mano.bank

mano.bank Payments API specification

Version 2.1

How to integrate with mano.bank Payments API

To be able to run given code samples with our payment system you'll need to have specific settings typically available to our customers. If you are planning to consider us to be your service provider and would like to try mano.bank Payments API, please contact us by email: support@mano.bank.

Exchange required info

Client should provide to mano.bank the following info

- Account/s (one or few) to be accessible by API and Payments limits per operation/day
- The name of the application that will use the API
- IP (one, or few) from which client will call mano.bank Payments API
- It's Private key's X.509 Certificate attributes:
    - **C - Country Name:** 2-digit country code where your organization is legally located.
    - **ST - State/Province:** Full name of the state where your organization is legally located.
    - **L - City:** Full name of the city where your organization is legally located.
    - **O - Organization Name:** Legal name of Client organization.
    - **OU - Organization Unit:** Name of the department (Not mandatory. Press Enter to skip)
    - **CN - Common Name:** Client's User (application) Name
    - **Email:** Clients representative email address

i.e.

```
C = LT,
ST = LT,
L = Vilnius,
O = Organization,
OU = IT,
CN = client-api-gw,
emailAddress = admin@client.com
```

Then mano.bank will provide

- *Client_Id*
- *User_Id*
- JWT *audience Id* string (base URl of mano.bank Payment Api i.e. api-test.mano.bank/payments)
- JWT *issuer Id* string (usually same as Client Id)
- JWT *subject* string (usually same as User Id)
- JWT maximum token expiration time (usually 60min)

These attributes should be used by Client to create JWT and in Requests Headers. See appropriate sections for more information.

Create key for signing JWT and Payment requests

To be able sign JWT and Request body, client should create asymmetric RSA key. Also, Client should create X.509 certificate using attributes provided and agreed with mano.bank (see section above).

Below is example how to create X.509 certificate (could be self-signed) and new private key by using ***openssl***

```
openssl req -nodes -newkey rsa:2048 -keyout client.key -out client.crt -x509 -days 730
```

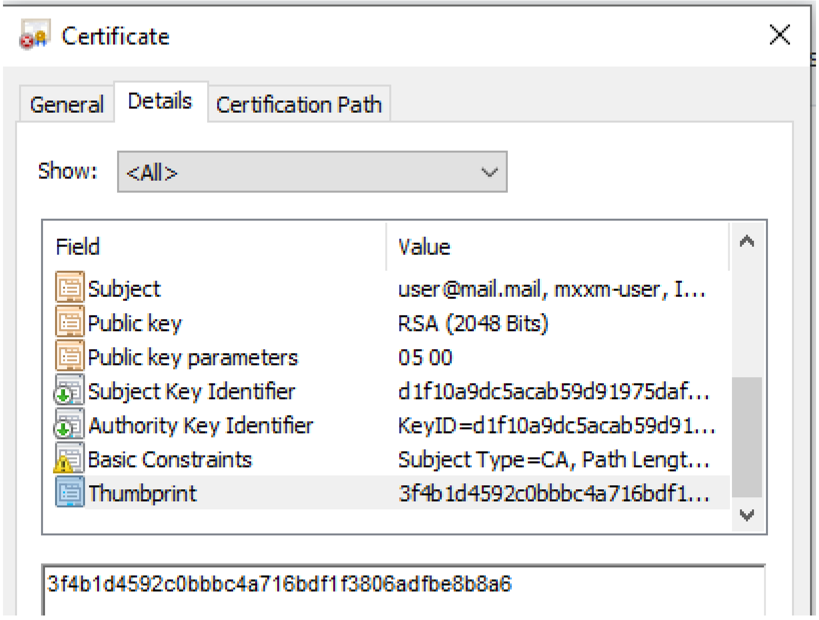
mano.bank requires to use only RSA algorithm with 2048 key length for private key generation.

Certificate valid period is minimum 2 years (730 days).

Provide X.509 certificate with Public key included

Send your certificate and public key (i.e. client.crt) to mano.bank representative.

Certificate thumbprint used to identify Client's key



Show: <All>

Field	Value
Subject	user@mail.mail, mxxm-user, I...
Public key	RSA (2048 Bits)
Public key parameters	05 00
Subject Key Identifier	d1f10a9dc5acab59d91975daf...
Authority Key Identifier	KeyID=d1f10a9dc5acab59d91...
Basic Constraints	Subject Type=CA, Path Lengt...
Thumbprint	3f4b1d4592c0bbbc4a716bdf1...

3f4b1d4592c0bbbc4a716bdf1f3806adfbe8b8a6

API Overview

Terminology

In this section are listed some terminology used in the documentation.

Term	Explanation
Resource Owner	The Resource Owner refers to the mano.bank who owns data and authorizes Client access to data
Client	The Client refers to the consumer of the API, which is commonly an application
API Call	API call is a request towards the API which receives a response. The API is by design stateless, and therefore it does not "remember" anything about previous requests, i.e., there is no session. Therefore, every request made towards the API must contain certain headers so that the API can authenticate and authorize the Client.
Access Authorization	Access Authorization is the process through which the Client obtains permission to access the Resource Owner's data and services at the bank.
Access Token	A token which is generated and signed by the Client. The access token is passed by the Client in all mano.bank Payment API calls.

API HTTP Methods

RESTful APIs use HTTP methods to perform actions to fetch, modify, add or delete resources. Here is the list of methods used in this API.

GET - This method reads a resource and returns it. It returns 200 on success.

POST - This method creates a new resource. It returns 201 on success.

Following table shows which methods are supported by the APIs.

API	GET	POST	PATCH	DELETE
Payments API	X	X	-	-

API Response codes

The API response HTTP codes are returned for the user application within the JSON object. These codes can be divided into four categories.

- 2xx Success
- 3xx Redirection
- 4xx Client errors
- 5xx Server error

Every response returned by this API has a response code. Response codes can be used to check the result of the requests, e.g., was the request successful or did it fail.

The following table shows the return codes used by Payment API:

HTTP Status Code	Text	Description
200	OK	Request was fulfilled.
201	Created	The request has been fulfilled, resulting in the creation of a new resource.
302	Found	Redirect.
401	Unauthorized	Like 403 Forbidden, but specifically for use when authentication is required and has failed or has not yet been provided.
403	Forbidden	The request was valid, but the server is refusing action. The user might not have the necessary permissions for a resource.
404	Not Found	The requested resource could not be found but may be available in the future.

Idempotency

mano.bank Payments APIs support idempotency, allowing to retry a request multiple times while only performing the action once. This helps avoid unwanted duplication in case of failures and retries.

If a transient error or a time-out occurred and the action was not completed in the previous request, the subsequent retry pushes the action to its completion. If the action is already completed, the action is to be performed only once and the same result is to be returned for the retry process.

mano.bank payment-related API is idempotent. The following table shows the payment idempotency rules.

Idempotency fields

The following table lists the idempotency fields of APIs.

API name	Idempotency Field	Rules
accounts-payment	referenceId	Client uses the *referenceId* field for idempotency control. For payment requests that are initiated with the same value of *referenceId*, the same result is to be returned for the request.

		Note: Pay attention, field name should be strongly case sensitive and in camelCase, as specified here, otherwise idempotency won't be applied. It's on client responsibility. **Note:** This API checks the consistency of these key request parameters such as *referenceId*, *payerAccountNumber*, *beneficiaryAccountNumber*, *amount* and *currency*. If any of the key values is different from that of the previous request, the error REPEAT_REQ_INCONSISTENT is to be returned.

Message encoding

To prevent errors or ambiguity caused by special characters enclosed in a message, encode the message properly before transmitting it.

Encoding scenarios	Encoding method
For the byte data, such as the JWT, Digest, Signature and the other encrypted content, encode the data with the base64UrlEncode algorithm before transmitting.	base64UrlEncode

Example for Base64UrlEncode in JavaScript Code Snippet

```
//BASE64URL function
function base64UrlEncode(source) {
   // Encode in classical base64
   encodedSource = CryptoJS.enc.Base64.stringify(source)

   // Remove padding equal characters
   encodedSource = encodedSource.replace(/=+$/, '')

   // Replace characters according to base64url specifications
   encodedSource = encodedSource.replace(/\+/g, '-')
   encodedSource = encodedSource.replace(/\//g, '_')

   return encodedSource
}
```

HTTP Message Headers

Request Headers (expected as part of the request)

Date	string **Required**

	HTTP header element for date and time represented as RFC 7231 Full Dates. Example date: Wed, 24 Apr 2019 14:00:37 EEST
Host	string **Required** HTTP header element for specification of the domain name of the server
X-MB-Client-Id	string **Required** Client ID provided by mano.bank, identifying the Client as organisation
X-MB-User-Id	string **Required** User ID provided by mano.bank, identifying the Client's application as api consumer
Authorization	string **Required** JWT Access Token, in "Authorization: Bearer JWT_TOKEN" format (<Access Token> provided by Client)
Digest	string **Required** Digest header as defined in [RFC3230] contains a Hash of the message body **Note**: this header should be Base64Url Encoded
Signature	string **Required** Signed Digest with other headers for non-repudiation Application-level signature of the request by the Client, using its Private X509 key [https://tools.ietf.org/id/draft-cavage-http-signatures-12.html]
Request-Id	string **Required** Unique Request Id

Digest Header format

SHA256 hash of the HTTP request's payload(messageBody), which is Base64Url Encoded and normalized as a **Digest** HTTP header. You must concatenate SHA-256=" with the digest string, as shown below.

Digest calculation example in JavaScript Code Snippet

```
function calculateDigest() {
  const requestData = resolveRequestBody();
  const sha256digest = CryptoJS.SHA256(requestData);
  //base64UrlEncode
  const base64sha256 = base64UrlEncode(sha256digest);
   //concatenate SHA-256=
  const calculatedDigest = 'SHA-256=' + base64sha256;
  return calculatedDigest;
}
```

Example: payload(messageBody) =

```
{
    "referenceId": "PMD-02498",
    "amount": 99.04,
    "currency": "EUR",
    "payerAccountNumber": "LT225030120000000198",
    "beneficiaryAccountNumber": "LT805030120000000221",
    "paymentDetailsType": "UNSTRUCTURED",
    "paymentDetails": "payment invoice nr.1032-26",
    "beneficiary": {
        "name": "Cargo357"
    }
}
```

Result of calculated Digest should look like this:

```
Digest: SHA-256= pS847o6ACtCqPVAntLRoZS5VTYbqfL1Cp4b-WYCICJc
```

Host Header format

The Payments API server name to which the request is sent.

In Test ENV

api-test.mano.bank

In Production ENV

api.mano.bank

Date Header format

Date and time the message originated, using GMT format, as defined by RFC7231.

Example:

```
Date: Wed, 25 Dec 2017 00:23:05 GMT
```

JavaScript Code Snippet

```
String gmtDateTime =
DateTimeFormatter.RFC_1123_DATE_TIME.format(ZonedDateTime.now(ZoneId.of("GMT")))
```

Authorization

Client Authorization is implemented using JSON Web Tokens (JWT).

Understand how JWT is generally used

Please look at https://jwt.io/introduction/
For debugging JWT requests you can use https://jwt.io/#debugger
To choose a library for JWT please see https://jwt.io
Notice that the JWT tokens are case sensitive.

Using JWT to communicate with mano.bank API

For each HTTP request, using the JWT library of your choice, create a JWT token, and set the following fields as described below. Set the JWT token in the request "Authorization" header as

```
"Authorization": "Bearer <JWT token>"
```

Access token (jwt) structure

JWT Structure: HEADER.PAYLOAD.SIGNATURE

HEADER: ALGORITHM & TOKEN TYPE

The JWT header should have these parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA and key id used to sign this access token.

alg	string The signing algorithm being used, in our case only value "RSA256"
typ	string The type of token being used, which in our case is always "JWT"
kid	string kid: Key id. Client Public key certificate thumbprint (see section Certificate thumbprint)

PAYLOAD:DATA

iss	string <= 100 characters

	Issuer. A unique identifier of the entity that issued the access_token,- use your Issuer ID string obtained from mano.bank. (see section Exchange required info)
aud	string <= 100 characters **Audience.** A value that identifies mano.bank resources allowed to access by this access_token. In our case its common path to Payments API URI. i.e. for test Env: api-test.mano.bank/payments/v1/
sub	string <= 100 characters **Subject**. the Subject identifies an application / user for which the access token is being issued. In our case, user ID assigned by mano.bank, identifying the Client's application/user.
nbf	string <= 10 characters The "**nbf**" (not before) claim identifies the time before which the JWT MUST NOT be accepted for processing. (usually it equals to **iat** claim value). While the serialization may differ by assertion format, it is REQUIRED that the time be expressed in UTC with no time zone component.
iat	string <= 10 characters **Issued at**. The time at which the access_token was issued. While the serialization may differ by assertion format, it is REQUIRED that the time be expressed in UTC with no time zone component.
exp	string <= 10 characters **Expires at**. The time at which the access_token expires. While the serialization may differ by assertion format, it is REQUIRED that the time be expressed in UTC with no time zone component. **Expires at** value is calculated by adding mano.bank provided JWT maximum token expiration time (usually 60min) to iat value.
jti	string <uuid4> **JWT Id** - The unique identifier of the access_token.

SIGNATURE

The jwt signature part is created from the encoded header and the encoded payload signed with secret (Client Private Key) using the algorithm specified in the header.

For example: in case of RSA SHA256 algorithm, the signature will be created like in this javaScript code snippet:

```
// Set header for JWT
var header = {
          'typ': 'JWT',
          'alg': 'RS256',
         'kid': pm.environment.get('jwt_kid')  //  Client Public key certificate thumbprint
value
};

// Prepare timestamp in seconds
```

```
var currentTimestamp = Math.floor(Date.now() / 1000)

// Set data payload for JWT
var data = {
            'iss': pm.environment.get('jwt_iss'),
            'aud': pm.environment.get('jwt_aud'),
           'sub': pm.environment.get('jwt_sub'),
           'nbf': currentTimestamp,
            'iat': currentTimestamp,
            'exp': currentTimestamp + 30, // expiry time is 30 seconds from time of creation
            'jti': 'jwt_nonce'  // const value is just for demonstration purpose
}

var sHeader = JSON.stringify(header);
var sPayload = JSON.stringify(data);

// build token and sign token
 var signedToken = KJUR.jws.JWS.sign(header.alg, sHeader, sPayload, pri_key);

  pm.environment.set('jwt_signed', signedToken);
```

The signature is used to verify the JWT payload wasn't changed along the way, and to verify that the issuer of the JWT is Client.

Example of JWT Header and Payload

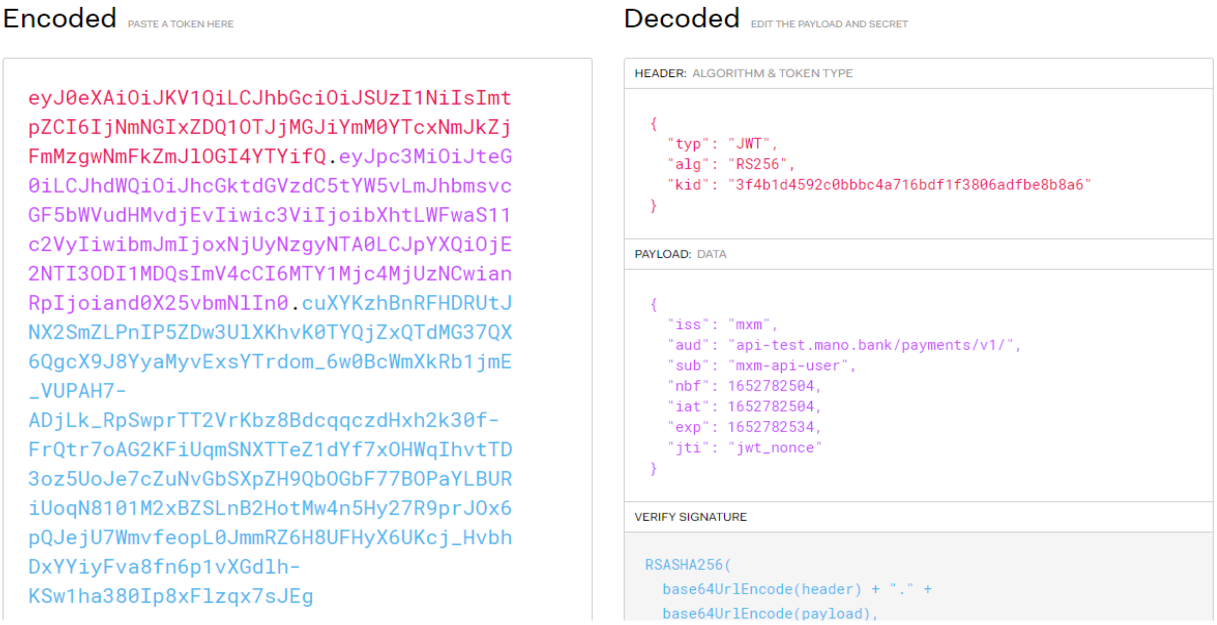
HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "3f4b1d4592c0bbbc4a716bdf1f3806adfbe8b8a6"
}
```

PAYLOAD:DATA

```
{
  "iss": "mxm",
  "aud": "api-test.mano.bank/payments/v1/",
  "sub": "mxm-api-user",
  "iat": 1650869127,
  "exp": 1650872127,
  "jti": "jwt_nonce"
}
```

Calculation Result with encoded and decoded values



Encoded PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImt
pZCI6IjNmNGIxZDQ1OTJjMGJiYmM0YTcxNmJkZj
FmMzgwNmFkZmJlOGI4YTYifQ.eyJpc3MiOiJteG
0iLCJhdWQiOiJhcGktdGVzdC5tYW5vLmJhbmsvc
GF5bWVudHMvdjEvIiwic3ViIjoibXhtLWFwaS11
c2VyIiwibmJmIjoxNjUyNzgyNTA0LCJpYXQiOjE
2NTI3ODI1MDQsImV4cCI6MTY1Mjc4MjUzNCwian
RpIjoiand0X25vbmNlIn0.cuXYKzhBnRFHDRUtJ
NX2SmZLPnIP5ZDw3UlXKhvK0TYQjZxQTdMG37QX
6QgcX9J8YyaMyvExsYTrdom_6w0BcWmXkRb1jmE
_VUPAH7-
ADjLk_RpSwprTT2VrKbz8BdcqqczdHxh2k30f-
FrQtr7oAG2KFiUqmSNXTTeZ1dYf7xOHWqIhvtTD
3oz5UoJe7cZuNvGbSXpZH9QbOGbF77BOPaYLBUR
iUoqN8101M2xBZSLnB2HotMw4n5Hy27R9prJOx6
pQJejU7WmvfeopL0JmmRZ6H8UFHyX6UKcj_Hvbh
DxYYiyFva8fn6p1vXGdlh-
KSw1ha380Ip8xFlzqx7sJEg
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "3f4b1d4592c0bbbc4a716bdf1f3806adfbe8b8a6"
}
```

PAYLOAD: DATA

```
{
  "iss": "mxm",
  "aud": "api-test.mano.bank/payments/v1/",
  "sub": "mxm-api-user",
  "nbf": 1652782504,
  "iat": 1652782504,
  "exp": 1652782534,
  "jti": "jwt_nonce"
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
```

Payment Request Data non-repudiation assurance

Before calling the ***payments/v1/accounts-payment*** endpoint, client must sign HTTP Request to ensure data non-repudiation.

Purpose

To ensure Payment data non-repudiation

- the Client Private key is used by the Client Application to digitally sign *payments/v1/accounts-payment* Request.
- from the other hand mano.bank payment API uses the Client user' certificate to verify received *payments/v1/accounts-payment* Request.

The Request signing process should be completed using the 'Draft Cavage HTTP Signature method' as defined by ietf.org in https://tools.ietf.org/id/draft-cavage-http-signatures-12.html.

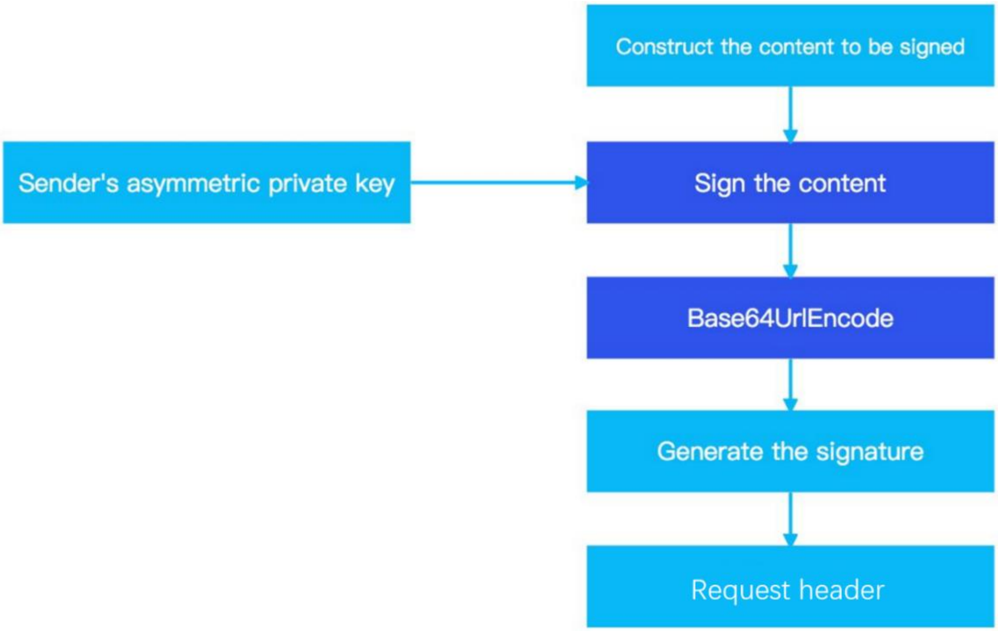
How does it work?

Payment API has a mandatory Signature Header which has the following format

Signature	string **Required** Signed Digest with other headers for non-repudiation Application-level signature of the request by the Client, using its Private X509 key [https://tools.ietf.org/id/draft-cavage-http-signatures-12.html] **Pay attention, instead of standard Base64, mano.bank requires to use Base64UrlEncode**

Sign a request

The following graphic shows an overview of the Request signature creation process:



The signature is sent in the 'Signature' HTTP Header as described in the RFC.

Payment API follows the HTTP Signature spec but with some restrictions as listed below.

- Signature header must have the following structure with these attributes:

```
Signature:
keyId="<thumbprint>",
algorithm="rsa-sha256",
headers="<signature headers>",
signature="<signature code>"
```

- The keyId attribute is the thumbprint number of the Client user X.509 certificate/key pair generated by Client (see section)
- The key size for the used RSA key pair must be at least 2048 bit
- The only allowed algorithm is rsa-sha256
- The headers attribute specifies required list of Request Headers included in Signature:
  - host date (request-target) x-mb-client-id x-mb-user-id request-id content-type digest
  - headers must be listed in lowercase
  - headers must be concatenated in this specified order
  - The request-target is a combination of the HTTP action verb and the request URI path, for example (request-target): post payments/v1/accounts-payment
- The Final signature attribute should be created using following algorithm:

```
signature="Base64UrlEncode(RSA-SHA256(signingString))"
```

Example – Request with Signature header

POST	https://api.mano.bank/payments/v1/accounts-payment

Let’s say we have request with Headers and Body

```
Digest: SHA-256=8VIlqcNA1L8UQfYyXHxJ2NsOYv2jPgA57mnaYrsemxE
Date: Tue, 17 May 2022 10:15:05 GMT
X-MB-Client-Id: mxm
X-MB-User-Id: mxm-api-user
Request-Id: 9e9ad826-df2c-4de6-9a52-ad754ee130bb
Content-Type: application/json
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjNmNGIxZDQ1OTJjMGJiYmM0YTcxNmJkZjFmM
zgwNmFkZmJlOGI4YTYifQ.eyJpc3MiOiJteG0iLCJhdWQiOiJhcGktdGVzdC5tYW5vLmJhbmsvcGF5
bWVudHMvdjEvIiwic3ViIjoibXhtLWFwaS11c2VyIiwibmJmIjoxNjUyNzgyNTA0LCJpYXQiOjE2NTI3O
DI1MDQsImV4cCI6MTY1Mjc4MjUzNCwianRpIjoiand0X25vbmNlIn0.cuXYKzhBnRFHDRUtJNX2SmZ
LPnIP5ZDw3UlXKhvK0TYQjZxQTdMG37QX6QgcX9J8YyaMyvExsYTrdom_6w0BcWmXkRb1jmE_V
UPAH7-ADjLk_RpSwprTT2VrKbz8BdcqqczdHxh2k30f-
FrQtr70AG2KFiUqmSNXTTeZ1dYf7xOHWqIhvtTD3oz5UoJe7cZuNvGbSXpZH9QbOGbF77BOPaYL
BURiUoqN8101M2xBZSLnB2HotMw4n5Hy27R9prJOx6pQJejU7WmvfeopL0JmmRZ6H8UFHyX6U
Kcj_HvbhDxYYiyFva8fn6p1vXGdlh-KSw1ha380Ip8xFlzqx7sJEg
Host: api-test.mano.bank

{
   "referenceId": "142-019",
   "amount": 99.04,
   "currency": "EUR",
   "payerAccountNumber":   "LT945030120000000163",
   "beneficiaryAccountNumber":           "LT897300010152608069",
   "paymentDetailsType":           "UNSTRUCTURED",
   "paymentDetails":       "payment invoice nr.1032-26",
   "beneficiary": {
       "name": "Metal and Metal"
   }
}
```

To calculate Signature header and its components you should:

Create signing string from (request target) and headers – like in this javaScript code snippet:

```
const url = new sdk.Url(resolveVariables(request.url));
const host = url.getHost().toLowerCase();
const path = url.getPathWithQuery();
const method = request.method.toLowerCase();
const date = moment().utc().format("ddd, DD MMM YYYY HH:mm:ss") + " GMT";
const x_client_id = pm.environment.get('client-id');
const x_user_id = pm.environment.get('user-id');
const content_type = pm.environment.get('content-type');
const request_id = guid.v4();
```

```
// prepare signingString from Request Headers
 var signingString =
     `host: ${host}\n` +
     `date: ${date}\n` +
     `(request-target): ${method} ${path}\n` +
     `x-mb-client-id: ${x_client_id}\n` +
     `x-mb-user-id: ${x_user_id}\n` +
     `request-id: ${request_id}` ;

 const digest = calculateDigest();
 signingString += `\ncontent-type: ${content_type}\ndigest: ${digest}`
```

Resulting signingString=

```
host: api-test.mano.bank\n
date: Tue, 17 May 2022 10:15:05 GMT\n
(request-target): post /payments/v1/accounts-payment\n
x-mb-client-id: mxm\n
x-mb-user-id: mxm-api-user\n
request-id: 9e9ad826-df2c-4de6-9a52-ad754ee130bb\n
content-type: application/json\n
digest: SHA-256=8VIlqcNA1L8UQfYyXHxJ2NsOYv2jPgA57mnaYrsemxE
```

Then signature code is signed with Client private key and Base64UrlEncoded, like in provided javaScript snippet below

```
function encryptSignature(signingString) {

  const messageDigest = forge.md.sha256.create();
  messageDigest.update(signingString, "utf8");

  var s = forge.util.encode64(getPrivateKey().sign(messageDigest));

   //base64UrlEncoded
   // Remove padding equal characters
   s = s.replace(/=+$/, '')

   // Replace characters according to base64url specifications
   s = s.replace(/\+/g, '-')
   s = s.replace(/\//g, '_')

   return s;
}
```

The final Request with all Headers and signature code as part of Signature header should look like this example below:

```
Digest: SHA-256=8VIlqcNA1L8UQfYyXHxJ2NsOYv2jPgA57mnaYrsemxE
Date: Tue, 17 May 2022 10:15:05 GMT
X-MB-Client-Id: mxm
X-MB-User-Id: mxm-api-user
```

```
Signature: keyId="3f4b1d4592c0bbbc4a716bdf1f3806adfbe8b8a6",algorithm="rsa-
sha256",headers="host date (request-target) x-mb-client-id x-mb-user-id request-id content-type
digest",signature="izPaXpArD7K3kx-VY5swGbVbBpeaVs02x-
ndsF8v382_0w3kFoxxZ2bdq_awNfTGvBcoWBPkHRn41gfwfUQnnBMdMOov7Na9HViBAZvdmSWLb
q8OEvGJwg_JyEXNwSWrxIMsFu2xQz7q_y_iKdraU_wKHXxp_4qSKOCmjS8jn9rTzhf8qc5vX0BntSPa
0hmJy2Y-QbVxqEWFZjy7QHSxEXLdLsK7NKdA3hLLCcBxHcDofX34G0F1yxfZ_YWzil6m8e-
tQGhdvMqC665pnLUMAN_qvpA_X0LNGxQe6Vf-ngFeMM-
ictZuDaV7FYOXkjEQ7j1bRbeiuX66UiyhgXEo5Q"
Request-Id: 9e9ad826-df2c-4de6-9a52-ad754ee130bb
Content-Type: application/json
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjNmNGIxZDQ1OTJjMGJiYmM0YTcxNmJkZjFmMz
gwNmFkZmJlOGI4YTYifQ.eyJpc3MiOiJteG0iLCJhdWQiOiJhcGktdGVzdC5tYW5vLmJhbmsvcGF5bW
VudHMvdjEvIiwic3ViIjoibXhtLWFwaS11c2VyIiwibmJmIjoxNjUyNzgyNTA0LCJpYXQiOjE2NTI3ODI1MD
QsImV4cCI6MTY1Mjc4MjUzNCwianRpIjoiand0X25vbmNlIn0.cuXYKzhBnRFHDRUtJNX2SmZLPnIP5Z
Dw3UlXKhvK0TYQjZxQTdMG37QX6QgcX9J8YyaMyvExsYTrdom_6w0BcWmXkRb1jmE_VUPAH7-
ADjLk_RpSwprTT2VrKbz8BdcqqczdHxh2k30f-
FrQtr70AG2KFiUqmSNXTTeZ1dYf7xOHWqIhvtTD3oz5UoJe7cZuNvGbSXpZH9QbOGbF77BOPaYLB
URiUoqN8101M2xBZSLnB2HotMw4n5Hy27R9prJOx6pQJejU7WmvfeopL0JmmRZ6H8UFHyX6UKcj_
HvbhDxYYiyFva8fn6p1vXGdlh-KSw1ha380Ip8xFlzqx7sJEg
User-Agent: PostmanRuntime/7.29.0
Accept: */*
Postman-Token: e01ca5e4-8015-48a9-b3b5-8083792a93f7
Host: api-test.mano.bank
{
   "referenceId": "142-019",
   "amount": 99.04,
   "currency": "EUR",
   "payerAccountNumber":   "LT945030120000000163",
   "beneficiaryAccountNumber":           "LT897300010152608069",
   "paymentDetailsType":          "UNSTRUCTURED",
   "paymentDetails":       "payment invoice nr.1032-26",
   "beneficiary": {
       "name": "Metal and Metal"
   }
}
```

In case of successful Request processing, you should get the following Response (see Payment API specification for more details: https://developers.mano.bank/api-details#api-payments-api&operation=payment)

```
Content-Type: application/json
{
   "operationId": "00008355",
   "status": "CONFIRMED",
   "metadata": {
      "responseId": "8e6bdaec-8bec-4cc8-894b-f77c34dfbaaf",
      "correlationId": "3a10a79a-f1a1-4e69-a8d4-b866e6561aa1",
      "hasErrorMessage": false,
      "messages": []
   }
}
```