/applications/"$client_id

# Curl request method must be in uppercase e.g
"POST", "GET"
response=$(curl -k -vv "$httpHost$reqPath" \
-H "Authorization: Bearer $access_token" \
--cert $tlsCertificate \
--key $tlsKey)

echo "$response" | jq '.'
```

2b. (optional) Update the registered application's metadata, especially the Redirect URL(s) and email address via PUT /oauth2/applications/{id}

As we are providing a pre-registered application and a set of example certificates updating the configuration is not possible in the Sandbox environment. In order to test this step, we are providing a script to test against our stubbed environenment to validate your request.

```
#!/bin/bash

# Note: current script was tested with the following combinations:
# curl 8.6.0 and OpenSSL 3.2.1 and bash version GNU bash, version 3.2.57(1)-release (arm64-apple-darwin23)
# curl 8.7.1 and OpenSSL 3.1.4 and bash version GNU bash, version 3.2.57(1)-release (arm64-apple-darwin23)
# curl 8.7.1 and LibreSSL 3.3.6 and bash version GNU bash, version 3.2.57(1)-release (x86_64-apple-darwin23)
# curl 8.9.1 and OpenSSL 3.2.2 and bash version GNU bash, version 5.2.26(1)-release (x86_64-pc-msys) windows 10

# In order to use the script for production environment, replace:
# httpHost with "https://api.ing.com"
# client_id with your own client id
# signCertificate and signKey with your own signing certificates
# tlsCertificate and tlsKey with your own tls certificates
# updatePayload with your request body
```

```
httpHost="https://api.sandbox.ing.com"

# client_id as provided in the documentation
client_id="5ca1ab1e-c0ca-c01a-cafe-154deadbea75"

## THE SCRIPT USES THE DOWNLOADED EXAMPLE EIDAS
CERTIFICATES
signCertificate="./example_client_signing.cer"
signKey="./example_client_signing.key"
tlsCertificate="./example_client_tls.cer"
tlsKey="./example_client_tls.key"

##################################################
#############################
#                 1. REQUEST APPLICATION ACCESS
TOKEN  (MTLS)                 #
##################################################
#############################

httpMethod="POST"
reqPath="/oauth2/token"

# client_id path param required for mtls flow
# You can also provide scope parameter in the body
E.g.
"grant_type=client_credentials&scope=granting"
# scope is an optional parameter. The downloaded
certificate contains all available scopes.
# If you don't provide a scope, the accessToken is
returned for all scopes available in certificate
tokenPayload="grant_type=client_credentials&client
_id=${client_id}"

response=$(curl -k -vv -X "$httpMethod"
"${httpHost}${reqPath}" \
            -H 'Accept: application/json' \
            -H 'Content-Type: application/x-www-
```

```
form-urlencoded' \
            -d "${tokenPayload}" \
            --cert $tlsCertificate \
            --key $tlsKey)

echo $response | jq '.'
access_token=`echo $response | jq -r
'.access_token'`

##################################################
#############################
#                        2. GENERATE JWS SIGNATURE
#
# Issues might appear for RSA-PSS with older
OpenSSL versions.                #
##################################################
#############################

httpMethod="PUT"
reqPath="/oauth2/applications/"$client_id

updatePayload="{ "contact": "test@ingsandbox.com",
"redirect_uris":[ "https://example.com",
"https://another-example.com"]}"

echo Creating x-jws-signature for calling
"$httpMethod" "$reqPath" with payload
"$updatePayload"

## Step 1: Create JWS Protected Header
# Description: Produce JWS header parameters which
define how the signature is created.

# generate the sha-256, Base64url, for the signing
certificate
base64UrlFingerprint=$(openssl x509 -noout -
fingerprint -sha256 -inform pem -in
$signCertificate | cut -d'=' -f2 | sed s/://g  |
xxd -r -p | base64 | tr -d '=' | tr '/+' '_-' | tr -d '\n')
echo Base64url for the signing certificate: "$base64UrlFingerprint"

# generate the current signing time, encoded using RFC 3339 Internet time format for UTC without fractional seconds (e.g. "2019-11-19T17:28:15Z")
sigT=$(date -u +"%Y-%m-%dT%H:%M:%SZ")
echo Current signing time: "$sigT"

# create the JWS Protected Header with the sigD parameters containing the mandatory headers
jwsHeader='{"b64":false,"x5t#S256":"'"$base64UrlFingerprint"'","crit":[ "sigT", "sigD", "b64"],"sigT":"'"$sigT"'","sigD":{ "pars":[ "(request-target)", "digest", "content-type" ], "mId":"http://uri.etsi.org/19182/HttpHeaders"},"alg":"PS256"}'
echo JWS Protected Header: "$jwsHeader"

## Step 2: Encode JWS Protected Header into Base64url
# Description: Convert JWS Protected Header (without line breaks or extra spaces) into a Base64url encoded string.
jwsHeaderBase64URL=$(echo -n "$jwsHeader" | openssl base64 -e -A | tr -d '=' | tr '/+' '_-' | tr -d '\n')
echo Encoded JWS Protected Header: "$jwsHeaderBase64URL"

## Step 3a: Compute Digest of HTTP Body
# Description: Calculate hash of the HTTP Body (payload without HTTP header and following empty line).
digest="SHA-256="$(echo -n "$updatePayload" | openssl dgst -binary -sha256 | openssl base64)
```

```
echo Digest of HTTP Body: "$digest"

## Step 3b: Collect HTTP Headers to be signed
# Description: Create HTTP header string, as
selected using the JWS header parameter sigD,
including Digest (base64 encoded).
# After each line, a line feed must be present.
lowerCaseHttpMethod=`echo $httpMethod | tr
[:upper:] [:lower:]`

signingString="(request-target):
$lowerCaseHttpMethod $reqPath
digest: $digest
content-type: application/json"
echo HTTP Headers to be signed: "$signingString"

## Step 4: Prepare input for Signature Value
Computation
# Description: Combine Base64url encoded JWS
Protected Header with HTTP Header to be signed,
separated by ".", ready for computation of
signature value.
inputForSignatureValueComputation="$jwsHeaderBase6
4URL.$signingString"

## Step 5: Compute JWS Signature Value
# Description: Compute the digital signature
cryptographic value calculated over a sequence of
octets derived from the JWS Protected Header and
HTTP Header Data to be Signed.
# This is created using the signing key associated
with the certificate identified in the JWS
Protected Header "x5t#S256" and using the
signature algorithm identified by "alg".
jwsSignatureValue=$(printf %s
"$inputForSignatureValueComputation"| openssl dgst
-sha256 -sign $signKey -sigopt
rsa_padding_mode:pss | openssl base64 -A  | tr -d
'=' | tr '/+' '_-' | tr -d '\n')
echo JWS Signature Value: "$jwsSignatureValue"

## Step 6: Build JSON Web Signature
# Description: Create JSON Web Signature
containing the Base64url encoded JWS Protected
header and ".." and the JWS Signature Value.
# This is encoded using JWS compact serialisation
with the HTTP Header Data to be Signed detached
from the signature.
jwsSignature=$jwsHeaderBase64URL..$jwsSignatureVal
ue
echo x-jws-signature: "$jwsSignature"


##################################################
#############################
#                        3. UPDATE APPLICATION'S
METADATA                              #
##################################################
#############################

response=$(curl -k -vv -X "$httpMethod"
"${httpHost}${reqPath}" \
            -H 'Content-Type: application/json' \
            -H "Digest: $digest" \
            -H "x-jws-signature: $jwsSignature" \
            -H "TPP-Signature-Certificate: $(tr -d
"\n\r" < $signCertificate)" \
            -H "Authorization: Bearer
$access_token" \
            -d "$updatePayload" \
            --cert $tlsCertificate \
            --key $tlsKey)

echo "$response" | jq '.'
```

Step 3: Request an authorization code and the customer access token

This step is only required for APIs that require customer consent (eg. for PSD2: AIS and CAF; for Premium: Open Account Information API) since these APIs allow you to access sensitive customer data. ING has implemented OAuth 2.0 authorization code flow to help you obtain the authorization of the customer. To learn more about the Authorization code flow refer to the Authorization code flow section in our OAuth 2.0 documentation

In this step a request for customer consent is initiated and an authorization code is generated. Using the application access token and the authorization code, your application obtains a customer access token. Finally, your application uses the customer access token to consume the API that requires customer consent.

In the production environment, the customer will log in and give consent that your application can retrieve account information. In the sandbox environment we have simulated this behavior by offering a set of pre-defined test customers with specific consents. In this simulation, you first need to choose the country of the customer, and then select a customer with a specific test scenario. Below is a step-by-step explanation of this process:

a) Redirect the customer to ING's authorization application and let the customer authorize your application

Important: Redirect the customer to their default browser in which they can confirm that they are communicating with the trusted ING website. You are not allowed to embed ING's authorization page in your application in any way! ING can block your application from using ING APIs when identifying such an implementation.

Redirect the customer to ING's authorization application URL, by appending your application specific parameters:

```
https://myaccount.ing.com/authorize/v2/[COUNTRY_CO
DE]?

client_id=[YOUR_CLIENT_ID]
&scope=[SCOPES_SPACE_SEPARATED_AND_URLENCODED]
&state=[SOME_ARBITRARY_BUT_UNIQUE_STRING]
&redirect_uri=[YOUR_URL_ENCODED_REDIRECT_URI]
&response_type=code
```

Because this is a simulation of production, you will have to select a customer with a specific test scenario you would like to test.

COUNTRY_CODE QUERY PARAMETER

Is an optional parameter and allows you to redirect your customer to the specified ING country login page. The format is a two-letter value (ISO 3166-1). If you don't add the country_code, the ING customer will be redirected to an ING country selection page.

Country	Code
Belgium	BE
The Netherlands	NL
Spain	ES
Romania	RO
Luxembourg	LU
Italy	IT
Germany	DE
Wholesale Banking	WB

client_id QUERY PARAMETER

Required parameter, the unique client_id obtained when you requested the application access token. If you are using our example certificates, its value is **5ca1ab1e-c0ca-c01a-cafe-154deadbea75**.

scope QUERY PARAMETER

Required parameter, a list of scopes separated by spaces for which an authorization request is being sent. This could be a subset of the scopes you application is allowed to request. The value must be URL encoded, where spaces may be encoded to either "+" or "%20".

state QUERY PARAMETER

Required parameter, can be used by your application to maintain state between HTTP redirects and achieve per-request customization of your redirect URL, please use the "state" parameter as described in the OAuth 2.0 RFC.

The state parameter must not be blank. Allowed characters for the "state" parameter are:

> Alphanumeric characters in range A-Z, a-z, 0-9
>
> Special characters =%#,;+/-_
>
> Any whitespace character \r \n \t \f \v

redirect_uri QUERY PARAMETER

Optional parameter, a specific redirect URI where the customer will return to once the authorization process is ended. The value must be URL encoded. This parameter is optional if your application has registered only a single redirect URL with ING. This is a mandatory parameter in case your application has registered multiple redirect URLs and should be one of them.

response_type QUERY PARAMETER

Required parameter, for PSD2 flow, the value must be "code".

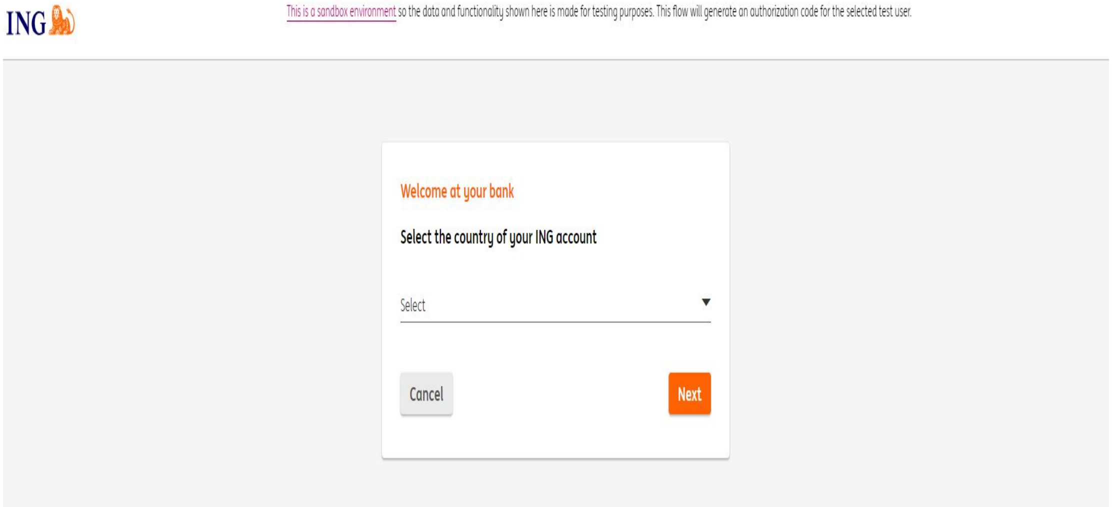
Example redirect url

An example URL to redirect the customer to start the authorization flow should look like this:

```
https://myaccount.sandbox.ing.com/authorize/v2/NL?&client_id=5ca1ab1e-c0ca-c01a-cafe-154deadbea75&state=ANY_ARBITRARY_VALUE&response_type=code&redirect_uri=https%3A%2F%2Fwww.example.com%2F&scope=payment-accounts%3Abalances%3Aview%20payment-accounts%3Atransactions%3Aview
```

Country selection screen

This screen will only be shown when you have not provided the country_code parameter in the redirect URL of the ING authorization application.



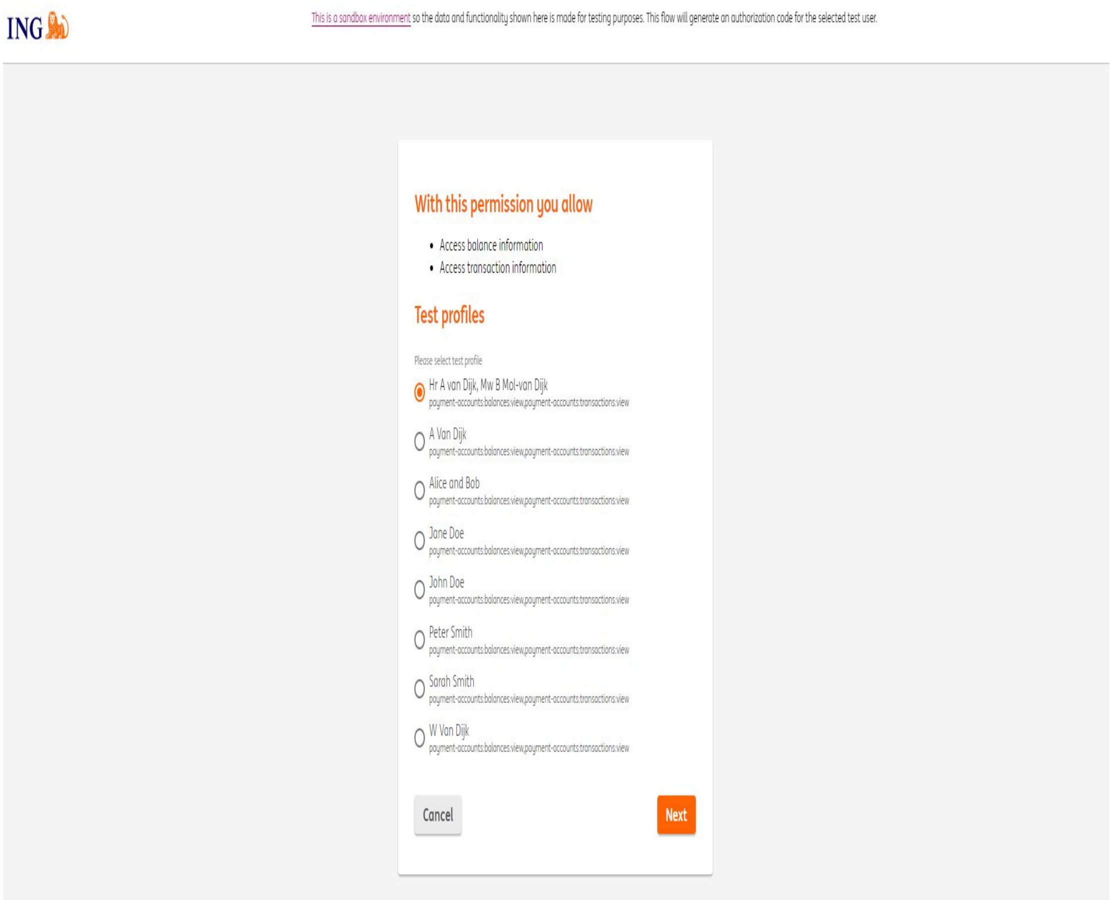
This is a sandbox environment so the data and functionality shown here is made for testing purposes. This flow will generate an authorization code for the selected test user.

Welcome at your bank

Select the country of your ING account

Test profile selection screen

Once you have selected the country, the application will display the list of test profiles to choose from as shown below. In production the ING customers will see all their accounts and would be able to select multiple accounts.



This is a sandbox environment so the data and functionality shown here is made for testing purposes. This flow will generate an authorization code for the selected test user.

With this permission you allow

- Access balance information
- Access transaction information

Test profiles

Please select test profile

● Hr A van Dijk, Mw B Mol-van Dijk
payment-accounts:balances:view,payment-accounts:transactions:view

○ A Van Dijk
payment-accounts:balances:view,payment-accounts:transactions:view

○ Alice and Bob
payment-accounts:balances:view,payment-accounts:transactions:view

○ Jane Doe
payment-accounts:balances:view,payment-accounts:transactions:view

○ John Doe
payment-accounts:balances:view,payment-accounts:transactions:view

○ Peter Smith
payment-accounts:balances:view,payment-accounts:transactions:view

○ Sarah Smith
payment-accounts:balances:view,payment-accounts:transactions:view

○ W Van Dijk
payment-accounts:balances:view,payment-accounts:transactions:view

This list is filtered by country (provided via country selection screen) and by scope (required query parameter).

b) Redirect the customer back to your application and acquire an authorization code

Once successful, ING's authorization application redirects the customer to the URL provided in the redirect_uri query parameter along with the authorization code. For example:

```
https://www.example.com/?state=
[YOUR_STATE_VALUE]&code=876fyb93-a597-44r2-838f-
71b2a57699tb
```

The "code" parameter in the redirected URL contains the dynamically generated authorization code for the selected test profile. This authorization code is used in order to request a customer access token.

c) Request an application access token

Here you can download the sample script to request an application access token for the PSD2 APIs in the sandbox.

An application access token is requested by calling ING's OAuth 2.0 API token endpoint: /oauth2/token. To call this API endpoint, you must create a signature and sign the HTTP request. We have explained below how to calculate the digest and the date for creating a signature.

Want to learn more about HTTP message Signing?

Calculate Digest

```
payload="grant_type=client_credentials"
payloadDigest=`echo -n "$payload" | openssl dgst -binary -sha256 | openssl base64`
digest=SHA-256=$payloadDigest
```

"Digest" header is used by our server to verify the integrity of the message body that you send to us when calling any of our APIs.

A request for an application access token can contain multiple scope-tokens (separated by spaces) in the "scope" parameter of the payload. The "scope" parameter value must be encoded using x-www-form-urlencoded-compliant URL encoding and the space character must be encoded with "+" instead of "%20" as explained below.

The scope parameter value with single scope-token "payment-accounts:balances:view".

```
grant_type=client_credentials&scope=payment-accounts%3Abalances%3Aview
```

The scope parameter value with multiple scope-tokens "payment-accounts:balances:view" and "payment-accounts:transactions:view".

```
grant_type=client_credentials&scope=payment-accounts%3Abalances%3Aview%20payment-
```

```
accounts%3Atransactions%3Aview
```

When no scope is explicitly requested the application access token will contain all the scope-tokens available in the certificate that you are using. Refer to the "Reference" section of the PSD2 API you want to consume to see the available scopes.

Calculate date

```
reqDate=$(LC_TIME=en_US.UTF-8 date -u "+%a, %d %b
%Y %H:%M:%S GMT")
```

💡 Note that there should not be a gap of more than 3 minutes between the time when the PSD2 API endpoint request is made and the time value in the "date" header field of the request.

Calculate signature for signing your request

```
httpMethod="post"
reqPath="/oauth2/token"
signingString="(request-target): $httpMethod
$reqPath
date: $reqDate
digest: $digest"
signature=`printf %s "$signingString" | openssl
dgst -sha256 -sign "${certPath}
example_client_signing.key" -passin
"pass:changeit" | openssl base64 -A`
```

Signature should be dynamically calculated for every PSD2 API endpoint request since the values that are used as input to create the signature change every time.

Once you have created the signature, you are ready to request an application access token using the below header and body parameters:

Request parameters (refer to the sample script available to download at the beginning of this section):

> *curl "${httpHost}${reqPath}"* - Full path of the API endpoint.
>
> *Digest: $(digest)* - Calculated value of digest.
>
> *Date: $(reqDate)* - Calculated value of date. This is the standard HTTP header (see RFC7231). Note that we enforce a strict time frame within which the request must be issued. If the Date header's time value diverges by more than 3 minutes from the time the API request is made, the request will be considered invalid.
>
> *keyId* - It has a format "SN=XXX, where “XXX" is the serial number of the certificate in hexadecimal coding. If you have downloaded the example certificates, the keyId is "SN=49BA5C71". If you are using your own eIDAS certificate, you can extract the serial number and full distinguished name of the CA using the below openssl command:
>
> *TPP-Signature-Certificate* - Copy and paste the downloaded example of certificate "example_client_signing.cer" as a single line string including the BEGIN CERTIFICATE and the END CERTIFICATE lines.
>
> *keyId* - It has a format "SN=XXX, where “XXX" is the serial number of the certificate in hexadecimal coding. If you have downloaded the example certificates, the keyId is "SN=49BA5C71".
>
> *signature* - Calculated value of the signature in order to sign your request.
>
> *Content-Type* - Value set to "application/x-www-form-urlencoded". It forces the body parameters to use the x-www-form-urlencoded compliant URL encoding. Note that space character should be encoded as "+" and not as "%20".

Example request:

```
curl -i -X POST "${httpHost}${reqPath}" \
-H 'Accept: application/json' \
-H 'Content-Type: application/x-www-form-
urlencoded' \
-H "Digest: ${digest}" \
-H "Date: ${reqDate}" \
-H "TPP-Signature-Certificate: -----BEGIN
CERTIFICATE-----
MIIENjCCAx6gAwIBAgIEXkKZv.....LkyWk4Mw1w0TnQLAq+s=
-----END CERTIFICATE-----" \
-H "authorization: Signature
keyId="$keyId",algorithm="rsa-sha256",headers="
(request-target) date
digest",signature="$signature"" \
-d "${payload}" \
--cert "/certs/example_client_tls.cer" \
--key "/certs/example_client_tls.key"
```

Response parameters

access_token - It contains the customer access token.

expires_in - It contains the expiration time in seconds for the customer access token.

client_id - It is a unique id. If you are using the example certificate, its value is **5ca1ab1e-c0ca-c01a-cafe-154deadbea75**. Note down this client_id since you will need it in the next steps to call the API.

expires_in - property contains the expiration time in seconds for the application access token. Once the application access token expires, you should execute this step (Step 2) again in order to re-authenticate your application and requested a new application access token.

scope - It contains the list of all the scopes associated with the "access_token".

Note that the application access token expires after 900 seconds after which you need to execute this step again in order to re-authenticate your application and request a new application access token. It is advisable to re-use the application access token instead of requesting a new token for each API call.

Example response for a successful request:

```
HTTP/2 200

date: Fri, 15 Nov 2019 12:41:32 GMT
content-type: application/json;charset=UTF-8
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
cache-control: no-cache, no-store, max-age=0,
must-revalidate
pragma: no-cache
expires: 0
strict-transport-security: max-age=31622400;
includeSubDomains
x-frame-options: deny

{"access_token":"eyJhbGciOiJkaXIiLCJlb.......","re
fresh_token":"eyJhbGciOiJkaX......","token_type":"
Bearer","expires_in":900,"scope":"payment-
accounts:balances:view payment-
accounts:transactions:view","refresh_token_expires
_in":2592000}
```

You can configure your TLS implementation to verify the trust chain of ING's TLS certificate used for setting up the tunnel between your application and ING. Our PKI certificate is signed by our Root CA which in turn is signed by a commercial CA.

d) Use the authorization code to request the customer access token

Here you can download the sample script to request the customer access token from the authorization code.

The authorization code is valid for only 10 minutes after issuing. Your application should exchange the authorization code for a pair of customer access token and a refresh token within this time frame using the endpoint /oauth2/token and the below request parameters.

You must use the authorization code while calculating the digest value as shown below:

```
payload="grant_type=authorization_code&code=876fyb
93-a597-44r2-838f-71b2a57699tb"
payloadDigest=`echo -n "$payload" | openssl dgst -
binary -sha256 | openssl base64`
digest=SHA-256=$payloadDigest
```

Request parameters (refer to the sample script available to download at the beginning of this section)

curl "${httpHost}${reqPath}" - Full path of the API endpoint.

Authorization: Bearer ${access_token} - The application access token obtained in step b.

Digest: $(digest) - Calculated value of digest using the authorization code.

keyId - Fixed client_id which was provided in Step 1 when downloading the certificates: **5ca1ab1e-c0ca-c01a-cafe-154deadbea75**.

signature - Calculated value of the signature in order to sign your request.

Content-Type - Value set to "application/x-www-form-urlencoded". It forces the body parameters to use the x-www-form-urlencoded compliant URL encoding. Note that space character should be encoded as "+" and not as "%20".

Example request:

```
curl -i -X POST "${httpHost}${reqPath}" \
-H "Accept: application/json" \
-H "Content-Type: application/x-www-form-
urlencoded" \
-H "Digest: ${digest}" \
-H "Date: ${reqDate}" \
-H "Authorization: Bearer ${accessToken}" \
-H "Signature: keyId=\"$keyId\",algorithm=\"rsa-
sha256\",headers=\"(request-target) date
digest\",signature=\"$signature\"" \
-d "${payload}" \
--cert "/certs/example_client_tls.cer" \
--key "/certs/example_client_tls.key"
```

Response parameters

access_token - It contains the customer access token.

expires_in - It contains the expiration time in seconds for the customer access token.

refresh_token - It contains the refresh token that should be used in case the customer access token expires. To learn more about refreshing an access token and revoking a refresh token refer to **"Refresh and revoke access token"** section of this documentation.

refresh_token_expires_in - It contains the expiration time in seconds for the refresh token.

x-ing-response-id - Unique tracing id which should be provided when reporting any issue.

Example response for a successful request:

```
HTTP/2 200

date: Fri, 15 Nov 2019 12:41:32 GMT
```

```
content-type: application/json;charset=UTF-8
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
cache-control: no-cache, no-store, max-age=0,
must-revalidate
pragma: no-cache
expires: 0
strict-transport-security: max-age=31622400;
includeSubDomains
x-frame-options: deny

{"access_token":"eyJhbGciOiJkaXIiLCJlb.......","re
fresh_token":"eyJhbGciOiJkaX......","token_type":"
Bearer","expires_in":900,"scope":"payment-
accounts:balances:view payment-
accounts:transactions:view","refresh_token_expires
_in":2592000}
```

Step 4: Call a PSD2 API with an application access token or the customer access token

With an application access token or the customer access token (in case customer consent is required) you can now call PSD2 API in sandbox. Check the API documentation of the API you want to connect to and determine whether it requires message signing (and which type of message signing) or whether it requires mTLS (mutual TLS) only. You can jump to the next section of this Get Started guide based on these requirements.

When the API requires an "X-JWS-Signature" header to be supplied, it requires JWS signing.

When the API requires a "Signature" header to be supplied, it requires HTTP signing.

When the API does not require the "X-JWS-Signature" or "Signature" headers, it requires mTLS only and no message signing.

Additional header that an API must contain are: Date, Digest, Signature and Authorization.

Refer to the "Reference" and "Documentation" sections of a particular PSD2 API you want to consume. Below we have provided you example scripts to call one of the endpoints of "Account Information API" and "Payment Initiation API":

Payment Initiation API (PIS) "/v1/periodic-payments/{payment-product}" endpoint using application access token. Account Information API (AIS) "/v3/accounts" endpoint using customer access token. Account Information API (AIS) "/v3/accounts/account-id/balances" endpoint using customer access token.

Production

This section explains how to consume PSD2 APIs (PIS, AIS or CAF) in the production environment from your application.

Pre-requisite to consume PSD2 APIs in the production environment

> You must be registered with a National Competent Authority (NCA, or Regulator) as Account Information Service Provider (AISP) and/or Account Servicing Payment Service Providers (ASPSP) and/or Payment Initiation Services Provider (PISP).
>
> You must have the PSD2 eIDAS or OBIE certificates.
>
> The certificate lifespan must not exceed 39 months.

Overview

We are currently in a transition phase and there are two distinct ways to request an application access token and two distinct signature (message integrity) protocols. The below table explains which is required for the specific interaction. Please be aware that other combinations are not possible.

Steps for new clients	Endpoint	Access token and message signature
1. Register your application and details (one time-only required action)	POST /oauth2/applications	no access token + JWS signature API call
2. Retrieve your application's registered metadata	GET /oauth2/applications/{id} GET /oauth2/applications/me	mTLS only access token (no signature)
2b) (optional). Update the registered application's metadata	PUT /oauth2/applications/{id}	mTLS only access token + JWS signature API call
3. Exchange auth code with a token (AIS, CAF)	POST /oauth2/token	HTTPS signature access token + HTTPS signature API call
4. Use the PSD2 APIs	AIS, PIS, CAF specific endpoints	PIS API: "X-JWS-Signature" acces token +X-JWS-Signature API call AIS / CAF API: HTTPS signature access token + HTTPS signature API call

1. Register your application and details (one time-only required action)

Registering will be realized by calling ING's OAuth 2.0 API onboarding endpoint: POST **/oauth2/applications**. This endpoint enables Third Party

Providers (TPPs) onboarding by generating an application identifier (client_id) that will be further used in the authentication flow. The flow will automatically subscribe your application to the allowed ING APIs based on the PSD2 roles in the certificates.

For this you will need to use the JWS signature protocol and the **x-jws-signature** header. Note: this message signature protocol is used only for the PUT and POST endpoints **/oauth2/applications/***.

Parameters

digest HEADER PARAMETER

The "Digest" header is used by our server to verify the integrity of the message body that you send to us when calling any of our APIs.

Creating the digest can be done by calculating the SHA-256 hash of the payload of the message and Base64 encoding the hash. In this example the payload is empty, so the digest will be:

```
digest="SHA-256=47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU="
```

content-type HEADER PARAMETER

The "content-type" header is used to indicate what type of media or resource is being used for the payload in the request, in this case this is "application/json".

TPP-Signature-Certificate HEADER PARAMETER

The PSD2 compliant Qualified Electronic Seal Certificate (QSealC) used to sign the request. The public certificate (PEM format) needs to be the Base64 encoded (including '-----BEGIN CERTIFICATE-----' and '-----END CERTIFICATE-----' parts).

x-jws-signature HEADER PARAMETER

The "x-jws-signature" header is used to send the JSON Web Signature (JWS) for this request.

Want to learn more about the JSON Web Signature specification for Open Banking?
Your signing private key is required to create a signature of the headers and their values that you will send in your request. Check the table under the "Sandbox and Production environments" section of this Get Started guide to check which private key to use for signing.

In the following three steps we will explain how to create the JWS signature that will be used as the value for the "x-jws-signature" header:

Create JWS protected header

The JWS protected header is a JSON object which stores information about the chosen algorithm and the fingerprint of the public certificate that should be used to verify the signature, as well as which HTTP headers are included in the signature and the timestamp when the signature was created.

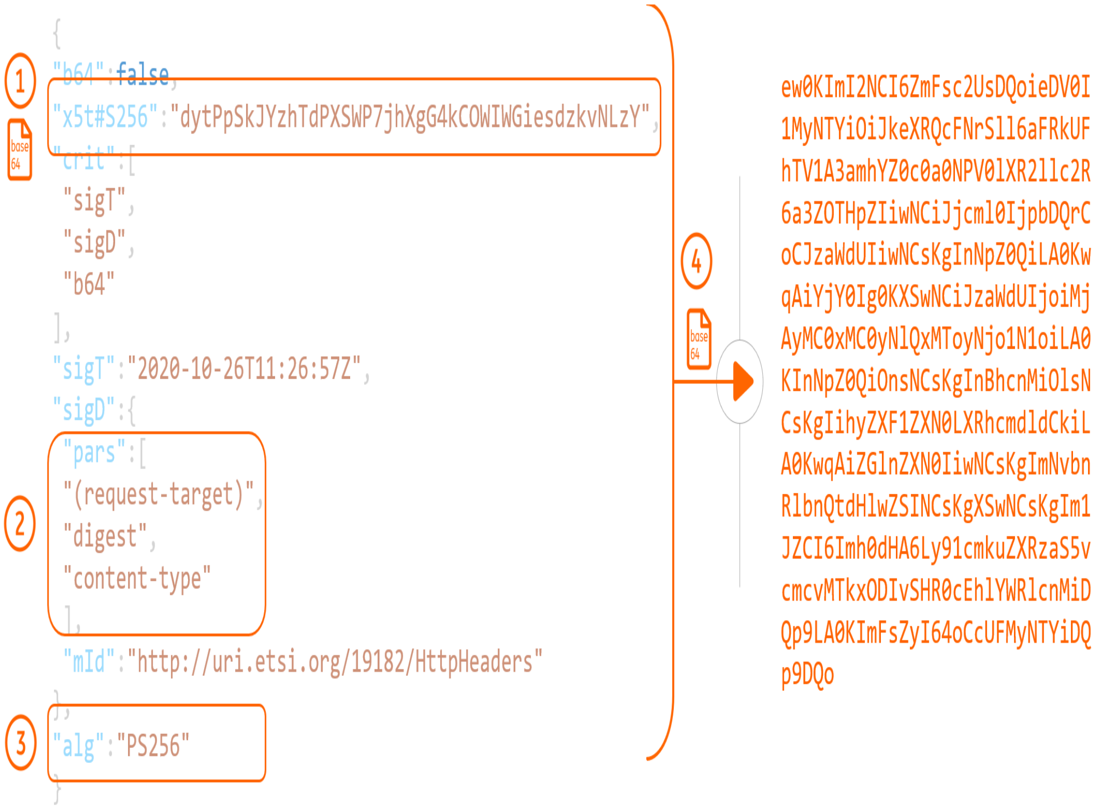
The following table explains the JWS protected header parameters:

JWS header parameter	Description
"b64": false	Means don't base64url encode header data to be signed
"x5t#S256": "...."	SHA-256 hash (fingerprint) of the public signing certificate (base64url encoded) that should be used to verify the signature
"crit":["sigT","sigD","b64"]	Non-standard JWS header parameters (ie. not defined in RFC 7515), which are critical
"sigT":"...."	Signing time (see RFC 3339). Format is "yyyy-MM-ddTHH:mm:ssZ", for example:

	"2022-04-26T11:26:04Z"
"sigD":{....}	List of HTTP Header fields to be signed (lower case) `"sigD":{ "pars":[ "(request-target)", "digest", "content-type" ], "mId":"http://uri.etsi.org/19182/HttpHeaders"}`
"alg": "...."	Signature algorithm. See API specifications on which algorithms are supported. The supported algorithms are SHA-256 with RSA, SHA-256 with ECDSA, SHA-384 with ECDSA, SHA-512 with ECDSA, SHA-512 with RSA and MGF1.

Follow the next steps to create the JWS protected header:

1. Base64URL encode the SHA-256 fingerprint of the signing certificate
2. Select HTTP headers to be signed (based on API specification) - in this case the mandatory are "(request-target)", "digest" and "content-type"
3. Select signing algorithm (based on API specification) - in this case PS256
4. Base64URL encode the JWS protected header



```
{
"b64":false,
"x5t#S256":"dytPpSkJYzhTdPXSWP7jhXgG4kCOWIWGiesdzkvNLzY",
"crit":[
 "sigT",
 "sigD",
 "b64"
],
"sigT":"2020-10-26T11:26:57Z",
"sigD":{
 "pars":[
 "(request-target)",
 "digest",
 "content-type"
 ],
 "mId":"http://uri.etsi.org/19182/HttpHeaders"
},
"alg":"PS256"
}
```

ew0KImI2NCI6ZmFsc2UsDQoieDV0I
1MyNTYiOiJkeXRQcFNrSll6aFRkUF
hTV1A3amhYZ0c0a0NPV0lXR2llc2R
6a3ZOTHpZIiwNCiJjcml0IjpbDQrC
oCJzaWdUIiwNCsKgInNpZ0QiLA0Kw
qAiYjY0Ig0KXSwNCiJzaWdUIjoiMj
AyMC0xMC0yNlQxMToyNjo1N1oiLA0
KInNpZ0QiOnsNCsKgInBhcnMiOlsN
CsKgIihyZXF1ZXN0LXRhcmdldCkiL
A0KwqAiZGlnZXN0IiwNCsKgImNvbn
RlbnQtdHlwZSINCsKgXSwNCsKgIm1
JZCI6Imh0dHA6Ly91cmkuZXRzaS5v
cmcvMTkxODIvSHR0cEhlYWRlcnMiD
Qp9LA0KImFsZyI6oCcUFMyNTYiDQ
p9DQo

Note that the JWS protected header does not contain any line breaks. The line breaks in the example above are just for readability.

The following script can be used for creating the JWS protected header:

```
# generate the sha-256 fingerprint for the signing
certificate, Base64urlencoded
base64UrlFingerprint=$(openssl x509 -noout -
fingerprint -sha256 -inform pem -in
${certPath}example_client_signing.cer | cut -d'='
-f2 | sed s/://g  | xxd -r -p | base64 | tr -d '='
| tr '/+' '_-' | tr -d '\n')

# generate the current signing time, encoded using
RFC 3339 Internet time format for UTC without
fractional seconds (e.g. "2019-11-19T17:28:15Z")
sigT=$(date -u +"%Y-%m-%dT%H:%M:%SZ")
```

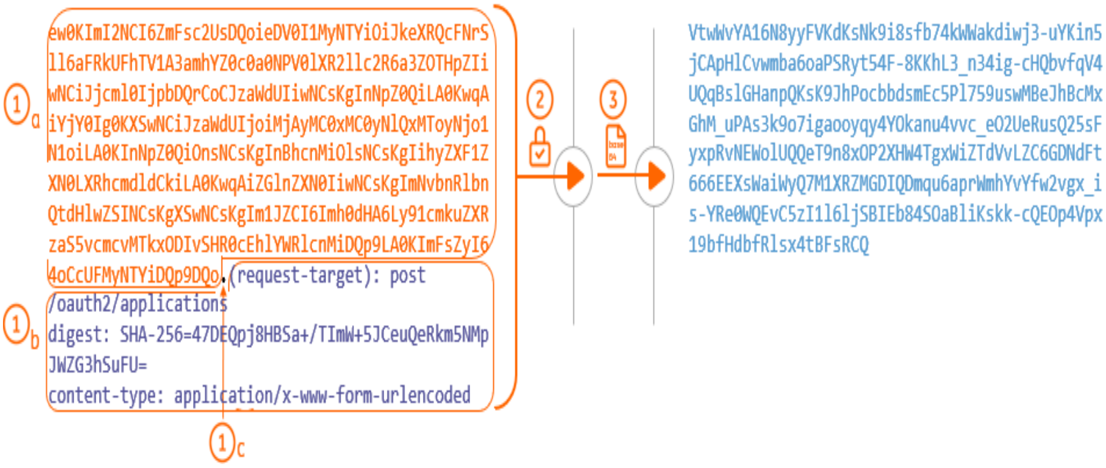
```
# create the JWS Protected Header with the sigD parameters containing the mandatory headers.
# Currently the json contains the minimal required headers to be signed: (request-target), digest and content-type.
# Please note that the list of mandatory headers can be different between APIs, check the API documentation.
jwsHeader='{"b64":false,"x5t#S256":"'"$base64UrlFingerprint"'","crit":[ "sigT", "sigD", "b64"],"sigT":"'"$sigT"'","sigD":{ "pars":[ "(request-target)", "digest", "content-type" ], "mId":"http://uri.etsi.org/19182/HttpHeaders"},"alg":"PS256"}'

# encode JWS Protected Header into Base64url
jwsHeaderBase64URL=$(echo -n "$jwsHeader" | openssl base64 -e -A | tr -d '=' | tr '/+' '_-' | tr -d '\n')
```

Create JWS signature value

Follow the next steps to create the JWS signature value:

1. Create JWS signature input
   a. Use the encoded JWS protected header, created in the previous step
   b. Create the HTTP header string, taking the mandatory HTTP headers in the exact order as they were listed in the sigD parameter of the JWS protected header. The HTTP headers are separated by line breaks.
   c. Concatenate the JWS protected header and HTTP header string, separated with a dot (.) character
2. Sign the JWS signature input with the private signing key using the chosen algorithm

3. Base64URL encode the JWS signature input to get the JWS signature value



```
ew0KImI2NCI6ZmFsc2UsDQoieDV0I1MyNTYiOiJkeXRQcFNrS
ll6aFRkUFhTV1A3amhYZ0c0a0NPV0lXR2llc2R6a3ZOTHpZIi
wNCiJjcml0IjpbDQrCoCJzaWdUIiwNCsKgInNpZ0QiLA0KwqA
iYjY0Ig0KXSwNCiJzaWdUIjoiMjAyMC0xMC0yNlQxMToyNjo1
N1oiLA0KInNpZ0QiOnsNCsKgInBhcnMiOlsNCsKgIihyZXF1Z
XN0LXRhcmdldCkiLA0KwqAiZGlnZXN0IiwNCsKgImNvbnRlbn
QtdHlwZSINCsKgXSwNCsKgIm1JZCI6Imh0dHA6Ly91cmkuZXR
zaS5vcmcvMTkxODIvSHR0cEhlYWRlcnMiDQp9LA0KImFsZyI6
4oCcUFMyNTYiDQp9DQo.(request-target): post
/oauth2/applications
digest: SHA-256=47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMp
JWZG3hSuFU=
content-type: application/x-www-form-urlencoded
```

```
VtwWvYA16N8yyFVKdKsNk9i8sfb74kWWakdiwj3-uYKin5
jCApHlCvwmba6oaPSRyt54F-8KKhL3_n34ig-cHQbvfqV4
UQqBslGHanpQKsK9JhPocbbdsmEc5Pl759uswMBeJhBcMx
GhM_uPAs3k9o7igaooyqy4YOkanu4vvc_eO2UeRusQ25sF
yxpRvNEWolUQQeT9n8xOP2XHW4TgxWiZTdVvLZC6GDNdFt
666EEXsWaiWyQ7M1XRZMGDIQDmqu6aprWmhYvYfw2vgx_i
s-YRe0WQEvC5zI1l6ljSBIEb84SOaBliKskk-cQEOp4Vpx
19bfHdbfRlsx4tBFsRCQ
```

Note that the JWS protected header does not contain any line breaks. The line breaks in the example above are just for readability.

The following script can be used for creating the JWS signature value:

```
httpMethod="post"
reqPath="/oauth2/applications"

# calculate digest of the request body parameters
digest="SHA-256="$(echo -n $payload | openssl dgst
-binary -sha256 | openssl base64)

# the HTTP header string must be declared exactly
as shown below, with each header on a separate
line
signingString="(request-target): $httpMethod
$reqPath
digest: $digest
content-type: application/json"

# combine Base64url encoded JWS Protected Header
```

```
with the signingString
payload="$jwsHeaderBase64URL.$signingString"

# sign and encode the payload with signing private
key
jwsSignatureValueBase64URL=$(printf %s "$payload"
| openssl dgst -sha256 -sign
${certPath}example_client_signing.key -sigopt
rsa_padding_mode:pss | openssl base64 -A | tr -d
'=' | tr '/+' '_-' | tr -d '\n')
```

Create JWS signature

Follow the next steps to create the JWS signature:

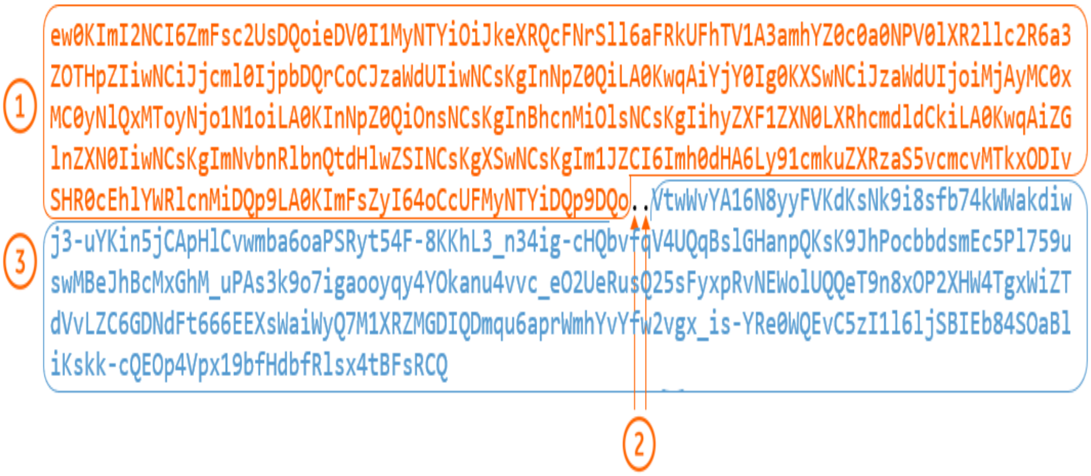
The format for the JWS signature is

(1) (2) (3)

[protected header]. [payload]. [signature value]

The payload value (2) is empty

Concatenate the JWS protected header (1), empty payload (2) and the JWS signature value (3), all separated with a dot (.) character (4)



ew0KImI2NCI6ZmFsc2UsDQoieDV0I1MyNTYiOiJkeXRQcFNrSll6aFRkUFhTV1A3amhYZ0c0a0NPV0lXR2llc2R6a3
ZOTHpZIiwNCiJjcml0IjpbDQrCoCJzaWdUIiwNCsKgInNpZ0QiLA0KwqAiYjY0Ig0KXSwNCiJzaWdUIjoiMjAyMC0x
MC0yNlQxMToyNjo1N1oiLA0KInNpZ0QiOnsNCsKgInBhcnMiOlsNCsKgIihyZXF1ZXN0LXRhcmdldCkiLA0KwqAiZG
lnZXN0IiwNCsKgImNvbnRlbnQtdHlwZSINCsKgXSwNCsKgIm1JZCI6Imh0dHA6Ly91cmkuZXRzaS5vcmcvMTkxODIv
SHR0cEhlYWRlcnMiDQp9LA0KImFsZyI6oCcUFMyNTYiDQp9DQo..VtwWvYA16N8yyFVKdKsNk9i8sfb74kWWakdiw
j3-uYKin5jCApHlCvwmba6oaPSRyt54F-8KKhL3_n34ig-cHQbvfqV4UQqBslGHanpQKsK9JhPocbbdsmEc5Pl759u
swMBeJhBcMxGhM_uPAs3k9o7igaooyqy4YOkanu4vvc_eO2UeRusQ25sFyxpRvNEWolUQQeT9n8xOP2XHW4TgxWiZT
dVvLZC6GDNdFt666EEXsWaiWyQ7M1XRZMGDIQDmqu6aprWmhYvYfw2vgx_is-YRe0WQEvC5zI1l6ljSBIEb84SOaBl
iKskk-cQEOp4Vpx19bfHdbfRlsx4tBFsRCQ

Note that the JWS protected header does not contain any line breaks. The line breaks in the example above are just for readability.

The following script can be used for creating the JWS signature:

```
# build JSON Web Signature
jwsSignature=$jwsHeaderBase64URL..$jwsSignatureVal
ueBase64URL
```

The signature should be dynamically calculated for every request since the values that are used as input to create the signature change every time, eg. the timestamp used in the sigT parameter of the JWS Protected Header.

Request

Once you have created the signature, you are ready to call the endpoint POST "/oauth2/applications" using all the header and body parameters.

The following script can be used as an example request:

```
curl -v -i -X POST "${httpHost}${reqPath}" \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-H "Digest: ${digest}" \
-H "TPP-Signature-Certificate:
${signCertificateHeader}" \
-H "x-jws-signature: ${jwsSignature}" \
--cert "${certPath}example_client_tls.cer" \
--key "${certPath}example_client_tls.key"
```

Request Body

```
{
  "contact": "one@example.com",
  "redirect_uris":
["https://client.example.org/callback"],
  "keys": {
    "tls": [
      {
        "kty": "RSA",
        "alg": "RS256",
        "use": "sig",
        "kid": "string",
        "n": "string",
        "e": "string",
        "x5t": "string",
        "x5c": ["string"]
      }
    ],
    "sign": [
      {
        "kty": "RSA",
        "alg": "RS256",
        "use": "sig",
        "kid": "string",
        "n": "string",
        "e": "string",
        "x5t": "string",
        "x5c": ["string"]
      }
    ]
  }
}
```

JWS Signature troubleshooting

If the JWS Signature verification was unsuccessful, here are potential fixes:

1. a field (for example 'date') is included in jws protected header 'pars' parameter, but not included in the signing input (HTTP headers to be signed).
2. a field (for example 'date') is included in the signing input (HTTP headers to be signed), but not included in jws protected header 'pars' parameter.
3. the http method or the path used in the signing input (HTTP headers to be signed) are not corresponding to the ones in the actual HTTP request.
4. when creating the signing input (collecting the HTTP headers to be signed), there might be an extra LF (line feed: 0x0A in UTF-8) after the last header.
5. one of the headers have different values in the http request than the ones used when creating the signature.
6. the signature was created with the wrong private key, one that does not correspond to the certificate hash fingerprint from the JWS Protected Header.
7. the signing input (HTTP headers to be signed) are not defined in the same order as defined in the SigD pars.
8. the 'alg' field is not corresponding to what was used for signing (ex: alg:PS256 but signed with PS384).
9. If you are on Windows, you can try a different cURL version
10. Make sure that sigT is +/- 3 min
11. If all the above were checked, you can try with a different Open SSL library. In our testing Open SSL version 3.2.1 provided the expected result.

Response

```
{
  "contact": "one@example.com",
  "application": {
    "client_id": "c188888a-6a7d-40b0-bdcb-
c612bc1f8ba1",
    "redirect_uris":
["https://client.example.org/callback"],
    "keys": {
      "tls": [
        {
          "kty": "RSA",
          "alg": "RS256",
          "use": "sig",
          "kid": "string",
          "n": "string",
          "e": "string",
          "x5t": "string",
          "x5c": ["string"]
        }
      ],
      "sign": [
        {
          "kty": "RSA",
          "alg": "RS256",
          "use": "sig",
          "kid": "string",
          "n": "string",
          "e": "string",
          "x5t": "string",
          "x5c": ["string"]
        }
      ]
    }
  }
}
```

2. Retrieve your application's registered metadata

2.1. Pre-requisite - Request an mTLS only application access token

When using the **/oauth2/applications/** endpoint, the way you request an access token from ING is different than for the other interactions. In broad lines, you will not have to sign this request, but rely on mutual TLS. We have described below the details:

An application access token is requested by calling ING's OAuth 2.0 API token endpoint: /oauth2/token. Requesting an application access token using mTLS requires the following parameters to be supplied.

grant_type

client_id

scope (optional)

Note that the application access token expires after 900 seconds. It is advisable to re-use the application access token as long as it has not expired, instead of requesting a new token for each API call.

Once the application access token expires, you should request a new application access token in order to re-authenticate your application.

Request

grant_type BODY PARAMETER

A request for an application access token requires the "grant_type" request parameter set to value "client_credentials".

```
grant_type=client_credentials
```

client_id BODY PARAMETER

The "client_id" for your registered application at lNG (which you have received in the API responses). Check the table under the "Sandbox and Production environments" section of this Get Started guide to check which value to use for the "client_id" parameter.

An example value for the "client_id" parameter for a PSD2 API in the Sandbox environment:

```
client_id=5ca1ab1e-c0ca-c01a-cafe-154deadbea75
```

scope BODY PARAMETER

A request for an application access token can contain multiple scope-tokens (separated by spaces) in the "scope" parameter of the payload. The "scope" parameter value must be encoded using x-www-form-urlencoded-compliant URL encoding, and the space character must be encoded with "+" instead of "%20".

The scope parameter value with single scope-token "greetings:view":

```
scope=greetings%3Aview
```

The scope parameter value with multiple scope-tokens "greetings:view" and "payment-requests:view":

```
scope=greetings%3Aview+payment-requests%3Aview
```

When no scope is requested, the application access token will contain all the registered scope-tokens by default. Refer to the "Reference" section of a particular API you want to consume to see the available scopes.

Now you are ready to request an application access token using all the header and body parameters.

The following script can be used as an example request:

```
#!/bin/bash


##################################################
#############################
#                   REQUEST APPLICATION ACCESS
TOKEN  (MTLS)                #
##################################################
```

```
#############################

certPath="./certificates/" # path of the
downloaded certificates and keys
# httpHost="https://api.ing.com" # production host

reqPath="/oauth2/token"

# clientId path param required for mtls flow
clientId="5ca1ab1e-c0ca-c01a-cafe-154deadbea75" #
client_id as provided in the documentation

# You can also provide scope parameter in the body
E.g.
"grant_type=client_credentials&scope=greetings%3Av
iew"
# scope is an optional parameter. The downloaded
certificate contains all available scopes. If you
don't provide a scope, the accessToken is returned
for all scopes available in certificate
payload="grant_type=client_credentials&client_id=$
{clientId}"

# Curl request method must be in uppercase e.g
"POST", "GET"
response=$(curl -k -X POST "${httpHost}${reqPath}"
\
-H 'Accept: application/json' \
-H 'Content-Type: application/x-www-form-
urlencoded' \
-d "${payload}" \
--cert "${certPath}example_client_tls.cer" \
--key "${certPath}example_client_tls.key")
```

Response

Header

x-ing-response-id - Unique tracing id which should be provided when reporting any issue.

Body

access_token - It contains the application access token.

expires_in - It contains the expiration time in seconds for the application access token. Once the application access token expires, you should request a new application access token in order to re-authenticate your application.

scope - It contains the list of all the scopes associated with the "access_token".

Example response for a successful request:

```
HTTP/2 200

date: Fri, 15 Nov 2019 08:44:05 GMT
content-type: application/json
x-frame-options: deny
x-content-type-options: nosniff
strict-transport-security: max-age=31622400;
includeSubDomains
x-xss-protection: 1; mode=block
cache-control: no-cache, no-store, max-age=0,
must-revalidate
pragma: no-cache
expires: 0
x-ing-response-id:
bb9c6fb19dc058db9794f6f8b9e28b2d
{"access_token":"iyK4bY.....","expires_in":905,"sc
ope":"payment-requests:view payment-
requests:create payment-requests:close
greetings:view virtual-ledger-accounts:fund-
reservation:create virtual-ledger-accounts:fund-
reservation:delete virtual-ledger-
accounts:balance:view","token_type":"Bearer","keys
":
[{"kty":"RSA","n":"34VPkdV.....","e":"AQAB","use":
"sig","alg":"RS256","x5t":"3c396700fc8cd70........
"}],"client_id":"5ca1ab1e-c0ca-c01a-cafe-
154deadbea75"}
```

2.2. Retrieve your application's registered metadata via GET /oauth2/applications/me or /oauth2/applications/{id}

Registered clients have a set of metadata values associated with their application identifier (client_id) at an authorization server, such as the list of valid redirection URIs, certificates and contact details.

Parameters

Header

Authorization string ***required*** HEADER PARAMETER

Application access token (bearer) received in the Client Credentials. grant flow

Path

id string ***required*** PATH PARAMETER

Application identifier - client_id received during onboarding.

Example request

```
#!/bin/bash


##################################################
#############################
#                    REQUEST APPLICATIONS METADATA
#
##################################################
```

```
#############################

certPath="./certificates/" # path of the
downloaded certificates and keys
httpHost="https://api.ing.com" # production host

reqPath="/oauth2/applications/me"

accessToken="" # mtls access_token received from
calling POST /oauth2/token endpoint

# Curl request method must be in uppercase e.g
"POST", "GET"
response=$(curl -k -X GET "${httpHost}${reqPath}"
\
-H "Authorization: Bearer $accessToken" \
--cert "${certPath}example_client_tls.cer" \
--key "${certPath}example_client_tls.key")
```

Example response

```
{
  "contact": "one@example.com",
  "applications": [
    {
      "client_id": "c188888a-6a7d-40b0-bdcb-
c612bc1f8ba1",
      "redirect_uris":
["https://client.example.org/callback"],
      "keys": {
        "tls": [
          {
            "kty": "RSA",
            "alg": "RS256",
            "use": "sig",
            "kid": "string",
            "n": "string",
            "e": "string",
            "x5t": "string",
            "x5c": ["string"]
          }
        ],
        "sign": [
          {
            "kty": "RSA",
            "alg": "RS256",
            "use": "sig",
            "kid": "string",
            "n": "string",
            "e": "string",
            "x5t": "string",
            "x5c": ["string"]
          }
        ]
      }
    }
  ]
}
```

2b. (optional) Update the registered application's metadata, especially the Redirect URL(s) and email address via PUT /oauth2/applications/{id}

For this you will need to use the JWS signature protocol and the x-jws-signature header. Note: this message signature protocol is used only for the PUT and POST on **/oauth2/applications/** endpoint

Parameters

authorization HEADER PARAMETER

When calling an API, the bearer authentication scheme is used for the authorization header. This means you need to supply the access token as the value for this header parameter.

For example:

```
# Please replace the value of this variable with a
valid application access token
authorization="Bearer
eyJhbGciOiJkaXIiLCJlbmMiOiJBMjU2Q0JDLUhTNTEyIiwia.
..
...bA06uS.vxbf2aRnZfa9fLwA7TOwpqzJIcPK8T10X95bPyEn
s3I"
```

The following script can be used for calculating the date:

```
reqDate=$(LC_TIME=en_US.UTF-8 date -u "+%a, %d %b
%Y %H:%M:%S GMT")
```

digest HEADER PARAMETER

The "Digest" header is used by our server to verify the integrity of the message body that you send to us when calling any of our APIs.

Creating the digest can be done by calculating the SHA-256 hash of the payload of the message and Base64 encoding the hash. In this example the payload is empty, so the digest will be:

```
digest="SHA-
256=47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU="
```

content-type HEADER PARAMETER

The "content-type" header is used to indicate what type of media or resource is being used for the payload in the request. The content type of PUT requests for updating the registered application's metadata is application/json, since they require a JSON body containing "contact", "redirect_uris" and "keys" fields.

TPP-Signature-Certificate HEADER PARAMETER

The PSD2 compliant Qualified Electronic Seal Certificate (QSealC) which you will use for the signature in the client credentials grant type. The certificate (PEM format) needs to be the Base64 encoded (and then surrounded by '-----BEGIN CERTIFICATE-----' and '-----END CERTIFICATE-----' tags).

x-jws-signature HEADER PARAMETER

The "x-jws-signature" header is used to store the JSON Web Signature (JWS) for this request.

Want to learn more about the JSON Web Signature specification for Open Banking?
Your signing private key is required to create a signature of the headers and their values that you will send in your request. Check the table under the "Sandbox and Production environments" section of this Get Started guide to check which private key to use for signing.

In the following three steps we will explain how to create the JWS signature that will be used as the value for the "x-jws-signature" header:

Create JWS protected header

The JWS protected header is a JSON object which stores information about the chosen algorithm and the fingerprint of the public certificate that should be used to verify the signature, as well as which HTTP headers are included in the signature and the timestamp when the signature was created.

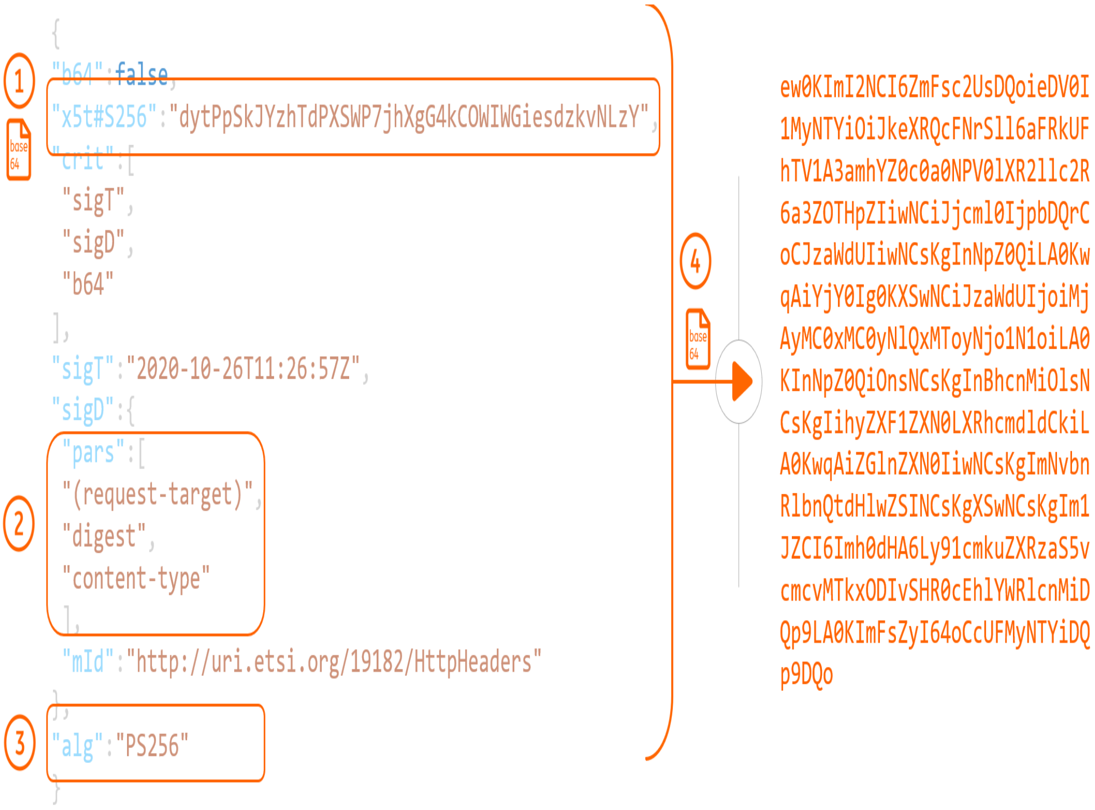
The following table explains the JWS protected header parameters:

JWS header parameter	Description
"b64": false	Means don't base64url encode header data to be signed
"x5t#S256": "...."	SHA-256 hash (fingerprint) of the public signing certificate (base64url encoded) that should be used to verify the signature

"crit":["sigT","sigD","b64"]	Non-standard JWS header parameters (ie. not defined in RFC 7515), which are critical
"sigT":"...."	Signing time (see RFC 3339). Format is "yyyy-MM-ddTHH:mm:ssZ", for example: "2022-04-26T11:26:04Z"
"sigD":{....}	List of HTTP Header fields to be signed (lower case) `"sigD":{ "pars":[ "(request-target)", "digest", "content-type" ], "mId":"http://uri.etsi.org/19182/HttpHeaders"}`
"alg": "...."	Signature algorithm. See API specifications on which algorithms are supported. The supported algorithms are PS256, PS384, PS512, ES256, ES384, ES512.

Follow the next steps to create the JWS protected header:

1. Base64URL encode the SHA-256 fingerprint of the signing certificate
2. Decide on HTTP headers to be signed (based on API specification)
3. Choose signing algorithm (eg. PS256)
4. Base64URL encode the JWS protected header



```
{
"b64":false,
"x5t#S256":"dytPpSkJYzhTdPXSWP7jhXgG4kCOWIWGiesdzkvNLzY",
"crit":[
 "sigT",
 "sigD",
 "b64"
],
"sigT":"2020-10-26T11:26:57Z",
"sigD":{
 "pars":[
 "(request-target)",
 "digest",
 "content-type"
 ],
 "mId":"http://uri.etsi.org/19182/HttpHeaders"
},
"alg":"PS256"
}
```

ew0KImI2NCI6ZmFsc2UsDQoieDV0I
1MyNTYiOiJkeXRQcFNrSll6aFRkUF
hTV1A3amhYZ0c0a0NPV0lXR2llc2R
6a3ZOTHpZIiwNCiJjcml0IjpbDQrC
oCJzaWdUIiwNCsKgInNpZ0QiLA0Kw
qAiYjY0Ig0KXSwNCiJzaWdUIjoiMj
AyMC0xMC0yNlQxMToyNjo1N1oiLA0
KInNpZ0QiOnsNCsKgInBhcnMiOlsN
CsKgIihyZXF1ZXN0LXRhcmdldCkiL
A0KwqAiZGlnZXN0IiwNCsKgImNvbn
RlbnQtdHlwZSINCsKgXSwNCsKgIm1
JZCI6Imh0dHA6Ly91cmkuZXRzaS5v
cmcvMTkxODIvSHR0cEhlYWRlcnMiD
Qp9LA0KImFsZyI6oCcUFMyNTYiDQ
p9DQo

Note that the JWS protected header does not contain any line breaks. The line breaks in the example above are just for readability.

The following script can be used for creating the JWS protected header:

```
# generate the sha-256 fingerprint for the signing certificate, Base64urlencoded
base64UrlFingerprint=$(openssl x509 -noout -fingerprint -sha256 -inform pem -in ${certPath}example_client_signing.cer | cut -d'=' -f2 | sed s/://g  | xxd -r -p | base64 | tr -d '=' | tr '/+' '_-' | tr -d '\n')

# generate the current signing time, encoded using RFC 3339 Internet time format for UTC without fractional seconds (e.g. "2019-11-19T17:28:15Z")
sigT=$(date -u +"%Y-%m-%dT%H:%M:%SZ")
```

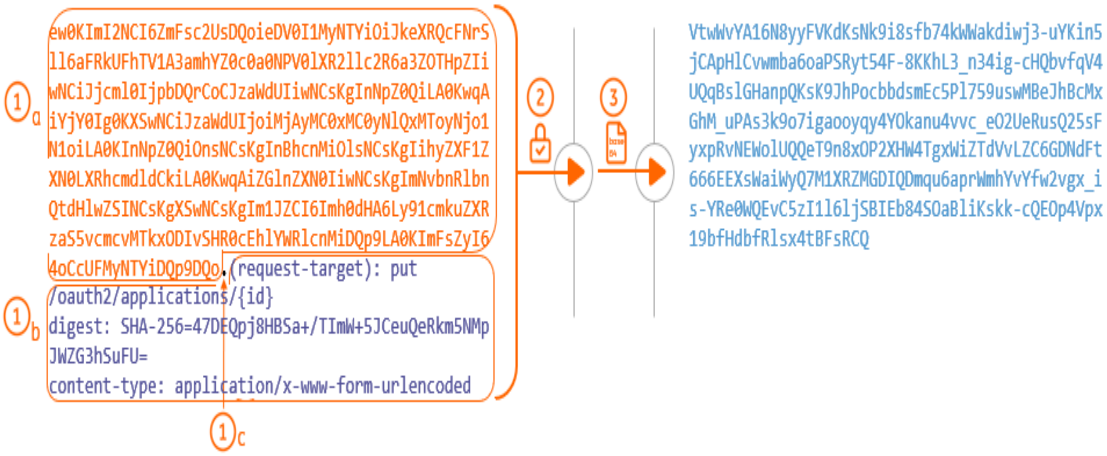
```
# create the JWS Protected Header with the sigD parameters containing the mandatory headers.
# Currently the json contains the minimal required headers to be signed: (request-target), digest and content-type.
# Please note that the list of mandatory headers can be different between APIs, check the API documentation.
jwsHeader='{"b64":false,"x5t#S256":"'"$base64UrlFingerprint"'","crit":[ "sigT", "sigD", "b64"],"sigT":"'"$sigT"'","sigD":{ "pars":[ "(request-target)", "digest", "content-type" ], "mId":"http://uri.etsi.org/19182/HttpHeaders"},"alg":"PS256"}'

# encode JWS Protected Header into Base64url
jwsHeaderBase64URL=$(echo -n "$jwsHeader" | openssl base64 -e -A | tr -d '=' | tr '/+' '_-' | tr -d '\n')
```

Create JWS signature value

Follow the next steps to create the JWS signature value:

1. Create JWS signature input
   a. Use the encoded JWS protected header, created in the previous step
   b. Create the HTTP header string, taking the mandatory HTTP headers in the exact order as they were listed in the sigD parameter of the JWS protected header. The HTTP headers are separated by line breaks.
   c. Concatenate the JWS protected header and HTTP header string, separated with a dot (.) character
2. Sign the JWS signature input with the private signing key using the chosen algorithm

3. Base64URL encode the JWS signature input to get the JWS signature value


1a
ew0KImI2NCI6ZmFsc2UsDQoieDV0I1MyNTYiOiJkeXRQcFNrS
ll6aFRkUFhTV1A3amhYZ0c0a0NPV0lXR2llc2R6a3ZOTHpZIi
wNCiJjcml0IjpbDQrCoCJzaWdUIiwNCsKgInNpZ0QiLA0KwqA
iYjY0Ig0KXSwNCiJzaWdUIjoiMjAyMC0xMC0yNlQxMToyNjo1
N1oiLA0KInNpZ0QiOnsNCsKgInBhcnMiOlsNCsKgIihyZXF1Z
XN0LXRhcmdldCkiLA0KwqAiZGlnZXN0IiwNCsKgImNvbnRlbn
QtdHlwZSINCsKgXSwNCsKgIm1JZCI6Imh0dHA6Ly91cmkuZXR
zaS5vcmcvMTkxODIvSHR0cEhlYWRlcnMiDQp9LA0KImFsZyI6
4oCcUFMyNTYiDQp9DQo.
1b
(request-target): put
/oauth2/applications/{id}
digest: SHA-256=47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMp
JWZG3hSuFU=
content-type: application/x-www-form-urlencoded
1c
2
3
VtwWvYA16N8yyFVKdKsNk9i8sfb74kWWakdiwj3-uYKin5
jCApHlCvwmba6oaPSRyt54F-8KKhL3_n34ig-cHQbvfqV4
UQqBslGHanpQKsK9JhPocbbdsmEc5Pl759uswMBeJhBcMx
GhM_uPAs3k9o7igaooyqy4YOkanu4vvc_eO2UeRusQ25sF
yxpRvNEWolUQQeT9n8xOP2XHW4TgxWiZTdVvLZC6GDNdFt
666EEXsWaiWyQ7M1XRZMGDIQDmqu6aprWmhYvYfw2vgx_i
s-YRe0WQEvC5zI1l6ljSBIEb84SOaBliKskk-cQEOp4Vpx
19bfHdbfRlsx4tBFsRCQ

Note that the JWS protected header does not contain any line breaks. The line breaks in the example above are just for readability.

The following script can be used for creating the JWS signature value:

```
# calculate digest of the request body parameters
digest="SHA-256="$(echo -n $payload | openssl dgst
-binary -sha256 | openssl base64)

# the HTTP header string must be declared exactly
as shown below, with each header on a separate
line
signingString="(request-target): $httpMethod
$reqPath
digest: $digest
content-type: application/json"

# combine Base64url encoded JWS Protected Header
with the signingString
payload="$jwsHeaderBase64URL.$signingString"
```

```
# sign and encode the payload with signing private
key
jwsSignatureValueBase64URL=$(printf %s "$payload"
| openssl dgst -sha256 -sign
${certPath}example_client_signing.key -sigopt
rsa_padding_mode:pss | openssl base64 -A | tr -d
'=' | tr '/+' '_-' | tr -d '\n')
```

Create JWS signature

Follow the next steps to create the JWS signature:

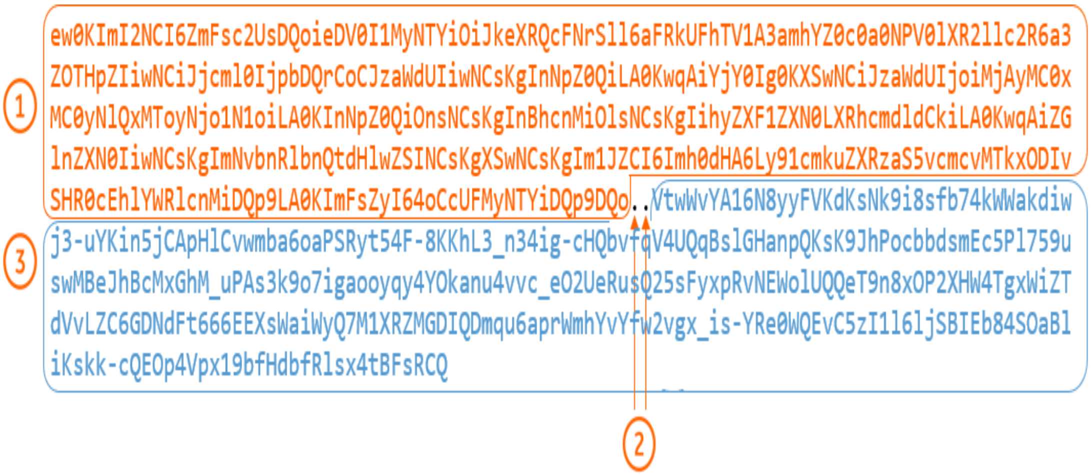
The format for the JWS signature is

(1) (2) (3)

[protected header]. [payload]. [signature value]

The payload value (2) is empty

Concatenate the JWS protected header (1), empty payload (2) and the JWS signature value (3), all separated with a dot (.) character (4)



```
ew0KImI2NCI6ZmFsc2UsDQoieDV0I1MyNTYiOiJkeXRQcFNrSll6aFRkUFhTV1A3amhYZ0c0a0NPV0lXR2llc2R6a3
ZOTHpZIiwNCiJjcml0IjpbDQrCoCJzaWdUIiwNCsKgInNpZ0QiLA0KwqAiYjY0Ig0KXSwNCiJzaWdUIjoiMjAyMC0x
MC0yNlQxMToyNjo1N1oiLA0KInNpZ0QiOnsNCsKgInBhcnMiOlsNCsKgIihyZXF1ZXN0LXRhcmdldCkiLA0KwqAiZG
lnZXN0IiwNCsKgImNvbnRlbnQtdHlwZSINCsKgXSwNCsKgIm1JZCI6Imh0dHA6Ly91cmkuZXRzaS5vcmcvMTkxODIv
SHR0cEhlYWRlcnMiDQp9LA0KImFsZyI6oCcUFMyNTYiDQp9DQo..VtwWvYA16N8yyFVKdKsNk9i8sfb74kWWakdiw
j3-uYKin5jCApHlCvwmba6oaPSRyt54F-8KKhL3_n34ig-cHQbvfqV4UQqBslGHanpQKsK9JhPocbbdsmEc5Pl759u
swMBeJhBcMxGhM_uPAs3k9o7igaooyqy4YOkanu4vvc_eO2UeRusQ25sFyxpRvNEWolUQQeT9n8xOP2XHW4TgxWiZT
dVvLZC6GDNdFt666EEXsWaiWyQ7M1XRZMGDIQDmqu6aprWmhYvYfw2vgx_is-YRe0WQEvC5zI1l6ljSBIEb84SOaBl
iKskk-cQEOp4Vpx19bfHdbfRlsx4tBFsRCQ
```

Note that the JWS protected header does not contain any line breaks. The line breaks in the example above are just for readability.

The following script can be used for creating the JWS signature:

```
# build JSON Web Signature
jwsSignature=$jwsHeaderBase64URL..$jwsSignatureValueBase64URL
```

The signature should be dynamically calculated for every request since the values that are used as input to create the signature change every time, eg. the timestamp used in the sigT parameter of the JWS Protected Header.

Request

Once you have created the signature, you are ready to call the "PUT /oauth2/applications/{id}" endpoint using all the header and body parameters.

The following script can be used as an example request:

```
curl -v -i -X PUT "${httpHost}${reqPath}" \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-H "Digest: ${digest}" \
-H "Authorization: Bearer ${accessToken}" \
-H "Date: ${reqDate}" \
-H "x-jws-signature: ${jwsSignature}" \
-H "TPP-Signature-Certificate: ${signCertificateHeader}" \
--cert "${certPath}example_client_tls.cer" \
--key "${certPath}example_client_tls.key"
```

id PATH PARAMETER

Application identifier - client_id received during onboarding.

Request Body

```
{
  "contact": "one@example.com",
  "redirect_uris":
["https://client.example.org/callback"],
  "keys": {
    "tls": [
      {
        "kty": "RSA",
        "alg": "RS256",
        "use": "sig",
        "kid": "string",
        "n": "string",
        "e": "string",
        "x5t": "string",
        "x5c": ["string"]
      }
    ],
    "sign": [
      {
        "kty": "RSA",
        "alg": "RS256",
        "use": "sig",
        "kid": "string",
        "n": "string",
        "e": "string",
        "x5t": "string",
        "x5c": ["string"]
      }
    ]
  }
}
```

Response

```
{
  "contact": "one@example.com",
  "application": {
    "client_id": "c188888a-6a7d-40b0-bdcb-
c612bc1f8ba1",
    "redirect_uris":
["https://client.example.org/callback"],
    "keys": {
      "tls": [
        {
          "kty": "RSA",
          "alg": "RS256",
          "use": "sig",
          "kid": "string",
          "n": "string",
          "e": "string",
          "x5t": "string",
          "x5c": ["string"]
        }
      ],
      "sign": [
        {
          "kty": "RSA",
          "alg": "RS256",
          "use": "sig",
          "kid": "string",
          "n": "string",
          "e": "string",
          "x5t": "string",
          "x5c": ["string"]
        }
      ]
    }
  }
}
```

⚠ Please keep in mind that the PUT call will clear all existing data and replace it with the new configuration. For example, if you have two sets of certificates registered and send only one, only the sent one will be saved.

3. Granting

Pending you have registered your Redirect URL(s) (as described above), this step is only required for APIs that require customer consent (eg. for PSD2: AIS and CAF). ING has implemented OAuth 2.0 authorization code flow to help you get access to customer data upon their explicit consent. To learn more about the Authorization code flow refer to the Authorization code flow section in our OAuth 2.0 documentation.

In this step a request for customer consent is initiated and an authorization code is generated. Using the application access token and the authorization code, your application obtains a customer access token. Finally, your application uses the customer access token to consume the API that requires customer consent.

a) Redirect the customer to ING's authorization application and let the customer authorize your application

Important: Redirect the customer to their default browser in which they can confirm that they are communicating with the trusted ING website. You are not allowed to embed ING's authorization page in your application in any way! ING can block your application from using ING APIs when identifying such an implementation.

Redirect the customer to ING's authorization application URL, by appending your application specific parameters:

```
https://myaccount.ing.com/authorize/v2/[COUNTRY_CODE]?

client_id=[YOUR_CLIENT_ID]
&scope=[SCOPES_SPACE_SEPARATED_AND_URLENCODED]
```

```
&state=[SOME_ARBITRARY_BUT_UNIQUE_STRING]
&redirect_uri=[YOUR_URL_ENCODED_REDIRECT_URI]
&response_type=code
```

COUNTRY_CODE QUERY PARAMETER

Is an optional parameter and allows you to redirect your customer to the specified ING country login page. The format is a two-letter value (ISO 3166-1). If you don't add the country_code, the ING customer will be redirected to an ING country selection page.

Country	Code
Belgium	BE
The Netherlands	NL
Spain	ES
Romania	RO
Luxembourg	LU
Italy	IT
Germany	DE
Wholesale Banking	WB

client_id QUERY PARAMETER

Required parameter, the unique client_id obtained when you registered your application with ING.

scope QUERY PARAMETER

Required parameter, a list of scopes separated by spaces for which an authorization request is being sent. This could be a subset of the scopes you application is allowed to request. The value must be URL encoded, where spaces may be encoded to either "+" or "%20".

state QUERY PARAMETER

Required parameter, can be used by your application to maintain state between HTTP redirects and achieve per-request customization of your redirect URL, please use the "state" parameter as described in the OAuth 2.0 RFC.

The state parameter must not be blank. Allowed characters for the "state" parameter are:

Alphanumeric characters in range A-Z, a-z, 0-9

Special characters =%#,;+/-_

Any whitespace character \r \n \t \f \v

redirect_uri QUERY PARAMETER

Optional parameter, a specific redirect URI where the customer will return to once the authorization process is ended. The value must be URL encoded. This parameter is optional if your application has registered only a single redirect URL with ING. This is a mandatory parameter in case your application has registered multiple redirect URLs and should be one of them.

response_type QUERY PARAMETER

Required parameter, for PSD2 flow, the value must be "code".

Example redirect url

An example URL to redirect the customer to start the authorization flow should look like this:

```
https://myaccount.ing.com/authorize/v2/NL?
client_id=6414808c-d8da-450e-b10d-
8a1c1dd37561&scope=payment-
accounts%3Atransactions%3Aview+payment-
accounts%3Abalances%3Aview&state=123456&response_t
ype=code&redirect_uri=http%3A%2F%2Fapi.example.com
```

b) Redirect the customer back to your application and acquire an authorization code

Once successful, ING's authorization application redirects the customer to the URL provided in the redirect_uri query parameter along with the authorization code. For example:

```
https://www.example.com/?state=[YOUR_STATE_VALUE]&code=486b7cfecc6220a0a97b001910fb4376d2605bc9c9ea2af9dd67ed5dfa63af70b0d50171512b4fe79c5c71b1998a3997
```

The "code" parameter in the redirected URL contains the dynamically generated authorization code, this authorization_code is used in the next step in order to request a customer access token.

c) Request an application access token

The interaction with the granting flow and subsequent PSD2 API interactions (AIS, PIS, CAF) use the HTTPS signature protocol for both the acess token requests and the resource API requests.

Here you can download the sample script to request an application access token for the PSD2 APIs in production.

An application access token is requested by calling ING's OAuth 2.0 API token endpoint: **/oauth2/token**. To call this API endpoint, you must create a signature and sign the HTTP request. We have explained both the steps below.

Want to learn more about HTTP message Signing?

Calculate digest

```
payload="grant_type=client_credentials"
payloadDigest=` echo -n "$payload" | openssl dgst -binary -sha256 | openssl base64`
digest=SHA-256=$payloadDigest
```

"Digest" header is used by our server to verify the integrity of the message body that you send to us when calling any of our APIs.

A request for an application access token can contain multiple scope-tokens (separated by spaces) in the "scope" parameter of the payload. The "scope" parameter value must be encoded using x-www-form-urlencoded-compliant URL encoding and the space character must be encoded with "+" instead of "%20" as explained below.

The scope parameter value with single scope-token "payment-accounts:balances:view".

```
grant_type=client_credentials&scope=payment-
accounts%3Abalances%3Aview
```

The scope parameter value with multiple scope-tokens "payment-accounts:balances:view" and "payment-accounts:transactions:view".

```
grant_type=client_credentials&scope=payment-
accounts%3Abalances%3Aview%20payment-
accounts%3Atransactions%3Aview
```

When no scope is explicitly requested the application access token will contain all the scope-tokens available in the certificate that you are using. Refer to the "Reference" section of the PSD2 API you want to consume to see the available scopes.

Calculate date

```
reqDate=$(LC_TIME=en_US.UTF-8 date -u "+%a, %d %b
%Y %H:%M:%S GMT")
```

Note that there should not be a gap of more than 3 minutes between the time when the PSD2 API endpoint request is made and the time value in the "Date" header field of the request.

Calculate signature for signing your request

```
httpMethod="post"
reqPath="/oauth2/token"
signingString="(request-target): $httpMethod
$reqPath
date: $reqDate
digest: $digest"
signature=`printf %s "$signingString" | openssl
dgst -sha256 -sign "/certs/client_signing.key" |
openssl base64 -A`
```

Signature should be dynamically calculated for every PSD2 API endpoint request since the values that are used as input to create the signature change every time.

Once you have created the signature, you are ready to request an application access token using the below header and body parameters:

Request parameters (refer to the sample script available to download at the beginning of this section):

curl "${httpHost}${reqPath}" - Full path of the API endpoint.

Digest: $(digest) - Calculated value of digest.

Date: $(reqDate) - Calculated value of date. This is the standard HTTP header (see RFC7231). Note that we enforce a strict time frame within which the request must be issued. If the Date header's time value diverges by more than 3 minutes from the time the API request is made, the request will be considered invalid.

TPP-Signature-Certificate - Copy and paste the certificate for HTTP request signing as a single line string including the BEGIN CERTIFICATE and the END CERTIFICATE lines.

keyId - It has a format "SN=XXX, where “XXX" is the serial number of the certificate in hexadecimal coding. You can extract the serial number and full distinguished name of CA using below openssl command:

```
openssl x509 -in -serial -noout
```

Example output:

```
serial=49BA5C71
```

signature - Calculated value of the signature in order to sign your request.

Content-Type - Value set to "application/x-www-form-urlencoded". It forces the body parameters to use the x-www-form-urlencoded compliant URL encoding. Note that space character should be encoded as "+" and not as "%20".

Example request:

```
curl -i -X POST "${httpHost}${reqPath}" \
-H 'Accept: application/json' \
-H 'Content-Type: application/x-www-form-urlencoded' \
-H "Digest: ${digest}" \
-H "Date: ${reqDate}" \
-H 'TPP-Signature-Certificate: -----BEGIN CERTIFICATE-----Y6Qqg2mjk+7Cq7zHty4E7q.......K-----END CERTIFICATE-----' \
-H "authorization: Signature keyId=\"SN=49BA5C71\",algorithm=\"rsa-sha256\",headers=\"(request-target) date digest\",signature=\"$signature\"" \
-d "${payload}" \
--cert "/certs/" \
--key "/certs/"
```

Response parameters

access_token - It contains the application access token.

client_id - It contains the client_id generated when registering the application in the previous step

expires_in - It contains the expiration time in seconds for the application access token. Once the application access token expires, you should execute this step (Step 1) again in order to re-authenticate your application and request a new application access token.

scope - It contains the list of all the scopes associated with the "access_token".

x-ing-response-id - Unique tracing id which should be provided when reporting any issue.

Note that the application access token expires after 900 seconds after which you need to execute this step again in order to re-authenticate your application and request a new application access token. It is advisable to re-use the application access token instead of requesting a new token for each API call.

Example response for a successful request:

```
HTTP/2 200

date: Fri, 15 Nov 2019 11:20:50 GMT
content-type: application/json
x-frame-options: deny
x-content-type-options: nosniff
strict-transport-security: max-age=31622400;
includeSubDomains
x-xss-protection: 1; mode=block
x-ing-respo-id: 495a8daf-486f-4193-80c2-
1a72e755a8db
cache-control: no-cache, no-store, max-age=0,
must-revalidate
pragma: no-cache
expires: 0
{"access_token":"eyJhb....","expires_in":905,"scop
e":"payment-accounts:orders:create
granting","token_type":"Bearer","keys":
[{"kty":"RSA","n":"3l3rdzkdV.....","e":"AQAB","use
":"sig","alg":"RS256","x5t":"3c396700fc8cd709cf9cb
5452a22bcde76985851"}],"client_id":"ff5d0aa0-95c3-
4a9f-8b77-........."}
```

You can configure your TLS implementation to verify the trust chain of ING's TLS certificate used for setting up the tunnel between your application and ING. Our PKI certificate is signed by our Root CA which in turn is signed by a commercial CA.

d) Use the authorization code to request the customer access token

Here you can download the sample script to request the customer access token from the authorization code.

The authorization code is valid for only 10 minutes after issuing. Your application should exchange the authorization code for a pair of customer access token and a refresh token within this time frame using the endpoint /oauth2/token and the below request parameters.

You must use the authorization code while calculating the digest value as shown below:

```
payload="grant_type=authorization_code&code=[ING_AUTHORIZATION_CODE]&redirect_uri=[YOUR_URL_ENCODED_REDIRECT_URI]"

payloadDigest=`echo -n "$payload" | openssl dgst -binary -sha256 | openssl base64`

digest=SHA-256=$payloadDigest
```

Request parameters (refer to the sample script available to download at the beginning of this section)

curl "${httpHost}${reqPath}" - Full path of the API endpoint.

Authorization: Bearer ${access_token} - The application access token obtained in step b.

Digest: $(digest) - Calculated value of digest using the authorization code.

Date: $(reqDate) - Calculated value of date. This is the standard HTTP header (see RFC7231). Note that we enforce a strict time frame within which the request must be issued. If the Date header's time value diverges by more than 3 minutes from the time the API request is made, the request will be considered invalid.

keyId - The client_id from step 1 obtained in the response to the request for an application access token.

signature - Calculated value of the signature in order to sign your request.

Content-Type - Value set to "application/x-www-form-urlencoded". It forces the body parameters to use the x-www-form-urlencoded compliant URL encoding. Note that space character should be encoded as "+" and not as "%20".

Example request:

```
curl -i -X POST "${httpHost}${reqPath}" \
-H "Accept: application/json" \
-H "Content-Type: application/x-www-form-
urlencoded" \
-H "Digest: ${digest}" \
-H "Date: ${reqDate}" \
-H "Authorization: Bearer ${accessToken}" \
-H "Signature: keyId=\"$keyId\",algorithm=\"rsa-
sha256\",headers=\"(request-target) date
digest\",signature=\"$signature\"" \
-d "${payload}" \
--cert "/certs/" \
--key "/certs/"
```

Response parameters

access_token - It contains the customer access token.

expires_in - It contains the expiration time in seconds for the customer access token.

refresh_token - It contains the refresh token that should be used when the customer access token expires so you don't need to perform a new authorization request. To learn more about refreshing an access token and revoking a refresh_token refer to "Refresh and revoke access token" section of this documentation.

refresh_token_expires_in - It contains the expiration time in seconds for the refresh token.

x-ing-response-id - Unique tracing id which should be provided when reporting any issue.

Example response for a successful request:

```
HTTP/2 200

date: Fri, 15 Nov 2019 12:41:32 GMT
content-type: application/json;charset=UTF-8
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
x-ing-id: 495a8daf-486f-4193-80c2-1a72e755a8db
cache-control: no-cache, no-store, max-age=0, must-revalidate
pragma: no-cache
expires: 0
strict-transport-security: max-age=31622400; includeSubDomains
x-frame-options: deny

{"access_token":"eyJhbGciOiJkaXIiLCJlb.......","refresh_token":"eyJhbGciOiJkaX......","token_type":"Bearer","expires_in":900,"scope":"payment-accounts:balances:view payment-accounts:transactions:view","refresh_token_expires_in":2592000}
```

4. Call a PSD2 API with an application access token or the customer access token

With an application access token or the customer access token (in case customer consent is required) you can now call PSD2 API in production. Check the API documentation of the API you want to connect to and determine whether it requires message signing (and which type of message signing) or whether it requires mTLS (mutual TLS) only. You can jump to the next section of this Get Started guide based on these requirements.

- When the API requires an "X-JWS-Signature" header to be supplied, it requires JWS signing.
- When the API requires a "Signature" header to be supplied, it requires HTTP signing.
- When the API does not require the "X-JWS-Signature" or "Signature" headers, it requires mTLS only and no message signing.

Additional header that an API must contain are: Date, Digest, Signature and Authorization.

Refer to the "Reference" and "Documentation" sections of a particular PSD2 API you want to consume.

Migration guide for the breaking change implemented on 13-11-2024

Overview change

- Dynamic Client Registration
  - new endpoints to manage your application, certificates, redirect URLs and contact details
    - **GET /oauth2/applications/{id} or /oauth2/applications/me** - to retrieve your application's registered metadata

        - **PUT /oauth2/applications/{id}** - to update the registered application's metadata
        - **POST /oauth2/applications** - for the first time users (PSD2 TPPs) that are onboarding to ING (one time only action)
    - a new message integrity protocol (and a different way to request an access token for Dynamic Client Registration)
- Granting
    - instead of calling **the /oauth2/authorization-server-url endpoint** → use a static URL to redirect your users to ING's authorization application
    - for redirecting your users back to your application after completing ING's authorization flow → use the pre-registered redirect URLs (via **PUT /oauth2/applications/{id}**)

Depending on the action you are doing, the combination of message signature and access token will be different. In the table below you can see the overview and below in the guide we explain the details.

		Steps for existing clients	Endpoint	Access token and message signature
Application management / Dynamic client registration	New	1. Retrieve your application's registered metadata	GET /oauth2/applications/{id} GET /oauth2/applications/me	mTLS only access token (no signature)
		2. Update the registered application's metadata (one time-only required action as	PUT /oauth2/applications/{id}	mTLS only access token + JWS signature API call

		prerequisite for point 3.a)		

Granting	New	3. a) Redirect the customer to ING's authorization application and let the customer authorize your application	`https://myaccount.ing.com/authorize/v2/[COUNTRY_CODE]?`
			`client_id=[YOUR_CLIENT_ID]&scope=[SCOPES_SPACE_SEPARATED_AND_URLENCODED]&state=[SOME_ARBITRARY_BUT_UNIQUE_STRING]&redirect_uri=[YOUR_URL_ENCODED_REDIRECT_URI]&response_type=code`	

	Same as before	3. b) - d) Exchange auth code with a token (AIS, CAF)	POST /oauth2/token	HTTPS signature access token + HTTPS signature API call
	Same as before	4. Use the PSD2 APIs	AIS, PIS, CAF	PIS API: “X-JWS-Signature" acces token +X-JWS-Signature API

PSD2 APIs		call AIS / CAF API: HTTPS signature access token + HTTPS signature API call

Migration steps

0. Pre-requisites for the onboarding flow - Request an mTLS only application access token

When using the suite of **/oauth2/applications/*** endpoints, the way you request an access token from ING is changing. In broad lines, you will not have to sign this request, but rely on mutual TLS. We have described below the details:

An application access token is requested by calling ING's OAuth 2.0 API token endpoint: /oauth2/token. Requesting an application access token using mTLS requires the following parameters to be supplied.

grant_type

client_id

scope(optional)

Note that the application access token expires after 900 seconds. It is advisable to re-use the application access token as long as it has not expired, instead of requesting a new token for each API call.

Once the application access token expires, you should request a new application access token in order to re-authenticate your application.

Parameters

content-type HEADER PARAMETER

The "content-type" header is used to indicate what type of media or resource is being used for the payload in the request. The content type of this request is always application/x-www-form-urlencoded.

grant_type BODY PARAMETER

A request for an application access token requires the "grant_type" request parameter set to value "client_credentials".

```
grant_type=client_credentials
```

client_id BODY PARAMETER

The "client_id" for your registered application at ING (which you have received in the API responses). Check the table under the "Sandbox and Production environments" section of this Get Started guide to check which value to use for the "client_id" parameter.

An example value for the "client_id" parameter for a PSD2 API in the Sandbox environment:

```
client_id=5ca1ab1e-c0ca-c01a-cafe-154deadbea75
```

scope BODY PARAMETER

A request for an application access token can contain multiple scope-tokens (separated by spaces) in the "scope" parameter of the payload. The "scope" parameter value must be encoded using x-www-form-urlencoded-compliant URL encoding, and the space character must be encoded with "+" instead of "%20".

The scope parameter value with single scope-token "granting":

```
scope=granting
```

Request

Now you are ready to request an application access token using all the header and body parameters.

The following script can be used as an example request:

```
#!/bin/bash


##################################################
#############################
#                    REQUEST APPLICATION ACCESS
TOKEN  (MTLS)                 #
##################################################
#############################

certPath="./certificates/" # path of the
downloaded certificates and keys
httpHost="https://api.ing.com" # production host

reqPath="/oauth2/token"

# clientId path param required for mtls flow
clientId="5ca1ab1e-c0ca-c01a-cafe-154deadbea75" #
client_id as provided in the documentation

# You can also provide scope parameter in the body
E.g.
"grant_type=client_credentials&scope=granting"
# scope is an optional parameter. The downloaded
certificate contains all available scopes. If you
don't provide a scope, the accessToken is returned
for all scopes available in certificate
payload="grant_type=client_credentials&client_id=$
{clientId}"

# Curl request method must be in uppercase e.g
```

```
"POST", "GET"
response=$(curl -k -X POST "${httpHost}${reqPath}"
\
-H 'Accept: application/json' \
-H 'Content-Type: application/x-www-form-
urlencoded' \
-d "${payload}" \
--cert "${certPath}example_client_tls.cer" \
--key "${certPath}example_client_tls.key")
```

Response

Header

x-ing-response-id - Unique tracing id which should be provided when reporting any issue.

Body

access_token - It contains the application access token.

expires_in - It contains the expiration time in seconds for the application access token. Once the application access token expires, you should request a new application access token in order to re-authenticate your application.

scope - It contains the list of all the scopes associated with the "access_token".

Example response for a successful request:

```
HTTP/2 200

date: Fri, 15 Nov 2019 08:44:05 GMT
content-type: application/json
x-frame-options: deny
x-content-type-options: nosniff
strict-transport-security: max-age=31622400;
```

```
includeSubDomains
x-xss-protection: 1; mode=block
cache-control: no-cache, no-store, max-age=0,
must-revalidate
pragma: no-cache
expires: 0
x-ing-response-id:
bb9c6fb19dc058db9794f6f8b9e28b2d
{"access_token":"iyK4bY.....","expires_in":905,"sc
ope":"granting","token_type":"Bearer","keys":
[{"kty":"RSA","n":"34VPkdV.....","e":"AQAB","use":
"sig","alg":"RS256","x5t":"3c396700fc8cd70........
"}],"client_id":"5ca1ab1e-c0ca-c01a-cafe-
154deadbea75"}
```

1. Retrieve your application's registered metadata via GET /oauth2/applications/me or /oauth2/applications/{id}

Registered clients have a set of metadata values associated with their application identifier (client_id) at an authorization server, such as the list of valid redirection URIs, certificates and contact details.

Parameters

Authorization string *required* HEADER PARAMETER

The mtls Application access token (bearer) received in the OAuth2 Client Credentials flow.

id string *required* PATH PARAMETER

Application identifier - client_id received during onboarding.

Example requests

```
#!/bin/bash
```

```
##################################################
#############################
#                    REQUEST APPLICATIONS METADATA
#
##################################################
#############################

certPath="./certificates/" # path of the
downloaded certificates and keys
httpHost="https://api.ing.com" # production host

reqPath="/oauth2/applications/me"

accessToken="" # mtls access_token received from
calling POST /oauth2/token endpoint

# Curl request method must be in uppercase e.g
"POST", "GET"
response=$(curl -k -X GET "${httpHost}${reqPath}"
\
-H "Authorization: Bearer $accessToken" \
--cert "${certPath}example_client_tls.cer" \
--key "${certPath}example_client_tls.key")
```

Example response

```
{
"contact": "one@example.com",
"applications": [
  {
    "client_id": "c188888a-6a7d-40b0-bdcb-
c612bc1f8ba1",
    "redirect_uris":
["https://client.example.org/callback"],
    "keys": {
      "tls": [
        {
          "kty": "RSA",
          "alg": "RS256",
          "use": "sig",
          "kid": "string",
          "n": "string",
          "e": "string",
          "x5t": "string",
          "x5c": ["string"]
        }
      ],
      "sign": [
        {
          "kty": "RSA",
          "alg": "RS256",
          "use": "sig",
          "kid": "string",
          "n": "string",
          "e": "string",
          "x5t": "string",
          "x5c": ["string"]
        }
      ]
    }
  }
]
}
```

2. Update the registered application's metadata, especially the Redirect URL(s) and email address via PUT /oauth2/applications/{id}

For this you will need to use the JWS signature protocol and the **x-jws-signature header**. Note: this message signature protocol is used only for the PUT (and POST) on **/oauth2/applications/** endpoint.

Parameters

authorization HEADER PARAMETER

When calling an API, the bearer authentication scheme is used for the authorization header. This means you need to supply the mtls application access token as the value for this header parameter. For example:

```
# Please replace the value of this variable with a
valid application access token
authorization="Bearer
eyJhbGciOiJkaXIiLCJlbmMiOiJBMjU2Q0JDLUhTNTEyIiwia.
..
...bA06uS.vxbf2aRnZfa9fLwA7TOwpqzJIcPK8T10X95bPyEn
s3I"
```

digest HEADER PARAMETER

The "Digest" header is used by our server to verify the integrity of the message body that you send to us when calling any of our APIs.

Creating the digest can be done by calculating the SHA-256 hash of the payload of the message and Base64 encoding the hash. In this example the payload is empty, so the digest will be:

```
digest="SHA-
256=47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU="
```

content-type HEADER PARAMETER

The "content-type" header is used to indicate what type of media or resource is being used for the payload in the request. The content type of this requests is always application/json.

TPP-Signature-Certificate HEADER PARAMETER

The PSD2 compliant Qualified Electronic Seal Certificate (QSealC) which you will use for the signature in the client credentials grant type. The certificate (PEM format) needs to be the Base64 encoded (and then surrounded by '-----BEGIN CERTIFICATE-----' and '-----END CERTIFICATE-----' tags).

x-jws-signature HEADER PARAMETER

The "x-jws-signature" header is used to store the JSON Web Signature (JWS) for this request.

Want to learn more about the JSON Web Signature specification for Open Banking?
Your signing private key is required to create a signature of the headers and their values that you will send in your request. Check the table under the "Sandbox and Production environments" section of this Get Started guide to check which private key to use for signing.

In the following three steps we will explain how to create the JWS signature that will be used as the value for the "x-jws-signature" header:

Create JWS protected header

The JWS protected header is a JSON object which stores information about the chosen algorithm and the fingerprint of the public certificate that should be used to verify the signature, as well as which HTTP headers are included in the signature and the timestamp when the signature was created.

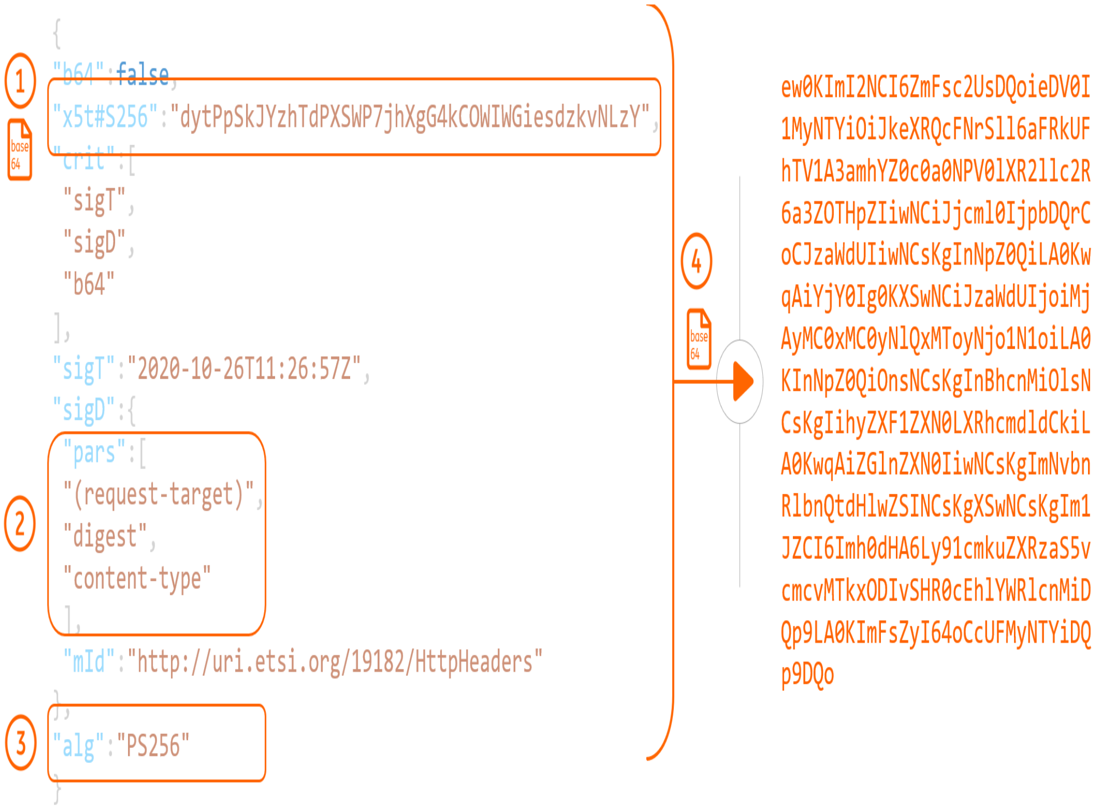
The following table explains the JWS protected header parameters:

JWS header parameter	Description
"b64": false	Means don't base64url encode header data to be signed
"x5t#S256": "...."	SHA-256 hash (fingerprint) of the public signing certificate (base64url encoded) that should be used to verify the signature
"crit":["sigT","sigD","b64"]	Non-standard JWS header

	parameters (ie. not defined in RFC 7515), which are critical
"sigT":"...."	Signing time (see RFC 3339). Format is "yyyy-MM-ddTHH:mm:ssZ", for example: "2022-04-26T11:26:04Z"
"sigD":{....}	List of HTTP Header fields to be signed (lower case) `"sigD":{ "pars":[ "(request-target)", "digest", "content-type" ], "mId":"http://uri.etsi.org/19182/HttpHeaders"}`
"alg": "...."	Signature algorithm. See API specifications on which algorithms are supported. The supported algorithms are PS256, PS384, PS512, ES256, ES384, ES512.

Follow the next steps to create the JWS protected header:

1. Base64URL encode the SHA-256 fingerprint of the signing certificate
2. Decide on HTTP headers to be signed (based on API specification)
3. Choose signing algorithm (based on API specification) - eg. PS256
4. Base64URL encode the JWS protected header



```
{
"b64":false,
"x5t#S256":"dytPpSkJYzhTdPXSWP7jhXgG4kCOWIWGiesdzkvNLzY",
"crit":[
 "sigT",
 "sigD",
 "b64"
],
"sigT":"2020-10-26T11:26:57Z",
"sigD":{
 "pars":[
 "(request-target)",
 "digest",
 "content-type"
 ],
 "mId":"http://uri.etsi.org/19182/HttpHeaders"
},
"alg":"PS256"
}
```

ew0KImI2NCI6ZmFsc2UsDQoieDV0I
1MyNTYiOiJkeXRQcFNrSll6aFRkUF
hTV1A3amhYZ0c0a0NPV0lXR2llc2R
6a3ZOTHpZIiwNCiJjcml0IjpbDQrC
oCJzaWdUIiwNCsKgInNpZ0QiLA0Kw
qAiYjY0Ig0KXSwNCiJzaWdUIjoiMj
AyMC0xMC0yNlQxMToyNjo1N1oiLA0
KInNpZ0QiOnsNCsKgInBhcnMiOlsN
CsKgIihyZXF1ZXN0LXRhcmdldCkiL
A0KwqAiZGlnZXN0IiwNCsKgImNvbn
RlbnQtdHlwZSINCsKgXSwNCsKgIm1
JZCI6Imh0dHA6Ly91cmkuZXRzaS5v
cmcvMTkxODIvSHR0cEhlYWRlcnMiD
Qp9LA0KImFsZyI6oCcUFMyNTYiDQ
p9DQo

Note that the JWS protected header does not contain any line breaks. The line breaks in the example above are just for readability.

The following script can be used for creating the JWS protected header:

```
# generate the sha-256 fingerprint for the signing
certificate, Base64urlencoded
base64UrlFingerprint=$(openssl x509 -noout -
fingerprint -sha256 -inform pem -in
${certPath}example_client_signing.cer | cut -d'='
-f2 | sed s/://g  | xxd -r -p | base64 | tr -d '='
| tr '/+' '_-' | tr -d '
')

# generate the current signing time, encoded using
RFC 3339 Internet time format for UTC without
fractional seconds (e.g. "2019-11-19T17:28:15Z")
```

```
sigT=$(date -u +"%Y-%m-%dT%H:%M:%SZ")

# create the JWS Protected Header with the sigD parameters containing the mandatory headers.
# Currently the json contains the minimal required headers to be signed: (request-target), digest and content-type.
# Please note that the list of mandatory headers can be different between APIs, check the API documentation.
jwsHeader='{"b64":false,"x5t#S256":"'"$base64UrlFingerprint"'","crit":[ "sigT", "sigD", "b64"],"sigT":"'"$sigT"'","sigD":{ "pars":[ "(request-target)", "digest", "content-type" ], "mId":"http://uri.etsi.org/19182/HttpHeaders"},"alg":"PS256"}'

# encode JWS Protected Header into Base64url
jwsHeaderBase64URL=$(echo -n "$jwsHeader" | openssl base64 -e -A | tr -d '=' | tr '/+' '_-' | tr -d '\n')
```

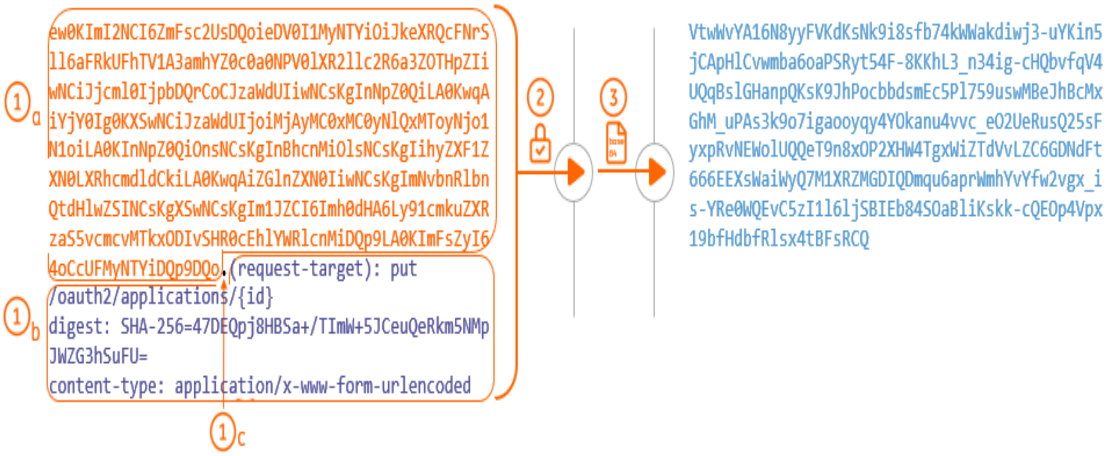
Create JWS signature value

Follow the next steps to create the JWS signature value:

1. Create JWS signature input
   a. Use the encoded JWS protected header, created in the previous step
   b. Create the HTTP header string, taking the mandatory HTTP headers in the exact order as they were listed in the sigD parameter of the JWS protected header. The HTTP headers are separated by line breaks.
   c. Concatenate the JWS protected header and HTTP header string, separated with a dot (.) character
2. Sign the JWS signature input with the private signing key using

the chosen algorithm

3. Base64URL encode the JWS signature input to get the JWS signature value



```
ew0KImI2NCI6ZmFsc2UsDQoieDV0I1MyNTYiOiJkeXRQcFNrS
ll6aFRkUFhTV1A3amhYZ0c0a0NPV0lXR2llc2R6a3ZOTHpZIi
wNCiJjcml0IjpbDQrCoCJzaWdUIiwNCsKgInNpZ0QiLA0KwqA
iYjY0Ig0KXSwNCiJzaWdUIjoiMjAyMC0xMC0yNlQxMToyNjo1
N1oiLA0KInNpZ0QiOnsNCsKgInBhcnMiOlsNCsKgIihyZXF1Z
XN0LXRhcmdldCkiLA0KwqAiZGlnZXN0IiwNCsKgImNvbnRlbn
QtdHlwZSINCsKgXSwNCsKgIm1JZCI6Imh0dHA6Ly91cmkuZXR
zaS5vcmcvMTkxODIvSHR0cEhlYWRlcnMiDQp9LA0KImFsZyI6
4oCcUFMyNTYiDQp9DQo.(request-target): put
/oauth2/applications/{id}
digest: SHA-256=47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMp
JWZG3hSuFU=
content-type: application/x-www-form-urlencoded
```

```
VtwWvYA16N8yyFVKdKsNk9i8sfb74kWWakdiwj3-uYKin5
jCApHlCvwmba6oaPSRyt54F-8KKhL3_n34ig-cHQbvfqV4
UQqBslGHanpQKsK9JhPocbbdsmEc5Pl759uswMBeJhBcMx
GhM_uPAs3k9o7igaooyqy4YOkanu4vvc_eO2UeRusQ25sF
yxpRvNEWolUQQeT9n8xOP2XHW4TgxWiZTdVvLZC6GDNdFt
666EEXsWaiWyQ7M1XRZMGDIQDmqu6aprWmhYvYfw2vgx_i
s-YRe0WQEvC5zI1l6ljSBIEb84SOaBliKskk-cQEOp4Vpx
19bfHdbfRlsx4tBFsRCQ
```

Note that the JWS protected header does not contain any line breaks. The line breaks in the example above are just for readability.

The following script can be used for creating the JWS signature value:

```
httpMethod="put"
reqPath="/oauth2/applications/$client_id"

# calculate digest of the request body parameters
digest="SHA-256="$(echo -n $payload | openssl dgst
-binary -sha256 | openssl base64)

# the HTTP header string must be declared exactly
as shown below, with each header on a separate
line
signingString="(request-target): $httpMethod
$reqPath
digest: $digest
content-type: application/json"
```

```
# combine Base64url encoded JWS Protected Header
with the signingString
payload="$jwsHeaderBase64URL.$signingString"

# sign and encode the payload with signing private
key
jwsSignatureValueBase64URL=$(printf %s "$payload"
| openssl dgst -sha256 -sign
${certPath}example_client_signing.key -sigopt
rsa_padding_mode:pss | openssl base64 -A | tr -d
'=' | tr '/+' '_-' | tr -d '\n')
```

Create JWS signature

Follow the next steps to create the JWS signature:

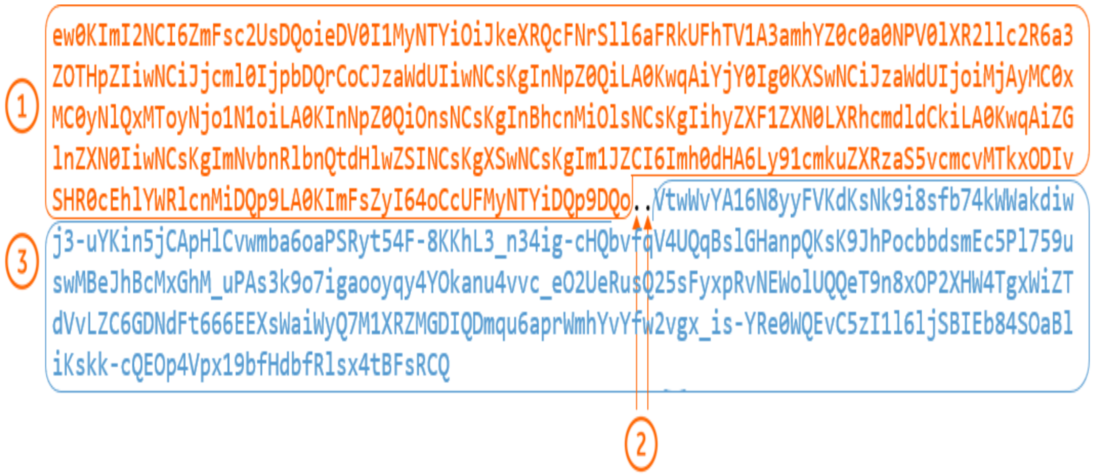
The format for the JWS signature is

(1) (2) (3)

[protected header]. [payload]. [signature value]

The payload value (2) is empty

Concatenate the JWS protected header (1), empty payload (2) and the JWS signature value (3), all separated with a dot (.) character (4)



ew0KImI2NCI6ZmFsc2UsDQoieDV0I1MyNTYiOiJkeXRQcFNrSll6aFRkUFhTV1A3amhYZ0c0a0NPV0lXR2llc2R6a3
ZOTHpZIiwNCiJjcml0IjpbDQrCoCJzaWdUIiwNCsKgInNpZ0QiLA0KwqAiYjY0Ig0KXSwNCiJzaWdUIjoiMjAyMC0x
MC0yNlQxMToyNjo1N1oiLA0KInNpZ0QiOnsNCsKgInBhcnMiOlsNCsKgIihyZXF1ZXN0LXRhcmdldCkiLA0KwqAiZG
lnZXN0IiwNCsKgImNvbnRlbnQtdHlwZSINCsKgXSwNCsKgIm1JZCI6Imh0dHA6Ly91cmkuZXRzaS5vcmcvMTkxODIv
SHR0cEhlYWRlcnMiDQp9LA0KImFsZyI6oCcUFMyNTYiDQp9DQo..VtwWvYA16N8yyFVKdKsNk9i8sfb74kWWakdiw
j3-uYKin5jCApHlCvwmba6oaPSRyt54F-8KKhL3_n34ig-cHQbvfqV4UQqBslGHanpQKsK9JhPocbbdsmEc5Pl759u
swMBeJhBcMxGhM_uPAs3k9o7igaooyqy4YOkanu4vvc_eO2UeRusQ25sFyxpRvNEWolUQQeT9n8xOP2XHW4TgxWiZT
dVvLZC6GDNdFt666EEXsWaiWyQ7M1XRZMGDIQDmqu6aprWmhYvYfw2vgx_is-YRe0WQEvC5zI1l6ljSBIEb84SOaBl
iKskk-cQEOp4Vpx19bfHdbfRlsx4tBFsRCQ

Note that the JWS protected header does not contain any line breaks. The line breaks in the example above are just for readability.

The following script can be used for creating the JWS signature:

```
# build JSON Web Signature
jwsSignature=$jwsHeaderBase64URL..$jwsSignatureVal
ueBase64URL
```

The signature should be dynamically calculated for every request since the values that are used as input to create the signature change every time, eg. the timestamp used in the sigT parameter of the JWS Protected Header.

Request

Once you have created the signature, you are ready to call the endpoint PUT "/oauth2/applications/{id}" using all the header and body parameters.

The following script can be used as an example request:

```
curl -v -i -X PUT "${httpHost}${reqPath}" \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
-H "Digest: ${digest}" \
-H "Authorization: Bearer ${accessToken}" \
-H "TPP-Signature-Certificate:
$signCertificateHeader" \
-H "x-jws-signature: ${jwsSignature}" \
--cert "${certPath}example_client_tls.cer" \
--key "${certPath}example_client_tls.key"
```

id *string* **PATH PARAMETER**

Application identifier - client_id received during onboarding.

Request Body

> If you want to register new certificates, make sure you create the TLS connection with the already onboarded certificates and specify the new certificates in the request body.

```
{
"contact": "one@example.com",
"redirect_uris":
["https://client.example.org/callback"],
"keys": {
  "tls": [
    {
      "kty": "RSA",
      "alg": "RS256",
      "use": "sig",
      "kid": "string",
      "n": "string",
      "e": "string",
      "x5t": "string",
      "x5c": ["string"]
    }
  ],
  "sign": [
    {
      "kty": "RSA",
      "alg": "RS256",
      "use": "sig",
      "kid": "string",
      "n": "string",
      "e": "string",
      "x5t": "string",
      "x5c": ["string"]
    }
  ]
}
}
```

JWS Signature troubleshooting

If the JWS Signature verification was unsuccessful, here are potential fixes:

1. a field (for example 'date') is included in jws protected header 'pars' parameter, but not included in the signing input (HTTP headers to be signed).
2. a field (for example 'date') is included in the signing input (HTTP headers to be signed), but not included in jws protected header 'pars' parameter.
3. the http method or the path used in the signing input (HTTP headers to be signed) are not corresponding to the ones in the actual HTTP request.
4. when creating the signing input (collecting the HTTP headers to be signed), there might be an extra LF (line feed: 0x0A in UTF-8) after the last header.
5. one of the headers have different values in the http request than the ones used when creating the signature.
6. the signature was created with the wrong private key, one that does not correspond to the certificate hash fingerprint from the JWS Protected Header.
7. the signing input (HTTP headers to be signed) are not defined in the same order as defined in the SigD pars.
8. the 'alg' field is not corresponding to what was used for signing (ex: alg:PS256 but signed with PS384).
9. If you are on Windows, you can try a different cURL version

10. Make sure that sigT is +/- 3 min
11. If all the above were checked, you can try with a different Open SSL library. In our testing Open SSL version 3.2.1 provided the expected result.

Response

```
{
"contact": "one@example.com",
"application": {
  "client_id": "c188888a-6a7d-40b0-bdcb-
c612bc1f8ba1",
  "redirect_uris":
["https://client.example.org/callback"],
  "keys": {
    "tls": [
      {
        "kty": "RSA",
        "alg": "RS256",
        "use": "sig",
        "kid": "string",
        "n": "string",
        "e": "string",
        "x5t": "string",
        "x5c": ["string"]
      }
    ],
    "sign": [
      {
        "kty": "RSA",
        "alg": "RS256",
        "use": "sig",
        "kid": "string",
        "n": "string",
        "e": "string",
        "x5t": "string",
        "x5c": ["string"]
      }
    ]
  }
}
}
```

> ⚠ Please keep in mind that the PUT call will clear all existing data and replace it with the new configuration. For example, if you have two sets of certificates registered and send only one, only the sent one will be saved.

RedirectURIs backwards compatibility during the migration period

During this migration period, if you use both the old and new granting flows simultaneously (i.e. registered at least one Redirect URL with the new flow via PUT /oauth2/applications/{id} while still utilizing other redirects via the old flow GET /oauth2/authorization-server-url), you must add the form parameter: redirect_uri on POST /oauth2/token (used for exchanging auth-code for a token). Failing to do so will result in a 400 error.

Alternatively, you can resolve this issues by deleting all redirects with the new endpoint PUT /oauth2/applications/{clientId} and by omitting the redirect_URI body parameter.

3. Granting

Pending you have registered your Redirect URLs (as described above), this step is only required for APIs that require customer consent (eg. for PSD2: AIS and CAF). ING has implemented OAuth 2.0 authorization code flow to help you get access to customer data upon their explicit consent. To learn more about the Authorization code flow refer to the Authorization code flow section in our OAuth 2.0 documentation.

In this step a request for customer consent is initiated and an authorization code is generated. Using the application access token and the authorization code, your application obtains a customer access token. Finally, your application uses the customer access token to consume the API that requires customer consent.

a) Redirect the customer to ING's authorization application and let the customer authorize your application

Important: Redirect the customer to their default browser in which they can confirm that they are communicating with the trusted ING website. You are not allowed to embed ING's authorization page in your application in any way! ING can block your application from using ING APIs when identifying such an implementation.

Redirect the customer to ING's authorization application URL, by appending your application specific parameters:

```
https://myaccount.ing.com/authorize/v2/[COUNTRY_CODE]?

client_id=[YOUR_CLIENT_ID]
&scope=[SCOPES_SPACE_SEPARATED_AND_URLENCODED]
&state=[SOME_ARBITRARY_BUT_UNIQUE_STRING]
&redirect_uri=[YOUR_URL_ENCODED_REDIRECT_URI]
&response_type=code
```

COUNTRY_CODE QUERY PARAMETER

Is an optional parameter and allows you to redirect your customer to the specified ING country login page. The format is a two-letter value (ISO 3166-1). If you don't add the country_code, the ING customer will be redirected to an ING country selection page.

Country	Code
Belgium	BE
The Netherlands	NL

Spain	ES
Romania	RO
Luxembourg	LU
Italy	IT
Germany	DE
Wholesale Banking	WB

client_id QUERY PARAMETER

Required parameter, the unique client_id obtained when you registered your application with ING.

scope QUERY PARAMETER

Required parameter, a list of scopes separated by spaces for which an authorization request is being sent. This could be a subset of the scopes you application is allowed to request. The value must be URL encoded, where spaces may be encoded to either "+" or "%20".

state QUERY PARAMETER

Required parameter, can be used by your application to maintain state between HTTP redirects and achieve per-request customization of your redirect URL, please use the "state" parameter as described in the OAuth 2.0 RFC.

The state parameter must not be blank. Allowed characters for the "state" parameter are:

> Alphanumeric characters in range A-Z, a-z, 0-9
>
> Special characters =%#,;+/-_
>
> Any whitespace character \r \n \t \f \v

redirect_uri QUERY PARAMETER

Optional parameter, a specific redirect URI where the customer will return to once the authorization process is ended. The value must be URL encoded. This parameter is optional if your application has registered only a single redirect URL with ING. This is a mandatory parameter in case your application has registered multiple redirect URLs and should be one of them.

response_type QUERY PARAMETER

Required parameter, for PSD2 flow, the value must be "code".

Example redirect url

An example URL to redirect the customer to start the authorization flow should look like this:

```
https://myaccount.ing.com/authorize/v2/NL?client_id=6414808c-d8da-450e-b10d-8a1c1dd37561&scope=payment-accounts%3Atransactions%3Aview+payment-accounts%3Abalances%3Aview&state=123456&response_type=code&redirect_uri=http%3A%2F%2Fapi.example.com
```

b) Redirect the customer back to your application and acquire an authorization code

Once successful, ING's authorization application redirects the customer to the URL provided in the redirect_uri query parameter along with the authorization code. For example:

```
https://www.example.com/?state=[YOUR_STATE_VALUE]&code=876fyb93-a597-44r2-838f-71b2a57699tb
```

The "code" parameter in the redirected URL contains the dynamically generated authorization code for the selected test profile. This authorization code is used in the next step in order to request a customer access token.

c) Request an application access token

Here you can download the sample script to request an application access token for the PSD2 APIs.

An application access token is requested by calling ING's OAuth 2.0 API token endpoint: /oauth2/token. To call this API endpoint, you must create a signature and sign the HTTP request. We have explained below how to calculate the digest and the date for creating a signature.

Want to learn more about HTTP message Signing?

Calculate Digest

```
payload="grant_type=client_credentials"
payloadDigest=`echo -n "$payload" | openssl dgst -binary -sha256 | openssl base64`
digest=SHA-256=$payloadDigest
```

"Digest" header is used by our server to verify the integrity of the message body that you send to us when calling any of our APIs.

A request for an application access token can contain multiple scope-tokens (separated by spaces) in the "scope" parameter of the payload. The "scope" parameter value must be encoded using x-www-form-urlencoded-compliant URL encoding and the space character must be encoded with "+" instead of "%20" as explained below.

The scope parameter value with single scope-token "payment-accounts:balances:view".

```
grant_type=client_credentials&scope=payment-accounts%3Abalances%3Aview
```

The scope parameter value with multiple scope-tokens "payment-accounts:balances:view" and "payment-accounts:transactions:view".

```
grant_type=client_credentials&scope=payment-
accounts%3Abalances%3Aview%20payment-
accounts%3Atransactions%3Aview
```

When no scope is explicitly requested the application access token will contain all the scope-tokens available in the certificate that you are using. Refer to the "Reference" section of the PSD2 API you want to consume to see the available scopes.

Calculate date

```
reqDate=$(LC_TIME=en_US.UTF-8 date -u "+%a, %d %b
%Y %H:%M:%S GMT")
```

💡 Note that there should not be a gap of more than 3 minutes between the time when the PSD2 API endpoint request is made and the time value in the "date" header field of the request.

Calculate signature for signing your request

```
httpMethod="post"
reqPath="/oauth2/token"
signingString="(request-target): $httpMethod
$reqPath
date: $reqDate
digest: $digest"
signature=`printf %s "$signingString" | openssl
dgst -sha256 -sign "${certPath}
example_client_signing.key" -passin
"pass:changeit" | openssl base64 -A`
```

Signature should be dynamically calculated for every PSD2 API endpoint request since the values that are used as input to create the signature change every time.

Once you have created the signature, you are ready to request an application access token using the below header and body parameters:

Request parameters (refer to the sample script available to download at the beginning of this section):

curl "${httpHost}${reqPath}" - Full path of the API endpoint.

Digest: $(digest) - Calculated value of digest.

Date: $(reqDate) - Calculated value of date. This is the standard HTTP header (see RFC7231). Note that we enforce a strict time frame within which the request must be issued. If the Date header's time value diverges by more than 3 minutes from the time the API request is made, the request will be considered invalid.

TPP-Signature-Certificate - Copy and paste the downloaded example of certificate "example_client_signing.cer" as a single line string including the BEGIN CERTIFICATE and the END CERTIFICATE lines.

keyId - It has a format "SN=XXX, where “XXX" is the serial number of the certificate in hexadecimal coding. You can extract the serial number and full distinguished name of CA using below openssl command:

```
openssl x509 -in -serial -noout
```

example output:

```
serial=49BA5C71
```

signature - Calculated value of the signature in order to sign your request.

Content-Type Value set to "application/x-www-form-urlencoded".

It forces the body parameters to use the x-www-form-urlencoded compliant URL encoding. Note that space character should be encoded as "+" and not as "%20".

Example request:

```
curl -i -X POST "${httpHost}${reqPath}" \
-H 'Accept: application/json' \
-H 'Content-Type: application/x-www-form-
urlencoded' \
-H "Digest: ${digest}" \
-H "Date: ${reqDate}" \
-H "TPP-Signature-Certificate: -----BEGIN
CERTIFICATE-----
MIIENjCCAx6gAwIBAgIEXkKZv.....LkyWk4Mw1w0TnQLAq+s=
-----END CERTIFICATE-----" \
-H "authorization: Signature
keyId=\"$keyId\",algorithm=\"rsa-
sha256\",headers=\"(request-target) date
digest\",signature=\"$signature\"" \
-d "${payload}" \
--cert "/certs/example_client_tls.cer" \
--key "/certs/example_client_tls.key"
```

Response parameters

access_token - It contains the application access token.

client_id - It contains the client_id generated when registering the application in the previous step

expires_in - It contains the expiration time in seconds for the application access token. Once the application access token expires, you should execute this step (Step 1) again in order to re-authenticate your application and request a new application access token.

scope - It contains the list of all the scopes associated with the "access_token".

x-ing-response-id - Unique tracing id which should be provided when reporting any issue.

Note that the application access token expires after 900 seconds after which you need to execute this step again in order to re-authenticate your application and request a new application access token. It is advisable to re-use the application access token instead of requesting a new token for each API call.

Example response for a successful request:

```
HTTP/2 200

date: Fri, 15 Nov 2019 11:20:50 GMT
content-type: application/json
x-frame-options: deny
x-content-type-options: nosniff
strict-transport-security: max-age=31622400;
includeSubDomains
x-xss-protection: 1; mode=block
cache-control: no-cache, no-store, max-age=0,
must-revalidate
pragma: no-cache
expires: 0
{"access_token":"eyJhb....","expires_in":905,"scop
e":"payment-accounts:orders:create
granting","token_type":"Bearer","keys":
[{"kty":"RSA","n":"3l3rdzkdV.....","e":"AQAB","use
":"sig","alg":"RS256","x5t":"3c396700fc8cd709cf9cb
5452a22bcde76985851"}],"client_id":"5ca1ab1e-c0ca-
c01a-cafe-154deadbea75"}
```

You can configure your TLS implementation to verify the trust chain of ING's TLS certificate used for setting up the tunnel between your application and ING. Our PKI certificate is signed by our Root CA which in turn is signed by a commercial CA.

d) Use the authorization code to request the customer access token

Here you can download the sample script to request the customer access token from the authorization code.

The authorization code is valid for only 10 minutes after issuing. Your application should exchange the authorization code for a pair of customer access token and a refresh token within this time frame using the endpoint /oauth2/token and the below request parameters.

You must use the authorization code while calculating the digest value as shown below:

```
payload="grant_type=authorization_code&code=[ING_AUTHORIZATION_CODE]&redirect_uri=[YOUR_URL_ENCODED_REDIRECT_URI]"

payloadDigest=` echo -n "$payload" | openssl dgst -binary -sha256 | openssl base64`

digest=SHA-256=$payloadDigest
```

Request parameters (refer to the sample script available to download at the beginning of this section)

curl "${httpHost}${reqPath}" - Full path of the API endpoint.

Authorization: Bearer ${access_token} - The application access token obtained in step b.

Digest: $(digest) - Calculated value of digest using the authorization code.

Date: $(reqDate) - Calculated value of date. This is the standard HTTP header (see RFC7231). Note that we enforce a strict time frame within which the request must be issued. If the Date header's time value diverges by more than 3 minutes from the time the API request is made, the request will be considered invalid.

keyId - The client_id from step 1 obtained in the response to the request for an application access token.

signature - Calculated value of the signature in order to sign your request.

Content-Type - Value set to "application/x-www-form-urlencoded". It forces the body parameters to use the x-www-form-urlencoded compliant URL encoding. Note that space character should be encoded as "+" and not as "%20".

Example request:

```
curl -i -X POST "${httpHost}${reqPath}" \
-H "Accept: application/json" \
-H "Content-Type: application/x-www-form-
urlencoded" \
-H "Digest: ${digest}" \
-H "Date: ${reqDate}" \
-H "Authorization: Bearer ${accessToken}" \
-H "Signature: keyId=\"$keyId\",algorithm=\"rsa-
sha256\",headers=\"(request-target) date
digest\",signature=\"$signature\"" \
-d "${payload}" \
--cert "/certs/example_client_tls.cer" \
--key "/certs/example_client_tls.key"
```

Response parameters

access_token - It contains the customer access token.

expires_in - It contains the expiration time in seconds for the customer access token.

refresh_token - It contains the refresh token that should be used in case the customer access token expires. To learn more about refreshing an access token and revoking a refresh token refer to

"Refresh and revoke access token" section of this documentation.

refresh_token_expires_in - It contains the expiration time in seconds for the refresh token.

x-ing-response-id - Unique tracing id which should be provided when reporting any issue.

Example response for a successful request:

```
HTTP/2 200

date: Fri, 15 Nov 2019 12:41:32 GMT
content-type: application/json;charset=UTF-8
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
cache-control: no-cache, no-store, max-age=0, must-revalidate
pragma: no-cache
expires: 0
strict-transport-security: max-age=31622400; includeSubDomains
x-frame-options: deny

{"access_token":"eyJhbGciOiJkaXIiLCJlb.......","refresh_token":"eyJhbGciOiJkaX......","token_type":"Bearer","expires_in":900,"scope":"payment-accounts:balances:view payment-accounts:transactions:view","refresh_token_expires_in":2592000}
```

4. Call a PSD2 API with an application access token or the customer access token

With an application acces token or the customer access token (in case customer consent is required) you can now call PSD2 API in production. All API requests should be signed using the "signature" HTTP header as described earlier and must contain at least the following headers: Date, Digest, Signature and Authorization.

Refer to the "Reference" and "Documentation" sections of a particular PSD2 API you want to consume.

Migration FAQ

Question	Answer
I received a 401 error code when performing the authorization code flow (granting) with a valid access token	Make sure that you have retrieved the access token via the correct flow - in this case, a signed access token is required.
When performing a granting flow I receive an error.	Ensure that you are following the oAuth2 documentation, specifically the part about redirect URLs - registering them and/or referencing them in the authorization code exchange.
As a PSD2 TPP, do I have to create a developer portal account and configure it?	No. You have to use the API interactions as referenced in the oAuth2 API.
I've been onboarded with ING before this breaking change, how do I retrieve my client_id?	Please check the API response you have received from ING when authenticating (via POST /oauth2/token). The client_id is one of the response attributes.
I tried to update my certificates and one of them is revoked. What should I do?	You cannot update your configuration with revoked certificates.
I received an error related to the JWS signature.	You can refer to the JWS troubleshooting part of the

	migration guide.
Is the PUT request needed when there is a TPP certificate renewal?	Yes.
What should we provide as digest for the PUT and POST calls?	You should provide the digest generated based on the payload of the request.

Terms of Use

Privacy Statement

Cookie Statement

Security