Handout iDEAL 2.0 Migration

Version 1.2
8-2-2024

Contents

1 Introduction

This handout is created to get you quickly started with the iDEAL 2.0 integration/migration, and is based on the available information provided in the following documents:

- iDEAL 2.0 Getting Started Guide
- Open Banking API v3 for iDEAL – Implementation Guide
- swagger-oauth_ver1.4.5.yaml
- open-banking-payment_initiation_service_swagger_3.4.0-v3_July_2023.yaml
- Open_Banking_Notification_Service_v3_ver3.3.1.yaml

The documents can be downloaded from the Rabobank iDEAL Dashboard under Documents.

This handout assumes that you already have an iDEAL contract with Rabobank, and focusses only on the provided test environment. The change from test to production is described in the iDEAL 2.0 Getting Started Guide document.

Please note that our examples are based on our own test code written in Python. The given code examples are converted versions from Postman based on our Python tests.

1.1 iDEAL's corporate identity requirements

To read more about the iDEAL corporate identity requirements, please read the **User Guide iDEAL MSP** document which can be found at the Rabo iDEAL Dashboard under Documents.

1.2 Impact Implementation

The iDEAL 2.0 migration differs on several points compared to the iDEAL 1.0 implementation. The most important changes are that …

… the interface changes from XML messaging to API calls
… the bank selection is done on the iDEAL site instead of the merchant's website/webshop
… the issuer ID should **not** be sent in the iDEAL API calls
… the payment status is a pushed notification instead of a pull request
… the communication is done with access tokens which are valid for 60 minutes
… on the iDEAL Dashboard additional mandatory fields are defined and should be filled, i.e. the Status Notification URL and the Notification Bearer Token

2 Global Variables

Before we start we need the necessary files for the test merchant. In our case the test merchant without sub-IDs. Please download the following files from the Rabobank iDEAL Dashboard and store them in a folder you can easily access.

- TestCertificatesiDEAL.2.0.pem
- TestCertificatesiDEAL.2.0.key
- PublicTPPCertificate.crt

Furthermore, we defined the following variables:

- BASE_DOMAIN = 'https://routingservice-rabo.awltest.de'
- APP = 'IDEAL'
- CLIENT = 'RaboiDEAL'
- ID = '002881'
- FINGERPRINT = None <-- We need to read this from the certificate (.pem)

In code this looks like:

```
# GLOBAL VARIABLES
CERT_LOC = 'TestCertificatesiDEAL.2.0.pem'
KEY_LOC = 'TestCertificatesiDEAL.2.0.key'
TPP_CERT = 'PublicTPPCertificate.crt'
BASE_DOMAIN = 'https://routingservice-rabo.awltest.de'

APP = 'IDEAL'
CLIENT = 'RaboiDEAL'
ID = '002881'
FINGERPRINT = None
```

2.1 Get the Current Date and Time

Before we can start with the iDEAL API calls, we need the current date and time in the correct format. The preferred datetime format is the standard ISO 8601 format, e.g. 2023-12-07T13:00:00.839+01:00.
For testing purposes, we only extract the date and time once per hour. This makes is easier to debug errors and to reperform code.

Example code to extract the datetime in ISO 8601 format:

```
# Update today per hour
current_dt = datetime.utcnow()
today = datetime(current_dt.year, current_dt.month, current_dt.day, current_dt.hour, 00,
00).isoformat() + '.839+01:00'
```

2.2 Get the Thumbprint from the Certificate

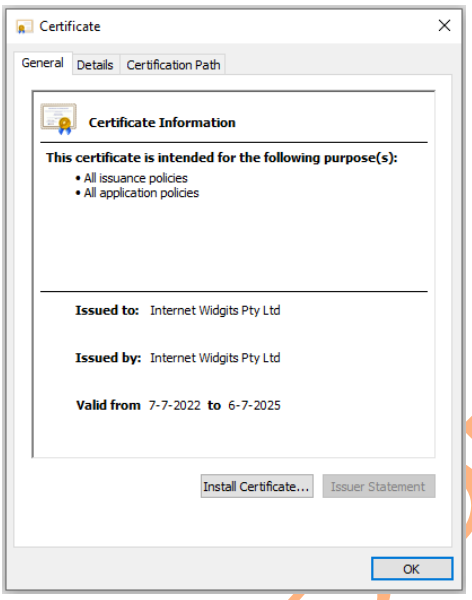
Next step is to get the thumbprint, or SHA1 hash, of the certificate. This can be done by checking the certificate itself, and copy paste the thumbprint. Or by reading the certificate with code and hash it.

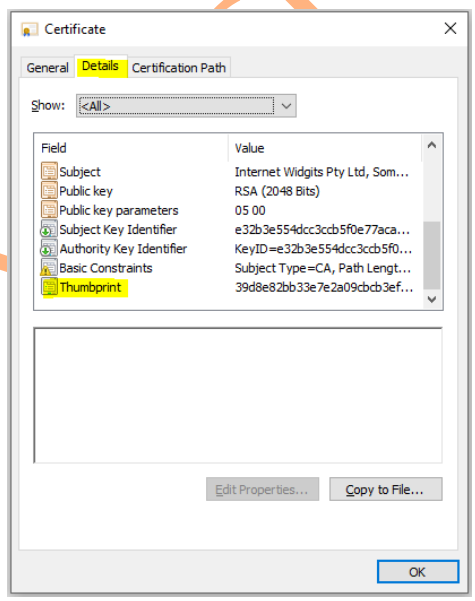
2.2.1 Read Thumbprint from Certificate

The thumbprint of the certificate can be found by opening the certificate file.

2.2.1.1 Test Certificate Thumbprint

On Windows, it is the easiest way to rename the certificate file from 'TestCertificatesiDEAL.2.0.pem' to 'TestCertificatesiDEAL.2.0.cer'. Now the file can be opened with a double click, and the following window will appear (in Windows):





Under the tab 'Details' you will find the thumbprint. For the test certificate the thumbprint is:

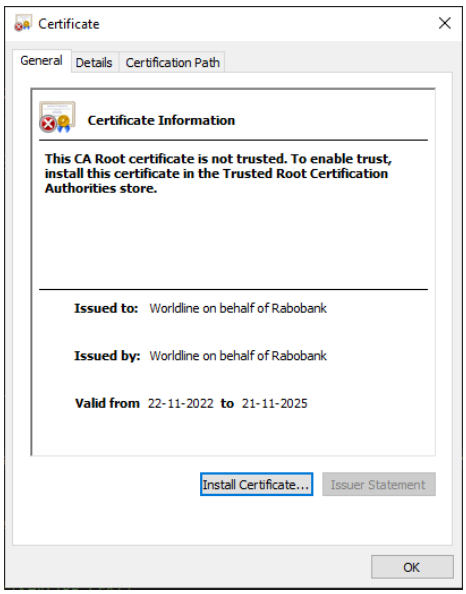
- `39d8e82bb33e7e2a09cbcb3ef3eab351ee1c5e8f`

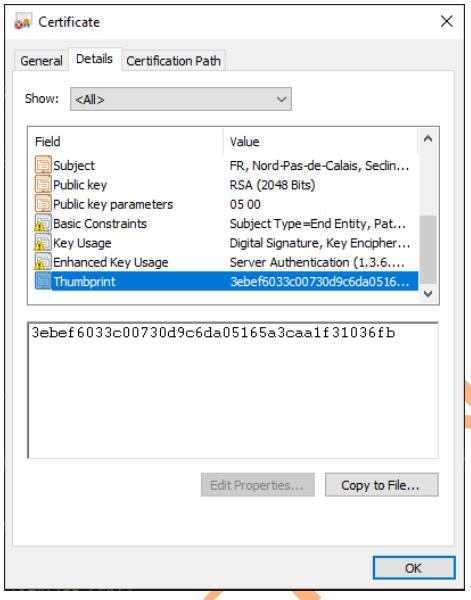
You can now update the FINGERPRINT variable in the code with this value.

Please note that you must update the fingerprint value when you move to production, and every time the certificate is no longer valid.

2.2.1.2 Public TPP Certificate Thumbprint

For the public TPP Certificate, which is already in .crt format, you only have to open the certificate with a double click, and the following window will appear (in Windows):





Under the tab 'Details' you will find the thumbprint. For the test certificate the thumbprint is:

- `3ebef6033c00730d9c6da05165a3caa1f31036fb`

You can now update the FINGERPRINT variable in the code with this value.

2.2.2 Extract the Thumbprint with Code

In case you want to read the fingerprint from the certificate, the certificate file must be read without its starting and end tags.

- -----BEGIN CERTIFICATE-----
- -----END CERTIFICATE-----

Read the characters between the tags as a single line and hash the string with the SHA-1 hashing algorithm.

The following code is an example of how the string can be encoded and hashed:

```
def getSHAHash(iStringToHash, iHashType = 'SHA1', iEncoding = 'ascii'):
    key = base64.b64decode(iStringToHash.encode(iEncoding))
    result = hashlib.sha1(key).hexdigest()
   return result
```

3 Get Access Token

To get the access token (the Bearer token), we need to make an API call to the server and identify ourselves. To do this, the following API call is available:

```
URL: https://routingservice-rabo.awltest.de/xs2a/routingservice/services/authorize/token

Headers:
    • Content-Type
    • App
    • Client
    • Id
    • Date
```

```
    • Authorization
    • Accept

Parameter:
    • grant_type
```

Most values of the API call are given and fixed. However, the 'Authorization' headers need to be calculated to identify ourselves as the authorized merchant.

3.1 Authorization Header

The authorization header consists of the concatenation of 4 elements. Depending on the API call the element values can differ.

Element	Description	Example value
Signature keyId	The thumbprint of the certificate to identify the merchant	39d8e82bb33e7e2a09cbcb3ef3eab351ee1c5e8f
algorithm	A fixed value how the message and signate is encrypted	SHA256withRSA
headers	The values and order of the encrypted signature values	app client id date
signature	The encrypted value of the header values	

Note that the headers contain the date time, and this will lead to different signatures.

The Authorization header will have the following format:

```
Authorization:  Signature keyId = <FINGERPRINT>, algorithm = <ALGORITHM>, headers = "app client id date", signature = <SIGNATURE>
```

3.1.1 Authorization Header Signature

The signature element of the authorization headers is the encrypted and signed value of the header element values. For the access token, the header element is defined as 'app client id date' which indicates that the following message is signed with the private key to create the signature element.

The header fields "app", "client", "id", and "date" are globally defined variables which we need to fill. In case of the test merchant 002881, the following header needs to be signed:

```
"app: IDEAL\n" \
"client: RaboiDEAL\n" \
"id: 002881\n" \
"date: 2023-12-15T15:00:00.839+01:00"
```

To sign the header, the private key must be used (without a password) to sign the message with the PKCS algorithm followed by a SHA-256 hash.
Please note that a SALT is not part of the algorithm and should not be used.

To give an example of signing a message with the private key, the following Python code could be used:

```
def getSignature(iPrivateKey, iMessage, iPassword = None):
    # Create certificate object
    from cryptography.hazmat.primitives import serialization, hashes
    from cryptography import x509
    from cryptography.hazmat.primitives.asymmetric import padding

    with open(iPrivateKey, "rb") as key_file:
        private_key = serialization.load_pem_private_key(
            key_file.read(),
            password=iPassword,
        )

    # Calculate signature
    signature = private_key.sign(bytes(iMessage, 'ascii')
                                 , padding.PKCS1v15()
                                 , hashes.SHA256()
                                )

    return base64.b64encode(signature).decode()
```

For the header

```
"app: IDEAL\n" \
"client: RaboiDEAL\n" \
"id: 002881\n" \
"date: 2023-12-15T15:00:00.839+01:00"
```

the signature is calculated as:

```
UZ1io6fv1faQpSSJrNeMLOt9lvFoXjDp6N4LYqETWGwQcA54GMQNtGstW2AHeW2zVHup977yBPEIpOnkjFfBSqvqeYCUjqngZWhR
WmvOdi9EJuqKFR02QQsxdxej0mBGsIGsv57WW2lJSUj2RURA/JJKMLpvvlwUBJleUrzfrNCBW2/YsUxzw0+cjTg9VDdSJFY6qJGG
Ygdm3LuayYo3PXxqBnVH2+9FTkOTm0CEOR9G16KJtEu5a+vic1dGUXja6BM8irY1nMxFzGi7jiNbuPByLbGrfY4TbDTxOu5WSkAh
+Jv9X7JSVoyIhC5MeqtElO4It3anTDiO49GkWXfAFQ==
```

3.2 Access Token API call

Now all the variables of the API call have a value, the API call to retrieve an access token can be made. The following call will give a valid access token (Bearer token).
cURL code:

```
curl -X POST \
  'https://routingservice-
rabo.awltest.de/xs2a/routingservice/services/authorize/token?grant_type=client_credentials' \
  -H 'Accept: application/json' \
```

```
  -H 'App: IDEAL' \
  -H 'Authorization: Signature keyId="39D8E82BB33E7E2A09CBCB3EF3EAB351EE1C5E8F",
algorithm="SHA256withRSA", headers="app client id date",
signature="UZ1io6fvlfaQpSSJrNeMLOt9lvFoXjDp6N4LYqETWGwQcA54GMQNtGstW2AHeW2zVHup977yBPEIpOnkjFfBSqvqe
YCUjqngZWhRWmvOdi9EJuqKFR02QQsxdxej0mBGsIGsv57WW2lJSUj2RURA/JJKMLpvvlwUBJleUrzfrNCBW2/YsUxzw0+cjTg9V
DdSJFY6qJGGYgdm3LuayYo3PXxqBnVH2+9FTkOTm0CEOR9G16KJtEu5a+vic1dGUXja6BM8irY1nMxFzGi7jiNbuPByLbGrfY4Tb
DTxOu5WSkAh+Jv9X7JSVoyIhC5MeqtElO4It3anTDiO49GkWXfAFQ==" \
  -H 'Client: RaboiDEAL' \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  -H 'Date: 2023-12-15T15:00:00.839+01:00'\
  -H 'Id: 002881' \
```

This will lead to the following result:

```
X-Request-ID:    5f6f9935-8732-48ab-a37b-c050cdaa4179
MessageDateTime: 2023-12-15T15:25:58.441Z
{
    "access_token": "96b2ea31f529c5cd61d534ad0c079decd81a75cb79743345a36ce016946c11a0",
    "token_type": "Bearer",
    "expires_in": 610
}
```

Note that the access token is only valid for **60 minutes**. So when executing the code above, a new access token will be generated.

4 Start the Payment Initiation

After receiving the access token, we are able to identify ourselves and can start a payment initiation. The payment initiation API can be called with the following elements:

```
URL: https://routingservice-rabo.awltest.de/xs2a/routingservice/services/ob/pis/v3/payments

Headers:
    • Accept
    • Content-Type
    • Digest
    • MessageCreateDateTime
    • Signature
    • Authorization
    • InitiatingPartyNotificationUrl*

Body:
    • Payment message
```

* The InitiatingPartyNotificationUrl is mandatory for the test environment, and optional for the production environment. For the production environment, the payment status is pushed to the Status Notification URL given in the Rabobank iDEAL Dashboard, or to the Initiating Party Notification URL if provided in the header (overwrites the Status Notification URL).

The following sections will describe the different elements to call the payment initiation API. It will start with the body of the call which contains the payment message, and continues with the header elements.

4.1 Payment Message

A payment message contains the information of the payment of the customer. Every initiate payment will lead to a payment status. To be sure all payment statuses will be handled in the right way, the test environment links specific amount to the different payment messages. The following amounts can be tested:

Payment Initiation Amount	Payment Status
€ 1,00	SettlementCompleted
€ 2,00	Cancelled
€ 3,00	Expired
€ 4,00	Open
€ 5,00	Error

A payment initiation message is formatted in JSON. Depending on if you are using a standard iDEAL payment without profile recognition, standard iDEAL payment with profile recognition, or Fast Checkout iDEAL payment, the JSON elements will change.

An example of a standard iDEAL payment without profile recognition looks like this (pretty printed):

```
{
    "PaymentProduct": [
      "IDEAL"
    ],
    "CommonPaymentData": {
      "Amount": {
        "Type": "Fixed",
        "Amount": "1.00",
        "Currency": "EUR"
      },
      "RemittanceInformation": "iDEAL | MERCHANT NAME",
      "RemittanceInformationStructured": {
        "Reference": "iDEALaankoop1"
      }
    },
    "IDEALPayments": {
      "UseDebtorToken": false,
      "FlowType": "Standard"
    }
}
```

Please update the example with your *merchant's name* for testing in the test environment. In production, the remittance information should be the (sub) merchant name, and will be shown on the transaction overview of the customer. For example, Rabobank Netherlands (merchant (sub id) name) or Rabo via Rabobank Netherlands (in case of a CPSP). The reference is the reference

information you can use to reconciliate the payment, for example the order ID, or a description. The reference will also be shown on the transaction overview of the customer.

For testing, it is recommended that you have at least 5 payment files to test all the different statuses. In case you want to use iDEAL with profile recognition and/or iDEAL fast checkout, please create those test payment files as well.

4.2 Payment Header Elements

The header of the payment initiation has several elements. Some of them need some calculations, where others are semi-fixed and/or easy to retrieve.

Element	Description	Example value
MessageCreateDateTime	The date and timestamp of the payment initiation	2023-12-15T15:00:00.839+01:00
Authorization	The authorization is the access token that is retrieved earlier. It should start with Bearer followed by the token.	Bearer d99c68d8d4bc87a27c7c64427431efa6f2a7ee1870ef8abdcdaff19755e90a38
InitiatingPartyNotificationUrl	The URL which catches the response of the iDEAL payment with the payment status. Note that this element is mandatory in the test environment, and optional in the production environment.	https://[merchantdomain.nl]/notification/v3
Content-Type	The type of the payment message. In this case, the format is JSON	application/json

The remaining elements require some additional calculations will be described in the following sub sections.

4.2.1 Payment Message - Digest Calculation

The digest is the base64-encoded value of the SHA-256 hashed payment message. The calculation of the digest is as follows:

1) Create a SHA-256 hash of the payload message
    a. Please make sure the payload message is binary data before applying the hash.
    b. The format of the payload message is important. If you choose a pretty printed JSON message, please make sure you also provide the pretty printed format in the API call.
2) Base64 encode the SHA-256 hash
3) Concatenate the hashing method with the base64 encoded hash

The payment message of 4.1 is calculated as follows:

Hexdigest: e991adfd59259bbdb409e78ccdcf85560328cdbfd64cae7e6c2ae4a7b622bfc2
Base64 encoded: 6ZGt/Vklm720CeeMzc+FVgMozb/WTK5+bCrkp7Yiv8l=

To give an example of the digest calculation, the following Python code could be used:

```
def getSHAHash(iStringToHash, iHashType = 'SHA1', iEncoding = 'ascii'):
    if iHashType == 'SHA1':
        key = base64.b64decode(iStringToHash.encode(iEncoding))
        result = hashlib.sha1(key).hexdigest()
    elif iHashType == 'SHA256':
        hexResult = hashlib.sha256(iStringToHash).hexdigest()
        result = base64.b64encode(bytes.fromhex(hexResult)).decode()
    else:
        result = ''
    return result
```

4.2.2 Signature of the Payment Message

The signature of the payment message consists of the concatenation of 4 elements. Depending on the API call the element values can differ.

Element	Description	Example value
keyId	The thumbprint of the certificate to identify the merchant	39d8e82bb33e7e2a09cbcb3ef3eab351ee1c5e8f
algorithm	A fixed value how the message and signate is encrypted	SHA256withRSA
headers	The values and order of the encrypted signature values	digest x-request-id messagecreatedatetime (request-target)
signature	The encrypted value of the header values	

Note that the headers contain the date time, and this will lead to different signatures.

Digest
The digest is the result that is calculated in the section before (see Payment Message - Digest Calculation).

X-Request-ID
The X-Request-ID is an *Universal Unique Identifier* (UUID) which is used to identify the unique payments. UUIDs are 36 characters strings containing numbers (0-9), letters (a-f) and dashes. UUIDs are written in 5 groups of hexadecimal digits separated by hyphens. The length of each group is: 8-4-4-4-12
An example of an X-Request-ID: 123e4567-e89b-12d3-a456-426655440000

Request-Target
The request target is a fixed element and can have different values depending on the API request.

API	Request Target
POST /payments	post /xs2a/routingservice/services/ob/pis/v3/payments

GET /payments/{paymentId}/status*	get /xs2a/routingservice/services/ob/pis/v3/payments/{{paymentId}}/status*
GET /preferences/{Psuid}*	get /xs2a/routingservice/services/ob/pis/v3/preferences/{{psuId}}*

* The placeholders {{paymentId}} and {{psuId}} must be substituted with the values as obtained within the respective flows.

The signature header will have the following format:

```
Signature:  keyId = <FINGERPRINT>, algorithm = <ALGORITHM>, headers = "digest x-request-id
messagecreatedatetime (request-id)", signature = <SIGNATURE>
```

The signature element is the encrypted and signed with the private key of the certificate. The encryption is done in the same way as for the authorization header to retrieve the access token (see Authorization Header Signature).

For the header:

```
"digest: [DIGEST VALUE]\n" \
"x-request-id: [REQUEST-ID]\n"\
"messagecreatedatetime: [DATETIME]\n" \
"(request-target): post /xs2a/routingservice/services/ob/pis/v3/payments"
```

the signature with request ID 'e1904971-8b52-47ea-8374-a229aaedb927' is calculated as:

```
NK7JLzOKHBrmUjJ5guqH/r7jhwvvcvJtNKU7RNL52IMY4MFvgUrKQ8Vq4+CQzQAS2J/VldhL2tq2dEjbjCpTH3q7fU9+zkJZOGHa
akVB0QcOPDdhYXokr8kN+K6Jia5FTy6iFjEwOu9/WbXXDienVxCaW4KwJa/ug3BU+yhllF0lS9Y0hhGFQRShR9MiGBag7KYzPhUN
Hclq9RPmiXv/aOz50AS89wT0O2baz/deWg27lWYlkP/P039t9pOV4hW5UXI4FYqg/em7w+CamDbke4G47yU2VYHj1yLsPi2rSQO+
SST3gHMQM9yad4NLCJOmdrIdl4ovODGwLXV2FBowWg==
```

4.3 Payment Initiation API call

Now all the variables of the API call have a value, the API call to retrieve an access token can be made. The following call will initiate a payment. The response of the API call will provide the transaction ID – payment ID – and some basic information. Please be aware that the payment status, indicating if the payment was successful or not, will be pushed to the provided URL as mentioned in the Rabobank iDEAL Dashboard (for production), or to the InitiatingPartyNotificationUrl in the test environment (or in production of the element is provided).

cURL code:

```
curl -X POST \
  https://routingservice-rabo.awltest.de/xs2a/routingservice/services/ob/pis/v3/payments \
  -H 'Accept: application/json' \
  -H 'Authorization: Bearer 952a33457b4500bb5ef06c9e5ad4626abe9fffa827baf4ce15801cc906b91863' \
  -H 'Content-Type: application/json' \
  -H 'Digest: SHA-256=ARJCbf0/Z6W5MWUVxR/tDNdHn+TMXhNbU4CDJNaOB9I=' \
  -H 'InitiatingPartyNotificationUrl: https://[MERCHANT DOMAIN]/notification/status/v3' \
  -H 'MessageCreateDateTime: 2023-12-15T15:00:00.839+01:00\
  -H 'Signature: keyId="39d8e82bb33e7e2a09cbcb3ef3eab351ee1c5e8f", algorithm="SHA256withRSA",
headers="digest x-request-id messagecreatedatetime (request-target)",
signature="NK7JLzOKHBrmUjJ5guqH/r7jhwvvcvJtNKU7RNL52IMY4MFvgUrKQ8Vq4+CQzQAS2J/VldhL2tq2dEjbjCpTH3q7f
U9+zkJZOGHaakVB0QcOPDdhYXokr8kN+K6Jia5FTy6iFjEwOu9/WbXXDienVxCaW4KwJa/ug3BU+yhllF0lS9Y0hhGFQRShR9MiG
Bag7KYzPhUNHclq9RPmiXv/aOz50AS89wT0O2baz/deWg27lWYlkP/P039t9pOV4hW5UXI4FYqg/em7w+CamDbke4G47yU2VYHj1
yLsPi2rSQO+SST3gHMQM9yad4NLCJOmdrIdl4ovODGwLXV2FBowWg==" \
```

```
  -H 'X-Request-ID: 27864cad-3edd-45c3-af3d-002c4285d2c7' \
  -d ' {
    "PaymentProduct": [
      "IDEAL"
    ],
    "CommonPaymentData": {
      "Amount": {
        "Type": "Fixed",
        "Amount": "1.00",
        "Currency": "EUR"
      },
      "RemittanceInformation": "iDEAL | MERCHANT NAME",
      "RemittanceInformationStructured": {
        "Reference": "iDEALaankoop1"
      }
    },
    "IDEALPayments": {
      "UseDebtorToken": false,
      "FlowType": "Standard"
    }
}'
```

This will lead to the following result:

```
{
    "CommonPaymentData": {
        "ExpiryDateTimestamp": "2023-12-15T16:51:27.177Z",
        "PaymentStatus": "Open",
        "PaymentId": "139835",
        "AspspPaymentId": "0001417695934485"
    },
    "Links": {
        "RedirectUrl": {
            "Href": "https://worldline.com"
        },
        "GetPaymentStatus": {
            "Href": "https://routingservice-
rabo.awltest.de/xs2a/routingservice/services/ob/pis/v3/payments/139835/status"
        }
    },
    "UseWaitingScreen": false
}
```

This result is not the final status of the payment, but provides the payment ID. In case the payment status notification is not pushed, or it is taking too long, the payment status can be requested with a GET API call when the payment ID is known.

4.4 Payment Status Notification

As mentioned earlier, the payment status notification is pushed to the provided URL. It is therefore important that the URL is able to receive this notification message. The notification will contain all the necessary information of the payment in JSON format.

An example of a payment status notification:

Signature	keyId="3EBEF6033C00730D9C6DA05165A3CAA1F31036FB",algorithm="rsa-sha256",headers="messagecreatedatetime x-request-id digest",signature="OH5FS0VZDMIzvteWUMG0X7fZL5SzE7BiS+G++Wt9HTjaBn5yPbLl/35dVOO2owwQ/cHqvrX/7pwCB0fJHPcVOvu6+t0H0s6l1Iwac9dpiDFIUVezeEqO956sBtBM/xfb3hY0tXjcyEcEL45RiUg9JGcp90DlrVDYKawVNnpJ3roojBnwSdsqQJ6angPr3Z4KUT3APcSj79CwE+PCLSaThjB50oEUrEuXCz4bnHYOQh6Rf5nqhEP4Bi6ZnXAHo7gKQ61riBn2JNH71YF/se5MxupmBrD1QTYzCilJySJ5htGNkIO7Veg2DOszEjtmE7QxFvZ9/8EOHwxXl6eROxlFmA=="
digest	SHA-256=1kxt5WrHJH7UXHUanV3TmgZgr3d2RvZi0VzXOgn2t28=
messagecreatedatetime	2023-12-15T16:57:01.071+01:00
x-request-id	0394f748-6928-4be6-b1f0-b16c0897af8e
authorization	Bearer iDEAL2.0testnotificationtoken


```
{
  "PaymentProductUsed": "IDEAL",
  "CommonPaymentData": {
    "GuaranteedAmount": "1.00",
    "PaymentStatus": "SettlementCompleted",
    "PaymentId": "139869",
    "AspspPaymentId": "0001667464638715",
    "AspspId": "RABONL2UXXX",
    "DebtorInformation": {
      "Name": "Edsger Wybe Dijkstra - Callback",
      "Agent": "ABNANL2AXXX",
      "Account": {
        "SchemeName": "IBAN",
        "Identification": "NL44RABO0123456789",
        "Currency": "EUR"
      },
      "ContactDetails": {
        "FirstName": "Edsger",
        "LastName": "Dijkstra",
        "PhoneNumber": "+31612345678",
        "Email": "edsger@domain.nl"
      }
    }
  },
  "UseWaitingScreen": false
}
```

5 Verification of the Payment Status

The response that is pushed to the provided URL is potentially at risk. It is therefore important to verify the payment status message. The verification can be done by checking the following:

- Does the header have the correct Bearer token as provided in the Rabo iDEAL Dashboard?
  - For the test environment the Bearer token is iDEAL2.0testnotificationtoken
- Calculate the digest, and compare with the received digest
  - SHA256 hash of the payment status response body
- Validate the signature with the public certificate of the TPP Solution

5.1 Example of payment Status Response Verification

As an example, we take the following response:

Signature	keyId="3EBEF6033C00730D9C6DA05165A3CAA1F31036FB",algorithm="rsa-sha256",headers="messagecreatedatetime x-request-id digest",signature="OH5FS0VZDMIzvteWUMG0X7fZL5SzE7BiS+G++Wt9HTjaBn5yPbLl/35dVOO2owwQ/cHqvrX/7pwCB0fJHPcVOvu6+t0H0s6l1Iwac9dpiDFIUVezeEqO956sBtBM/xfb3hY0tXjcyEcEL45RiUg9JGcp90DlrVDYKawVNnpJ3roojBnwSdsqQJ6angPr3Z4KUT3APcSj79CwE+PCLSaThjB50oEUrEuXCz4bnHYOQh6Rf5nqhEP4Bi6ZnXAHo7gKQ61riBn2JNH71YF/se5MxupmBrD1QTYzCilJySJ5htGNkIO7Veg2DOszEjtmE7QxFvZ9/8EOHwxXl6eROxlFmA=="
digest	SHA-256=1kxt5WrHJH7UXHUanV3TmgZgr3d2RvZi0VzXOgn2t28=
messagecreatedatetime	2023-12-15T16:57:01.071+01:00
x-request-id	0394f748-6928-4be6-b1f0-b16c0897af8e
authorization	Bearer iDEAL2.0testnotificationtoken


```
{
  "PaymentProductUsed": "IDEAL",
  "CommonPaymentData": {
    "GuaranteedAmount": "1.00",
    "PaymentStatus": "SettlementCompleted",
    "PaymentId": "139869",
    "AspspPaymentId": "0001667464638715",
    "AspspId": "RABONL2UXXX",
    "DebtorInformation": {
      "Name": "Edsger Wybe Dijkstra - Callback",
      "Agent": "ABNANL2AXXX",
      "Account": {
        "SchemeName": "IBAN",
        "Identification": "NL44RABO0123456789",
        "Currency": "EUR"
      },
      "ContactDetails": {
        "FirstName": "Edsger",
        "LastName": "Dijkstra",
        "PhoneNumber": "+31612345678",
        "Email": "edsger@domain.nl"
      }
```

```
    }
  },
  "UseWaitingScreen": false
}
```

Bearer Token

The bearer token only needs to be checked to be correct. In the test environment the Bearer token is 'iDEAL2.0testnotificationtoken', which is equal to the Bearer token in the header of the payment status response.

Digest

To calculate the digest of the payment message (body), please note that the digest is calculated on the JSON string without formatting. In this case it looks like this:

```
{"PaymentProductUsed":"IDEAL","CommonPaymentData":{"GuaranteedAmount":"1.00","PaymentStatus":"Settle
mentCompleted","PaymentId":"139869","AspspPaymentId":"0001667464638715","AspspId":"RABONL2UXXX","Deb
torInformation":{"Name":"Edsger Wybe Dijkstra -
Callback","Agent":"ABNANL2AXXX","Account":{"SchemeName":"IBAN","Identification":"NL44RABO0123456789"
,"Currency":"EUR"},"ContactDetails":{"FirstName":"Edsger","LastName":"Dijkstra","PhoneNumber":"+3161
2345678","Email":"edsger@domain.nl"}}},"UseWaitingScreen":false}
```

With digest: NkPTOHRpuSH5hdcLVVuVCGJxNR6SwMmrf8Y/k3QeMwc=
This is equal to the digest mentioned in the header of the payment status response.

Signature validation

The signature can be validated with the public certificate of the TPP Solution. The certificate can be downloaded from the Rabo iDEAL dashboard.
To verify the signature, you need to provide the signature in base64 decoded format, the signature message (based on the header information), the signing method (PKCS1v15), and the SHA256 hash. Please note that the signature message is formatted as described in 4.2.2 Signature of the Payment Message.

6 Manual Payment Status Request (Optional)

In case the pushed payment status notification message is taking too long to receive, a GET API call can be made to request the payment status. The payment status request API can be called with the following elements:

```
URL: https://routingservice-rabo.awltest.de/ /xs2a/routingservice/services/ob/pis/v3/payments/[
PAYMENTID]/status

Headers:
    • Accept
    • Content-Type
    • MessageCreateDateTime
    • Authorization
    • Signature
```

Please note that the URL needs to be updated with the PaymentID to retrieve the status.

Element	Description	Example value
Accept	The type of the payment message. In this case, the format is JSON	application/json
Content-Type	The type of the payment message. In this case, the format is JSON	application/json
MessageCreateDateTime	The date and timestamp of the payment initiation	2023-12-14T10:00:00.839+01:00
Authorization	The authorization is the access token that is retrieved earlier. It should start with Bearer followed by the token.	Bearer d99c68d8d4bc87a27c7c64427431efa6f2a7ee1870ef8abdcdaff19755e90a38
Signature	The encrypted value of the header values	

As most elements are seen before, only the signature header is explained in more detail in the next section.

6.1 Signature header

The signature of the payment status request consists of the concatenation of 4 elements. Depending on the API call the element values can differ.

Element	Description	Example value
keyId	The thumbprint of the certificate to identify the merchant	`39d8e82bb33e7e2a09cbcb3ef3eab351ee1c5e8f`
algorithm	A fixed value how the message and signate is encrypted	SHA256withRSA
headers	The values and order of the encrypted signature values	x-request-id messagecreatedatetime (request-target)
signature	The encrypted value of the header values	

Note that the headers contain the date time, and this will lead to different signatures.

X-Request-ID

The X-Request-ID is an *Universal Unique Identifier* (UUID) which is used to identify the unique payments. UUIDs are 36 characters strings containing numbers (0-9), letters (a-f) and dashes. UUIDs are written in 5 groups of hexadecimal digits separated by hyphens. The length of each group is: 8-4-4-4-12

An example of an X-Request-ID: 123e4567-e89b-12d3-a456-426655440000

Request-Target

The request target is a fixed element and can have different values depending on the API request.

API	Request Target
POST /payments	post /xs2a/routingservice/services/ob/pis/v3/payments

GET /payments/{paymentId}/status*	get /xs2a/routingservice/services/ob/pis/v3/payments/{{paymentId}}/status*
GET /preferences/{Psuid}*	get /xs2a/routingservice/services/ob/pis/v3/preferences/{{psuId}}*

* The placeholders {{paymentId}} and {{psuId}} must be substituted with the values as obtained within the respective flows.

The signature header will have the following format:

```
Signature:  keyId = <FINGERPRINT>, algorithm = <ALGORITHM>, headers = "x-request-id
messagecreatedatetime (request-id)", signature = <SIGNATURE>
```

The signature element is the encrypted and signed with the private key of the certificate. The encryption is done in the same way as for the authorization header to retrieve the access token (see Authorization Header Signature).

For the header:

```
"x-request-id: [REQUEST ID]\n"\
"messagecreatedatetime: [DATE TIME]\n" \
"(request-target): get /xs2a/routingservice/services/ob/pis/v3/payments[PAYMENT ID]/status"
```

the signature – with the correctly filled parameters - is calculated as:

```
mIKRwcjeL9wn2GSJL+pwSSDoMqJQZA20M4FtALxSJZEOsjHNcFiYYISDdDN3MR/pSuAEDQ4+Jvf2Sg7FDSWlswvK6gaOjGzyZBjQ
32dcYZPKZ7Qzld5eovU2PynHHqVntiMsqloCE7TaTB6QH674Hy6Osiv0QnJJ7QtSg/b1tobEn4yqfs0+7KbCpEnodPtnCAtksmPH
D3bdOVuSEj8k3ApQak48SxEhkoVZb4MsbRGeLrdE0ftmrRIbGQVSI2eopN5mteEi8vQcPEcTdoJPrtOCEOzGgY09hy83/v1O+HDy
gWasuFmeej1zsn7U0RFbarImx0X7fMBOU2lEB0u+KA==
```

6.2 Payment Status Request Response

Now all the variables of the API call have a value, the API call to request the payment status can be made. The response of the API request be the same as when the payment status notification that will be pushed.

cURL code:

```
curl -X GET \
  https://routingservice-
rabo.awltest.de/xs2a/routingservice/services/ob/pis/v3/payments/139613/status \
  -H 'Accept: application/json' \
  -H 'Authorization: Bearer  3b8e9e579609b0298ab2d18f168fbf25962388d3120da7cd8bff84115ca2cd22' \
  -H 'Content-Type: application/json' \
  -H 'MessageCreateDateTime: 2023-12-15T15:00:00.839+01:00' \
  -H 'Signature: keyId="39d8e82bb33e7e2a09cbcb3ef3eab351ee1c5e8f", algorithm="SHA256withRSA",
headers="x-request-id messagecreatedatetime (request-target)",
signature="mIKRwcjeL9wn2GSJL+pwSSDoMqJQZA20M4FtALxSJZEOsjHNcFiYYISDdDN3MR/pSuAEDQ4+Jvf2Sg7FDSWlswvK6
gaOjGzyZBjQ32dcYZPKZ7Qzld5eovU2PynHHqVntiMsqloCE7TaTB6QH674Hy6Osiv0QnJJ7QtSg/b1tobEn4yqfs0+7KbCpEnod
PtnCAtksmPHD3bdOVuSEj8k3ApQak48SxEhkoVZb4MsbRGeLrdE0ftmrRIbGQVSI2eopN5mteEi8vQcPEcTdoJPrtOCEOzGgY09h
y83/v1O+HDygWasuFmeej1zsn7U0RFbarImx0X7fMBOU2lEB0u+KA==" \
  -H 'X-Request-ID: 95d862a0-3a95-4bcf-9d09-684b67426f2a' \
```

This will lead to the following result:

```
{
    "PaymentProductUsed": "IDEAL",
```

```
    "CommonPaymentData": {
        "PaymentStatus": "SettlementCompleted",
        "PaymentId": "139869",
        "AspspPaymentId": "0001667464638715",
        "AspspId": "10002",
        "DebtorInformation": {
            "Name": "Edsger Wybe Dijkstra - Callback",
            "Agent": "ABNANL2AXXX",
            "Account": {
                "SchemeName": "IBAN",
                "Identification": "NL44RABO0123456789"
            }
        }
    }
}
```


As stated for the payment status notification, it is strongly advised to verify the response to ensure that the message is not intercepted.