THE Berlin GROUP

A EUROPEAN STANDARDS INITIATIVE

openFinance API Framework

Implementation Guidelines

Protocol Functions and Security Measures

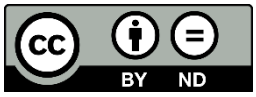
Version 2.1

31 July 2024

License Notice

This Specification has been prepared by the Participants of the openFinance Taskfoce*. This Specification is published by the Berlin Group under the following license conditions:

- "Creative Commons Attribution-NoDerivatives 4.0 International Public License"

  This means that the Specification can be copied and redistributed in any medium or format for any purpose, even commercially, and when shared, that appropriate credit must be given, a link to the license must be provided, and indicated if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. In addition, if you remix, transform, or build upon the Specification, you may not distribute the modified Specification.

- Implementation of certain elements of this Specification may require licenses under third party intellectual property rights, including without limitation, patent rights. The Berlin Group or any contributor to the Specification is not, and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.
- Any right, title and interest in and to the copyright and all related rights in topic-related Scheme Rulebooks, belong to the respective Scheme Manager (amongst others, the European Payments Council AISBL - EPC).
- The Specification, including technical data, may be subject to export or import regulations in different countries. Any user of the Specification agrees to comply strictly with all such regulations and acknowledges that it has the responsibility to obtain licenses to export, re-export, or import (parts of) the Specification.

* The openFinance Taskforce brings together participants of the Berlin Group with additional European banks (ASPSPs), banking associations, payment associations, payment schemes and interbank processors.

Contents

1 Introduction

1.1 From Core XS2A Interface to openFinance API

With [PSD2] the European Union has published a directive on payment services in the internal market. Among others [PSD2] contains regulations on services to be operated by socalled Third Party Payment Service Providers (TPP) on behalf of a Payment Service User (PSU). These services are

- Payment Initiation Service (PIS) to be operated by a Payment Initiation Service Provider (PISP) TPP as defined by article 66 of [PSD2],
- Account Information Service (AIS) to be operated by an Account Information Service Provider (AISP) TPP as defined by article 67 of [PSD2], and
- Confirmation on the Availability of Funds Service (FCS) to be used by a Payment Instrument Issuing Service Provider (PIISP) TPP as defined by article 65 of [PSD2].

To implement these services (subject to PSU consent) a TPP needs to access the account of the PSU. The account is managed by another PSP called the Account Servicing Payment Service Provider (ASPSP). To support the TPP in accessing the accounts managed by an ASPSP, each ASPSP has to provide an "access to account interface" (XS2A interface). Such an interface has been defined in the Berlin Group NextGenPSD2 XS2A Framework.

This XS2A Framework now has been opened up to extended services. This interface is addressed in the following as **openFinance API**. This openFinance API differs from the XS2A interface in several dimensions:

- The extended services might not rely anymore solely on PSD2.
- Other important regulatory frameworks which apply are e.g. GDPR.
- The openFinance API can address different types of **API Clients** as access clients, e.g. TPPs regulated by an NCA according to PSD2, or corporates not regulated by an NCA.
- The extended services might require contracts between the access client and the ASPSP.
- While the client identification at the openFinance API can still be based on eIDAS certificates, they do not need to be necessarily PSD2 compliant eIDAS certificates.
- The extended services might require e.g. the direct involvement of the access client's bank for KYC processes.

Note: The notions of API Client and ASPSP are used because of the technical standardisation perspective of the openFinance API. These terms are analogous to "asset broker" and "asset holder" resp. in the work of the ERPB on a SEPA API access scheme.

Note: In implementations, the API services of several ASPSPs might be provided on an aggregation platform. Such platforms will be addressed in the openFinance API Framework as "API provider".

Note: At some places, the openFinance API Framework is still using the technical TPP notion also for the openFinance APIs, e.g. in redirection related header parameters. In this context, the TPP is not necessarily a licensed TPP as mentioned above, but for certain services may just be any "third party provider".

The following account access methods are covered by this framework:

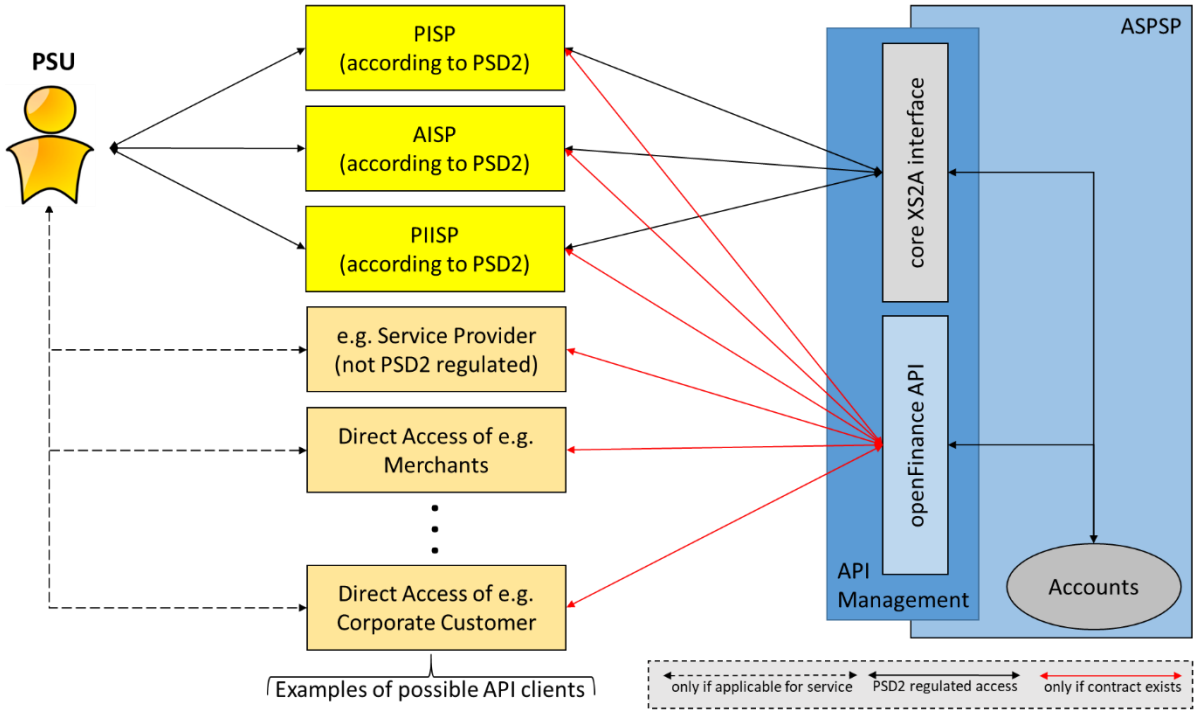


Figure 1: Core XS2A interface and openFinance API

The ASPSP may restrict the access to the services offered at its openFinance API and require dedicated onboarding. The requirements for the rights to access to services offered at the openFinance API are out of scope of this document. These requirements are in detail in [oFA-IG-ADM].

In contrast to the services of the openFinance API the ASPSP has to offer access to the services of his core XS2A API to any TPP without any discrimination as long as the TPP has got the necessary licence to access a service as a PISP, AISP or PIISP from an NCA according to the regulations of [PSD2].

1.2 Protocol Functions supported by XS2A and openFinance API

The addressed services within the XS2A API and the openFInance API share many protocol functions. Such functions are defined in this Framework document and cover e.g.

- dynamic protocol steering by hyperlinks,
- how to address transaction authorisation resource,
- how to push resource status changes to API clients or
- how to handle protocol preferences of the API Clients.

Such generic protocol functions are explained only within this document. Service specifications using these functions then just refer to this openFinance API Framework documentation for better readability. For better readability, the document refers in most parts to the openFinance API Framework only instead of addressing both, the XS2A API and the openFinance API. Where needed, the two APIs are addressed separately.

1.3 Security Measures supported by XS2A and openFinance API

Specific API protocol functions are security measures to be supported by an API. XS2A and openFinance API according to the specifications of the Berlin Group support different security measures and approaches for the authentication of a PSU. Examples are the authentication of an API Client either at the transport level or the support of different approaches for the execution of a strong customer authentication of a PSU.

Some of these security measures and approaches for PSU authentication have to be supported by an API mandatorily, for other the support is an option for the ASPSP. In any case an API Client has to secure his access to an XS2A or openFinance API by a security measure or has to use a special approach for PSU authentication if requested by an ASPSP.

This document describes how different security measures and approaches for PSU authentication are supported by an XS2A or openFinance API. For each it is defined if the support at an XS2A API and/or at an openFinance API is mandatory (i.e. has to be supported by each ASPSP) or optional (i.e. the ASPSP can decide to support it or not). If special restrictions for values, parameters, variants, etc. for the support of a security measure exist, these are also described.

Please notice that this document contains only the specification of how security measures are supported by an API, but not the specification of the security measures itself.

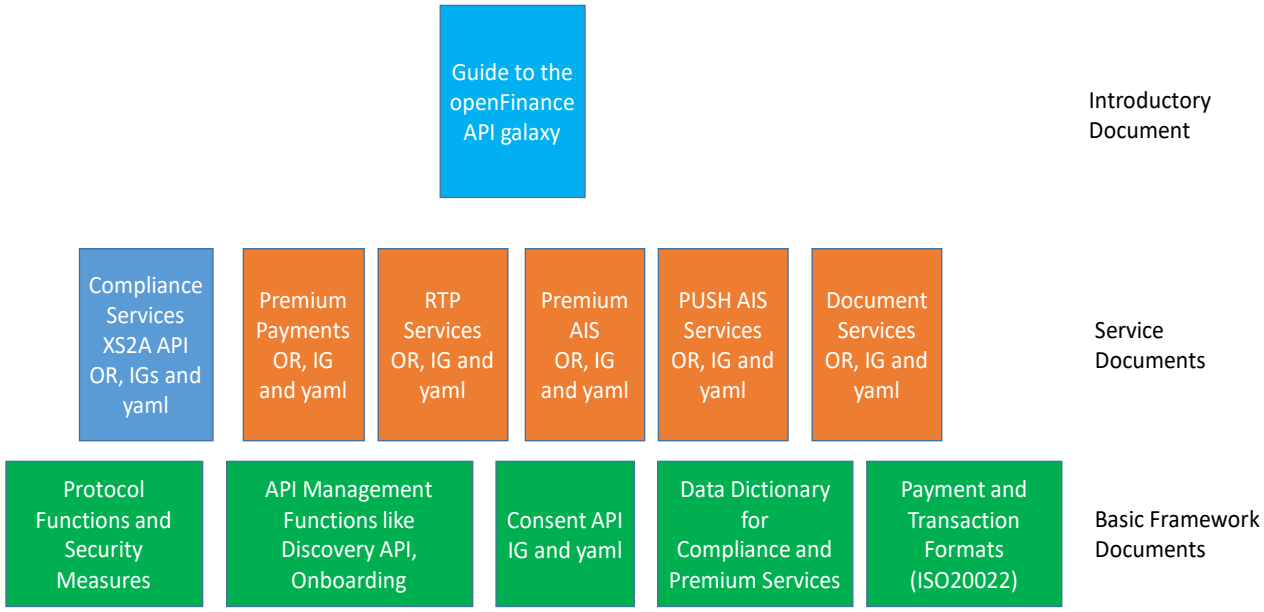
1.4 Document Structure

This document specifies the Protocol Functions and Security Measures supported by an XS2A or openFinance API (if based on one of the Berlin Group specifications) in detail. It has to be

considered in combination with any other Implementation Guidelines published by the Berlin Group. This document defines the following building bricks:

- Character sets and primitive data types supported by all services in Section 2.1
- Notations used within all services definitions and in the openFinance API Framework documentation in Section 2.2
- REST API principles within the openFinance API Framework in Section 3
- Error handling in Section 4
- Authentication of API Clients and ASPSP at transport layer in Section 5
- Signing messages on application layer in Section 6
- Signing message bodies in Section 7.1
- Encryption of message bodies in Section 7.2
- SCA Approaches supported in Section 8
- Authorisation processes supported in Section 9
- Signing basket function to group transactions for authorisation in Section 10
- Resource status notification function (formerly called Lean Push Service) in Section 11

This document fits into the overall structure of the documentation as follows:



1.5 Document History

Version	Change/Note	Approved
Version 2.0	Initial Version of the openFInance API Framework documentation.	2023-10-5 openFinance TF
Version 2.1	Enhancing the Framwork by the SCA approach "ASPSP-Channels" when using asynchronous SCA procedures via e.g. the ASPSP online channels. Renaming some authorisation related HTTP headers to prepare the API Framework for direct access for corporates. Extending the embedded and decoupled SCA approach by an option to add the debtor account during the authorisation process, via an account selection process. Extending message signing routines by headers protecting the resources addressed. Errata and clarifications.	

Version	Change/Note	Approved
	Details are covered in the Annex of this document.	

2 Character Sets and Notations

This section introduces character sets and notations used throughout the openFinance API Framework.

2.1 Character Set and Data Types

HTPP body parameters

The character set within the openFinance API Framework for parameters transported in HTTP bodies is UTF 8 encoded. This specification is only using the basic data elements "String", "Boolean", "ISODateTime", "ISODate", "UUID" and "Integer" (with a byte length of 32 bits) and ISO based code lists. For codes defined by ISO, a reference to the corresponding ISO standard is given on service specification or data dictionary level.

Max35Text, Max70Text, Max140Text Max500Text, Max1000Text, Max2000Text are defining strings with a maximum length of 35, 70, 140, 500, 1000 and 2000 characters respectively.

ASPSPs will accept for strings at least the following character set:

a b c d e f g h i j k l m n o p q r s t u v w x y z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

/ - ? : ( ) . , ' +

Space

ASPSPs may accept further character sets for text fields like names, addresses, text. Corresponding information will be contained in the ASPSP documentation of the related service. ASPSPs might convert certain special characters of these further character sets, before forwarding e.g. submitted payment data.

Complex data types and code lists are defined in the data dictionary [oFA DD] or in related service specifications.

HTTP query and header parameters

Encoding follows the HTTP specification. Specific additions might be provided for HTTP parameters introduced specifically by this specification.

2.2 Notation

2.2.1 Notation for Requests

For API request calls, query parameters, HTTP header parameters and body content parameters are specified within this openFinance API Framework as follows:

Attribute	Type	Condition	Description
attribute tag	type of attribute	condition	description of the semantic of the attribute and further conditions.

The "Type" is referring to either basic or complex data types as introduced in Section 2.1.

The following conditions may be used when describing data to be submitted by the client:

- Optional: The attribute is supported by the server, usage is optional for the client. The server may ignore the parameter if mentioned in the “Description” column of the table above.
- Conditional: The attribute is supported by the server and might be mandated by
  - the server provider in its own documentation of the support of the related API or
  - by certain rules as defined in the “Description” column of the table above.
- Mandatory: The attribute is supported by the server and shall be used by the client.
- Optional if supported by API provider: It is optional for the server to support this attribute. If the server is supporting the attribute as indicated in its own documentation of the related service, it might be used by the client optionally. If the server is not supporting the attribute, then the request is rejected when it is contained.

Remark: Please note that the conditions “Optional if supported by API provider” is used rarely in the openFinance API Framework.

2.2.2 Notation for Responses

For API call responses, parameters, HTTP header parameters and body content parameters are specified within the openFinance API Framework as follows:

Attribute	Type	Condition	Description
attribute tag	type of attribute	condition	description of the semantic of the attribute and further conditions.

The "Type" is referring to either basic or complex data types as introduced in Section 2.1.

The following conditions can be set on data to be provided by the server:

- Optional: The attribute is supported optionally by the server
- Conditional: The attribute is supported by the server under certain conditions as indicated in the “Description” column of the table above.
- Mandatory: The attribute is always supported by the server.

2.2.3 Notations used for Requests as well as Responses

The following additional conditions apply to both, requests from the client to the server as well as responses from the server to the client:

Attribute	Type	Condition	Description
		{Or	
		Or	
		Or}	
		{Or – Optional	
		Or – Optional	
		Or – Optional}	

- {Or: The **first** element in a sequence of elements of which **exactly one** has to be included.
- Or: An element in a sequence of elements of which **exactly one** has to be included. The element is **neither the first nor the last** within this sequence.
- Or}: The **last** element in a sequence of elements of which **exactly one** has to be included.
- {Or – Optional: The **first** element in a sequence of elements of which **at most one** may be included.
- Or – Optional: An element in a sequence of elements of which **at most one** may be included. The element is **neither the first nor the last** within this sequence.
- Or – Optional}: The **last** element in a sequence of elements of which **at most one** may be included.

Note: Specifications for the openFinance API are accompanied by an openAPI interface description (as a yaml file). Within these openAPI descripitions, elements with conditions "{Or", "Or", "Or}", "{Or – Optional", "Or – Optional", "Or – Optional}" will be treated as (pure) optional

elements at the moment. It is the responsibility of the implementer to ensure the additional checks.

2.2.4 Base64 Notations

Base64 encoding:

> Base64 encoding according to [RFC4648].

Base64url encoding:

> According to [RFC7515] base64 encoding using the URL- and filename-safe character set defined in Section 5 of [RFC4648], with all trailing '=' characters omitted (as permitted by Section 3.2) and without the inclusion of any line breaks, whitespace, or other additional characters. Note that the base64url encoding of the empty octet sequence is the empty string. (See Appendix C of [RFC7515] for notes on implementing base64url encoding without padding.)

2.2.5 Notion of a Transaction

Transaction is used in two different contexts within this document respectively the openFinance API Framework:

- "Transaction" as meta description of a service interaction with the API to simplify the definitions in this document:
  - The notion of a transaction subsumes service interactions like payment initiation, consents for API interaction, subscriptions or signing baskets.
  - Such transactions are started by the API Client with a related initiation request, called throughout this document "Transaction Initiation Request", e.g. a
    - Payment Initiation Request,
    - Establish Consent Request,
    - Establish Signing Basket Request, or
    - Initiate Subscription Request.
  - The response to these initiations by the ASPSP is called "Transaction Initiation Response" throughout this document.
  - Transactions need normally to be authorised by the PSU by applying a Strong Customer Authentication (SCA).
  - Where transaction data is posted to the API, a related resource is created.

- Transaction as a banking transaction on an account:
    - Entries on an account are also addressed as transactions.

2.2.6 Notation for Access Methods

All service definitions in the openFinance API Framework will come with an overview on the http access methods supported.

These access methods are denoted by the following tables:

Endpoints	Method	Condition	Description
service-endpoint	POST or GET or PUT or DELETE	Mandatory or Optional or Conditional	Description and reference to the section, where the related access point is

The following definitions apply for such tables throughout the openFinance API Framework documentation:

- Endpoints: This is the endpoint name.
- Method: This is the HTTP method to be applied. The HTTP methods POST; GET PUT and Delete are supported.
- Condition: The condition defines the condition for the ASPSP.
    - Mandatory: The access method shall be provided by the ASPSP
    - Optional: The access method may be provided by the ASPSP
    - Conditional: The access method shall be provided given certain conditions defined in the Description column.

    Please note that the condition is given relative to the parent node of the path, i.e. the condition e.g. on a method on /service-endpoint/{resourceId} applies only if the endpoint /service-enpoint is supported at all.
- Description: Gives an overview on the method and potentially conditions for the method as well as a reference to the section in the document, where the method is described in detail.

3 REST API Approach: Guiding Principles

This section introduces some basic technical rules on how to implement the REST API approach within the openFinance API Framework.

3.1 Location of Message Parameters

The openFinance API Framework definitions follow the REST service approach. This approach allows to transport message parameters at different levels:

- message parameters as part of the HTTP level (HTTP header)
- message parameters by defining the resource path (URL path information) with additional query parameters and
- message parameters as part of the HTTP body.

The content parameters in the corresponding HTTP body will be encoded either in JSON or in XML syntax. XML syntax is only used where

- an ISO 20022 based payment initiation (pain.001 message) with the corresponding payment initiation report (pain.002 message) or
- ISO 20022 based account information message (camt.052, camt.053 or camt.054 message)

is contained.

As an exception, response messages might contain plain text format in account information messages to support MT940, MT941 or MT942 message formats related to transaction reports.

The parameters are encoded

- in spinal-case (small letters) on path level,
- in spinal-case (starting capital letters) on HTTP header level and
- in lowerCamelCase for query parameters and JSON based content parameters.

The following principle is applied when defining the API:

Message parameters as part of the HTTP header:

- Definition of the content syntax,
- Certificate and Signature Data where needed,
- PSU identification data (the actual data from the online banking frontend or access token),

- Protocol level data like Request Timestamps, Request/Transaction Identifiers or protocol preferences of the API Client.

Message parameters as part of the path level:

- All data addressing a resource:
  - Provider identification,
  - Service identification,
  - Payment product identification,
  - Account Information subtype identification,
  - Resource ID

Query Parameters:

- Additional information needed to process the GET request for filtering information,

Message parameters as part of the HTTP body:

- Business data content,
- PSU authentication data,
- Messaging Information
- Hyperlinks to steer the full TPP – ASPSP process

3.2 API Structure

The XS2A Interface resp. openFinance API is resource oriented. Resources can be addressed under the API endpoints

`https://{provider}/v2/{service}{?query-parameters}`

using additional content parameters {parameters}

where

- {provider} is the host and path of the related API, which is not further mentioned. The host or path may contain release version information of the ASPSP.
- v2 is denoting the final version 2.x of the related implementation guidelines within the openFinance API Framework.

NOTE: The addressed exact version of the Implementation Guidelines of the addressed service will be provided in a dedicated response header, cp. Section 3.3.

- {service} has the values consents, payments, bulk-payments, periodic-payments, accounts, card-accounts, signing-baskets or funds-confirmations for the XS2A Interface eventually extended by more information on product types like resource identification and request scope. For the openFinance API, more services are offered as provided in the extended service specifications of the openFinance API Framework.

  Remark: It is strongly recommended that the related resourceId for identifying a dedicated resource is a UUID.

- {?query-parameters} are parameters detailing GET based access methods, e.g. for filtering content data

- {parameters} are content attributes defined in JSON or XML and transported in the http body encoding according to the following

  - XML encoding appears only when ISO 20022 pain.001 messages are transported when demanded by the ASPSP for the corresponding payment product

  - all other request bodies are encoded in JSON

Resources

The structure of the request/response is described according to the following categories

- Path: Attributes encoded in the Path, e.g. "payments/sepa-credit-transfers" for {resource}

- Query Parameters: Attributes added to the path after the "?" sign as process steering flags or filtering attributes for GET access methods. Query parameters of type Boolean shall always be used in a form query-parameter=true or query-parameter=false.

- Header: Attributes encoded in the HTTP header of request or response

- Request: Attributes within the content parameter set of the request

- Response: Attributes within the content parameter set of the response, defined in XML, text or JSON:

  - XML encoding appears only, when camt.052, camt.053 or camt.054 messages (reports, notifications or account statements) or pain.002 payment status messages are transported. pain.002 messages will only be delivered for the GET Status Request, and only in cases where the payment initiation was performed by using pain.001 messages.

  - Text encoding appears only, when MT940, MT941 or MT942 messages (reports, notifications or account statements) are transported.

  - All other response bodies are encoded in JSON.

The HTTP response codes which might be used in the openFinance API Framework are specified in Section 3.8. This is not repeated for every API call definition.

> **Remark:** For JSON based responses, this specification defines body attributes which are responded from ASPSP to API Clients following POST or PUT API calls. The ASPSP is free to return the whole addressed resource within the response, following usual REST methodologies.

3.3 API Versioning

For calls on a dedicated service, the ASPSP might use the following HTTP header in responses:

Attribute	Type	Condition	Description
X-Reference-API-Name	String	Optional	"Berlin Group openFinance API"
X-Reference-API-Document	String	Optional	The name of the Implementation Guideline document, where the service is based on, e.g. "Extended Payment Initiation Services".
X-Reference-API-Version	String	Optional	This is the exact version number of the openFinance API Framework Implementation Guidelines of the related service which is implemented, e.g. "2.1.3".

Note: The usage of these header is recommended to use at least on root endpoints for an addressed service. It should be used independently of the http response code.

3.4 API Request Header Parameter

The following headers are defined generically within this document and will be instantiated to service definitions explicitly only via the related OpenAPI files. Such lean documentation is intended to enhance the readability of the service definitions.

3.4.1 Transaction Initiation Request Headers

Transaction Initiation Request messages (cp. Section 2.2.5 for a definition) are specific within the openFinance API Framework. These initiation messages for a business transaction not only transport business related information, but also context data and data steering the usage of openFinance API Framework functions. These headers are defined within this document and will be instantiated to service definitions explicitly only via the related OpenAPI files. Such lean documentation is intended to enhance the readability of the service definitions.

The openFinance API Framework foresees the following header parameters for Transaction Initiation Request messages

- PSU context data, cp. Section 8.2

- PSU identification data, cp. Section 8.3
- SCA Approach Preferences, cp. Section 8.4
- API Client Notification data, cp. Section 11
- Client Brand Information Header, see Section 3.4.2

The usage of the headers mentioned above might lead to the usage of specific header parameters in the Transaction Initiation Response, as defined in the sections above.

The same headers (besides the Client Brand Information Header) also apply to Payment Cancellation Request messages.

3.4.2 Client Brand Information Header

The following header may be used within all Transaction Initiation Requests:

Attribute	Type	Condition	Description
Client-Brand-Logging-Information	String	Optional	This header might be used by API Clients to inform the ASPSP about the brand used by the API Client towards the PSU. This information is meant for logging entries to enhance communication between ASPSP and PSU or ASPSP and API Client. This header might be ignored by the ASPSP.

3.4.3 Signature related headers

In addition, the following header parameters might be used in all API request headers, where applicable:

- Signature related headers, cp. Section 6

3.4.4 Technical headers

The http requests come with certain technical headers, which are not specifically addressed in this specification. One of these headers is the "date" header which is typically added automatically on http protocol level by the API Client system.

Please further note that the http message signing procedure defined in Section 6 specifies a dedicated timestamp for the openFinance API Framework which is protected itself by signing.

3.5 Header Parameters for Idempotency

All requests **and** responses in this API framework come with the unique X-Request-Id, a UUID in requests and responses. The X-Request-Id in the response equals the X-Request-Id in the related request.

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.

The uniqueness implies the following:

A request B with the same X-Request-Id as request A should be responded as follows to achieve idempotency:

- If the business content of request B (e.g. path, query parameters, body and headers defined in this specification) equals the business content of request A and if the time deviation is not too high (implementation/request specific) and if potentially the internal state of the related resource has not changed in the meantime: The response message from the server to request B then also should equal the response message to request A.
- Otherwise the requested message should be rejected with a 40x response code.

3.6 API Steering Process by Hyperlinks (HATEOAS)

The services defined within the openFinance API Framework require several requests from the API Client towards the ASPSP. With the Transaction Initiation Request specified in each service, a resource presentation is generated by the ASPSP for the related business transaction, e.g. a payment resource or a consent resource for AIS access. The location header of the response will usually contain a link to the created resource.

In addition, the ASPSP can embed a hyperlink together with a "tag" for the semantics of this hyperlink into the response to these first requests and to all succeeding requests within the services. This hyperlink must be a URI reference as defined in [RFC3986] and can be either a relative link, which is recommend to save space, for the host starting e.g. with "/psd2/v2/payments/sepa-credit-transfers" or it can be a global link like https://www.testbank.com/psd2/v2/payments/sepa-credit-transfers/asdf-asdf-asdf-1234.

The global links might be needed in some circumstances, e.g. a re-direct. The tag of the hyperlink transports the functionality of the resource addressed by the link, e.g. "authorise-transaction". This link indicates that results of a SCA method are to be posted to the resource addressed by this link to authorise e.g. a payment.

The steering hyperlinks are transported in the "_links" data element of type Link, cp. the related data dictionary. It may contain one or several hyperlinks. Services might define additional

hyperlinks before they are integrated mid-term to the overall data dictionary within the openFinance API Framework maintenance.

The "_links" data element may contain more hyperlinks than specified in the related call. In this case, this will be documented in the ASPSP's API documentation or the hyperlinks can be ignored by the API Client.

Some hyperlinks might require additional data in the same response body which are then needed when following this hyperlink. The following table gives an overview on these specific steering hyperlinks to explain interconnection with the data elements.

Hyperlink	**Additional Link Related Data**	**Description**
startAuthorisationWith PsuAuthentication	(challengeData)	The link to an endpoint where the authorisation of a transaction or of a transaction cancellation shall be started, where PSU authentication data shall be uploaded with the corresponding call. Remark: In rare cases the ASPSP will ask only for some dedicated ciphers of the passwords. This information is then transported to the TPP by using the "challenge" data element, normally used only in SCA context.
startAuthorisationWith EncryptedPsuAuthentication	(challengeData)	Same as startAuthorisactionWith PsuAuthentication, but password is encrypted on application layer when uploaded.
updatePsuAuthentication	(challengeData)	The link to the authorisation sub resource, which needs to be updated by a PSU password and eventually the PSU identification if not delivered yet. **Remark**: In rare cases the ASPSP will ask only for some dedicated ciphers of the passwords. This information is then transported to the TPP by using the "challenge" data element, normally used only in SCA context.
updateEncryptedPsu Authentication	(challengeData)	Same as updatePsuAuthentication, but password is encrypted on application layer when uploaded.

Hyperlink	Additional Link Related Data	Description
startAuthorisationWith AuthenticationMethodSelection	scaMethods	This is a link to and endpoint where the authorisation of a transaction or of a transaction cancellation shall be started, where the selected SCA method shall be uploaded with the corresponding call.
selectAuthenticationMethod	scaMethods	This is a link to a resource, where the TPP can select the applicable strong customer authentication methods for the PSU, if there were several available authentication methods.
authoriseTransaction	challengeData, chosenScaMethod	A link to the resource, where a "Transaction Authorisation Request" can be sent to. This request transports the result of the SCA method performed by the customer, generating a response to the challenge data.
startAuthorisationWith TransactionAuthorisation	challengeData, chosenSCAMethod	A link to an endpoint, where an authorisation of a transaction or a cancellation can be started, and where the response data for the challenge is uploaded in the same call for the transaction authorisation or transaction cancellation at the same time in the Embedded SCA Approach.

3.7 Links in Transaction Initiation Response

The Transaction Initiation Response message may contain many hyperlinks to indicate the next steps to be taken to authorise the transaction. These hyperlinks are independent of the actual transaction initiated. This is why they are defined once here generically and are not repeated on Implementation Guideline level, but only in related Open API files. The same applies to response messages of other transactions which might require authorisation, e.g. a Payment Cancellation Request.

Attribute	Type	Condition	Description
_links	Links	Mandatory	A list of hyperlinks to be recognised by the API Client. The actual hyperlinks used in the response depend on the dynamical decisions of the ASPSP when processing the request. **Remark**: All links can be relative or full links, to be decided by the ASPSP. Type of links admitted in this response, (further links might be added for ASPSP defined extensions): "scaRedirect": In case of an SCA Redirect Approach, the ASPSP is transmitting the link to which to redirect the PSU browser. "startAuthorisationWithPsuIdentification": The link to the authorisation end-point, where the authorisation sub-resource has to be generated while uploading the PSU identification data. "startAuthorisationWithPsuAuthentication": The link to the authorisation end-point, where the authorisation sub-resource has to be generated while uploading the PSU authentication data. "startAuthorisationWithEncryptedPsuAuthentication": Same as startAuthorisactionWithPsuAuthentication, but the authentication data need to be encrypted on application level while uploading. "startAuthorisationWithAuthentication MethodSelection": The link to the authorisation end-point, where the authorisation sub-resource has to be generated

Attribute	Type	Condition	Description
			while selecting the authentication method. This link is contained under exactly the same conditions as the data element "scaMethods"
			"startAuthorisationWithTransactionAuthorisation": The link to the authorisation end-point, where the authorisation sub-resource has to be generated while authorising the transaction e.g. by uploading an OTP received by SMS.
			"self": The link to the transaction initiation resource created by this request. This link can be used to retrieve the resource data. "status": The link to retrieve the transaction status of the transaction initiated. "scaStatus": The link to retrieve the scaStatus of the corresponding authorisation sub-resource. This link is only contained, if an authorisation sub-resource has been already created.

3.8 HTTP Response Codes

The HTTP response code is communicating the success or failure of an API Client request message. The 4XX HTTP response codes should only be given if the current request cannot be fulfilled, e.g. a payment initiation cannot be posted or account transactions cannot be retrieved. For further details on error handling cp. Section 4. A request to get the status of a transaction usually returns HTTP response code 200 since the actual request to retrieve the status succeeded, regardless if that transaction state is set to failure or not.

The openFinance API Framework supports the following HTTP response codes:

Status Code	Description
200 OK	PUT, GET Response Codes This return code is permitted if a request was repeated due to a time-out. The response in that might be either a 200 or 201 code depending on the ASPSP implementation. The POST for a Funds request will also return 200 since it does not create a new resource. DELETE Response Code where a payment resource has been cancelled successfully and no further cancellation authorisation is required.
201 Created	POST response code where a Transaction Request was correctly performed and a related resource was created, which is addressable further on by the API Client.
202 Accepted	DELETE response code, where a transaction resource, e.g. a payment resource can be cancelled in general, but where a cancellation authorisation is needed in addition.
204 No Content	DELETE response code where a transaction resource, e.g. a consent resource was successfully deleted. The code indicates that the request was performed, but no content was returned.
400 Bad Request	Validation error occurred. This code will cover malformed syntax in request or incorrect data in payload.
401 Unauthorized	The TPP or the PSU is not correctly authorized to perform the request. Retry the request with correct authentication information.
403 Forbidden	Returned if the resource that was referenced in the path exists but cannot be accessed by the API Client or the PSU. This code should only be used for non-sensitive id references as it will reveal that the resource exists even though it cannot be accessed.

Status Code	Description
404 Not found	Returned if the resource or endpoint that was referenced in the path does not exist or cannot be referenced by the API Client or the PSU. When in doubt if a specific id in the path is sensitive or not, use the HTTP response code 404 instead of the HTTP response code 403.
405 Method Not Allowed	This code is only sent when the HTTP method (PUT, POST, DELETE, GET etc.) is not supported on a specific endpoint. It has nothing to do with the related transaction data model. DELETE Response code in case of cancellation of a transaction resource, e.g. a payment resource, where the related transaction, e.g. a payment initiation cannot be cancelled due to legal or other operational reasons.
406 Not Acceptable	The ASPSP cannot generate the content that the API Client specified in the accept header.
408 Request Timeout	The server is still working correctly, but an individual request has timed out.
409 Conflict	The request could not be completed due to a conflict with the current state of the target resource.
415 Unsupported Media Type	The API Client has supplied a media type which the ASPSP does not support.
429 Too Many Requests	The API Client has exceeded the number of requests allowed by the consent or by the RTS.
500 Internal Server Error	Internal server error occurred.
503 Service Unavailable	The ASPSP server is currently unavailable. Generally, this is a temporary state.

3.9 Data Extensions by the ASPSP

The ASPSP might add more data attributes to response messages. Such extensions then shall be documented in the ASPSP's documentation of its XS2A interface. These data attributes can be either ignored by the API Client or can be interpreted as defined by the above mentioned documentation.

The ASPSP might add additional optional data attributes to be submitted, e.g. for setting up additional services. In addition, an ASPSP can ask the API Client for a submission of proprietary data in a second step via the "proprietaryData" hyperlink. This shall be published by the ASPSP in its documentation.

Remark: Before defining these additional proprietary data elements, the ASPSP is requested to submit the attribute description to the Berlin Group openFinance Taskforce, where it will be decided on a standardised approach for the related data attributes.

3.10 Multicurrency Accounts

Definition: A multicurrency account is an account which is a collection of different sub-accounts which are all addressed by the same account identifier like an IBAN by e.g. payment initiating parties. The sub-accounts are legally different accounts and they all differ in their currency, balances and transactions. An account identifier like an IBAN together with a currency always addresses uniquely a sub-account of a multicurrency account.

This specification supports to address multicurrency accounts either on collection or on sub-account level. The currency data attribute in the corresponding data structure "Account Reference" allows to build structures like

```
{"iban": "DE40100100103307118608"}
```

or

```
{"iban": "DE40100100103307118608",
 "currency": "EUR"}
```

If the underlying account is a multicurrency account, then

- the first reference is referring to the collection of all sub-accounts addressable by this IBAN, and
- the second reference is referring to the euro sub-account only.

This interface specification is acting on sub-accounts of multicurrency accounts in exactly the same way as on regular accounts. This applies to payment initiation as well as to account information.

3.11 Interval Borders Including Rules

Data attributes for dates, amounts or other discrete data come sometimes with the suffix "To" or "From", e.g. "dateTo", "validTo" or "baseAmountFrom". These attributes always define intervals. In **all** such cases, the "From" and "To" is including the value at the addressed border of the addressed interval. That means, that e.g. "dateFrom": "2023-04-05" means, that the 5th Aprill 2023 is included in the addressed interval.

4 Error Handling

This section introduces the error handling in the openFinance API Framework. API data structures as defined here will not be repeated in service specifications. Instantiations to the related services will only be provided via the related OpenAPI files.

4.1 Responses in Error Cases

In order to achieve a better readability, the service definitions in the openFinance API Framework specify responses in case of a positive processing result only. The following section gives specific rules for the case of a negative processing result applicable to all services.

4.1.1 Header

In general, the same rules regarding the presence of header elements apply for both positive and negative responses. An exception is made for cases, where an error occurred before functional processing. Examples for such error cases are general server errors (typically with 50x http response code) and – depending on the implementation – the validation of the certificate. In those error cases, the ASPSP may omit the specified headers. However, when functional processing has already taken place, the ASPSP is still required to include the mandated (and if applicable conditional) headers also in a negative response.

4.1.2 Body

All descriptions of body elements in the service documents only apply to cases with a positive response. The related attributes need also be provided in the body, where possible and applicable. In addition, the API shall offer additional error information as described in Section 4.2, when the API server is technically able to provide it.

4.2 Additional Error Information

If necessary, the ASPSP **might** communicate additional error information to the API Client within a request/response dialogue which results in 4xx or 5xx HTTP response codes, in some exemptions also for HTTP response code 2xx, see also Section 3.8 This specification offers two possibilities for ASPSPs to communicate additional error information. The ASPSP might choose one of the solutions. Note that the major additional error information is the detailed error code which is of type "Message Code" as defined in [oFA DD] is used in both variants of additional error information.

In cases, where no message code is defined for an HTTP response code, the additional error information is not used, since the messageCode is a mandatory subfield. In this case, the HTTP code gives sufficient information about the error situation.

4.2.1 openFinance API Framework Specific Solution

The openFinance API Framework offers a proprietary way to transport additional error information. In this solution, the additional error information is sent to the API Client using the data element apiClientMessages with the attribute category set to "ERROR". The attribute "code" indicates the error, and the attribute "path" the path of the element of the request

message which provoked this error message, if applicable. It will further offer a free text field to describe the error context or actions to be taken to the API Client.

Usually, the usage of apiClientMessages accompanies a negative response. However, there are cases where the ASPSP sends a positive response but still includes an apiClientMessages attribute. This might occur, when the API Client sends a status request and the request itself is technically accepted but the requested status indicates some kind of banking processing issue/error or the requirement of additional action by the API Client or the PSU. In the same way, the requirement of additional actions can be indicated when (generally) accepting a payment initiation.

In addition, the response message might optionally contain a _links section containing a hyperlink to tell the API Client the next step to avoid further errors, cp. Section 3.6. This applies especially in case of PSU authentication errors where a resubmission of credentials by the API Client might be needed after new entering of credentials by the PSU.

Response Code

The HTTP response code is 4xx or 5xx as defined in Section 3.8 for response codes in case of errors.

Response Header

Attribute	Type	Condition	Description
Content-Type	String	Mandatory	The string application/json is used.

Response Body

Attribute	Type	Condition	Description
apiClientMessages	Array of Client Message Information	Optional	Error information
_links	Links	Optional	Should refer to next steps if the problem can be resolved e.g. for re-submission of credentials.

Example 1 (Access token not correct):

```
{ "apiClientMessages": [{
        "category": "ERROR",
        "code": "TOKEN_INVALID",
        "text": "additional text information of the ASPSP up to 500
characters"
   }]
```

```
}
```

Example 2 (Password incorrect):

```
{ "apiClientMessages": [{
        "category": "ERROR",
        "code": "PSU_CREDENTIALS_INVALID",
        "text": "additional text information of the ASPSP up to 500
characters"
   }],
  "_links": {
        "updatePsuAuthentication": {"href": "/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations/3d9a81b3-
a47d-4130-9999-a9c0ff861100"}
            }
}
```

4.2.2 Standardised Additional Error Information

In [RFC7807] a standardised definition of reporting error information is described. In the following, requirements of how to use this standardised error information reporting in the context of openFinance API Framework are defined.

RFU: This section might be replaced in future by [RFC9457] which is the successor of [RFC7807].

Response Code

The HTTP response code is 4xx or 5xx as defined in Section 3.8 for response codes in case of errors. However, with the same reasoning as in Section 4.2.1, Additional Error Information may also be included in certain responses with positive response codes.

Response Header

Attribute	Type	Condition	Description
Content-Type	String	Mandatory	The string application/problem+json is used.

Response Body

Attribute	Type	Condition	Description
type	Max70Text	Mandatory	A URI reference [RFC3986] that identifies the problem type. **Remark for Future**: These URI will be provided by NextGenPSD2 in future.

Attribute	Type	Condition	Description
title	Max70Text	Optional	Short human readable description of error type. Could be in local language. To be provided by ASPSPs.
status	Integer	Optional	HTTP response code generated by the server. If contained, this is more relevant as the actual http response code in the actual response, because it is introduced by the application server.
detail	Max500Text	Optional	Detailed human readable text specific to this instance of the error.
instance	Max256Text	Optional	This attribute is containing a JSON pointer (as defined in [RFC6901]) or XPath expression to indicate the path to an issue generating the error in the related request.
code	Message Code	Mandatory	Message code to explain the nature of the underlying error.
additionalErrors	Array of Error Information	Optional	Might be used if more than one error is to be communicated
_links	Links	Optional	Should refer to next steps if the problem can be resolved e.g. for re-submission of credentials.

Example 1

```
  HTTP/1.1 401 Unauthorized
  Content-Type: application/problem+json
  Content-Language: en
   {
    "type": "https://berlingroup.com/error-codes/TOKEN_INVALID",
    "title": " The OAuth2 token is associated to the TPP but is not valid
for the addressed service/resource.",
   "detail": " additional text information of the ASPSP up to 500
characters ",
    "code": "TOKEN_INVALID",
    "additionalErrors": [ {
            "title": "The PSU-Corporate-ID cannot be matched by the
addressed ASPSP.",
            "detail": "additional text information of the ASPSP up to 500
characters",
            "code": "CORPORATE_ID_INVALID"
```

```
    },… ],
    "_links": { }
   }
```

Example 2

The following example reflects a decline to a Bulk Payment request, where the first payment of the bulk contains an incorrect IBAN to identify the creditor account,

```
  HTTP/1.1 400 Bad Request
  Content-Type: application/problem+json
  Content-Language: en
   {
    "type": "https://berlingroup.com/error-codes/PAYMENT_FAILED",
    "title": "The payment initiation POST request failed during the initial
process. Additional information may be provided by the ASPSP.",
   "detail": "IBAN check failed.",
   "instance": "/creditTransfers/0/creditorAccount/iban"
    "code": " PAYMENT_FAILED",
    "_links": { }
   }
```

5 Authentication of API Client and ASPSP at transport layer

The communication between an API Client and an ASPSP is always secured by using a TLS-secured connection. Client authentication is always used, i.e. a mutual authentication between the API Client and the ASPSP will be part of the setup of the TLS-secured connection.

TLS version 1.2 or higher has to be used. For the choice of cipher suite selections, NIST recommendations on the cryptographical strength should be followed. For ASPSPs, further cipher suite requirements of their national IT security agency or of a related API Access Scheme might apply.

Two TLS-secured connections have to be distinguished:

- TLS-secured connection established by the API Client (as TLS-client) to access the BG API provided by an ASPSP (as TLS-server) for usual API services defined by the XS2A Framework or the openFinance API Framework. See section 5.1.
- TLS-secured connection established by an ASPSP (as TLS-client) to access an API provided by an API Client (as TLS-server) for pushed-based services defined by the openFinance Framework. See section 5.2.

5.1 TLS-secured connection established by API Client

Support for this security measures is mandatory for any ASPSP and any API Client.

For all services offered at an XS2A API the TLS-secured connection has to be established by the API Client as TLS-client. The ASPSP acts as TLS-server in this case.

This is also true for most of the extended services offered at an openFinance API. Only for push-based extended services the TLS-secured connection has to be established by the ASPSP. See section 5.2 for this case.

For the case of an XS2A API:

The API Client has to establish the TLS-secured connection including client authentication, i.e. authentication of the API Client as TLS-client. For this authentication the API Client has to use a qualified certificate for website authentication (QWAC). This qualified certificate has to be issued by a qualified trust service provider according to the eIDAS regulation [eIDAS]. The content of the certificate has to be compliant with the requirements of [EBA-RTS] and the technical specification of ETSI [ETSI TS 119 495]. The certificate of the API Client has to indicate all roles the API Client is authorised to use (according to [PSD2] and [EBA-RTS].

Remark: For getting a QWAC compliant with the requirements of [EBA-RTS] and [ETSI TS 119 495] the API Client has to get the necessary licence to access a service as a PISP, AISP or PIISP from an NCA according to the regulations of [PSD2].

For the case of an openFinance API:

The requirements for the certificate for website authentication to be used by the API Client for the client authentication as TLS-client will be defined by the ASPSP. In any case a QWAC of

an API Client enabling him to access the XS2A API of an ASPSP can also be used by the API Client to access the openFinance API of the ASPSP.

In any case within the openFinance API Framework:

This specification does not define any further requirements for the certificates used by an ASPSP as TLS-server. Of course, the certificates should be compliant with general requirements defined by [RFC5280] and the best practise recommendations of the CA/Browser Forum.

If an ASPSP wants to support any push-based services it is recommended that the certificate of the ASPSP supports also the client authentication as TLS-client. See section 5.2 for further details.

5.2 TLS-secured connection established by ASPSP

Support for this security measures is conditional for an ASPSP and for an API Client. It shall be supported by an ASPSP if the ASPSP wants to offer pushed-based services. It shall be supported by an API Client if the API Client wants to use pushed-based services offered by an ASPSP.

For push-based extended services it is necessary that the TLS-secured connection is established by the ASPSP. In this case the ASPSP acts as TLS-client and the API Client as TLS-server.

The ASPSP **shall** use the same certificate for his authentication as TLS-client as he will use for his authentication as TLS-server as described in section 5.1. For this it this necessary that the certificate of the ASPSP supports also the client authentication, i.e. the certificate shall contain the field extendedKeyUsage with both attributes clientAuth and serverAuth.

The API Client **should** use the same QWAC for his authentication as TLS-server as he will use for his authentication as TLS-client as described in section 5.1.

The URIs which are provided by an API Client to an ASPSP for push-based services shall comply with the domain secured by the certificate of the API Client (used for his authentication as TLS-server), i.e. as contained in one of the fields CN or SubjectAltName of the certificate of the API Client. Please note that in case of example-TPP.com, a certificate entry Client-Notification-URI like

- https://example-TPP.com/xs2a-client/v2/ASPSPidentifcation/mytransaction-id/notifications or
- https://push-server.example-TPP.com/xs2a-client/v2/ASPSPidentifcation/mytransaction-id/notifications

would be compliant.

Wildcard definitions shall be taken into account for compliance checks by the ASPSP.

NOTE: The URI should be addressable by the ASPSP just by presenting his certificate for his authentication as TLS-client as part of the establishment of the TLS-secured connection. No further pre-steps of any kind should be mandated by the API Client.

5.3 Certificate Requirements on Redirect URIs

The TPP can provide several URIs to the ASPSP as parameters for succeeding protocol steps. For security reasons, it should be ensured that these URIs are secured by the TPP eIDAS QWAC used for identification of the TPP. The following applies:

URIs which are provided by TPPs in Client-Redirect-URI or Client-Nok-Redirect-URI should comply with the domain secured by the eIDAS QWAC certificate of the TPP in the field CN or SubjectAltName of the certificate. Please note that in case of example-TPP.com as certificate entry Client-Redirect-URI like

- www.example-TPP.com/xs2a-client/v2/ASPSPidentifcation/mytransaction-id or
- redirections.example-TPP.com/xs2a-client/v2/ASPSPidentifcation/mytransaction-id

would be compliant.

Wildcard definitions shall be taken into account for compliance checks by the ASPSP.

NOTE: In premium services, ASPSPs may reject requests, if the provided URIs do not comply.

6 Signing HTTP request messages at the application level

Support for this security measure is optional for an ASPSP.

Support for the security measure is mandatory for an API Client. If an ASPSP requests signing a HTTP request message at the application level the API Client shall sign the HTTP request message at application level.

If requested by an ASPSP only the HTTP request message has to be signed at application level by the API Client. HTTP response messages are not signed by the ASPSP, if not stated otherwise explicitly in the ASPSP documentation.

Remark for Future: Future versions might also support explicit specification of signing HTTP response messages at application level.

Signing of HTTP request messages at application level used for push-based services is also possible. The ASPSP (or related API access scheme) decides if he signs an HTTP request message used for a pushed-based service at the application level.

Signing of an HTTP request message at application level is always done using the signature profile based on JSON Web signature [RFC7515] adapted for signing HTTP request messages (taking [OBEsign] as best practise approach into account). See section 6.2 for the description of this signature profile.

6.1 Certificates to be used

For the case of an XS2A API:

For signing an HTTP request message at application level, the API Client has to use a qualified certificate for electronic seals (QSealC). This qualified certificate has to be issued by a qualified trust service provider according to the eIDAS regulation [eIDAS]. The content of the certificate has to be compliant with the requirements of [EBA-RTS] and the technical specification of ETSI [ETSI TS 119 495] . The certificate of the API Client has to indicate all roles the API Client is authorised to use (according to [PSD2] and [EBA-RTS].

Remark: For getting a QSealC compliant with the requirements of [EBA-RTS] and [ETSI TS 119 495] the API Client has to get the necessary licence to access a service as a PISP, AISP or PIISP from an NCA according to the regulations of [PSD2].

For the case of an openFinance API:

The requirements for the certificate for signing HTTP request messages to be used by the API Client will be defined by the ASPSP. In any case a QSealC of an API Client enabling him to sign HTTP request messages at the XS2A API of an ASPSP can also be used by the API Client to sign HTTP request messages at the openFinance API of the ASPSP.

In any case of a BG API:

This specification does not define any further requirements for the certificates used by an ASPSP to sign HTTP request messages as part of a push-based service. Of course, the certificates should be compliant with general requirements defined by [RFC5280].

6.2 Signing HTTP messages based on [RFC7515]

The following signature profile is compliant with the requirements of [RFC7515] (JSON Web Signature) and [ETSI TS 119 182-1] (JAdES profile of ETSI). It is based on the best practise approach defined by [OBEsign].

6.2.1 Extensions to the HTTP message

6.2.1.1 Path

None.

6.2.1.2 Header parameter

The following parameters have to be added to the header of the HTTP request message:

Attribute	Type	Condition	Description
Digest	String	Conditional	Contains the hash value calculated over the content of the body of the HTTP request message. See section 6.2.2.1.
x-jws-signature	String	Conditional	JSON Web Signature containing the base64url encoded JWS Protected Header and the base64url encoded JWS Signature Value (separated by ".."). See section 6.2.2.2.

Table 1: Header parameters for HTTP message signature according to [OBEsign].

Remark on the row "condition":

If the HTTP request message has to be signed using this signature profile both attributes are mandatory.

Remark on the transport of the certificate:

The attribute TPP-Signature-Certificate used in former versions of the specifications for the transport of the certificate is not available for version 2 of the specification. Instead the recommendations of [OBEsign] are used for the transport of the certificate. These are summarized as follows:

- If the certificate used by the API Client to sign the HTTP request message is not already known by the ASPSP the API Client shall include his certificate into the JWS Protected Header using the element x5c. See requirement 9 of [OBEsign].

- If the certificate used by the API Client to sign the HTTP request message is already known by the ASPSP (by some prior arrangements out of scope of this specification) one of the elements x5c (containing the certificate) or x5t#S256 (containing a digest of the certificate using SHA 256) shall be part of the JWS Protected Header. Not both elements shall be part of the JWS Protected Header at the same time. See requirement 7 of [OBEsign].

- The ASPSP may require in any case that the certificate is included into the JWS Protected Header using the element x5c.

- If the JWS Protected Header contains the element x5t#S256 (containing a digest of the certificate using SHA 256), the JWS Protected Header may also contain the element x5u with further information to identify the certificate. See option 14 of [OBEsign].

6.2.1.3 Message body

None.

6.2.2 How to build the extensions?

6.2.2.1 Attribute Digest

When an API Client includes a signature according to this signature profile, he also must include a "Digest" header as defined in [RFC3230]. The "Digest" Header contains a Hash of the message body. If the message does not contain a body, the "Digest" header must contain the hash of an empty byte list. The only hash algorithms that may be used to calculate the Digest within the context of this specification are SHA-256 and SHA-512 as defined in [RFC5843].

Remark: In case of a multipart message the same method is used to calculate the digest. I.e. a hash of the (whole) message body is calculated including all parts of the multipart message as well as the separators.

6.2.2.2 Attribute x-jws-signature

The attribute x-jws-signature contains the JSON Web Signature. This consists of the following three elements:

Element	Type	Condition	Description
JWS Protected Header	String	Mandatory	Contains the JWS Protected Header base64url encoded.
Delimiter	String	Mandatory	Constant string **".."**.
JWS Signature Value	String	Mandatory	Contains the JWS Signature Value base64url encoded.

Table 2: Elements of the JSON Web Signature according to [OBEsign].

JWS Protected Header

The JWS Protected Header is a JSON structure which defines how the signature is created. It contains the following elements:

Element	Type	Condition	Description
typ	String	Mandatory	Fixed string "JOSE".
b64	Boolean	Mandatory	Parameter according to [RFC 7797] indicating that the payload does not need to be base64url re-encoded According to [OBEsign] this element has to be included and has to be set to "false".
x5c	String	Conditional	Certificate used by the API Client to sign the HTTP request message. The certificate has to be included base64url encoded. This element may include the full certificate chain up to a trust anchor. See option 10 in [OBEsign]. This element shall be part of the JWS Protected Header if the certificate is not already known by the ASPSP (by some prior arrangements out of scope of this specification). The ASPSP may require that this element is included. The ASPSP informs about this as part of the discovery API.

Element	Type	Condition	Description
x5t#S256	String	Conditional	Hash value over the certificate of the API Client used for signing the request message. SHA 256 has to be used for calculating this hash value. The hash value has to be included base64url encoded. This element shall be part of the JWS Protected Header if and only if the element x5c is not part of this JWS protected header.
x5u	String	Optional	An URI pointing to the resource where the X.509 signing certificate (with or without the certification path) may be retrieved from. This element may be part of the JWS Protected Header only if also the element x5t#S256 is also part of the JWS Protected Header
crit	List	Mandatory	List of names of elements of the JWS Protected Header which are marked critical. According to [OBEsign] this element has to be included and has to contain the following list: ["b64", "sigT", "sigD"] Other elements shall not be marked critical for BG API.
sigT	DateTime	Mandatory	Claimed signing time. The time shall be UTC (ending in "Z") and shall indicate date and time to the second.
sigD	JSON	Mandatory	This element shall contain the two sub-elements pars and mid. The sub-element pars contains the list of HTTP header parameters which are used to create the data to be signed. The names of the HTTP header parameters shall be given lowercase in this list. The sub-element mid identifies the mechanism used to identify the data to be signed. This sub-element shall contain the fixed value http://uri.etsi.org/19182/HttpHeaders

Element	Type	Condition	Description
			For using this signature profile with BG API, the following restrictions concerning the sub-element pars shall be considered: The list shall contain at least "digest" and "x-request-id". Conditionally the list shall include: • "api-contract-id", If and only if "API-Contract-ID" is included as a header of the HTTP-Request. • "psu-id", if and only if "PSU-ID" is included as a header of the HTTP-Request. • "psu-corporate-id", if and only if "PSU-Corporate-ID" is included as a header of the HTTP-Request. • "Client-Redirect-uri", if and only if "Client-Redirect-URI" is included as a header of the HTTP-Request. The list may include further HTTP header parameters.
alg	String	Mandatory	This element identifies the cryptographic algorithm used to create the JWS Signature Value. According to [OBEsign] this element has to be included and shall not have the value "none". Identifiers should be used (if possible) according to [RFC7518]. For using this signature profile with BG API, the following restrictions concerning the algorithm to be used shall be considered: The algorithm shall identify the same algorithm for the signature as described for the public key (Subject Public Key Info) in the certificate (already known by the ASPSP or contained in the element "x5c") of this Request. Otherwise the request message shall be rejected.

Element	Type	Condition	Description
aud	String	Mandatory	Parameter according to [RFC7519] containing the method and the target of the HTTP-Request as a string. The target is identified by the relative URI starting with /v2/. If the HTTP-Request contains query parameters, these are also included in this element. Examples for the content of the aud element: "POST /v2/deferred-payments/sepa-credit-transfers" "GET /v2/deferred-payments/sepa-credit-transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/initiations?dateFrom=2024-04-15"

Table 3: Elements of the JWS Protected Header according to [OBEsign].

Remarks:

- The element aud according to [RFC7519] is an extension to [OBEsign]. It is used to protect the method and the relative path of the HTTP-request by the signature.
- If the element x5t#S256 is used: For the current version the algorithm SHA 256 shall be used to calculate the hash value over the certificate. This might be enhanced to further hash algorithms in future versions of this specification.
- If the element x5t#S256 is used: It is recommended that the relying party checks the hash value contained in X5t#S256 against the certificate of the API Client (already known to the ASPSP by some prior arrangements out of scope of this specification).
- Remark: If the element x5t#S256 is used, the hash value can be used as key id to identify a certificate/public key exchanged between API Client and ASPSP before.
- For the element sigD the RECOMMENDATION-23 of [OBEsign] is not followed, i.e. (request-target), Content-Type, Content-Encoding and Host may not be part of the sub-element pars.
- For the element sigD the ASPSP may not require more HTTP header parameters to be included. The API Client may decide which HTTP header parameters are included into the generation of the signature as long as the restrictions mentioned above are met.
- For the element alg the ASPSP may introduce further restrictions about the algorithms to be used. Information will be given by the discovery API of the ASPSP.

JWS Signature Value

The signature shall be calculated using the algorithm indicated by the element alg of the JWS Protected Header. The key to be used shall be the private key corresponding to the certificate (e.g. QSealC) of the API Client.

To calculate the signature, proceed as follows:

1. Encode the JSON structure JWS Protected Header into base64url.

2. Collect the HTTP header parameters as indicated by the sub-element pars of the element sigD of the JWS Protected Header (using the sequence defined by the content of pars) to form a string as follows:

   a) For an HTTP header name contained in the element pars create the header field string by concatenating the lowercased header field name followed with a colon ':', a space character, and the header field value. Any leading and trailing white spaces are removed. If there are multiple instances of the same header field, all header field values associated with the header field shall be concatenated, separated by an ASCII comma and an ASCII space ', ', and used in the order in which they will appear in the transmitted HTTP message.

   b) Insert newline character after all but the last HTTP header value.

   c) Concatenate the header field strings.

   See 5.2.8.2 of [ETSI TS 119 182-1] [for this procedure.

3. Prepare the signature input by combining the JWS Protected Header base64url coded (see step 1.) with the result string of step 2 separated by **"."**.

4. Compute the JWS Signature value using the signature input from step 3 using the signature algorithm determined by the element alg of the JWS Protected Header.

6.2.3 Example

Disclaimer:

The following example is only an informative part of this specification. Its objective is just to give an example for the different steps to build the signed HTTP message starting from a given HTTP message. It should not be taken as a reference point for implementations for signing Http messages. The key pair used has been generated without any connection to a CA. No certificate exists for the public key. The hash value of the certificate used in the example is only a dummy value. In addition, the sequence of steps shown in the example could be changed by a real implementation if the same result is produced.

For the purposes of this example, it is assumed that an RSA key and corresponding X.509 certificate with the following Properties are used:

RSA public Modulus (hex)	97 55 5b cf bb 51 d5 a0 01 db cb 34 3a 56 b6 1d 18 ff 5a e0 c6 43 79 cd b1 48 83 8d 02 80 b5 df c7 85 eb 4c 0b cb 97 07 e3 0b bb e6 83 ec a3 09 73 d2 9a fe 7a f4 69 f6 e1 a6 0a f9 fc 9e 35 06 37 d9 08 4b 42 cb 11 69 4d 8f f1 5d ee ef 18 10 a4 73 d4 1e cb a5 d2 ca d9 da 91 b7 06 9d 54 03
RSA public Exponent (hex)	03
RSA private Key (hex)	19 38 e4 a2 9f 38 4e 45 55 a4 a1 de 09 b9 1e 5a 2e d5 39 d0 21 0b 3e f7 9d 8c 15 ec d5 c0 1e 4f f6 96 51 e2 01 f7 43 d6 a5 d7 49 fb c0 a7 70 81 51 b1 90 73 6e 31 90 4a 2e 0c f8 85 72 90 34 9d 66 7e f6 74 66 4c 69 9c 46 01 41 0c 3a a5 0d c8 9b 05 d2 68 c0 7c 44 a6 64 da 69 19 28 b6 05 af
Certificate SHA-256 hash (hex)	77 2b 4f a5 29 09 63 38 53 74 f5 d2 58 fe e3 85 78 06 e2 40 8e 58 85 86 89 eb 1d ce 4b cd 2f 36
Certificate SHA-256 hash (base64)	dytPpSkJYzhTdPXSWP7jhXgG4kCOWIWGiesdzkvNLzY

Step 0: Take the HTTP message (HTTP Headers + HTTP Body)

Description: Prepare unsigned HTTP message input to JWS function.

Note: Each line is terminated with a pair of carriage return and line feed character (0x0D0A), including the last line.

POST /v2/payments/sepa-credit-transfers HTTP/1.1

Host: api.testbank.com

Content-Type: application/json

X-Request-ID: 99391c7e-ad88-49ec-a2ad-99ddcb1f7721

PSU-IP-Address: 192.168.8.78

PSU-GEO-Location: GEO:52.506931,13.144558

PSU-User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:54.0) Gecko/20100101

Firefox/54.0

Date: Mon, 26 Oct 2020 11:24:37 GMT

{

"instructedAmount": {"currency": "EUR", "amount": "123.50"},

"debtorAccount": {"iban": "DE40100100103307118608"},

"creditor": {"name": "Merchant123"},
"creditorAccount": {"iban": "DE02100100109307118603"},
"remittanceInformationUnstructured": ["Ref Number Merchant"]
}

Step 1: Create JWS Protected Header

Description: Produce JWS header parameters which define how the signature is created:

"b64": false means don't base64url encode header data to be signed

"x5t#S256": "…." Hash of signing certificate (base64url encoded)

"crit": ["sigT","sigD","b64"]

"sigT": "…." Claimed signing time

"sigD": {….} Lower case HTTP Header fields to be signed

"alg": "RS256" Signature algorithm

Note: This is shown below in a pretty print layout. It will be sent as a single string without line brakes or extra spaces. Escape characters are not used.

```
{
"b64": false,
"x5t#S256": "dytPpSkJYzhTdPXSWP7jhXgG4kCOWIWGiesdzkvNLzY",
"crit": [ "sigT", "sigD", "b64"],
"sigT":"2020-10-26T11:26:57Z",
"sigD":{
    "pars":["x-request-id", "digest" ],
    "mId":"http://uri.etsi.org/19182/HttpHeaders"
    },
"alg":"RS256"
}
```

Step 2: Encode JWS Protected Header into Base64url

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

```
eyJiNjQiOmZhbHNlLCJ4NXQjUzI1NiI6ImR5dFBwU2tKWXpoVGRQWFNXUDdqaFhnRzRr
Q09XSVdHaWVzZHprdk5MelkiLCJjcml0IjpbInNpZ1QiLCJzaWdEIiwiYjY0Il0sInNp
Z1QiOiIyMDIwLTEwLTI2VDExOjI2OjU3WiIsInNpZ0QiOnsicGFycyI6WyJ4LXJlcXVl
```

c3QtaWQiLCJkaWdlc3QiXSwibUlkIjoiaHR0cDovL3VyaS5ldHNpLm9yZy8xOTE4Mi9I
dHRwSGVhZGVycyJ9LCJhbGciOiJSUzI1NiJ9

Step 3a: Compute Digest of HTTP Body

Description: Calculate hash of the HTTP Body (payload without HTTP header and following empty line).

With the example body above, SHA-256 would be 0x98420e321d1e9514e464261450dce761f57d546d4c4efef1c1313d4bcc8da632 (hexadecimal) or in base64 : `mEIOMh0elRTkZCYUUNznYfV9VG1MTv7xwTE9S8yNpjI=`.

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).

```
x-request-id: 99391c7e-ad88-49ec-a2ad-99ddcb1f7721

digest: SHA-256=mEIOMh0elRTkZCYUUNznYfV9VG1MTv7xwTE9S8yNpjI=
```

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.

eyJiNjQiOmZhbHNlLCJ4NXQjUzI1NiI6ImR5dFBwU2tKWXpoVGRQWFNXUDdqaFhnRzRr
Q09XSVdHaWVzZHprdk5MelkiLCJjcml0IjpbInNpZ1QiLCJzaWdEIiwiYjY0Il0sInNp
Z1QiOiIyMDIwLTEwLTI2VDExOjI2OjU3WiIsInNpZ0QiOnsicGFycyI6WyJ4LXJlcXVl
c3QtaWQiLCJkaWdlc3QiXSwibUlkIjoiaHR0cDovL3VyaS5ldHNpLm9yZy8xOTE4Mi9I
dHRwSGVhZGVycyJ9LCJhbGciOiJSUzI1NiJ9.x-request-id: 99391c7e-ad88-
49ec-a2ad-99ddcb1f7721

digest: SHA-256=mEIOMh0elRTkZCYUUNznYfV9VG1MTv7xwTE9S8yNpjI=

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".

The RSA Signature of the example is

Hex	36 e1 a0 95 60 6d 1d c5 da be 86 f6 66 65 be 3e 84 a9 32 68 49 4b d5 3a cf cc cf a8 71 68 13 de ef 1c 8d f3 f3 0b 5a 55 d6 f0 b2 3e 56 ce 03 e3 b5 55 94 52 79 3c 73 13 f8 26 32 d7 77 0c f0 86 11 c1 da e2 79 14 8e 19 64 67 52 41 0c 3e f6 95 cf 53 1d dc 08 94 b3 1e 6a e0 90 9f 9d 8b e0 38

base64URL	NuGglWBtHcXavob2ZmW-PoSpMmhJS9U6z8zPqHFoE97vHI3z8wtaVdbwsj5WzgPjtVWUUnk8cxP4JjLXdwzwhhHB2uJ5FI4ZZGdSQQw-9pXPUx3cCJSzHmrgkJ-di-A4

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.

eyJiNjQiOmZhbHNlLCJ4NXQjUzI1NiI6ImR5dFBwU2tKWXpoVGRQWFNXUDdqaFhnRzRrQ09XSVdHaWVzZHprdk5MelkiLCJjcml0IjpbInNpZ1QiLCJzaWdEIiwiYjY0Il0sInNpZ1QiOiIyMDIwLTEwLTI2VDExOjI2OjU3WiIsInNpZ0QiOnsicGFycyI6WyJ4LXJlcXVlc3QtaWQiLCJkaWdlc3QiXSwibUlkIjoiaHR0cDovL3VyaS5ldHNpLm9yZy8xOTE4Mi9IdHRwSGVhZGVycyJ9LCJhbGciOiJSUzI1NiJ9..NuGglWBtHcXavob2ZmW-PoSpMmhJS9U6z8zPqHFoE97vHI3z8wtaVdbwsj5WzgPjtVWUUnk8cxP4JjLXdwzwhhHB2uJ5FI4ZZGdSQQw-9pXPUx3cCJSzHmrgkJ-di-A4

Step 7: Insert JSON Web Signature into HTTP Message to form HTTP Signed Message

Description: The HTTP message as sent over the network with the JSON Web Signature inserted.

POST /v2/payments/sepa-credit-transfers HTTP/1.1

Host: api.testbank.com

Content-Type: application/json

X-Request-ID: 99391c7e-ad88-49ec-a2ad-99ddcb1f7721

PSU-IP-Address: 192.168.8.78

PSU-GEO-Location: GEO:52.506931,13.144558

PSU-User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:54.0) Gecko/20100101

Firefox/54.0

Date: Mon, 26 Oct 2020 11:24:37 GMT

Digest: SHA-256=mEIOMh0elRTkZCYUUNznYfV9VG1MTv7xwTE9S8yNpjI=

x-jws-signature:
eyJiNjQiOmZhbHNlLCJ4NXQjUzI1NiI6ImR5dFBwU2tKWXpoVGRQWFNXUDdqaFhnRzRrQ09XSVdHaWVzZHprdk5MelkiLCJjcml0IjpbInNpZ1QiLCJzaWdEIiwiYjY0Il0sInNpZ1QiOiIyMDIwLTEwLTI2VDExOjI2OjU3WiIsInNpZ0QiOnsicGFycyI6WyJ4LXJlcXVlc3QtaWQiLCJkaWdlc3QiXSwibUlkIjoiaHR0cDovL3VyaS5ldHNpLm9yZy8xOTE4Mi9IdHRwSGVhZGVycyJ9LCJhbGciOiJSUzI1NiJ9..NuGglWBtHcXavob2ZmW-PoSpMmhJS9U6z8zPqHFoE97vHI3z8wtaVdbwsj5WzgPjtVWUUnk8cxP4JjLXdwzwhhHB2uJ5FI4ZZGdSQQw-9pXPUx3cCJSzHmrgkJ-di-A4{

"instructedAmount": {"currency": "EUR", "amount": "123.50"},

"debtorAccount": {"iban": "DE40100100103307118608"},

"creditor": {"name": "Merchant123"},

"creditorAccount": {"iban": "DE02100100109307118603"},
"remittanceInformationUnstructured": "Ref Number Merchant"
}

7 Security measures for securing (parts of) the message body

7.1 Signing the body

The body of a request or response message may be signed by one or more responsible entities. Who is responsible for signing the body depends on the single use case. The following are possible examples:

- For a request message sent by an API Client to the openFinance API of an ASPSP the body may be signed by one or more PSU.
- For a response message to a request of an API Client the body may be signed by a responsible department or one or more responsible employees of the ASPSP.
- For a request message as part of a pushed-based service the body may be signed by a responsible department or one or more responsible employees of the ASPSP.

If the body of a request message sent by an API Client to the openFinance API of an ASPSP is signed by one or more PSU this will be called a **signed payment request**, if the request message is part of a payment service and contains payment data. For a signed payment request further SCA procedures of the PSU involved will not be necessary if the creation of the corresponding signatures is compliant with the requirements of [EBA-RTS].

Remark:

> Signing the body of a request message for a signed payment request should not be confused with the signing of an HTTP request message according to section 6.. Signing the HTTP message according to section 6 will authenticate the API Client sending the request message to openFinance API of the ASPSP. It will not authenticate the PSU and for this reason cannot substitute an SCA of the PSU.

It is up to the ASPSP to decide if it supports signed bodies for messages. For some use cases an ASPSP might mandate that the body of a request message sent by an API Client has to be signed by one or more PSUs.

This specification does not define any requirements about the quality of the process to generate the signature over the body of the http message. It is possible that the ASPSP will define further requirements for this as for example

- special algorithms to be used,
- key length to be used,
- quality of certificates to be used.

Different signature procedures and profiles based on asymmetric and symmetric cryptography will be supported in future by this specification. The phrase "signing a body" will be used regardless if the body will be secured by an electronic signature or an electronic seal based on asymmetric cryptography or by some kind of cryptogram based on symmetric cryptography.

7.1.1 Extension to the http message

7.1.1.1 Path

None.

7.1.1.2 Query parameters

None.

7.1.1.3 Header parameters

If the body is signed the following header parameters may be added to the header of the HTTP request or response message:

Attribute	Type	Condition	Description
Body-Sig-Profile	String	Conditional	Indicates the signature profile used for signing (parts of) the body. Shall be used if the body is signed.

Table 4: Header parameters to indicate that (parts of) the body have been signed.

Body-Sig-Profile

This parameter indicates the profile used for signing (parts of) the message body.

If this parameter is missing the information has to be known by some other means.

Currently the following values are supported by this specification:

JAdES_JS	The body is signed based on [RFC7515] using JWS JSON Serialization taking the requirements of [ETSI TS 119 182-1] for JAdES into account.
XAdES	The body is signed based on [W3C XMLSig] taking the requirements of [ETSI EN 319 132-1] for XAdES into account.
EMV_AC	The body is signed using an EMV AC cryptogram.

Table 5: Supported signature profiles.

Future versions of the specification may support further values for the signature profile.

7.1.1.4 Body

The content of the unsigned body has to be replaced by a signed data structure.

The procedure for building the signed data structure depends on the signature profile used for signing the body of the message. See the following subsections.

NOTE: The signing body solution for JSON encoded bodies is not yet part of market consultation. This introductory text is only provided to inform the market about next steps in enhancing the openFinance API Framework by e.g. SCA mechanisms by signing JSON transaction content in the message body. The XML version is more mature and already contained

7.1.2 Signing the body using JAdES_JS

NOTE: This signing function is still under internal review and will not be published yet. A new version of this document expected for Q4 2023 will add this section.

7.1.3 Signing the body using XAdES

The profile XAdES can be used to sign the body of a message regardless of the coding of the content. Nevertheless, in the following only the case of an XML coded body is considered.

ETSI European Norm [ETSI EN 319 132-1] and [W3C XMLSig] will be used for the XAdES profile. Note that for the XML Signature Syntax and Processing also a newer version exists [W3C XMLSig V2], but for compliance reasons with the European Norm version 1.1 as defined by [W3C XMLSig] is used.

The body of the message will be replaced by an XML document representing the signed body as follows:

Unsigned body:

```
<any_tag> body data to be signed </any_tag>
```

Signed body:

```
<SignedBody>
      <Object Id="ID_bodyToBeSigned">
            <any_tag> body data to be signed </any_tag>
      </Object>
      <Object>
            <Manifest Id="ID_manifest"> … </Manifest>
      </Object>
      </Signature Id="ID_signature_1"> … </Signature>
      …
      </Signature Id="ID_signature_n"> … </Signature>
</SignedBody>
```

Table 6: Elements of a signed XML message body.

Namespaces have to be included as described by [ETSI EN 319 132-1] and [W3C XMLSig].

Only detached signatures signing local data are used, i.e. the data to be signed is contained in a sibling element. The data to be signed can be protected by n signatures (with n >= 1). Note that these are independent signatures and not counter signatures. If one signature has to be protected by another signatures, counter signatures as described in section 5.2.7 of [ETSI EN 319 132-1] shall be used.

The manifest is introduced to increase efficiency for the generation and verification of the signatures. The hash value over the body data to be signed has to be calculated only once and not separately for each signature.

The IDs shown above are only place holders. During creation of the XML document it has to be taken care that the concrete values used for these IDs do not produce any collisions that violate the ID uniqueness.

Algorithms

For transformation, canonicalization, hash value calculation and signature creation the algorithms have to be supported as defined by [ETSI EN 319 132-1]. Out of this set of algorithms the ASPSP can mandate algorithm to be used.

Element Manifest

The element Manifest has got the following content:

```
<Manifest Id="ID_manifest>
     <Reference URI="#ID_bodyToBeSigned">
          <Transforms>
               <Transform Algorithm="URI_TransformAlgorithm"/>
          </Transforms>
          <DigestMethod Algorithm="URI_HashAlgorithm"/>
          <DigestValue> base64 coded hash value </DigestValue>
     </Reference
</Manifest>
```

Table 7: Content of the element Manifest.

This manifest contains only the digest value for the body to be signed. It will be referenced in each element containing a signature. This manifest is only created once regardless how many signatures will be generated to protect the body. Also, during verification of the signatures, the content of this manifest is only created once.

Element Signature

An element Signature containing a single signature protecting the body has got the following content:

```
<Signature Id="ID_signature_k">
     <SignedInfo> … </SignedInfo>
```

```
        <SignatureValue>
              base64 coded signature value
        </SignatureValue>
        <KeyInfo> … </KeyInfo>
        <Object>
              <QualifyingProperties>
                    <SignedProperties Id="ID_signedProp_k"> …
                    </SignedProperties>
                    <UnsignedProperties> … </UnsignedProperties>
              </QualifiedProperties>
        </Object>
</Signature>
```

Table 8: Content of a single signature element

Element QualifyingProperties

The Element QualifyingProperties contains the signed and not signed properties of the data to be signed and of the signature according to [ETSI EN 319 132-1]. The element SignedProperties will be referenced in the element SignedInfo and by this protected by the signature contained in the element SignatureValue.

The signature has to be compliant at least with XAdES baseline signatures of level B-B defined in section 6 of [ETSI EN 319 132-1]. For this reason, the element SignedProperties may not be empty (see section 6.3 of [ETSI EN 319 132-1]). It has got the following sub elements:

```
<SignedProperties Id="ID_signedProp_k">
      <SignedSignatureProperties> …
      </SignedSignatureProperties>
      <SignedDataObjectProperties> …
      </SignedDataObjectProperties>
</SignedProperties>
```

Table 9: Content of the element SignedProperties

The element SignedSignatureProperties has to contain at least the following sub elements:

- SigningTime (see section 5.2.1 of [ETSI EN 319 132-1]).
- SigningCertificateV2 (see section 5.2.2 of [ETSI EN 319 132-1]).

The element SignedDataObjectProperties hat to contain at least the following sub elements:

- DataObjectFormat (see section 5.2.4 of [ETSI EN 319 132-1]) containing at least the MIME type indicating the format of the body data to be signed.

The element UnsignedProperties may be empty. If it is empty it is missing. Empty elements are not allowed according to [ETSI EN 319 132-1].

An ASPSP can mandate higher levels defined by section 6 of [ETSI EN 319 132-1] depending on the nature of the service.

Element SignedInfo

The element SignedInfo contains the information which data is signed (data of the unsigned body and the signed properties) and how the signature has to be created. It has got the following content:

```
<SignedInfo>
     <CanonicalizationMethod Algorithm= URI_CanoniAlgorithm/>
     <SignatureMethod Algorithm="URI_SignatureAlgorithm"/>
     <Reference URI="#ID_bodyToBeSigned">
          Type="http://www.w3.org/2000/09/xmldsig#Manifest">
          <Transforms>
               <Transform Algorithm="URI_TransformAlgorithm"/>
          </Transforms>
          <DigestMethod Algorithm="URI_HashAlgorithm"/>
          <DigestValue> base64 coded hash value </DigestValue>
     </Reference>
     <Reference URI="#ID_signedProp_k"
          <Transforms>
              <Transform Algorithm="URI_TransformAlgorithm"/>
          </Transforms>
          <DigestMethod Algorithm="URI_HashAlgorithm"/>
          <DigestValue> base64 coded hash value </DigestValue>
     </Reference>
</SignedInfo>
```

Table 10: Content of the element SignedInfo

Element KeyInfo

The element KeyInfo contains information about the key needed for the verification of the signature. According to section 6.3 of [ETSI EN 319 132-1]

- it **shall** contain at least the signing certificate, i.e. the X.509 certificate belonging to the public key needed for the verification of the signature, and
- it **should** contain all certificates not already available to the relying party needed to verify the signing certificate.

The element has got at least the following content:

```
<KeyInfo>
      <X509Data>
            <X509Certificate>
                  base64 coded X.509 signer certificate
            </X509Certificate>
      </X509Data>
</KeyInfo>
```

Table 11: Content of the element KeyInfo

The sub element X509Data may contain more than one certificate building a path from the signer certificate to a root certificate or to a CA contained in a trusted list.

7.1.4 Signing the body using EMV_AC

Not supported for the current version of the openFinance Framework. Support will be added as part of future versions.

7.2 Encryption of (parts of) the body

Parts of a body or the complete body of a request or response message can be encrypted, if the body is JSON or XML encoded.

For the support of the encryption of (parts of) the body of a message the following holds.

- An ASPSP may support the encryption or decryption of (parts of) the body of a request or response message optionally.
- An ASPSP shall encrypt (parts of) a response message to be sent to an API Client only if the API Client accepts encrypted (parts of the) message bodies.
- An ASPSP shall encrypt (parts of) a request message to be sent to an API Client as part of a pushed-based service only if the API Client accepts encrypted (parts of the) message bodies.
- An API Client shall support the encryption of (parts of) a body of a request message to be sent to an ASPSP.
- An API Client shall encrypt (parts of) the body of a request message to be sent to an ASPSP if this is requested by an ASPSP.
- An API Client may support the decryption of (parts of) a body of a response message receive from an ASPSP optionally.

- For pushed-based services: An API Client may support the decryption of (parts of) a body of a request message receive from an ASPSP or the encryption of (parts of) a body of a response message to be sent to an ASPSP optionally.

Remark:

API access schemes may define further requirements.

7.2.1 Overview

Encryption of (part of) the body of an http (request or response) message will be based on the specification of

- JSON Web Encryption ([RFC7516]) for the case of JSON encoded bodies,
- XML Encryption [XML ENC] for the case of XML encoded bodies.

For JSON encryption:

- If the complete body of the message will be encrypted the complete plain text content of the body will be replaced by a JWE data structure representing the encrypted (and integrity) protected content of the body.
- If only an attribute of a body of the message will be encrypted the plain text content of this attribute will be replaced by a JWE data structure representing the encrypted (and integrity protected) content of the attribute.

For XML encryption:

- If the complete body of the message will be encrypted the complete plain text content of the body will be replaced by an XML element representing the encrypted (and integrity protected) content of the body.
- If only an element of a body of the message will be encrypted the plain text content of this element will be replaced by an XML element representing the encrypted (and integrity protected) content of the element.

Remark: The protection of the integrity of the encrypted content depends on the algorithm used for the encryption. If A256GCM is used the integrity of the content is protected in addition to the encryption of the content.

Only key encryption will be used as key management mode. A randomly generated CEK (content encryption key) is used for the encryption of the plain text using a symmetric encryption algorithm. A public key of the receiver is used for the encryption of the CEK using an asymmetric encryption algorithm.

For the current version of this specification only the following encryption algorithms may be used (see also [RFC7518] and [XML ENC]):

- A256GCM/AES256-GCM: Symmetric AES using the Galois/Counter mode of operation for the encryption of the plain text. For this the CEK shall be a 256-bit AES key.

- RSAES-OAEP: Asymmetric RSA with OAEP padding for the encryption of the CEK. MGF1 with SHA-1 mask generation function shall be used.

For JSON encryption only JWE Compact Serialisation according to section 3.1 of [RFC7516] is used.

Remark: It is possible that an ASPSP or a scheme mandates the support and usage of further encryption algorithms.

7.2.2 Extension to the http message

7.2.2.1 Path

None.

7.2.2.2 Query parameters

None.

7.2.2.3 Header parameters

If (parts of) the body contains encrypted information the following header parameters may be added to the header of the HTTP request or response message:

Attribute	Type	Condition	Description
Body-Enc-Profile	String	Conditional	Indicates the encryption profile used for the encryption of (parts of) the body.
Body-Enc-List	List of names	Optional	Contains a List of names of data elements/ attributes of the body which contain encrypted information.

Table 12: Header parameters to indicate the encryption of (parts of) the body.

Body-Enc-Profile

This parameter indicates the profile used for the encryption of (parts of) the message body. Currently only the values

- JWE_CS for "JSON Web Encryption using Compact Serialization" and
- XML_ENC for "XML Encryption" are supported.

Future versions of the specification may support further values for the encryption profile.

Note: This header parameter shall exist if encryption of the body or parts of the body is used.

Body-Enc-List

This parameter indicates which of the data elements/attributes of the body contain encrypted content. Example: ["password"]. If this parameter equals ["BODY"], the complete body of the message will be encrypted.

If this parameter is missing it has to be indicated by some other means which parts of the body contain encrypted information. For example, the name of a data element/attribute may by itself indicate that the content is encrypted, like for example the attribute encryptedPassword.

7.2.2.4 Body

If the header parameter Body-Enc-List contains the name of a data element/attribute of the message body or if the name of a data element/attribute indicates that the content has to be encrypted, the plain text content of this data element/attribute has to be exchanged by

- (in the case of JSON encoding) the JWE built over using plain text content of that data element/attribute as described in section 7.2.3, or
- (in the case of XML encoding) the EncryptedElementTag element build based on the plain text content of that element as described in section 7.2.4.

If the header parameter Body-Enc-List equals ["BODY"] the plain text of the complete body has to be exchanged by

- (in the case of JSON encoding) the JWE built using the complete plain text content of the message body as described in section 7.2.3, or
- (in the case of XML encoding) the EncryptedBody element build based on the plain text XML body as described in section 7.2.4.

For JSON encoding encryption can only be used for special attributes for which the message definition allows strings without further restrictions.

For XML encoding encryption can only be used for special elements for which the message definition allows the EncryptedElementTag element as content without further restrictions.

7.2.3 How to build a JWE

For the current version of this specification only Compact Serialisation according to section 3.1 of [RFC7516] is used. In this case the JWE is a string built by the concatenation of the following base64url encoded elements:

BASE64URL(UTF8(JWE Protected Header)) || '.' ||

BASE64URL(JWE Encrypted KEY) || '.' ||

BASE64URL(JWE Initialisation Vector) || '.' ||

BASE64URL(JWE Ciphertext) || '.' ||

BASE64URL(Authentication Tag)

The single elements are built as follows:

JWE Protected Header

The JWE Protected Header is a JSON structure with the following elements:

Element	Type	Condition	Description
alg	String	Mandatory	Algorithm to be used for the encryption of the content encryption key (CEK). For the current version of this specification only the value RSAES-OAEP is supported.
enc	String	Mandatory	Algorithm to be used for the encryption of the plain text using the CEK. For the current version of this specification only the value A256GCM is supported.
x5c	JSON	{Or	The "x5c" (X.509 certificate chain) Header Parameter contains the X.509 public key certificate or certificate chain corresponding to the key used to encrypt the CEK. The certificate or certificate chain is represented as a JSON array of certificate value strings. Each string in the array is a base64-encoded (**not** base64url-encoded) DER PKIX certificate value. The certificate containing the public key corresponding to the key used to the encrypt the CEK MUST be the first certificate. This MAY be followed by additional certificates, with each subsequent certificate being the one used to certify the previous one. The recipient MUST validate the certificate chain according to RFC 5280 and consider the certificate or certificate chain to be invalid if any validation failure occurs.

Element	Type	Condition	Description
X5u	URI	Or}	The "x5u" (X.509 URL) Header Parameter is a URI that refers to a resource for the X.509 public key certificate or certificate chain corresponding to the key used to encrypt the CEK. The identified resource MUST provide a representation of the certificate or certificate chain that conforms to RFC 5280 in PEM-encoded form, with each certificate delimited as specified in Section 6.1 of RFC 4945. The certificate containing the public key used to encrypt the CEK MUST be the first certificate. This MAY be followed by additional certificates, with each subsequent certificate being the one used to certify the previous one. The protocol used to acquire the resource MUST provide integrity protection; an HTTP GET request to retrieve the certificate MUST use TLS; and the identity of the server MUST be validated, as per Section 6 of RFC 6125.

Table 13: JSON structure of a JWE Protected Header.

Remarks:

- One of the elements x5u or x5c shall be contained in the JWE Protected Header.
- The ASPSP may define further restriction for the content of the elements x5u and x5c. For example, the ASPSP may require that the element x5c is contained and that this element contains only the certificate itself but not the complete certificate chain. These additional requirements have to be described by the ASPSP specific documentation.

JWE Encrypted Key

The sender of the message shall generate a random content encryption key (CEK). This CEK shall be usable with the algorithm indicated by the element "enc" as part of the JWE protected header.

For the current version of this specification the CEK shall be a 256-bit AES key.

The JWE Encrypted Key equals the encrypted CEK with the algorithm indicated by the element "alg" as part of the JWE protected header using the public key of the receiver of the message.

JWE Initialisation Vector

The sender of the message shall generate a random JWE Initialisation Vector. This initialisation vector shall be usable with the algorithm indicated by the element "enc" as part of the JWE protected header.

For the current version of this specification the JWE Initialisation Vector shall be a 128 bit long random value.

JWE Ciphertext

The plain text of the complete body or of the data element/attribute of the body has to be encrypted with the CEK using the encryption algorithm given by the element "enc" of the JWE Protected Header. For the current version of this specification only A256GCM is supported, i.e. the Galois/Counter mode of operation [GCMencryp]. By this the plain text is not only encrypted but also integrity protected.

For the Galois/Counter mode of operation "additional authenticated data" (AAD) is needed. This is defined by the following:

AAD = ASCII(BASE64URL(UTF8(JWE Protected Header)))

JWE Authentication Tag

The output of the Galois/Counter mode of operation of the AES algorithm delivers not only the ciphertext but also a 128-bit long authentication tag. This 128-bit output has to be used as JWE Authentication Tag.

7.2.4 How to build an EncryptedBody or EncryptedElementTag element

If the complete body has to be encrypted the corresponding encrypted body has the following content:

```
<EncryptedBody>
     <EncryptedData>
          See below
     </EncryptedData>
     <EncryptedKey ID="ID_CEK">
          See below
     </EncryptedKey>
</EncryptedBody>
```

Table 14: Elements of an encrypted XML message body

If an element with the tag ElementTag has to be encrypted the corresponding encrypted element has the following content:

```
<EncryptedElementTag>
      <EncryptedData>
            See below
      </EncryptedData>
      <EncryptedKey ID="ID_CEK">
            See below
      </EncryptedKey>
</EncryptedElementTag>
```

Table 15: Elements of an encrypted XML element with tag ElementTag

Element EncryptedData

This element contains the encrypted data and shall contain the following:

```
<EncryptedData>
      <EncryptionMethod Algorithm="URI_encryptionAlgorithm" />
      <KeyInfo>
         <RetrivalMethod URI="#ID_CEK"
         Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
      </KeyInfo>
      <CipherData>
            <CipherValue>
            Base 64 coded cipher value of the encrypted content
            </CiperValue>
      </CipherData>
</EncryptedData>
```

URI_encryptionAlgorithm identifies the (symmetric encryption) algorithm which has been used for the calculation of the cipher value using the symmetric CEK (content encryption key). This CEK can be retrieved from the element EncryptedKey identified by "ID=ID_CEK".

URI_encryptionAlgorithm = http://www.w3.org/2009/xmlenc11#aes256-gcm has to be supported, i.e. AES encryption with 256-bit CEK using the Galois/Counter mode of operation for the encryption of the plain text. An ASPSP can mandate the usage of another algorithm.

Element EncryptedKey

This element contains the encrypted CEK (content encryption key) and shall contain the following:

```
<EncryptedKey ID="ID_CEK">
     <EncryptionMethod Algorithm="URI_CEK_encryptionAlg" />
     <KeyInfo>
          <X509Data>
               <X509Certificate>
               base64 coded X.509 encryption certificate
               </X509Certificate>
          </X509Data>
     </KeyInfo>
     <CipherData>
          <CipherValue>
          Base 64 coded cipher value of the encrypted CEK
          </CiperValue>
     </CipherData>
</EncryptedKey>
```

URI_CEK_encryptionAlg identifies the (asymmetric encryption) algorithm which has been used for the calculation of the cipher value of the CEK using the public key of the receiver of the message. X.509 certificate identifying this public key is contained in the sub-element KeyInfo.

URI_CEK_encryptionAlg = http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p has to be supported, i.e. RSA-OAEP encryption using MGF1 with SHA-1 mask generation function. An ASPSP can mandate the usage of another algorithm.

The element `X509Data shall contain the X.509 certificate containing the public key of the receiver of the message used to encrypt the CEK.`

7.2.5 Exchange of certificates with public encryption keys

For the encryption of (parts of) the body of an HTTP-message it is necessary that the sender of the message knows the public key of the receiver of the message to be used to encrypt the CEK to be used for the encryption of the content. This public key has to be exchanged as part of a corresponding certificate over this public key. The openFinance API Framework supports two different mechanisms to exchange these encryption certificates.

Exchange of the certificate within an attribute of a message body:

A message may contain the following attribute if the sender of this message wishes that the receiver of this message will encrypt (parts of) the body of following messages:

Attribute	Type	Condition	Description
encryptionCertificate	String	Optional	The certificate to be used by the receiver for encryption of (parts of) the body of a following message

The attribute encryptionCertificate contains the X.509 public key certificate or certificate chain corresponding to the public key to be used to encrypt the CEK for the encryption of (parts of) the body of following messages. The certificate or certificate chain is represented as a JSON array of certificate value strings. Each string in the array is a base64-encoded (**not** base64url-encoded) DER PKIX certificate value. The certificate containing the public key corresponding to the key used to the encrypt the CEK MUST be the first certificate. This MAY be followed by additional certificates, with each subsequent certificate being the one used to certify the previous one.

The ASPSP may define further restrictions such that this attribute may contain only the certificate of the public key itself but not a certificate chain.

Exchange using a special link to the certificate:

The "_links" attribute of a message may contain the following special link if the sender of this message wishes that the receiver of this message will encrypt (parts of) the body of following messages:

Attribute	Type	Condition	Description
_links	String	Optional	May contain the following special link: "encryptionCertificates": {"href":…}

The href-link is an URI that refers to a resource for the X.509 public key certificate or certificate chain corresponding to the public key to be used to encrypt the CEK for the encryption of (parts of) the body of following messages. The identified resource MUST provide a representation of the certificate or certificate chain that conforms to RFC 5280 in PEM-encoded form, with each certificate delimited as specified in Section 6.1 of RFC 4945. The certificate containing the public key to be used for the encryption of the CEK MUST be the first certificate. This MAY be followed by additional certificates, with each subsequent certificate being the one used to certify the previous one.

8 Strong customer authentication of a PSU

Remark: The term PSU is used for the user of a services even in the case of services which are not related to a payment service.

8.1 Optional Usage of OAuth2 for PSU Authentication or Authorisation

The XS2A API respectively the openFinance APIs will allow an ASPSP to implement OAuth2 as a support for the authorisation of the PSU towards the API Client for several services. In this case, the API Client will be the client, the PSU the resource owner and the ASPSP will be the resource server in the abstract OAuth2 model.

This specification supports two ways of integrating OAuth2. The first support is an authentication of a PSU in a pre-step, translating this authentication into an access token to be used at the XS2A interface or openFinance API afterwards. This usage of OAuth2 will be referred to in this specification as "if OAuth2 has been used as PSU authentication". Further details shall be defined in the documentation of the ASPSP of this XS2A interface or openFinance API.

Remark: When implementing the OAuth pre-step, the requirements on e.g. registration steps or no mandatory two SCA application in specific PIS only scenarios as defined by [EBA-OP2] should be recognised by the ASPSP.

The second option to integrate OAuth2 is an integration as an OAuth2 SCA Approach to be used for authorisation of transactions. In all services, OAuth2 will in this option be used in an integrated way, by using the following steps:

Integrated OAuth for the authorisation of transactions which are reflected as resources in the related services endpoints of the API:

1.) The transaction data is posted to the corresponding service endpoint of the XS2A API resp. the openFinance API. This service endpoint might be extended by a product-type, e.g. sepa-credit-transfers in the case of the /payments endpoint.
2.) The OAuth2 protocol is used with the "Authorization Code Grant" flow to get the authorisation of the PSU for the related transaction, while using the "scope" attribute in OAuth2 to refer to the data from Step 1.).
3.) The corresponding transaction is then automatically initiated by the ASPSP after a successful authorisation by the PSU, if no further interaction between API Client and ASPSP is required.

NOTE: For AIS services, the API Client needs to get a consent resource authorised first via the routine described above. The API Client then can use the access token received during the OAuth2 protocol to access the related accounts endpoint for authorised account information for the validity period of the authorised consent resp. the validity period of the technical access token.

For Step 2.), details are described in Section 8.8.

When using OAuth2, the XS2A API or openFinance API calls will work with an access token instead of using the PSU credentials.

8.2 Header Parameter for PSU Context Data

The following data elements are forwarding information about the interface between PSU and API Client to enhance the risk management procedures of the ASPSP integrated into SCA resp. SCA exemption procedures. It is **strongly recommended** to send these data elements within all Transaction Initiation Requests, that involve PSU authentication, e.g. in Payment Initiation Requests.

Header parameters as defined in this section will not be repeated in the implementation guidelines of service specifications to enhance document readability, but the implementation guidelines will refer to this document in the relevant sections. However, the detailed specifications of service calls via the related OpenAPI files will still contain all applicable headers.

The only exception is where conditions other than "optional" apply on specific request messages, e.g. for the PSU IP Address. More details might be provided in the data overview within service definitions.

Attribute	Format	Condition	Description
PSU-IP-Address	String	Conditional	The forwarded IP Address header field consists of the corresponding HTTP request IP Address field between PSU and TPP. Conditions will be defined within service specifications, If applicable.
PSU-IP-Port	String	Optional	The forwarded IP Port header field consists of the corresponding HTTP request IP Port field between PSU and TPP, if available.
PSU-Accept	String	Optional	The forwarded IP Accept header fields consist of the corresponding HTTP request Accept header fields between PSU and TPP, if available.
PSU-Accept-Charset	String	Optional	see above
PSU-Accept-Encoding	String	Optional	see above
PSU-Accept-Language	String	Optional	see above
PSU-User-Agent	String	Optional	The forwarded Agent header field of the HTTP request between PSU and TPP, if available.
PSU-Http-Method	String	Optional	HTTP method used at the PSU – TPP interface, if available. Valid values are: • GET

Attribute	Format	Condition	Description
			• POST • PUT • PATCH • DELETE
PSU-Device-ID	String	Optional	UUID (Universally Unique Identifier) or IMEI for a device, which is used by the PSU, if available. UUID/IMEI identifies either a device or a device dependant application installation. In case of an installation identification this ID need to be unaltered until removal from device.
PSU-Geo-Location	Geo Location	Optional	The forwarded Geo Location of the corresponding HTTP request between PSU and TPP if available.

Note: Information about the interface between PSU and API Client might be used by the ASPSP as input for his fraud detection and risk management systems. Some ASPSPs use this information also to exclude certain authentication methods (for example some ASPSPs do not allow to receive an OTP by SMS on the same smartphone used also for the transaction itself). In addition, the ASPSP might need to receive specific device related information to be able to support an optimised app-2-app redirection procedure for the API Client. For these reasons it is highly recommended that an API Client includes all of this information in the related Transaction Initiation Request messages. Missing information may result in an assessment of the user device as not useable for the authentication method or in a classification of the current transaction as a "higher risk transaction" e.g. due to session attacks. By this, the probability of a rejection of that transaction due to the result of fraud detection and/or risk management might be increased.

Remark: These requirements do not apply in a "direct access" scenario, where no third party provider is involved and where the ASPSP can detect the related information directly from the then PSU/ASPSP interface.

8.3 Header Parameters for PSU Identification Data

The openFinance API Framework supports the transmission of PSU identification data within request parameters for all Transaction Initiation Request messages. The ASPSP might mandate these elements, which then needs to be reflected in the related ASPSP documentation.

Header parameters as defined in this section will not be repeated in the implementation guidelines of service specifications to enhance document readability, but the implementation guidelines will refer to this document in the relevant sections. However, the detailed specifications of service calls via the related OpenAPI files will still contain all applicable headers.

Attribute	Type	Condition	Description
PSU-ID	Max140Text	Conditional	Client ID of the PSU in the ASPSP client interface. Might be mandated in the ASPSP's documentation. It might be contained even if an Oauth2 based authentication was performed in a pre-step. In this case the ASPSP might check whether PSU-ID and token match, according to ASPSP documentation.
PSU-ID-Type	Max35Text	Conditional	Type of the PSU-ID; needed in scenarios where PSUs have several PSU-IDs as access possibility. In this case, the mean and use are then defined in the ASPSP's documentation.
PSU-Corporate-ID	Max140Text	Conditional	Identification of a Corporate in the Online Channels Might be mandated in the ASPSP's documentation. Only used in a corporate context.
PSU-Corporate-ID-Type	Max35Text	Conditional	This is describing the type of the identification needed by the ASPSP to identify the PSU-Corporate-ID content as used in online channels. Typically, this is a proprietary definition. Mean and use is defined in the ASPSP's documentation. Only used in a corporate context.

Note: If the ASPSP is mandating PSU identification related attributes, then the ASPSP needs to assure to be compliant with related sections from [EBA-OP2] if PSD2 compliance related services are addressed.

Note: The ASPSP might support special characters for PSU identification in their client interfaces. To support this on API level, the recommended character set for PSU identification related header parameters is ISO8859-1 based. In addition to the plain ISO8859-1 encoding following the HTTP specification, the ASPSP might offer a Base64 encoding. In the latter variant, this results in the following expression:

```
=?ISO-8859-1?B?<Base64 coded ISO-8859-1 String>?=
```

An UTF-8 encoding of header parameters used by the API Client thus might lead to rejections with Message Code PSU_CREDENTIALS_INVALID.

Deviations from these recommendations should be documented in detail by the ASPSP.

8.4 Header Parameters for strong customer authentication

Header parameters as defined in this section will not be repeated in the implementation guidelines of service specifications to enhance document readability, but the implementation guidelines will refer to this document in the relevant sections. However, the detailed specifications of service calls via the related Open API files will still contain all applicable headers.

8.4.1 Request Header Parameters Steering SCA Approaches

The following request headers might be used by an API Client in a Transaction Initiation Request to communicate SCA preferences to the ASPSP for transaction authorisation.

Attribute	Type	Condition	Description
Client-SCA-Approach-Preference	Max35Text	Optional	A comma separated list of attributes, where the first entry will have a higher priority than the next or to every SCA Approach which is not indicated at all, e.g. "decoupled, redirect, embedded" or "decoupled" This attribute may be ignored by the ASPSP
Client-Redirect-URI	String	Conditional	URI of the TPP, where the transaction flow shall be redirected to after a Redirect. Mandated for the Redirect SCA Approach. See Section 8.7 for further requirements on this header. It is recommended to always use this header field.
Client-Nok-Redirect-URI	String	Optional	If this URI is contained, the TPP is asking to redirect the transaction flow to this address instead of the Client-Redirect-URI in case of a negative result of the redirect SCA method. This might be ignored by the ASPSP. See Section 8.7 for further requirements on this header.
Client-Explicit-Authorisation-Preferred	Boolean	Optional	If it equals "true", the API Client prefers to start the authorisation process separately, e.g. because of the usage of a signing basket or because of asynchronous authorisation. This preference might be ignored by the ASPSP, if a signing

Attribute	Type	Condition	Description
			basket is not supported as functionality or if asynchronous authorisation is not supported. If it equals "false" or if the parameter is not used, there is no preference of the API Client. This especially indicates that the API Client assumes a direct authorisation of the transaction in the next step, without using a signing basket.

8.4.2 Related Response Headers

The following header is used by the ASPSP to transmit decisions of the chosen SCA Approach to the third party.

Attribute	Type	Condition	Description
ASPSP-SCA-Approach	String	Conditional	This data element must be contained, if the SCA Approach is already fixed. Possible values are: • EMBEDDED • DECOUPLED • REDIRECT • ASPSP-CHANNEL The OAuth SCA approach will be subsumed by REDIRECT.

8.5 Embedded SCA Approach

The Embedded SCA Approach is defined as the scenario where all PSU credentials are transmitted via the XS2A API resp. the openFinance API. PSU identification data are transported using the related header parameters as defined in Section 8.3. After a Transaction Initiation Request and Response, an authorisation sub-resource is either implicitly generated by the ASPSP or needs to be explicitly created by the API Client in a next step, cp. Section 9.1.

The API calls to create an authorisation sub-resource as well as the API calls to transmit PSU credentials to the related authorisation resource within an Embedded SCA Approach are defined in Section 9.4 and Section 9.5. The specific functions to be used will always be indicated by related hyperlinks presented by the ASPSP before.

8.6 Decoupled SCA Approach

The Decoupled SCA Approach differs from the Embedded SCA Approach by using an ASPSP authentication app for the strong customer authentication and dynamic linking if applicable. The app is used by the PSU also for authentication of transaction in the online channels. Since

the XS2A API resp. the openFinance API is not involved in this app authentication, no authentication specific API support is defined in this case.

NOTE: A more standardised security profile based on re-direct like the FAPI CIBA profile are not supported yet by the openFinance API Framework.

As a hybrid between Embedded and Decoupled SCA approach, the ASPSP might require the PSU to transmit a first factor (a password) first via the XS2A API resp. the openFinance API and only afterwards authentication with a second factor via the Decoupled Approach. The transmission of the password is handled as in the Embedded SCA Approach via the related authorisation sub resource and the API calls defined in Section 9.4. Such an implementation would still be called a Decoupled SCA Approach in the openFinance API Framework.

8.7 Redirect SCA Approach

In the Redirect SCA Approach, the TPP will at some point receive a "scaRedirect" link in the response from the ASPSP. The link leads to an authorisation page related to the transaction to be authorised and to the TPP must redirect the PSU agent to that link as a next step. When accessing the page, the PSU will authenticate himself with the related PSU credentials, review the presented information of the transaction and can finalise the authorisation from there. When the authorisation has been finalised by the PSU, the PSU agent is redirected again by the ASPSP to the Client-Redirect-URI, which has been submitted within the related Transaction Initiation Request. If the authorisation was not successful, the ASPSP may use the Client-Nok-Redirect-URI submitted in the same call, if supported.

Refer to Section 5.3 for requirements on TPP redirection links.

NOTE: The "scaRedirect" link may include query parameters. Due to definitions for a potential later confirmation flow, cp. Section 9.8.2.2, ASPSPs shall not include a query parameter named "state" in their "scaRedirect" links. Such a parameter is reserved for the TPP to be used for session control.

Remark for Future: For migration reasons, this specification mandates the TPP to keep the Client-Redirect-URI used within all authorisation processes for a specific transaction during the lifecycle of this transaction constant. This might be removed in the next version of the specification.

8.8 OAuth SCA Approach

The OAuth2 protocol as used optionally for this API is defined in [RFC6749]. In this section, additional requirements on the protocol are defined. The related protocol shall be used by the TPP, if the ASPSP is transmitting the link of type "scaOAuth".

NOTE: A more standardised security profile based on re-direct like the FAPI PAR profile are not supported yet by the openFinance API Framework.

The requirements on the data exchange between the TPP and the OAuth Server of the ASPSP regarding the transport layer are identical to the data exchange requirements between TPP and the XS2A Interface resp. the openFinance API, cp. Section 5 .

Note: Specifically, the requirements on using MTLS also apply to the usage of the OAuth2 protocol. However, the general requirements on the application layer such as e.g. signing of Requests (see Section 6) do not apply to the OAuth2 messages.

The response type "code" and the grant types "authorization_code" and "refresh_token" are recommended by this specification. It is further strongly recommended to TPPs and ASPSPs to follow the security best practices defined in [OA-SecTop].

Note: In case of the OAuth SCA Approach, the TPP has to generate in addition a nonce for the challenge parameter. This has also to be bound to the session of the user agent.

The ASPSP is required to provide TPPs with configuration data conforming to the "OAuth 2.0 Authorisation Server Metadata" specification.

8.8.1 Authorization Request

For the "authorization request" (see [RFC6749], section 4.1.1) to the OAuth2authorization endpoint provided in the Server Metadata, the following parameters are defined:

Query Parameters

Attribute	Condition	Description
response_type	Mandatory	"code" is recommended as response type.
client_id	Mandatory	organizationIdentifier as provided in the eIDAS certificate. PSD2 related access clients, the organizationIdentifier attribute shall contain information using the following structure in the presented order: - "PSD" as 3 character legal person identity type reference; - 2 character ISO 3166 country code representing the NCA country; - hyphen-minus "-" and - 2-8 character NCA identifier (A-Z uppercase only, no separator) - hyphen-minus "-" and - PSP identifier (authorization number as specified by NCA). For the usage in the context of premium services, a related API Access Scheme might define other requirements.
scope	Mandatory	PIS: The scope is the reference to the payment resource in the form "PIS:<paymentId>".

Attribute	Condition	Description
		AIS: The scope is the reference to the consent resource for account access in the form "AIS:<consentId>" PIIS: The scope is the reference to the consent resource for granting consent to confirmation of funds in the form "PIIS:<consentId>". Signing Basket Services: The scope is the reference to the signing basket resource for authorisation of a bundle of access functions in the form "SB:<basketId>. Subscriptions: The scope is the reference to the subscription resource or subscription entry sub-resource in the form "SUB:<subscriptionId>" resp. "SUB-E"<subscriptionId>-<entryId>." Mandate Services: The scope is the reference to the mandate resource in the form "MAN:<resoruceId>", where the resourceId is the resource identification of the related mandate. **Note:** The resource ids chosen by the ASPSP need to be unique to avoid resource conflicts during the SCA process.
state	Mandatory	A dynamical value set by the TPP and used to prevent XSRF attacks.
redirect_uri	Mandatory	the URI of the TPP where the OAuth2 server is redirecting the PSU's user agent after the authorization.
code_challenge	Mandatory	PKCE challenge according to cryptographic RFC 7636 (https://tools.ietf.org/html/rfc7636) used to prevent code injection attacks.
code_challenge_method	Optional	Code verifier transformation method, is "S256" or "plain". "S256" is recommended by this specification.

Example

```
GET /authorize?response_type=code&client_id=PSDES-BDE-3DFD21 &
scope=ais%3A3d9a81b3-a47d-4130-8765-a9c0ff861100+offline_access&
state= S8NJ7uqk5fY4EjNvP_G_FtyJu6pUsvH9jsYni9dMAJw&
```

```
redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb&
code_challenge_method="S256"
code_challenge=5c305578f8f19b2dcdb6c3c955c0aa709782590b4642eb890b97e43917cd
0f36 HTTP/1.1
    Host: api.testbank.com
```

8.8.2 Authorization Response

The Authorization Response (see [RFC6749], section 4.1.2) of the ASPSP will deliver the following data:

Remark: As the request is not sent by the TPP but the PSU user agent, it will not be secured by the TPP's QWAC.

http Response Code

302

Query Parameters

Attribute	Condition	Description
Location:	Mandatory	redirect URI of the API Client
code	Mandatory	Authorisation code
state	Mandatory	Same value as for the request.

Example

```
HTTP/1.1 302 Found

Location: https://client.example.com/cb
?code=SplxlOBeZQQYbYS6WxSbIA
&state=S8NJ7uqk5fY4EjNvP_G_FtyJu6pUsvH9jsYni9dMAJw
```

8.8.3 Access Token Request

As described in [RFC6749], section 4.1.3, the TPP sends a POST request to the token endpoint in order to exchange the authorization code provided in the authorization response for an access token and, optionally, a refresh token. The following parameters are used:

Request Parameters

Attribute	Condition	Description
grant_type	Mandatory	"authorization_code" is recommended as response type.

Attribute	Condition	Description
client_id	Mandatory	cp. Definition in Section 8.8.1
code	Mandatory	Authorization code from the authorization response
redirect_uri	Mandatory	the exact uri of the TPP where the OAuth2 server redirected the user agent to for this particular transaction
code_verifier	Mandatory	PKCE verifier according to cryptographic RFC 7636 (https://tools.ietf.org/html/rfc7636) used to prevent code injection attacks.

Example

```
POST /token HTTP/1.1
Host: https://api.testbank.com
Content-Type: application/x-www-form-urlencoded
client_id=PSDES-BDE-3DFD21
&grant_type=authorization_code
&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri= https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
&code_verifier=7814hj4hjai87qqhjz9hahdeu9qu771367647864676787878
```

The TPP is authenticated during this request by utilising "OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens" (see [RFC 8705]) in conjunction with the TPP's eIDAS certificate.

8.8.4 Access Token Response

The ASPSPS responds with the following parameters:

Response Parameters

Attribute	Condition	Description
access_token	Mandatory	Access Token bound to the scope as requested in the authorisation request and confirmed by the PSU.
token_type	Mandatory	Set to "Bearer"
expires_in	Optional	The lifetime of the access token in seconds
refresh_token	Optional	Refresh Token, which can be utilised to obtain a fresh access tokens in case the previous access token expired or was revoked. Especially useful in the context of AIS.

Attribute	Condition	Description
scope	Mandatory	the scope of the access token

Example

HTTP/1.1 200 OK

```
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
 "access_token": "SlAV32hkKG",
 "token_type": "Bearer",
 "expires_in": 3600,
 "refresh_token": "tGzv3JokF0XG5Qx2TlKWIA",
 "scope": "exampleScope"
}
```

8.8.5 Refresh Token Grant Type

The ASPSP may issue refresh tokens at its discretion, e.g. if an AISP uses the standard scope value "offline_access" or if the recurringIndicator is set to true.

8.8.6 API Requests

When using the OAuth SCA approach, subsequent API requests are being authorised using the respective OAuth Access Token. The Access Token is sent to the API using the “Authorization” Header and the “Bearer” authorization scheme as defined in RFC 6750.

This is an example API request

```
  GET /psd2/v2/payments/sepa-credit-transfers/3d9a81b3-a47d-4130-8765-
a9c0ff861100/status HTTP/1.1
  Host: https://api.testbank.com
  Authorization: Bearer SlAV32hkKG
```

8.9 ASPSP Channel SCA Approach

The ASPSP might offer to authorise a transaction **not** within the session

- started with the first resource data submission of the ASPSP within a Transaction Initiation procedure, or
- started with the POST /authorisations request

but instead later in other authorisation channels of the ASPSP like the online channels.

This SCA approach is addressed by the API Client by using the term "ASPSP-CHANNEL" in the attribute "Client-SCA-Approach-Preference".[2]

NOTE: The new ASPSP-SCA-Approach "ASPSP-CHANNEL" might be changed later on dynamically to e.g. "Redirect" by the API Client in case of re-starting the authorisation procedure explicitly via the API. It just sticks to "ASPSP-SCA-Approach", if the related transaction is authorised in a different authorisation channel of the ASPSP (e.g. online channel).

[2] This SCA approach is added to support extended services, where asynchronous SCA approaches are provided. This is not supposed to be supported for compliance services under PSD2 which cover synchronous authorisation processes only.

9 Authorisation Processes used commonly in all Services

Many business transactions need authorisation processes to be initiated via the openFinance API resp. the XS2A API. Authorisation processes include PSU authentication supported in the openFinance API Framework via the different SCA approaches as defined in Section 8. A very specific business transaction is a cancellation process for a transaction resource via the API where in some cases (e.g. payment initiation) also an authorisation process via SCA by the PSU might be mandated.

These authorisations are reflected in the openFinance API Framework in two perspectives:

- The impact of an authorisation process on an existing transaction resource, which will lead to new sub-resources of the addressed transaction resource as defined in Section 9.1 and 9.2.
- The impact on access methods for the related authorisation or cancellation authorisation endpoints as defined in Section 9.3 ff.

The description in this section is generic by defining the access methods in detail but abstractly. The related endpoints will not be documented in the implementation guidelines of service specification to enhance document readability. However, detailed specifications including the authorisation endpoints will be available via the related OpenAPI files.

Remark: The API design differs across the various SCA approaches (Embedded, Redirect, OAuth2 or Decoupled, cp. Section 8), but mostly between the Embedded SCA Approach and the others, since the Embedded SCA Approach demands the support of the full SCA complexity within the API itself. For that reason, all data or processes which are only needed for the Embedded SCA Approach are shown in a light blue background, to increase the readability of the specification.

9.1 Authorisation Endpoints

The openFinance API Framework is supporting dedicated authorisation endpoints in order to handle transaction authorisation by PSUs. These authorisation endpoints are supporting the following features in a common structured way

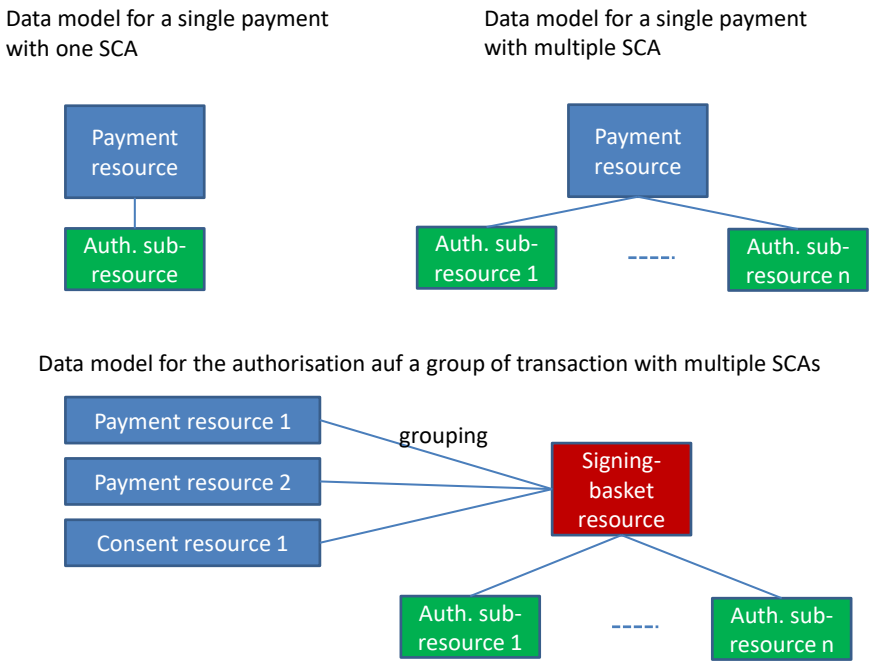
- multiple level SCA, where a transaction needs an authorisation by more than one PSU, e.g. in a corporate context,
- signing of a group of transactions with one SCA, as it is offered by ASPSPs today in online banking, cp. the definition of signing baskets in Section 10
- signing of a group of transactions with multi-level SCA, where this group of transactions need an authorisation by more than one PSU, e.g. in a corporate context.

To support this, the resources resulting from the submission of transaction data, like payment data or consent data are separated from authorisation (sub-)resources. For example, payment

which needs to be signed *n* times then will end up in a payment resource with *n* SCA (sub-)resources in a normal successful process.

The optional function of grouping several transactions for one common authorisation process is supported by the signing-baskets endpoint, which might be offered by the ASPSP. If this function is offered by an ASPSP, the API Client can first submit transaction data without starting the authorisation. After having grouped the related transaction resources by using a grouping command through the signing-baskets endpoint, the authorisation then can be started by authorising this basket content. This results in a basket resource with the corresponding authorisation sub-resource.

The following picture gives an overview on the abstract data model for the different scenarios in the example of a payment:



Remark: When offering the signing basket function, the ASPSP might restrict the grouping e.g.

- to payments as such,
- to individual payments,
- to the same payment product.

This restriction on groupings will then be detailed in the ASPSP's documentation.

Note: The grouping of transaction is only a "signing vehicle", bundling authorisation processes for the grouped transactions. The authorisation rules for transactions can be very complex in a corporate context. The signing basket gets the status of being fully authorised as soon as all grouped transactions have been successfully authorised by the applied SCA mechanism. A transaction with less authorisation requirements might then be authorised earlier than the

whole signing basket and also already processed. In addition, single transactions of the signing basket could be authorised with additional SCAs directly on transaction level, depending on the implementations of the ASPSPs – the signing basket is a non-exclusive mechanism to bundle authorisations. Current implementations of this functionality differ in Europe, specifically in a corporate context. For this reason, more complex functionality as DELETE processes on partially authorised signing baskets are not supported yet.

Remark for Future: The upcoming versions of the specification might implement more advanced functionality of the signing basket function and cancellation processes around it.

Optimisation process for the submission of e.g. single payments

The general model introduced above requires the API Client to start two sub-processes when initiating a transaction. For example, in a payment initiation of a sepa credit transfer this would result in

```
POST /payments/sepa-credit-transfers {payment data}
```

which is generating the payment resource and returns paymentId as a resource identification.

```
POST /payments/sepa-credit-transfers/paymentId/authorisations
```

is then starting the authorisation process with creating an authorisation sub-resource and returning an authorisationId for addressing this sub-resource in the following.

Applying this requirement to all authorisations of transactions e.g. in the Redirect SCA Approach would significantly augment the calls on the resulting API. For this reason, this specification still enables the ASPSP to directly start e.g. a Redirect SCA processing after the submission of transaction data, like e.g. a payment or a consent, if no other data from the API Client has to be submitted anyhow. In this case, the ASPSP will create the related authorisation sub-resources automatically and will give access to these sub-resources to the API Client by returning corresponding hyperlinks, cp. Section 3.6. As a consequence, the authorisation status would still be provided by submitting the command

```
GET /payments/sepa-credit-
transfers/paymentId/authorisations/authorisationId,
```

where the authorisation resource with identification authorisationId has been created by the ASPSP implicitly.

9.2 Transaction Cancellation Endpoints e.g. for Payments

The openFinance API Framework is supporting the cancellation of strongly protected resources, like payment initiations or payment authorisations by PISPs. This process is divided into two steps:

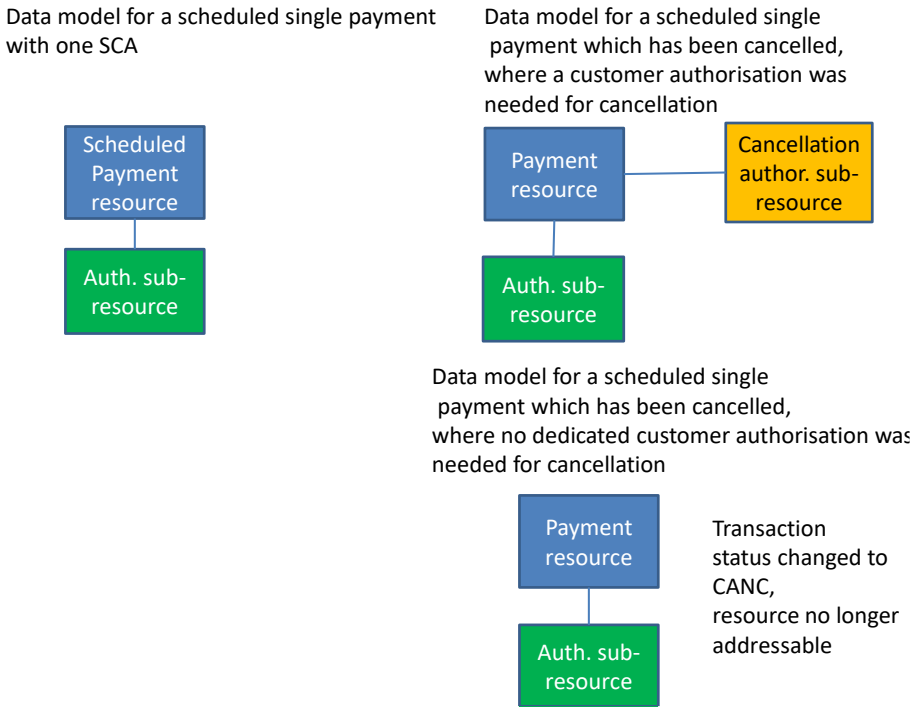
1. DELETE the corresponding resource.
2. Start an authorisation process for the cancellation by the PSU where needed by submitting a

   `POST payments/sepa-credit-transfers/paymentId/cancellation-authorisations`

   command.

The second step might be omitted, where a dedicated authorisation of the cancellation is not foreseen by the ASPSP. The need to authorise the cancellation will be communicated by sending corresponding hyperlinks to the API Client, cp. Section 3.6.

In the two-step approach, this cancellation process will be handled by cancellation-authorisation sub-resources in analogy to the actual authorisations. The authorisation sub resources will stay unchanged. The following picture shows the changes on resource level in case of a scheduled payment:



The corresponding original authorisation sub-resources stay unchanged.

For transactions, where a multilevel SCA is needed for authorisation, also a multilevel SCA might be needed for cancellation, depending on ASPSP role management. In equivalence to authorisation, the model would then be extended by more cancellation sub-resources.

NOTE: The logic of cancellation sub resources will apply also to other transactions, where a cancellation via the API Client requires a cancellation authorisation by the PSU.

9.3 Access Methods for Authorisations

Processes on starting authorisations, update PSU identification or PSU authentication data and explicit authorisation of transactions by using SCA are very similar in all services. The API calls supporting these processes are described in the following independently from the service/endpoint. These processes usually are used following a hyperlink of the ASPSP. The usage is defined at the beginning of the following sections. The API access methods are provided only in a schematic way in this document. The related OpenAPI files will provide the actual access methods by instantiating the term {resource-path}/{resourceId} with the addressed service, product types, where applicable, and the correct resource identification.

Thus, the following API access methods are supported for authorisation processes.

Endpoints/Resources	Method	Condition	Description
{resource-path}/{resourceId}/authorisations	POST	Mandatory	Create an authorisation sub-resource and start the authorisation process, might in addition transmit authentication and authorisation related data. This method is iterated n times for an n times SCA authorisation in a corporate context, each creating an own authorisation sub-endpoint for the corresponding PSU authorising the transaction. The ASPSP might make the usage of this access method unnecessary in case of only one SCA process needed, since the related authorisation resource might be automatically created by the ASPSP after the submission of the resource data with the first `POST {resource-path}` call. Section 9.4.
{resource-path}/{resourceId}/authorisations	GET	Mandatory	Read a list of all authorisation sub-resources IDs which have been created. Section 9.6.

Endpoints/Resources	Method	Condition	Description
{resource-path}/{resourceId}/authorisations/{authorisationId}	PUT	Mandatory for Embedded SCA Approach, Conditional for other approaches	Update data on the authorisation resource if needed. It may authorise a payment within the Embedded SCA Approach where needed. Independently from the SCA Approach it supports e.g. the selection of the authentication method and a non-SCA PSU authentication. Section 9.4.1, Section 9.5.
{resource-path}/{resourceId}/authorisations/{authorisationId}	PUT	Conditional for Redirect or OAuth SCA Approach	Update the authorisation resource with a confirmation code to counter potential session attacks, where supported by ASPSPs. Section 9.8.
{resource-path}/{resourceId}/authorisations/{authorisationId}	GET	Mandatory	Read the SCA status of the authorisation. Section 9.7.
{resource-path}/{resourceId}/cancellation-authorisations	POST	Optional	Starts the authorisation of the cancellation of the addressed resource with resource identification resourceId if mandated by the ASPSP (i.e. the DELETE access method is not sufficient) and if applicable to the related service, and received in product related timelines (e.g. before end of business day for scheduled payments of the last business day before the scheduled execution day). **NOTE:** Such a cancellation authorisation for now only applies to payments. But the openFinance API Framework allows to apply it to any other service. Section 9.4.

Endpoints/Resources	Method	Condition	Description
{resource-path}/{resourceId}/cancellation-authorisations	GET	Optional	Retrieve a list of all created cancellation authorisation sub-resources. If the POST command on this endpoint is supported, then also this GET method needs to be supported. Section 9.6.
{resource-path}/{resourceId}/cancellation-authorisations/{authorisationId}	PUT	Mandatory for Embedded SCA Approach, Conditional for other approaches	Update data on the cancellation authorisation resource if needed. It may authorise a cancellation of the payment within the Embedded SCA Approach where needed. Independently from the SCA Approach it supports e.g. the selection of the authentication method and a non-SCA PSU authentication. Section 9.4.1 and Section 9.5.
{resource-path}/{resourceId}/cancelation-authorisations/{authorisationId}	GET	Mandatory	Read the SCA status of the cancellation authorisation. Section 9.7.

Remark for Future: The PUT HTTP methods might be adapted to technical PATCH methods in a future version of the specification. A corresponding decision will reflect current market practices and the work in ISO TC68/SC9/WG2 on Financial API services.

9.4 Start Authorisation Process

Usage

If the further processing of a resource reflecting a business transaction (e.g. a payment) requires an authorisation of one or more PSUs, each authorisation is represented by an authorisation sub-resource of that resource, cp. also Section 9.1 for the example of payment initiation. This transaction resource has been created after the submission of Transaction Initiation Request, cp. Section 2.2.5..

In some cases, an authorisation by the PSU is required, when an already existing and authorised resource is to be cancelled. Currently, the only occasion of that situation is the

cancellation of a payment. In such a case, a "Transaction Cancellation Request" (with corresponding "Transaction Cancellation Response") has been sent by the API Client for an already existing resource. This leads to a situation, where a "Cancelation-Authorisation" will be created to represent the authorisation of the cancellation of the existing resource by the PSU. Therefore, the Cancellation-Authorisation resources also are sub-resources of the "Transaction" resource itself, cp. Section 9.2.

The "start authorisation process" is the process which creates a new authorisation or cancellation-authorisation sub-resource. The related sub-resource is reflecting the PSU authentication steps as authorisation process or cancellation authorisation process of the PSU. Please note that all the processes and conditions are identical for authorisations and cancellation-authorisations. The only difference appears in their respective names and effects on the underlying resource. The "start authorisation process" appears in the following scenarios:

- The ASPSP has indicated with a "startAuthorisation" hyperlink in the preceding Transaction Initiation Response or Transaction Cancellation Response that an explicit start of the authorisation process is needed by the API Client. The "startAuthorisation" hyperlink can transport more information about data which needs to be uploaded by using the extended forms
  - "startAuthorisationWithPsuIdentfication",
  - "startAuthorisationWithPsuAuthentication",
  - "startAuthorisationWithEncryptedPsuAuthentication",
  - "startAuthorisationWithAuthentciationMethodSelection"
  - "startAuthorisationWithTransactionAuthorisation".
- The related business transaction or business transaction cancellation cannot yet be executed since a multilevel SCA is mandated.

Call

```
POST /v2/{resource-path}/{resourceId}/{authorisation-category}
```

Starts an authorisation process or a cancellation authorisation process for the business transaction reflected by the resource identified with resourceId, depending on the authorisation category `authorisations` or `cancellation-authorisations`.

Path Parameters

Attribute	Type	Description
resource-path	String	This resource path can be a one-level parameter {service} or a two-level parameter {service}/{product-type}, where • {service} stands for the service type of the related business transaction, e.g. /payments or /consents

Attribute	Type	Description
		• {product-type} stands for the product-type of the related business transaction where applicable, e.g. sepa-credit-transfers in the case of payments or account-access in case of consents.
authorisation-category	String	The following two categories are supported: • `authorisations`: used in case of an authorisation of the related business transaction. • `cancellation-authorisations`: used in case of the cancellation authorisation of the related business transaction. Used only if applicable to the addressed {service}
resourceId	String	Resource identification of the related payment initiation, signing basket, consent, subscription or other related business transaction resource.

Query Parameters

No specific query parameters.

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
PSU-ID	String	Conditional	Client ID of the PSU in the ASPSP client interface. Shall be transmitted if this Request is indicated by "startAuthorisationWithPsuIdentification" or "startAuthorisationWithPsuAuthentication" or "startAuthorisationWithEncryptedPsuAuthentication" and this field has not yet been transmitted before.
PSU-ID-Type	String	Conditional	Type of the PSU-ID, needed in scenarios where PSUs have several PSU-IDs as access possibility. Shall be transmitted in this case, if this Request is indicated by "startAuthorisationWithPsuIdentification" or "startAuthorisationWithPsuAuthentication" or

Attribute	Type	Condition	Description
			"startAuthorisationWithEncryptedPsuAuthentication" and this field has not yet been transmitted before.
PSU-Corporate-ID	String	Conditional	Identification of a Corporate in the Online Channels. Shall be transmitted if this Request is indicated by "startAuthorisationWithPsuIdentification" or "startAuthorisationWithPsuAuthentication" or "startAuthorisationWithEncryptedPsuAuthentication" and this field has not yet been transmitted before, and only where generally needed in a corporate context.
PSU-Corporate-ID-Type	String	Conditional	This is describing the type of the identification needed by the ASPSP to identify the PSU-Corporate-ID content. Shall be transmitted if this Request is indicated by "startAuthorisationWithPsuIdentification". or "startAuthorisationWithPsuAuthentication" or "startAuthorisationWithEncryptedPsuAuthentication" and this field has not yet been transmitted before. Mean and use is defined in the ASPSP's documentation. Only used in a corporate context.
Authorization	String	Conditional	Bearer Token. Is contained only, if an OAuth2 based authentication was performed in a pre-step or an OAuth2 based SCA was performed in a preceding AIS service in the same session.
Client-SCA-Approach-Preference	String	Optional	A comma separated list of attributes, where the first entry will have a higher priority than the next or to every SCA Approach which is not indicated at all, e.g. "decoupled, redirect, embedded" or "decoupled". This attribute may be ignored by the ASPSP.
Client-Redirect-URI	String	Conditional	URI of the TPP, where the transaction flow shall be redirected to after a Redirect. Mandated for the Redirect SCA Approach, specifically when Client-Redirect-Preferred equals "true". See Section 5.3for further requirements on this header.

Attribute	Type	Condition	Description
			This field may be ignored by the ASPSP for migration reasons. For this reason, the same Client-Redirect-URI as used when creating the related resource shall be provided by the TPP. This specifically applies to the authorisation of a payment cancellation, where the same Client-Redirect-URI as for the corresponding payment initiation shall be used. This applies also to multilevel SCA, where the Client-Redirect-URI for all authorisation processes for one transaction shall be equal. It is recommended to always use this header field. **Remark for Future:** The condition on keeping the Client-Redirect-URI equal during a transaction lifecycle might be removed in the next version of the specification.
Client-Nok-Redirect-URI	String	Optional	If this URI is contained, the TPP is asking to redirect the transaction flow to this address instead of the Client-Redirect-URI in case of a negative result of the redirect SCA method. This may be ignored by the ASPSP. See Section 5.3 for further requirements on this header. The same condition as for Client-Redirect-URI on keeping the URI equal during a transaction lifecycle applies also to this header.

Request Body

No request body.

Note: If the hyperlinks in the following extended forms are used in the Transaction Initiation Response message or Transaction Cancellation Response message before, additional conditions on request body parameters apply as indicated in the following:

- "startAuthorisationWithPsuIdentification": Cp. Section 9.4.2
- "startAuthorisationWithPsuAuthentication": Cp. Section 9.4.3
- "startAuthorisationWithEncryptedPsuAuthentication": Cp. Section 9.4.3.
- "startAuthorsiationWithAuthenticationMethodSelection": Cp. Section 9.4.4.

The differences in the calls then are only whether to use a POST command to create the authorisation sub-resource and update the specified data at the same time or to use a PUT command to update the specified data to an already created authorisation sub-resource.

Response Code

HTTP response code equals 201.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
ASPSP-SCA-Approach	String	Conditional	Possible values are: • EMBEDDED • DECOUPLED • REDIRECT • ASPSP-CHANNEL OAuth will be subsumed by the value REDIRECT

Response Body

Attribute	Type	Condition	Description
transactionFees	Amount	Optional	Might be used by the ASPSP to transport the total transaction fee relevant for the underlying payments in case of a payment resource. This field includes the entry of the currencyConversionFees if applicable.
currencyConversion Fees	Amount	Optional	Might be used by the ASPSP to transport specific currency conversion fees related to the initiated credit transfer in case of a payment resource id.
estimatedTotalAmount	Amount	Optional	The amount which is estimated to be debted from the debtor account in case of a payment resource. Note: This amount includes fees.
estimatedInterbank SettlementAmount	Amount	Optional	The estimated amount to be transferred to the payee in case of a payment resource.

Attribute	Type	Condition	Description
scaStatus	SCA Status	Mandatory	
authorisationId	String	Mandatory	Unique resource identification of the created authorisation or cancellation authorisation sub-resource.
scaMethods	Array of Authentication Objects	Conditional	This data element might be contained, if SCA is required and if the PSU has a choice between different authentication methods. Depending on the risk management of the ASPSP this choice might be offered before or after the PSU has been identified with the first relevant factor, or if an access token is transported. If this data element is contained, then there is also a hyperlink of type "selectAuthenticationMethod" contained in the response body. These methods shall be presented towards the PSU for selection by the API Client.
chosenSca Method	Authentication Object	Conditional	This data element is only contained in the response if the ASPSP has chosen the Embedded SCA Approach, if the PSU is already identified e.g. with the first relevant factor or alternatively an access token, if SCA is required and if the authentication method is implicitly selected.
challengeData	Challenge	Conditional	It is contained in addition to the data element "chosenScaMethod" if challenge data is needed for SCA. In rare cases this attribute is also used in the context of the "updatePsuAuthentication" or "updateEncryptedPsuAuthentication" link.

Attribute	Type	Condition	Description
_links	Links	Mandatory	A list of hyperlinks to be recognised by the API Client. The actual hyperlinks used in the response depend on the dynamical decisions of the ASPSP when processing the request. **Remark**: All links can be relative or full links, to be decided by the ASPSP. Type of links admitted in this response, (further links might be added for ASPSP defined extensions): "scaRedirect": In case of an SCA Redirect Approach, the ASPSP is transmitting the link to which to redirect the PSU browser.
			"scaOAuth": In case of a SCA OAuth2 Approach, the ASPSP is transmitting the URI where the configuration of the Authorisation Server can be retrieved. The configuration follows the OAuth 2.0 Authorisation Server Metadata specification. "confirmation": Might be added by the ASPSP if either the "scaRedirect" or "scaOAuth" hyperlink is returned in the same response message. This hyperlink defines the URL to the resource which needs to be updated with • a confirmation code as retrieved after the plain redirect authentication process with the ASPSP authentication server or • an access token as retrieved by submitting an authorization code after the integrated OAuth based authentication process with the ASPSP authentication server.

Attribute	Type	Condition	Description
			"updatePsuIdentification": The link to the authorisation or cancellation authorisation sub-resource, where PSU identification data needs to be uploaded. "udpatePsuAuthentication": The link to the authorisation or cancellation authorisation sub-resource, where PSU authentication data needs to be uploaded. "udpateEncryptedPsuAuthentication": The link to the authorisation or cancellation authorisation sub-resource, where encrypted PSU authentication data needs to be uploaded "selectAuthenticationMethod": The link to the authorisation or cancellation authorisation sub-resource, where the selected authentication method needs to be uploaded. This link is contained under exactly the same conditions as the data element "scaMethods" "authoriseTransaction": The link to the authorisation or cancellation authorisation sub-resource, where the authorisation data has to be uploaded, e.g. the TOP received by SMS. "scaStatus": The link to retrieve the scaStatus of the corresponding authorisation sub-resource. "transactionFees": The link is to the status resource. This link is only added in case fee information is available via the status resource.
psuMessage	Max500Text	Optional	

Note: If the hyperlinks in the following extended forms are used in the Transaction Initiation Response or Transaction Cancellation Response message before, additional response parameters apply as indicated in the following:

- In case of "startAuthorisationWithPsuIdentification": Cp. Section 9.4.2
- In case of: "startAuthorisationWithPsuAuthentication": Cp. Section 9.4.3
- In case of: "startAuthorisationWithEncryptedPsuAuthentication": Cp. Section 9.4.3
- In case of: "startAuthorisationWithAuthenticationMethodSelection": Cp. Section 9.4.4.

Example

Request

```
POST https://api.testbank.com/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
PSU-ID:                 PSU-1234
```

Response

```
HTTP/1.x 201 CREATED
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
ASPSP-SCA-Approach:     DECOUPLED
Date:                   Sun, 06 Aug 2017 15:05:47 GMT
Location:               https://www.testbank.com/psd2/v2/payments/sepa-
credit-transfers/3d9a81b3-a47d-4130-8765-
a9c0ff861100/authorisations/3d9a81b3-a47d-4130-9999-a9c0ff861100
Content-Type:           application/json
{
 "scaStatus": "received",
 "authorisationId": "3d9a81b3-a47d-4130-9999-a9c0ff861100",
 "psuMessage": "Please use your BankApp for transaction authorisation.",
 "_links": {
   "scaStatus":  {"href": "/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/
authorisations/3d9a81b3-a47d-4130-9999-a9c0ff861100"}
   }
}
```

9.4.1 Update PSU Data

Independent of the SCA approach, the ASPSP might offer to select the SCA method before the SCA approach is defined, because e.g. the SCA Approach to be chosen might depend on the SCA method.

- A specific Update PSU Data Request is applicable for
    - the selection of authentication methods (see section 9.4.4), before choosing the actual SCA approach.

Depending on the SCA approach, different further PSU Data needs to be updated:

- Decoupled SCA Approach: A specific Update PSU Data Request is only applicable for
    - adding the PSU Identification (see section 9.4.2) or a PSU Authentication (see section 9.4.3, if not provided yet in the related Transaction Initiation Request and if no OAuth2 access token is used, or
    - the selection of authentication methods (see section 9.4.4).
- Embedded SCA Approach: The Update PSU Data Request might be used
    - to add credentials as a first factor authentication data of the PSU (see section 9.4.3) and
    - to select the authentication method (see section 9.4.4).

These different Update PSU Data Requests are differentiated in the following sub sections.

9.4.2 Update PSU Data (Identification)

This call is used, when in the preceding call the hyperlink of type "updatePsuIdentification" was contained, e.g. in case of a Decoupled Approach in the response and is now followed by the TPP.

Call

`PUT /v2/{resource-path}/{resourceId}/{authorisation-category}/{auhtorisationId}`
Updates the addressed authorisation sub-resource data on the server by PSU data, if requested by the ASPSP.

Path Parameters

Attribute	Type	Description
resource-path	String	This resource path can be a one-level parameter {service} or a two-level parameter {service}/product-type}, where • {service} stands for the service type of the related business transaction, e.g. /payments or /consents • {product-type} stands for the product-type of the related business transaction where applicable, e.g. sepa-

Attribute	Type	Description
		credit-transfers in the case of payments or account-access in case of consents.
resourceId	String	Resource identification of the related payment initiation, signing basket, consent, subscription or other related business transaction resource.
authorisation-category	String	The following two categories are supported: • `authorisations`: used in case of an authorisation of the related business transaction. • `cancellation-authorisations`: used in case of the cancellation authorisation of the related business transaction. Used only if applicable to the addressed {service}
authorisationId	String	Resource identification of the related authorisation sub-resource.

Query Parameters

No specific query parameters.

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
PSU-ID	String	Conditional	Contained if not yet contained in a preceding request, and mandated by the ASPSP in the related response
PSU-ID-Type	String	Conditional	Type of the PSU-ID, needed in scenarios where PSUs have several PSU-IDs as access possibility.
PSU-Corporate-ID	String	Conditional	Contained if not yet contained in a preceding request, and mandated by the ASPSP in the related response. This field is relevant only in a corporate context.
PSU-Corporate-ID-Type	String	Conditional	Might be mandated by the ASPSP in addition if the PSU-Corporate-ID is contained.

Request Body

No request body.

Response Code

HTTP response code is 200.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
ASPSP-SCA-Approach	String	Conditional	Possible values are: • EMBEDDED • DECOUPLED • REDIRECT • ASPSP-CHANNEL OAuth will be subsumed by the value REDIRECT

Response Body

Attribute	Type	Condition	Description
transactionFees	Amount	Optional	Might be used by the ASPSP to transport the total transaction fee relevant for the underlying payments. This field includes the entry of the currencyConversionFees if applicable.
currencyConversion Fees	Amount	Optional	Might be used by the ASPSP to transport specific currency conversion fees related to the initiated credit transfer.
estimatedTotalAmount	Amount	Optional	The amount which is estimated to be debted from the debtor account. **Note:** This amount includes fees.
estimatedInterbank SettlementAmount	Amount	Optional	The estimated amount to be transferred to the payee.
scaMethods	Array of Authentication Objects	Conditional	Might be contained, if several authentication methods are available. (name, type)

Attribute	Type	Condition	Description
_links	Links	Mandatory	A list of hyperlinks to be recognised by the TPP. The actual hyperlinks used in the response depend on the dynamical decisions of the ASPSP when processing the request. **Remark**: All links can be relative or full links, to be decided by the ASPSP. Type of links admitted in this response, (further links might be added for ASPSP defined extensions): "scaStatus": The link to retrieve the scaStatus of the corresponding authorisation sub-resource. "transactionFees": The link is to the status resource. This link is only added in case fee information is available via the status resource.
			"selectAuthenticationMethod": This is a link to a resource, where the TPP can select the applicable second factor authentication methods for the PSU, if there are several available authentication methods and if the PSU is already sufficiently authenticated.. If this link is contained, then there is also the data element "scaMethods" contained in the response body
scaStatus	SCA Status	Mandatory	
psuMessage	Max500Text	Optional	

Example

Request

```
PUT https://api.testbank.com/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations/3d9a81b3-
a47d-4130-9999-a9c0ff861100
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
PSU-ID:                 PSU-1234
```

Response

```
HTTP/1.x 200 OK
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
ASPSP-SCA-Approach:     DECOUPLED
Date:                   Sun, 06 Aug 2017 15:05:47 GMT
Content-Type:           application/json
{
 "scaStatus": "psuIdentified",
 "psuMessage": "Please use your BankApp for transaction Authorisation.",
 "_links": {
   "scaStatus":  {"href": "/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/
authorisations/3d9a81b3-a47d-4130-9999-a9c0ff861100"}
   }
}
```

9.4.3 Update PSU Data (Authentication) in the Decoupled or Embedded Approach

This call is used, when in the preceding call the hyperlink of type "updatePsuAuthentication", "updateEncryptedPsuAuthentication", or was contained in the response and is followed by the API Client.

Call

```
PUT /v2/{resource-path}/{resourceId}/{authorisation-
category}/{auhtorisationId}
```

Updates the addressed authorisation sub-resource data on the server by PSU data, if requested by the ASPSP.

Path Parameters

Attribute	Type	Description
resource-path	String	This resource path can be a one-level parameter {service} or a two-level parameter {service}/product-type}, where • {service} stands for the service type of the related business transaction, e.g. /payments or /consents • {product-type} stands for the product-type of the related business transaction where applicable, e.g. sepa-credit-transfers in the case of payments or account-access in case of consents.
resourceId	String	Resource identification of the related payment initiation, signing basket, consent, subscription or other related business transaction resource.

Attribute	Type	Description
authorisation-category	String	The following two categories are supported: • `authorisations`: used in case of an authorisation of the related business transaction. • `cancellation-authorisations`: used in case of the cancellation authorisation of the related business transaction. Used only if applicable to the addressed {service}
authorisationId	String	Resource identification of the related authorisation sub-resource.

Note: The openFinance API Framework allows ASPSPs to mandate a payload encryption to protect the password contained in the payload.

Query Parameters

No specific query parameters.

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
PSU-ID	String	Conditional	Contained if not yet contained in a preceding request, and mandated by the ASPSP in the related response
PSU-ID-Type	String	Conditional	Contained if not yet contained in a preceding request, and mandated by the ASPSP in the related response
PSU-Corporate-ID	String	Conditional	Contained if not yet contained in a preceding request, and mandated by the ASPSP in the related response. This field is relevant only in a corporate context.
PSU-Corporate-ID-Type	String	Conditional	Contained if not yet contained in a preceding request, and mandated by the ASPSP documentation. Might be mandated by the ASPSP in addition if the PSU-Corporate-ID is contained.

Request Body

Attribute	Type	Condition	Description
psuData	PSU Credentials	Mandatory	The password, or encryptedPassword, subfield is used, depending on encryption requirements of the ASPSP as indicated in the corresponding hyperlink contained in the preceding response message of the ASPSP. The related encryption requirements are defined in Section 7.2.

Response Code

HTTP response code equals 200.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
ASPSP-SCA-Approach	String	Conditional	Possible values are: • EMBEDDED • DECOUPLED • REDIRECT • ASPSP-CHANNEL OAuth will be subsumed by the value REDIRECT

Response Body

Attribute	Type	Condition	Description
chosenScaMethod	Authentication Object	Conditional	A definition of the provided SCA method is contained, if only one authentication method is available, and if the Embedded SCA approach is chosen by the ASPSP.
challengeData	Challenge	Conditional	Challenge data might be contained, if only one authentication method is available, and if the Embedded SCA approach is chosen by the ASPSP.

Attribute	Type	Condition	Description
scaMethods	Array of Authentication Objects	Conditional	Might be contained, if several authentication methods are available. (name, type)
accounts	Array of Account Details	Conditional	Provided, if the PSU account reference has not been provided in the related resource data submission and if more than one account is available.
_links	Links	Conditional	A list of hyperlinks to be recognised by the TPP. Might be contained, if several authentication methods are available for the PSU. Type of links admitted in this response: "updateAdditionalPsuAuthentication" The link to the payment initiation or account information resource, which needs to be updated by an additional PSU password. This link is only contained in rare cases, where such additional passwords are needed for PSU authentications. "updateAdditionalEncryptedPsuAuthentication" The link to the payment initiation or account information resource, which needs to be updated by an additional encrypted PSU password. This link is only contained in rare cases, where such additional passwords are needed for PSU authentications. "updateResourceByDebtorAccountResource" The link to the business resource which needs an update by any potential debtor account delivered in the "accounts" attribute above. "selectAuthenticationMethod": This is a link to a resource, where the TPP can select the applicable second factor authentication methods for the PSU, if there were several available authentication methods. This link is only contained, if the PSU is already identified or authenticated with the first

Attribute	Type	Condition	Description
			relevant factor or alternatively an access token, if SCA is required and if the PSU has a choice between different authentication methods. If this link is contained, then there is also the data element "scaMethods" contained in the response body "authoriseTransaction": The link to the resource, where the "Transaction Authorisation Request" is sent to. This is the link to the resource which will authorise the transaction by checking the SCA authentication data within the Embedded SCA approach. "scaStatus": The link to retrieve the scaStatus of the corresponding authorisation sub-resource. "transactionFees": The link is to the status resource. This link is only added in case fee information is available via the status resource.
scaStatus	SCA Status	Mandatory	
psuMessage	Max500Text	Optional	

NOTE: In case of an incorrect password, the ASPSP informs the TPP via the message code PSU_CREDENTIALS_INVALID. The TPP then needs to ask the PSU for re-entering the password. The newly entered password needs to be updated to the same path. It is recommended that the ASPSP is informing the TPP about this by adding a _links section in the additional error information and presenting a corresponding updatePsuAuthentication or updateEncryptedPsuAuthentication hyperlink. In case an incorrect password is entered too often, the ASPSP will at least put the related resource to a "Rejected" status. The ASPSP might inform the PSU via psuMessages about further impact.

NOTE: If an account list is provided for the selection of an account via the access method defined in Section 9.9, then the status of the related payment resource will be moved from "RCVD" to "PNDG", to indicate that further data needs to provided on resource level, before the authorisation can be completed. In addition, the ASPSP should provide a status reason code, providing more information on the pending reason.

Example

Request in case of Embedded Approach

```
PUT https://api.testbank.com/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations/3d9a81b3-
a47d-4130-9999-a9c0ff861100
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
PSU-ID:                 PSU-1234
{
"psuData": {
  "password": "start12"
  }
}
```

Response in case of the embedded approach

```
HTTP/1.x 200 OK
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
ASPSP-SCA-Approach:     EMBEDDED
Date:                   Sun, 06 Aug 2017 15:05:47 GMT
Content-Type:           application/json

{
  "scaStatus": "psuAuthenticated",
  "_links": {
   "authoriseTransaction": {"href": "/psd2/v2/payments/sepa-credit-
transfers/3A3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations/3d9a81b3-
a47d-4130-9999-a9c0ff861100"}
  }
}
```

9.4.4 Update PSU Data (Select Authentication Method)

This call is used, when in the preceding call the hyperlink of type "selectAuthenticationMethod" was contained in the response and was followed by the TPP.

Call

```
PUT /v2/{resource-path}/{resourceId}/{authorisation-
category}/{auhtorisationId}
```

Updates the addressed authorisation sub-resource data on the server by PSU data, if requested by the ASPSP.

Path Parameters

Attribute	Type	Description
resource-path	String	This resource path can be a one-level parameter {service} or a two-level parameter {service}/product-type}, where

Attribute	Type	Description
		• {service} stands for the service type of the related business transaction, e.g. /payments or /consents • {product-type} stands for the product-type of the related business transaction where applicable, e.g. sepa-credit-transfers in the case of payments or account-access in case of consents.
resourceId	String	Resource identification of the related payment initiation, signing basket, consent, subscription or other related business transaction resource.
authorisation-category	String	The following two categories are supported: • `authorisations`: used in case of an authorisation of the related business transaction. • `cancellation-authorisations`: used in case of the cancellation authorisation of the related business transaction. Used only if applicable to the addressed {service}
authorisationId	String	Resource identification of the related authorisation sub-resource.

Query Parameters

No specific query parameters.

Response Code

The HTTP response code equals 200.

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.

Request Body

Attribute	Type	Condition	Description
authentication MethodId	String	Mandatory	The authentication method ID as provided by the ASPSP.

Response Code

HTTP response code equals 200.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
ASPSP-SCA-Approach	String	Optional	Possible values are: • EMBEDDED • DECOUPLED • REDIRECT • ASPSP-CHANNEL OAuth will be subsumed by the constant REDIRECT

Response Body

Attribute	Type	Condition	Description
chosenSca Method	Authentication object	Conditional	A definition of the provided SCA method is contained, if the Embedded SCA approach is chosen by the ASPSP.
challengeData	Challenge	Conditional	Challenge data might be contained, if the Embedded SCA approach is chosen by the ASPSP.

Attribute	Type	Condition	Description
_links	Links	Conditional	A list of hyperlinks to be recognised by the TPP. The actual hyperlinks used in the response depend on the dynamical decisions of the ASPSP when processing the request. **Remark**: All links can be relative or full links, to be decided by the ASPSP. **Remark**: This method can be applied before or after PSU identification. This leads to many possible hyperlink responses. Type of links admitted in this response, (further links might be added for ASPSP defined extensions): "scaRedirect": In case of an SCA Redirect Approach, the ASPSP is transmitting the link to which to redirect the PSU browser. "scaOAuth": In case of a SCA OAuth2 Approach, the ASPSP is transmitting the URI where the configuration of the Authorisation Server can be retrieved. The configuration follows the OAuth 2.0 Authorisation Server Metadata specification. "confirmation": Might be added by the ASPSP if either the "scaRedirect" or "scaOAuth" hyperlink is returned in the same response message. This hyperlink defines the URL to the resource which needs to be updated with • a confirmation code as retrieved after the plain redirect authentication process with the ASPSP authentication server or • an access token as retrieved by submitting an authorization code after the integrated OAuth based authentication process with the ASPSP authentication server.

Attribute	Type	Condition	Description
			"updatePsuIdentification": The link to the authorisation or cancellation authorisation sub-resource, where PSU identification data needs to be uploaded. "updatePsuAuthentication": The link to the authorisation or cancellation authorisation sub-resource, where PSU authentication data needs to be uploaded. "updateEncryptedPsuAuthentication": The link to the authorisation or cancellation authorisation sub-resource, where encrypted PSU authentication data needs to be uploaded. "authoriseTransaction": The link to the authorisation or cancellation authorisation sub-resource, where the authorisation data has to be uploaded, e.g. the TOP received by SMS. "scaStatus": The link to retrieve the scaStatus of the corresponding authorisation sub-resource. "transactionFees": The link is to the status resource. This link is only added in case fee information is available via the status resource.
scaStatus	Sca Status	Mandatory	
psuMessage	Max500Text	Optional	

Example

Request in case of Embedded Approach

```
PUT https://api.testbank.com/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations/3d9a81b3-
a47d-4130-9999-a9c0ff861100
X-Request-ID:           asdfoeljkasdfoelkjasdf-123479093

{
authenticationMethodId: "myAuthenticationID"
}
```

Response in case of the embedded approach

```
HTTP/1.x 200 OK
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
ASPSP-SCA-Approach:     EMBEDDED
Date:                   Sun, 06 Aug 2017 15:05:47 GMT
Content-Type:           application/json

{
  "scaStatus": "scaMethodSelected",
  "chosenScaMethod": {
      "authenticationType": "SMS_OTP",
      "authenticationMethodId": "myAuthenticationID"},
  "challengeData": {
      "otpMaxLength": "6",
      "otpFormat": "integer"},
  "_links": {
     "authoriseTransaction": {"href": "/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations/3d9a81b3-
a47d-4130-9999-a9c0ff861100"}
  }
}
```

9.5 Transaction Authorisation

This call is only used in case of an Embedded SCA Approach.

Call

```
PUT /v2/{resource-path}/{resourceId}/{authorisation-
category}/{auhtorisationId}
```

Transmit response data to the challenge for SCA checks by the ASPSP.

Path Parameters

Attribute	Type	Description
resource-path	String	This resource path can be a one-level parameter {service} or a two-level parameter {service}/product-type}, where • {service} stands for the service type of the related business transaction, e.g. /payments or /consents • {product-type} stands for the product-type of the related business transaction where applicable, e.g. sepa-credit-transfers in the case of payments or account-access in case of consents.
resourceId	String	Resource identification of the related payment initiation, signing basket, consent, subscription or other related business transaction resource.
authorisation-category	String	The following two categories are supported: • `authorisations`: used in case of an authorisation of the related business transaction. • `cancellation-authorisations`: used in case of the cancellation authorisation of the related business transaction. Used only if applicable to the addressed {service}
authorisationId	String	Resource identification of the related authorisation sub-resource.

Query Parameter

No specific query parameters.

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
Authorization	String	Conditional	Is contained only, if the optional Oauth Pre-Step was performed.

Request Body

Attribute	Type	Condition	Description
scaAuthenticationData	String	Mandatory	SCA authentication data, depending on the chosen authentication method. If the data is binary, then it is base64 encoded.

Response Code

HTTP response code equals 200.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.

Response Body

Attribute	Type	Condition	Description
scaStatus	SCA Status	Mandatory	
_links	Links	Conditional	. A list of hyperlinks to be recognised by the TPP. The actual hyperlinks used in the response depend on the dynamical decisions of the ASPSP when processing the request. **Remark**: All links can be relative or full links, to be decided by the ASPSP. "transactionFees": The link is to the status resource. This link is only added in case fee information is available via the status resource.

NOTE: In case of incorrect scaAuthenticationData, the ASPSP informs the TPP via the message code PSU_CREDENTIALS_INVALID. The TPP then needs to ask the PSU for re-entering the authentication data by repeating the SCA method first. Depending on the implementation of the corresponding SCA method, the TPP needs

- either to re-start the full authorisation process by generating a new authorisation sub-resource, e.g. in case of an SMS OTP,

- or to submit newly generated authentication data generated on a customer device to the same path as the first time, and where no new challenge data from the ASPSP is needed, e.g. in case of a CHIP OTP.

The ASPSP is informing the TPP about this by adding a _links section in the additional error information and presenting a corresponding startAuthorisation, or transactionAuthorisation hyperlink. In case an incorrect OTP is entered too often, the ASPSP will at least put the related resource to a "Rejected" status. The ASPSP might inform the PSU via psuMessages about further impact.

Example

Request

```
PUT https://api.testbank.com/psd2/v2/payments/sepa-credit-transfers/
3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations/3d9a81b3-a47d-4130-
9999-a9c0ff861100
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
{
 "scaAuthenticationData": "123456"
}
```

Response in case of the embedded approach

Response Code 200

Response Body

```
{
  "scaStatus": "finalised",
  "_links": {
   "scaStatus":  {"href": "/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations/3d9a81b3-
a47d-4130-9999-a9c0ff861100"}
   }
}
```

9.6 Get Authorisation Sub-Resources Request

Call

```
GET /v2/{resource-path]/{resourceId}/{authorisation-category}
```

Will deliver an array of resource identifications of all generated authorisation or cancellation authorisation sub-resources.

Path Parameters

Attribute	Type	Description
resource-path	String	This resource path can be a one-level parameter {service} or a two-level parameter {service}/product-type}, where • {service} stands for the service type of the related business transaction, e.g. /payments or /consents • {product-type} stands for the product-type of the related business transaction where applicable, e.g. sepa-credit-transfers in the case of payments or account-access in case of consents.
resourceId	String	Resource identification of the related payment initiation, signing basket, consent, subscription or other related business transaction resource.
authorisation-category	String	The following two categories are supported: • `authorisations`: used in case of an authorisation of the related business transaction. • `cancellation-authorisations`: used in case of the cancellation authorisation of the related business transaction. Used only if applicable to the addressed {service}

Query Parameters

No specific query parameters defined.

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
Authorization	String	Conditional	Is contained only, if an OAuth2 based authentication was performed in a pre-step or an OAuth2 based SCA was performed in the current business transaction or in a preceding AIS service in the same session, if no such OAuth2 SCA approach was chosen in the current PIS transaction.

Request Body

No request body.

Response Code

The HTTP response code equals 200.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.

Response Body

Attribute	Type	Condition	Description
authorisationIds	Array of String	Mandatory	An array of all authorisationIds connected to the related business transaction.

Example

Request

```
GET https://api.testbank.com/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations
Accept:                 application/json
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7723
Date:                   Sun, 06 Aug 2017 15:04:07 GMT
```

Response

```
HTTP/1.x 200 Ok
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7723
Date:                   Sun, 06 Aug 2017 15:04:08 GMT
Content-Type:           application/json

{
  "authorisationIds": ["3d9a81b3-a47d-4130-9999-a9c0ff861100"]
}
```

9.7 GET Authorisation Status Request

Call

```
GET /v2/{resource-path}/{resourceId}/{authorisation-
category}/{auhtorisationId}
```

Checks the SCA status of an authorisation or cancellation authorisation sub-resource.

Path Parameters

Attribute	Type	Description
resource-path	String	This resource path can be a one-level parameter {service} or a two-level parameter {service}/product-type}, where • {service} stands for the service type of the related business transaction, e.g. /payments or /consents • {product-type} stands for the product-type of the related business transaction where applicable, e.g. sepa-credit-transfers in the case of payments or account-access in case of consents.
resourceId	String	Resource identification of the related payment initiation, signing basket, consent, subscription or other related business transaction resource.
authorisation-category	String	The following two categories are supported: • `authorisations`: used in case of an authorisation of the related business transaction. • `cancellation-authorisations`: used in case of the cancellation authorisation of the related business transaction. Used only if applicable to the addressed {service}
authorisationId	String	Resource identification of the related authorisation sub-resource.

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
Authorization	String	Conditional	Is contained only, if an OAuth2 based authentication was performed in a pre-step or an OAuth2 based SCA was performed in the current PIS transaction or in a preceding AIS service in the same session, if no such OAuth2 SCA approach was chosen in the current PIS transaction.

Query Parameters

No specific query parameters defined.

Request Body

No request body.

Response Code

The HTTP response code equals 200.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.

Response Body

Attribute	Type	Condition	Description
scaStatus	SCA Status	Mandatory	This data element is containing information about the status of the SCA method applied.
psuName	Max140Text	Optional	Name of the PSU[3] In case of a corporate account, this might be the person acting on behalf of the corporate.
_links	Links	Optional	Should refer to next steps if the problem can be resolved via the interface e.g. for re-submission of credentials.
apiClientMessages	Array of Client Message Information	Optional	Messages to the TPP on operational issues.

Example

Request

```
GET https://api.testbank.com/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations/3d9a81b3-
a47d-4130-9999-a9c0ff861100
Accept:                 application/json
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
```

[3] Usage is following the mandate resulting from EBA Q&A 2020_5165.

```
Date:                   Sun, 06 Aug 2017 15:04:07 GMT
```

Response

```
HTTP/1.x 200 Ok
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
Date:                   Sun, 06 Aug 2017 15:04:08 GMT
Content-Type:           application/json

{
  "scaStatus": "finalised"
}
```

9.8 Confirmation Request Procedure

The Confirmation Request Procedure is technically an optional extension of the authorisation process in case of the Redirect or OAuth SCA Approach. Its purpose is to prevent Cross-Site-Request-Forgery attacks (XSRF attacks) following the approach presented in [XS2A-SecB], chapter 3. Further details motivating the extension are also presented there.

The Confirmation Request is used, when in the preceding response the hyperlink of type "confirmation" was contained. This may (only) happen, if a redirection or OAuthSCA approach has been applied.

The processes connected to the Confirmation Request differ in details depending on whether the authorization has taken place using a Redirect approach that **does not involve the OAuth** protocol (described as "**Redirect SCA Approach**" in the following) or a Redirect Approach that **makes use of the OAuth protocol** ("**OAuth SCA Approach**"). Before the call can be submitted by the TPP, an authorization code (in case of the OAuth SCA Approach), respectively a confirmation code (in case of the Redirect SCA Approach) needs to be retrieved by the TPP after the SCA processing in a redirect to the ASPSP authentication server.

- In case of the OAuth SCA Approach, the overall procedure to receive the related authorization code and the access token succedingly is described in Section
- In case of the Redirect SCA Approach, the procedure to retrieve the confirmation code is described in Section 9.8.2. The actual Confirmation Request Message is described in Section 9.8.4 for both the OAuth2 SCA Approach and the Redirect SCA Approach after having defined the pre-conditions in both approaches in Section 9.8.3

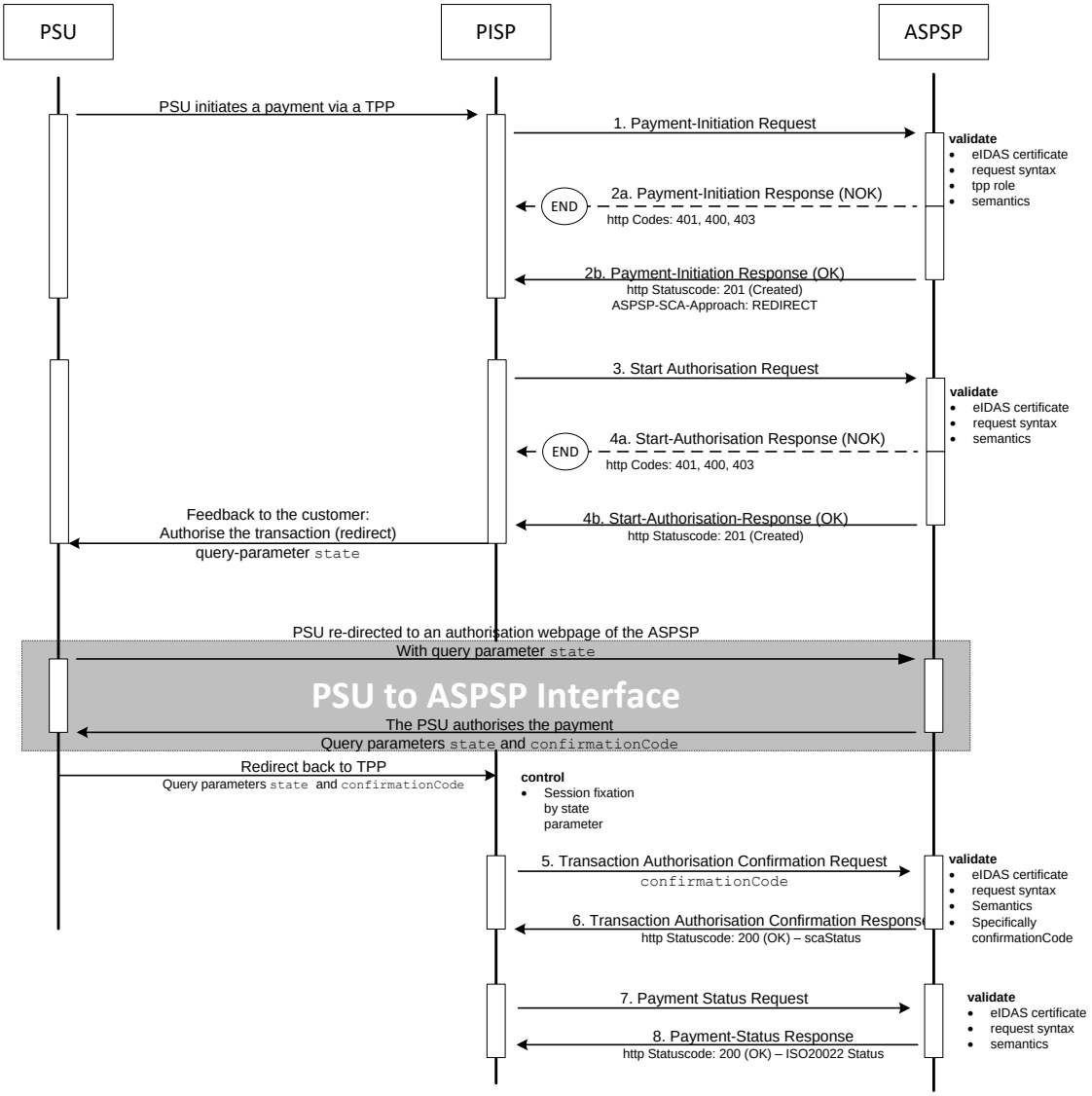
Due to the complexity of the process, flows are added below for payment initiation as an example before specifying the related requests in detail.

9.8.1 Confirmation Request Flow Examples for Payment Initiation

9.8.1.1 Redirect SCA Approach: Explicit Start of the Authorisation Process with Confirmation

If the ASPSP supports the Redirect SCA Approach, the message flow within the payment initiation service is simple. The Payment Initiation Request is followed by an explicit request of the TPP to start the authorisation. This is followed by a redirection to the ASPSP SCA

authorisation site. An authorisation confirmation request is requested by the TPP after the session is re-redirected to the TPP's system and after the TPP's control on session fixation. In the end, a payment status request might be needed by the TPP to control the exact status of the payment initiation.

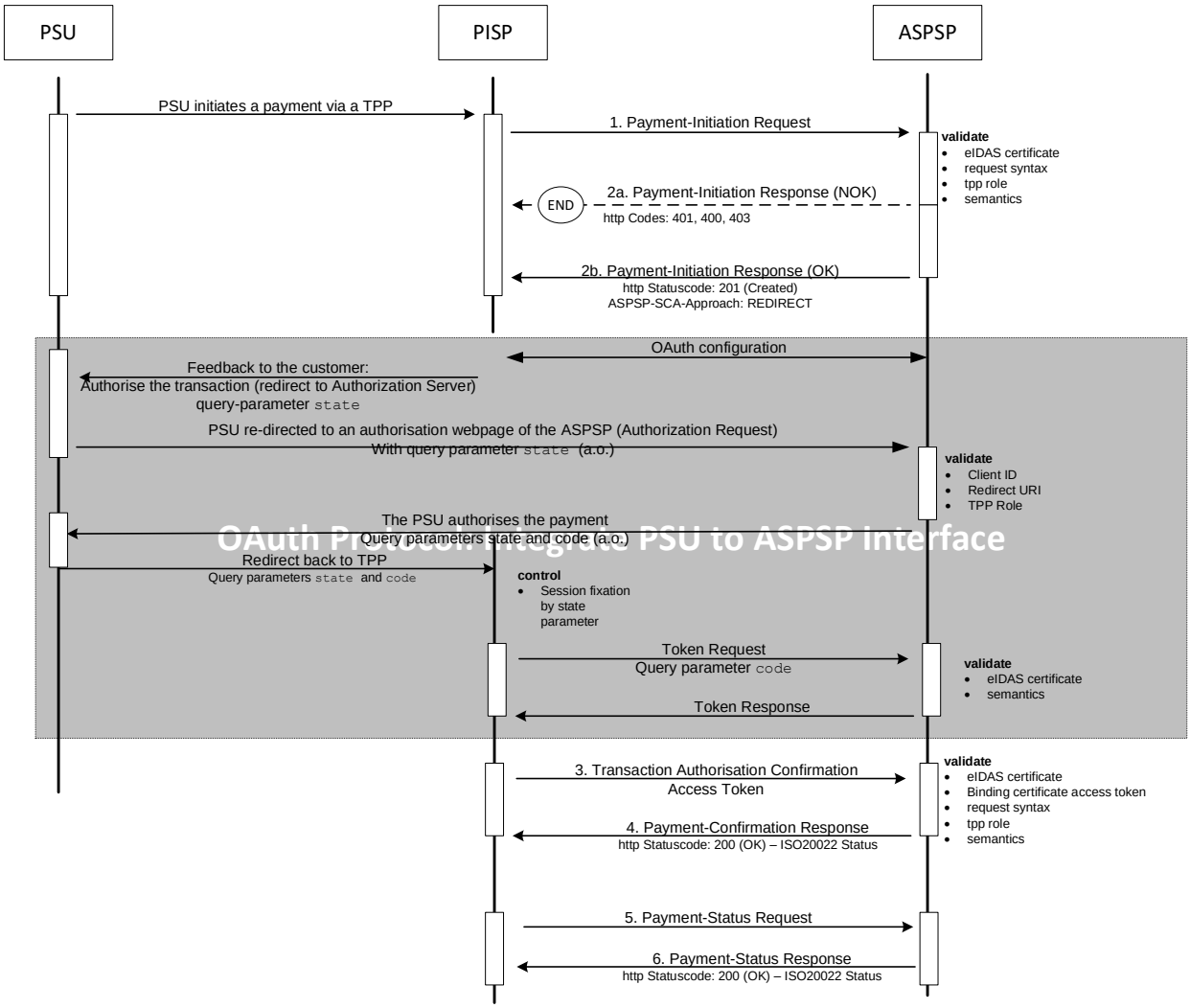


9.8.1.2 OAuth2 SCA Approach: Implicit Start of the Authorisation Process with Confirmation

If the ASPSP supports the OAuth2 SCA Approach, the flow is very similar to the Redirect SCA Approach with implicit start of the Authorisation Process. Instead of redirecting the PSU directly to an authentication server, the OAuth2 protocol is used for the transaction authorisation process. An authorisation confirmation request is required for the TPP in this scenario after the session is re-redirected to the TPP's system and after the TPP's control on

session fixation. In the end, a payment status request might be needed by the TPP to control the exact status of the payment initiation.

Remark: The OAuth2 SCA Approach with explicit start of the Authorisation Process is treated analogously.



It is further recommended for ASPSPs and TPPs in this case to follow the Security Best Practice definitions as defined in [OA-SecTop]

9.8.2 Retrieving the Confirmation Code in Redirect SCA approach

In the Confirmation Request Flow, the TPP will receive the confirmationCode as a query parameter in the HTTP request from the PSU agent that redirects the PSU back to the TPP after successful authorisation.

9.8.2.1 Preparation of the Confirmation Request Flow

Any authorisation process always involves the PSU directly and therefore requires a session between the PSU and the TPP. To support the Confirmation Request flow that might occur during the session between PSU and TPP, the TPP needs to fix the session of the PSU agent (browser or mobile app) on its server with a nonce, where part of it is a unique state parameter.

Before redirecting the PSU agent to send the authorization request (see section 9.8.2.2) to the ASPSP authorisation website, the TPP shall

- create a one-time use token to prevent XSRF attacks to be conveyed to the ASPSP in the query parameter state and,
- bind this value to the current session in the user (PSU) agent.

The confirmationCode is then retrieved from the response of the HTTP request of the PSU agent as described in following sub-section.

.

9.8.2.2 Redirecting the PSU agent (Authorisation Request)

To prepare the Confirmation Request flow, the TPP shall add the generated token to prevent XSRF attacks (see section 9.8.2.1) as an additional query parameter to that URI.

Additional Query Parameter in the PSU Authorisation Request (GET command)

Attribute	Type	Condition	Description
state	string	mandated	state parameter as defined by the TPP as a unique parameter to counter XSRF attacks and bound to the PSU/TPP session.

As a consequence of this requirement, it follows that ASPSPs shall not include a query parameter named "state" in their "scaRedirect" links.

Example

If the TPP received "`scaRedirect`" link containing the URI "`https://authserver.testbank.com`" only, the TPP shall forward the PSU agent to send a request

```
GET https://authserver.testbank.com?state=1234567er
```

If the TPP received "`scaRedirect`" link containing the URI "`https://authserver.testbank.com?bankProvidedQueryParameter=A`" the TPP shall forward the PSU agent to send a request

```
GET
https://authserver.testbank.com?bankProvidedQueryParameter=A&state=1
234567er
```

Response from the ASPSP Redirecting the PSU Agent back to the TPP

If the customer authentication via a Redirect SCA Approach ultimately fails, the customer is redirected to the URI assigned by the TPP in header field "Client-Nok-Redirect-URI". In this case, the business transaction / cancellation will anyhow not be authorised based in the

current flow. Therefore no additional steps are required from the TPP. The last part of this paragraph as well as the whole sections 9.8.3 to 9.8.4 only apply, if SCA has been successful.

After the customer authentication via SCA has taken place on the ASPSP server, the ASPSP responds to the PSU agent redirecting it to the URI provided by the TPP in the header "`Client-Redirect-URI`". To support the Confirmation Request flow, the ASPSP shall add the received query parameter "`state`" and a unique confirmationCode bound to the authorisation resource . The confirmationCode will only be contained if SCA has been successfully performed. Redirection is done by sending an HTTP response code 302 and a location header containing the URI to which the PSU is redirected. This would result in a response from the ASPSP to the PSU agent as described in the follows:

Response Code

The HTTP response code equals 302.

Response Header

Attribute	Type	Condition	Description
Location	String	Mandatory	The ASPSP's Authorization server must create the content of the Location Header in the following Form: [Client-Redirect-URI]?confirmationCode=[confirmation code]&state=[state] Where [Client-Redirect-URI] is the value of the "Client-Redirect-URI" header of the underlying request from the TPP [confirmationCode] is the ASPSP's confirmation code as described above [state] is the value of query parameter "state" from the PSU authorisation request

Remark: In case of the OAuth SCA approach, the PSU agent is forwarded similarly to the redirect_uri as defined in OAuth2 protocol. This redirection will also include a parameter state and – instead of the "confirmationCode" a parameter "code".

Example in case of Redirect SCA Approach

```
http 302
```

```
Location: www.example-TPP.com/xs2a-
client/v2/ASPSPidentifcation/mytransaction-
id?state=1234567er&confirmationCode=2256ffgh
```

9.8.3 Confirmation Request Message Pre-Condition

When the PSU is again redirected to the TPP as described above, the TPP will receive the additional query parameters "`state`" and "`confirmationCode`". The TPP must check whether the state parameter is linked to the current session as described in Section 9.8.2. If the check is positive then the TPP further processes

- within context of the OAuth SCA Approach with retrieving the access Bearer token as described in Section 8.8.3 of this document and then proceed as described in Section 9.8.4.
- within context of the Redirect SCA Approach directly as described in Section 9.8.4.

If the check fails, the transaction must be stopped by the TPP and the above defined request messages shall not be used.

9.8.4 Authorisation Confirmation Request Message

Call

```
PUT /v2/{resource-path}/{resourceId}/{authorisation-
category}/{auhtorisationId}
```

In case of the Redirect SCA approach, this call updates the addressed authorisation or cancellation authorisation data on the ASPSP server by a confirmation code.

In case of the OAuth2 SCA approach, this call updates the addressed authorisation or cancellation authorisation data on the ASPSP server with the OAuth2 access token, after having sent the related attribute "code" to the token server, cp. Section 8.8.3.

Both calls shall only be performed if requested by the ASPSP via the related hyperlink "confirmation" and if the related checks by the TPP as defined in Section 9.8.3 have been successful.

Path Parameters

Attribute	Type	Description
resource-path	String	This resource path can be a one-level parameter {service} or a two-level parameter {service}/product-type}, where • {service} stands for the service type of the related business transaction, e.g. /payments or /consents • {product-type} stands for the product-type of the related business transaction where applicable, e.g. sepa-

Attribute	Type	Description
		credit-transfers in the case of payments or account-access in case of consents.
resourceId	String	Resource identification of the related payment initiation, signing basket, consent, subscription or other related business transaction resource.
authorisation-category	String	The following two categories are supported: • `authorisations`: used in case of an authorisation of the related business transaction. • `cancellation-authorisations`: used in case of the cancellation authorisation of the related business transaction. Used only if applicable to the addressed {service}
authorisationId	String	Resource identification of the related authorisation sub-resource.

Query Parameters

No specific query parameters.

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
Authorization	String	Conditional	Authorization Bearer Token as retrieved by the TPP in case the OAuth SCA Approach as described in Section 8.8.4

Request Body

Attribute	Type	Condition	Description
confirmationCode	String	Conditional	Confirmation code as retrieved by the TPP within the Redirect SCA Approach as described in Section 9.8.2

Response Code

HTTP response code is 200.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.

Response Body

Attribute	Type	Condition	Description
scaStatus	SCA Status	Mandatory	Value "finalised" if the transaction authorisation/cancellation authorisation and confirmation was successful. Value "failed" if the transaction authorisation/cancellation authorisation or confirmation was not successful.
_links	Links	Mandatory	A list of hyperlinks to be recognised by the TPP. The actual hyperlinks used in the response depend on the dynamical decisions of the ASPSP when processing the request. **Remark**: All links can be relative or full links, to be decided by the ASPSP. Type of links admitted in this response, (further links might be added for ASPSP defined extensions): "status": The link to retrieve the status of the corresponding transaction resource.
psuMessage	Max512Text	Optional	

Example for OAuth SCA solution

Request

```
PUT https://api.testbank.com/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations/3d9a81b3-
a47d-4130-9999-a9c0ff861100
```

```
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
Authorization:          Bearer 1234567
```

Response

```
HTTP/1.x 200 OK
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
Date:                   Sun, 06 Aug 2017 15:05:47 GMT
Content-Type:           application/json
{
 "scaStatus": "finalised",
 "_links": {
   "status": {"href": "/psd2/v2/payments/sepa-credit-transfers/3d9a81b3-
a47d-4130-8765-a9c0ff861100/status"}
   }
```

Example for redirect solution

Request

```
PUT https://api.testbank.com/psd2/v2/payments/sepa-credit-
transfers/3d9a81b3-a47d-4130-8765-a9c0ff861100/authorisations/3d9a81b3-
a47d-4130-9999-a9c0ff861100
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
{"confirmationCode": "2256ffgh"}
```

Response

```
HTTP/1.x 200 OK
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
Date:                   Sun, 06 Aug 2017 15:05:47 GMT
Content-Type:           application/json
{
 "scaStatus": "finalised",
 "_links": {
   "status":  {"href": "/psd2/v2/payments/sepa-credit-transfers/3d9a81b3-
a47d-4130-8765-a9c0ff861100/status"}
   }
}
```

9.9 Update Resource with Debtor Account

The following method is used to update a payment related resource by an account which is associated to a PSU, and which needs to be received from the ASPSP first in the response of the "Update PSU Data with Authentication Request", cp. Section 9.4.3.

Remark: This procedure might be offered by the ASPSP in decoupled or embedded SCA channels to support "noManualIbanEntry" requirements from regulators for payment initiation.

Call

```
PUT /v2/{resource-path}/{resourceId}
```

Updates the addressed resource by an account related to the PSU.

Path Parameters

Attribute	Type	Description
resource-path	String	This resource path can be a one-level parameter {service} or a two-level parameter {service}/{product-type}, where • {service} stands for the service type of the related business transaction, e.g. /payments or /mandates • {product-type} stands for the product-type of the related business transaction where applicable, e.g. sepa-credit-transfers in the case of payments
resourceId	String	Resource identification of the related payment initiation or mandate submission.

Query Parameters

No specific query parameters.

Response Code

The HTTP response code equals 200.

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.

Request Body

Attribute	Type	Condition	Description
debtorAccount	Account Resource Reference	Mandatory	The resourceId of the account as provided in the response defined in Section 9.4.3 shall be used as the technical account reference.

NOTE: The support of this feature mandates the ASPSP to support UUIDs in the attributes resourceId of the data type "account details".

Response Code

HTTP response code equals 200.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
ASPSP-SCA-Approach	String	Optional	Possible values are: • EMBEDDED • DECOUPLED • REDIRECT • ASPSP-CHANNEL OAuth will be subsumed by the constant REDIRECT

Response Body

Attribute	Type	Condition	Description
chosenSca Method	Authentication Object	Conditional	A definition of the provided SCA method is contained, if only one authentication method is available, and if the Embedded SCA approach is chosen by the ASPSP.
challengeData	Challenge	Conditional	Challenge data might be contained, if only one authentication method is available, and if the Embedded SCA approach is chosen by the ASPSP.
scaMethods	Array of Authentication Objects	Conditional	Might be contained, if several authentication methods are available. (name, type)

Attribute	Type	Condition	Description
_links	Links	Conditional	A list of hyperlinks to be recognised by the TPP. The actual hyperlinks used in the response depend on the dynamical decisions of the ASPSP when processing the request. **Remark**: All links can be relative or full links, to be decided by the ASPSP. **Remark**: This method can be applied before or after PSU identification. This leads to many possible hyperlink responses. Type of links admitted in this response, (further links might be added for ASPSP defined extensions): "selectAuthenticationMethod": This is a link to a resource, where the TPP can select the applicable second factor authentication methods for the PSU, if there were several available authentication methods. This link is only contained, if the PSU is already identified or authenticated with the first relevant factor or alternatively an access token, if SCA is required and if the PSU has a choice between different authentication methods. If this link is contained, then there is also the data element "scaMethods" contained in the response body "authoriseTransaction": The link to the authorisation or cancellation authorisation sub-resource, where the authorisation data has to be uploaded, e.g. the TOP received by SMS. "scaStatus": The link to retrieve the scaStatus of the corresponding authorisation sub-resource. "transactionFees": The link is to the status resource. This link is only added in case fee information is available via the status resource.
psuMessage	Max500Text	Optional	

10 Signing Baskets

The signing basket is a function of the openFinance API Framework to bundle several transactions for authorisation. The SCA is then covering all transactions within the signing basket simultaneously. The ASPSP offering this service can restrict the bundling of transactions to certain transaction types, e.g. payments only.

10.1 Access Methods for Signing Baskets

Please note that the Establish Signing Basket Request is one instantiation of the Transaction Initiation Request as introduced in Section 2.2.5, i.e. the notion of authorisation sub resources and related access methods do also apply here but are specified generically in Section 9.

The following access methods are provided for signing baskets:

Endpoints	Method	Condition	Description
signing-baskets	POST	Optional	Creates a signing basket resource, where all transactions to be signed will be addressed within the body. Section 10.2
signing-baskets/{basketId}	GET	Mandatory	Retrieve the content of a signing basket resize. Section 10.3
signing-baskets/{basketId}/status	GET	Mandatory	Retrieve the singing basket status Section 10.4
signing-baskets/{basketId}	DELETE	Optional	Remove an existing signing basket. The transactions contained in the basket are not impacted by this call. Section 10.5

10.2 Establish Signing Basket Request

`POST /v2/signing-baskets/`

Generates a signing basket

NOTE: Please note that the Establish Signing Basket Request is one instantiation of the Transaction Initiation Request as introduced in Section 2.2.5, i.e. all related header parameters regarding PSU context, PSU Identification etc. will apply but will not be described in detail here. The related Open API files will offer all header parameters in detail.

Path Parameters

None.

Query Parameters

No Query Parameter

Request Header

Attribute	Type	Condition	Description
Content-Type	String	Mandatory	application/json
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
Authorization	String	Conditional	Bearer Token. Is contained only, if an OAuth2 based authentication was performed in a pre-step or an OAuth2 based SCA was performed in an preceding AIS service in the same session.
Consent-ID	String	Optional	This data element may be contained, if the signing basket transaction is part of a session, i.e. combined AIS/PIS service. This then contains the "consentId" of the related AIS one off consent, which was performed prior to this bulk signing.
PSU-IP-Address	String	Mandatory	The forwarded IP Address header field consists of the corresponding HTTP request IP Address field between PSU and TPP. If not available, the TPP shall use the IP Address used by the TPP when submitting this request.

Request Body

Attribute	Type	Condition	Description
paymentIds	Array of Max70Text	Optional	A non empty array of paymentIds.
consentIds	Array of Max70Text	Optional	A non empty array of consentIds.
subscriptionIds	Array of Max70Text	Optional	A non empty array of subcriptionIds.
subscriptionEntryIds	Array of Max70Text	Optional	A non empty array of subscriptionEntryIds.

Attribute	Type	Condition	Description
mandateResourceIds	Array of Max70Text	Opitonal	A non-empty array of mandateResourceIds

The body shall contain at least one entry.

Response Code

The HTTP response code equals 201.

Response Header

Attribute	Type	Condition	Description
Location	String	Mandatory	Location of the created resource (if created)
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.

Response Body

Attribute	Type	Condition	Description
transactionStatus	Transaction Status	Mandatory	The non payment related values might be used like RCVD or ACTC. For a list of all transactionStatus codes permitted for signing baskets, cp. Section 10.4.
basketId	Max70Text	Mandatory	resource identification of the generated signing basket resource.
scaMethods	Array of Authentication Objects	Conditional	This data element might be contained, if SCA is required and if the PSU has a choice between different authentication methods. Depending on the risk management of the ASPSP this choice might be offered before or after the PSU has been identified with the first relevant factor, or if an access token is transported. If this data element is contained, then there is also a hyperlink of type "startAuthorisationWith AuthenticationMethodSelection" contained in the response body.

Attribute	Type	Condition	Description
			These methods shall be presented towards the PSU for selection by the TPP.
chosenScaMethod	AuthenticationObject	Conditional	This data element is only contained in the response if the ASPSP has chosen the Embedded SCA Approach, if the PSU is already identified e.g. with the first relevant factor or alternatively an access token, if SCA is required and if the authentication method is implicitly selected.
challengeData	Challenge	Conditional	It is contained in addition to the data element "chosenScaMethod" if challenge data is needed for SCA.
			In rare cases this attribute is also used in the context of the "startAuthorisationWithPsuAuthentication" link.
_links	Links	Mandatory	A list of hyperlinks to be recognised by the API Client. The actual hyperlinks used in the response depend on the dynamical decisions of the ASPSP when processing the request. The potential links for this response message are generically defined in Section 3.7 for all Transaction Initiation Response messages which applies also to the Establish Signing Basket Response. These links will also be contained in the related Open API files. **Remark**: All links can be relative or full links, to be decided by the ASPSP.
psuMessage	Max500Text	Optional	Text to be displayed to the PSU
apiClientMessages	Array of Client Message Information	Optional	Messages to the API Client on operational issues.

Example

Request

```
POST https://api.testbank.com/psd2/v2/signing-baskets
Content-Type:           application/json
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
PSU-IP-Address:         192.168.8.78
```

```
PSU-GEO-Location:       GEO:52.506931;13.144558
PSU-User-Agent:         Mozilla/5.0 (Windows NT 10.0; WOW64; rv:54.0)
Gecko/20100101 Firefox/54.0
Date:                   Sun, 06 Aug 2017 15:02:37 GMT

{
   "paymentIds": ["3d9a81b3-a47d-4130-8765-a9c0ff861100", "3d9a81b3-a47d-
4130-8765-a9c0ff861110"]
}
```

Response (always with explicit authorisation start)

```
HTTP/1.x 201 Created
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
ASPSP-SCA-Approach:     REDIRECT
Date:                   Sun, 06 Aug 2017 15:02:42 GMT
Location:               https://www.testbank.com/psd2/v2/signing-
baskets/3d9a81b3-a47d-4130-8766-a9c0ff861100
Content-Type:           application/json

{
  "transactionStatus": "RCVD",
  "basketId": "3d9a81b3-a47d-4130-8766-a9c0ff861100",
  "_links": {
        "self": {"href": "/psd2/v2/signing-baskets/3d9a81b3-a47d-4130-8766-
a9c0ff861100"},
        "status": {"href": "/psd2/v2/signing-baskets/3d9a81b3-a47d-4130-
8766-a9c0ff861100/status"},
        "startAuthorisation": {"href": "/psd2/v2/signing-baskets/3d9a81b3-
a47d-4130-8766-a9c0ff861100/authorisations"}
  }
}
```

10.3 Get Signing Basket Request

Call

```
GET /v2/signing-baskets/{basketId}
```

Returns the content of a signing basket object.

Path Parameters

Attribute	Type	Description
basketId	String	ID of the corresponding signing basket object.

Query Parameters

No specific query parameter.

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
Authorization	String	Conditional	Is contained only, if an Oauth2 based authentication was performed in a pre-step or an Oauth2 based SCA was performed in the current PIS transaction or in a preceding AIS service in the same session, if no such Oauth2 SCA approach was chosen in the current signing basket transaction.

Request Body

No request body.

Response Code

The HTTP response code equals 200.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.

Response Body

Attribute	Type	Condition	Description
paymentIds	Array of Max70Text	Optional	A non-empty array of paymentIds.
consentIds	Array of Max70Text	Optional	A non-empty array of consentIds.
subscriptionIds	Array of Max70Text	Optional	A non-empty array of subcriptionIds.
subscriptionEntryIds	Array of Max70Text	Optional	A non-empty array of subscriptionEntryIds.
mandateResourceIds	Array of Max70Text	Optional	A non-empty array of mandateResourceIds.

Attribute	Type	Condition	Description
transactionStatus	Transaction Status	Mandatory	Only the not explicitly payment related codes like RCVD, PATC, ACTC, RJCT are used. For a list of all transactionStatus codes permitted for signing baskets, cp. Section 10.4.
_links	Links	Optional	The ASPSP might integrate hyperlinks to indicate next (authorisation) steps to be taken.

Example

Request

```
GET https://api.testbank.com/psd2/v2/signing-baskets/3d9a81b3-a47d-4130-8766-a9c0ff861100
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
Date:                   Sun, 06 Aug 2017 15:05:46 GMT
```

Response

```
HTTP/1.x 200 Ok
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
Date:                   Sun, 06 Aug 2017 15:05:47 GMT
Content-Type:           application/json

{
"paymentIds": ["3d9a81b3-a47d-4130-8765-a9c0ff861100", "3d9a81b3-a47d-4130-8765-a9c0ff861110"],
"transactionStatus": "ACTC"
}
```

10.4 Get Signing Basket Status Request

Call

```
GET /v2/signing-baskets/{basketId}/status
```

Returns the status of a signing basket object.

Path Parameters

Attribute	Type	Description
basketId	String	ID of the corresponding signing basket object.

Query Parameters

No specific query parameter.

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
Authorization	String	Conditional	Is contained only, if an OAuth2 based authentication was performed in a pre-step or an OAuth2 based SCA was performed in the current PIS transaction or in a preceding AIS service in the same session, if no such OAuth2 SCA approach was chosen in the current signing basket transaction.

Request Body

No request body.

Response Code

The HTTP response code equals 200.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.

Response Body

Attribute	Type	Condition	Description
transactionStatus	Transaction Status	Mandatory	Only the codes RCVD, PATC, ACTC, CANC and RJCT are supported for signing baskets.

Example

Request

```
GET https://api.testbank.com/psd2/v2/signing-baskets/3d9a81b3-a47d-4130-
8766-a9c0ff861100/status
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
Date:                   Sun, 06 Aug 2017 15:05:49 GMT
```

Response

```
HTTP/1.x 200 Ok
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
Date:                   Sun, 06 Aug 2017 15:05:51 GMT
Content-Type:           application/json

{
"transactionStatus": "ACTC"
}
```

10.5 Cancellation of Signing Baskets

A cancellation of a Signing Basket is only permitted where no (partial) authorisation has been applied for the Signing Basket.

Call

```
DELETE /v2/signing-baskets/{basketId}
```

Deletes a created signing basket if it is not yet (partially) authorised.

Path Parameters

Attribute	Type	Description
basketId	String	Contains the resource-ID of the signing basket to be deleted.

Query Parameters

No specific query parameters.

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.
Authorization	String	Conditional	Is contained only, if an OAuth2 based SCA has been used in a pre-step.

Request Body

No Request Body.

Response Code

The HTTP response code is 204 in case of successful deletion.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.

Response Body

No Response Body

Example

Request

```
DELETE https://api.testbank.com/psd2/v2/signing-baskets/3d9a81b3-a47d-4130-
8766-a9c0ff861100
X-Request-ID            99391c7e-ad88-49ec-a2ad-99ddcb1f7757
Date                    Sun, 13 Aug 2017 17:05:37 GMT
```

Response

```
HTTP/1.x 204 No Content
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7757
Date:                   Sun, 13 Aug 2017 17:05:38 GMT
```

11 Resource Status Notification

The openFinance API Framework is supporting a resource status notification function, formally specified in an alone standing document as Resource Status Notification Service before. This is now covered within this section as a basic building block for the openFinance API Framework V2.

This resource status notification function allows ASPSP to push status changes of a resource created by the API Client within the openFinance API Framework. The API Client needs to register for receiving such notification messages for every resource where such a function is envisaged. This registration is steered by the API Client by adding certain headers within the related Transaction Initiation Request message, which creates the addressed resource.

The related header parameters as defined in Section 11.4 will not be repeated on service implementation guideline level, but will still be contained in OpenAPI files as an instantiation to service specifications.

The resource status notification function is an optional feature within the openFinance API Framework, i.e. a related registration by the API Client might be ignored by the ASPSP. Requirements on TLS support for the actual push function (turning the ASPSP into a technical http client towards the API Client) are defined in Section 5.2.

11.1 API Access Methods

The following table gives an overview on the HTTP access methods supported by the notification API endpoints of the API Client for receiving notification messages on resource status changes by the ASPSP.

Endpoint	Method	Condition	Description
<Client-Notification-URI>	POST	Conditional	Notification initiated by ASPSP, endpoint provided by the API Client. This command posts notification content to the provided endpoint. This access method shall be supported by the API Client if a Client-Notification-URI is provided by the API Client in a previous call to the XS2A Interface or openFinance API of the ASPSP.

11.2 HTTP Response Codes for Notifications

The HTTP response code is communicating the success or failure of an API Client request message. The 4XX HTTP response codes should only be given if the current request cannot be fulfilled, e.g. the syntax of the body content is not correct.

This specification supports the following HTTP response codes for the Client Notification API:

Status Code	Description
200 OK	POST for a notification
400 Bad Request	Validation error occurred. This code will cover malformed syntax in request or incorrect data in payload.
401 Unauthorized	The API Client or the PSU is not correctly authorized to perform the request. Retry the request with correct authentication information.
403 Forbidden	Returned if the resource that was referenced in the path exists but cannot be accessed by the ASPSP. This code should only be used for non-sensitive id references as it will reveal that the resource exists even though it cannot be accessed.
404 Not found	Returned if the endpoint that was referenced in the path does not exist or cannot be referenced by the ASPSP. When in doubt if a specific id in the path is sensitive or not, use the HTTP response code 404 instead of the HTTP response code 403.
405 Method Not Allowed	This code is only sent when the HTTP method (PUT, POST, DELETE, GET etc.) is not supported on a specific endpoint.
408 Request Timeout	The server is still working correctly, but an individual request has timed out.
415 Unsupported Media Type	The ASPSP has supplied a media type which the API Client does not support.
500 Internal Server Error	Internal server error occurred.
503 Service Unavailable	The API Client server is currently unavailable. Generally, this is a temporary state.

11.3 Implicit Subscription for Resource Status Notification

This section describes how an API Client is registering for the resource status notification function within the Transaction Initiation Request message, e.g. a Payment Initiation Message or an Establish Consent Request message.

NOTE: The notification services will also be available for cancellation processes which require SCA based authentication of PSUs. These services will then be supported by the ASPSP if requested before by the API Client for the related resource initiation process. So, the "notification service" support function is stored within the created resource.

11.4 Communicate Notification URI of API Clients to the ASPSP

Call

```
Any POST command
```

- `creating a resource to be authorised within the openFinance API Framework.`

Creates a corresponding resource in the ASPSP server.

Path Parameters

No specific requirements for the subscription of the resource status notification function.

Query Parameters

No specific requirements for the subscription of the resource status notification function.

Request Header

The following table contains only the request headers which have to be supported by the TPP or the API Client more generally in addition to headers defined for the corresponding resource creation request.

Attribute	Type	Condition	Description
Client-Notification-URI	String	Optional	URI for the Endpoint of the Client API to which the status of the resource should be sent. This header field may by ignored by the ASPSP if the resource status push function is not supported for the related API client.
Client-Notification-Content-Preferred	String	Optional	The string has the form status=X1, …, Xn where Xi is one of the constants SCA, PROCESS, LAST and where constants are not repeated. The usage of the constants supports the following semantics: SCA: A notification on every change of the scaStatus attribute for all related authorisation processes is preferred by the API Client.

Attribute	Type	Condition	Description
			PROCESS: A notification on all changes of resource status attributes is preferred by the API Client. LAST: Only a notification on the last resource status as available in the XS2A interface or openFinance API is preferred by the API Client. This header field may be ignored, if the ASPSP does not support the resource notification push function for the related API Client.

Request Body

No specific requirements.

Response Code

No specific requirements

Response Header

The following table contains only the response headers which have to be supported by the ASPSP in addition to headers defined for the corresponding resource creation response if the resource status push function is supported.

Attribute	Type	Condition	Description
ASPSP-Notification-Support	Boolean	Conditional	true if the ASPSP supports the resource status push function for the created resource. false if the ASPSP supports resource status push function in general, but not for the current request. One of the reasons of no support is that the API Client had not explicitly asked for the function in the request. Not used, if the resource status notification function is generally not supported by the ASPSP. Shall be supported if the ASPSP generally supports the resource status notification function.
ASPSP-Notification-Content	String	Conditional	The string has the form status=X1, …, Xn

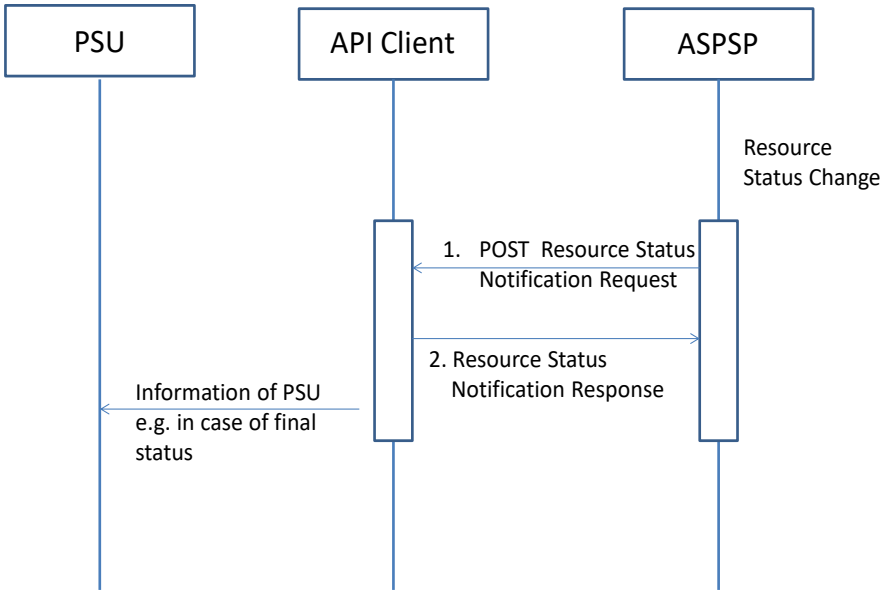
Attribute	Type	Condition	Description
			where Xi is one of the constants SCA, PROCESS, LAST and where constants are not repeated. The usage of the constants supports the following semantics: SCA: Notification on every change of the scaStatus attribute for all related authorisation processes is provided by the ASPSP for the related resource. PROCESS: Notification on all changes of the resource status attributes is provided by the ASPSP for the related resource. LAST: Notification on the last resource status after authorisation or cancellation as available in the related serviceI is provided by the ASPSP for the related resource. This field shall be provided if the ASPSP-Notification-Support =true. The ASPSP might consider the notification content as preferred by the TPP, but can also respond independently of the preferred request.

Response Body

No specific requirements.

11.5 Resource Status Notification Message Flow

The following flow shows the simple request and response flow for a resource status notification function:



Remark: In case, where the ASPSP is only pushing a status hyperlink to the API Client, the API Client needs to check the resource status after step 2.) before informing e.g. the PSU.

11.6 Push Resource Status with JSON encoding

Call

```
POST <Client-Notification-URL>
```

Creates a Resource Notification on the API Client server.

Path Parameters

No Path Parameter

Query Parameters

No Query Parameter

Request Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the request, unique to the call, as determined by the initiating party.

Request Body

Attribute	Type	Condition	Description
paymentId	Max70Text	{Or	This shall be contained, if the push notification is about a payment or RTP initiation.
consentId	Max70Text	Or	This shall be contained if the push notification is about establishing a consent.
subscriptionId	Max70Text	Or	This shall be contained if the push notification is about establishing a subscription.
basketId	Max70Text	Or	This shall be contained if the push notification is about signing a basket.
mandateResourceId	Max70Text	Or}	This shall be contained if the push notification is about establishing a mandate.
entryId	Max70Text	{Or – Optional	This may be used if the status relates to an entry of an RTP bulk.
subscriptionEntryId	Max70Text	Or – Optional}	This may be used if the status relates to an entry of a subscription.
authorisationId	Max70Text	{Or - Optional	This attribute should be contained if the push notification is about a specific SCA status.
cancellationId	Max70Text	Or - Optional}	This attribute should be contained if the push notification is about a specific SCA status of a cancellation authorisation sub-resource.
transactionStatus	Transaction Status	{Or Optional	This attribute might be contained if the related resource contains a transaction status which has changed.
consentStatus	Consent Status	Or Optional	This attribute might be contained if the consent status of the addressed resource has changed.

Attribute	Type	Condition	Description
subscriptionStatus	Subscripton Status	Or Optional	This attribute might be contained if the subscription status of the addressed resource has changed.
subscriptionEntryStatus	Subscription Entry Status	Or Optional	This attribute might be contained if the subscription entry status of the addressed resource has changed.
mandateStatus	Mandate Status	Or optional}	This attribute might be contained if the mandate status of the addressed resource has changed.
scaStatus	SCA Status	Optional	This attribute might be contained if the authorisation status of the addressed authorisation resource has changed.
requestStatus	Request Status	Optional	The status of the related request to pay transaction. To be delivered by the API Server if not agreed otherwise.
reasonCode	Status Reason Code	{Or Optional	Additional information on the reason for e.g. rejecting the request
reasonProprietary	Max35Text	Or Optional}	Proprietary additional information on the reason for e.g. rejecting the request.
debtorDecision DateTime	ISO Date Time	Optional	The date and time when the PSU has decided on accepting/rejecting the related request.
acceptedAmount	Amount	Optional	Contained only if the accepted amount deviates from the instructed amount.
acceptanceDateTime	ISODateTime	Optional	Contained only if the agreed requested execution date deviates from the requested execution date in the request.
acceptedPaymentInstrument	Max105Text	Optional	“SCT” or “SCT inst” as default values.
statusIdentification	Max35Text	Optional	Reference added by the debtor.

Attribute	Type	Condition	Description
_links	Links	Optional	The following link types are supported.
			scaStatus This shall be contained if the related SCA status is not reported at the same time by the scaStatus attribute. The API Client then needs to get the scaStatus by a GET command using this hyperlink.
			status This shall be contained if the related consent or transaction status is not reported at the same time. The API Client then needs to get the resource status by a GET command using this hyperlink.

HTTP Response Code

200

Remark: All response codes which do not equal 200 are ignored by the ASPSP. The notification will not be repeated.

Response Header

Attribute	Type	Condition	Description
X-Request-ID	UUID	Mandatory	ID of the corresponding request, unique to the call, as determined by the initiating party.

Response Body

No Response Body

Example
Request

```
POST https://notifications.testclient.com/v2/transaction-12345
Content-Type:           application/json
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
Date:                   Sun, 06 Aug 2017 15:02:37 GMT

{
  "paymentId": "3d9a81b3-a47d-4130-8765-a9c0ff861100",
```

```
  "transactionStatus": "ACFC"
}
```

Response

```
HTTP/1.x 200
Content-Type:           application/json
X-Request-ID:           99391c7e-ad88-49ec-a2ad-99ddcb1f7721
Date:                   Sun, 06 Aug 2017 15:04:08 GMT
```

12 Annex

12.1 List of tables

12.2 References

12.2.1 Documents of the NextGenPSD2 XS2A Framework

[XS2A-SecB] NextGenPSD2 XS2A Framework, Security Bulletin, Version 1.1, 30 October 2020

12.2.2 Documents of the openFinance API Framework

[oFA DD] openFinance API Framework, Data Dictionary for V2.x, Version 2.2, 31 July 2024

[oFA-IG-ADM] openFinance API Framework, Implementation Guidelines for Administrative Service, Version 1.0, 21 February 2024

12.2.3 Further documents

[EBA-FR] Final Report, Draft Regulatory Technical Standards, amending Commission Delegated Regulation (EU) 2018/389 supplementing Directive (EU) 2015/2366 of the European Parliament and of the Council with regard to regulatory technical standards for strong customer authentication and common and secure open standards of communication, published 5 April 2022

[EBA-OP2] Opinion of the European Banking Authority on obstacles under Article 32(3) of the RTS on SCA and CSC, EBA/OP/2020/10, published 4 June 2020

[EBA-RTS] Commission Delegated Regulation (EU) 2018/389 of 27 November 2017 supplementing Directive 2015/2366 of the European Parliament and of the Council with regard to Regulatory Technical Standards for Strong Customer Authentication and Common and Secure Open Standards of Communication, C(2017) 7782 final, published 13 March 2018

[eIDAS] Regulation (EU) No 910/2014 of the European Parliament and of the Council on Electronic Identification and Trust Services for Electronic Transactions in the Internal Market, 23 July 2014, published 28 August 2014

[ETSI EN 319 132-1] ETSI European Standard, Electronic Signatures and Infrastructures (ESI); XAdES digital signatures; Part 1: Building blocks and XAdES baseline signatures, V1.1.1 (2016-04)

[ETSI TS 119 182-1] ETSI Technical Specification, Electronic Signatures and Infrastructures (ESI); JAdES digital signatures; Part 1: Building blocks and JAdES baseline signatures, V1.2.1 (2022-02)

[ETSI TS 119 495] ETSI Technical Specification, Electronic Signatures and Infrastructures (ESI); Sector Specific Requirements; Certificate Profiles and TSP Policy Requirements for Open Banking, V1.5.1 (2021-04)

[FAPI-CBPIA] OpenID Foundation, Financial-grade API (FAPI) Working Group, Cross-Browser Payment Initiation Attack, https://bitbucket.org/openid/fapi/src/master/TR-Cross_browser_payment_initiation_attack.md, 3.01.2019

[GCMencryp] D.A. McGrew, J. Viega, The Galois/Counter Mode of Operation (GCM), https://luca-giuzzi.unibs.it/corsi/Support/papers-cryptography/gcm-spec.pdf

[HAL] Kelley, M., "HAL - Hypertext Application Language", 2013-09-18, http://stateless.co/hal_specification.html

[OA-SecTop] OAuth 2.0 Security Best Current Practice draft-ietf-oauth-security-topics-13, Lodderstedt et al., 8 July 2019, https://tools.ietf.org/html/draft-ietf-oauth-security-topics-13

[OBEsign] Open Banking Europe: JSON Web Signature Profile for Open Banking, Version 001-001, 18.05.2021, https://www.openbankingeurope.eu/media/2095/obe-json-web-signature-profile-for-open-banking.pdf

[PSD2] Directive (EU) 2015/2366 of the European Parliament and of the Council on payment services in the internal market, published 23 December 2015

[RFC2426] Dawson, F. and T. Howes, T., "vCard MIME Directory Profile", September 1998, https://tools.ietf.org/html/rfc2426

[RFC3230] Mogul, J. and A. Van Hoff, "Instance Digests in HTTP", RFC 3230, DOI 10.17487/RFC3230, January 2002,https://www.rfc-editor.org/info/rfc3230

[RFC3986] T. Berners-Lee, R. Fielding and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005, https://tools.ietf.org/html/rfc3986

[RFC4648] Josefsson, S.," The Base16, Base32, and Base64 Data Encodings", October 2006, https://tools.ietf.org/html/rfc4648

[RFC5280] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, May 2008, https://datatracker.ietf.org/doc/html/rfc5280

[RFC5843] Bryan, A, "Additional Hash Algorithms for HTTP Instance Digests", RFC 5843, DOI 10.17487/RFC5843, April 2010, https://www.rfc-editor.org/info/rfc5843

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", October 2012, https://tools.ietf.org/html/rfc6749

[RFC6901] Bryan, P., Nottingham, M., "JavaScript Object Notation (JSON) Pointer", April 2013, https://tools.ietf.org/html/rfc6901

[RFC7231] R. Fielding, J. Reschke, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content

[RFC7515] Jones, Bradley, Sakimura, "JSON Web Signatures (JWS)", May 2015, https://datatracker.ietf.org/doc/rfc7515/

[RFC7516] Jones, Hildebrand: "JSON Web Encryption (JWE)", May 2015, https://datatracker.ietf.org/doc/rfc7516/

[RFC7518] Jones, "JSON Web Algorithms (JWA)", May 2015, https://datatracker.ietf.org/doc/rfc7518/

[RFC7519] Jones, Bradley, Sakimura: "JSON Web Token (JWT), May 2015, https://datatracker.ietf.org/doc/html/rfc7519

[RFC 7797] Jones, "JSON Web Signature (JWS) Unencoded Payload Option", February 2016, https://datatracker.ietf.org/doc/rfc7797/

[RFC7807] M. Nottingham, Akamai, E. Wilde, „Problem Details for HTTP APIs“, March 2016, https://tools.ietf.org/html/rfc7807

[RFC 8414] M. Jones, N. Sakimura, J. Bradley; "OAuth 2.0 Authorization Server Metadata"; June 2018, https://www.rfc-editor.org/rfc/rfc8414.html

[RFC 8705] B. Campbell, J. Bradley, N. Sakimura; "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens"; February 2020,

[RFC9457] M. Nottingham, E. Wilde, S. Dalal, "Problem Details for HTTP APIs", July2023, https://www.rfc-editor.org/rfc/rfc9457.html

[XML ENC] W3C Recommendation: "XML Encryption Syntax and Processing, Version 1.1, 11 April 2013

[W3C XMLSig] W3C Recommendation: "XML Signature and Processing", Version 1.1, 11 April 2013

[W3C XMLSig V2] W3C Recommendation: "XML Signature and Processing", Version 2.0, 23 July 2015

12.3 Detailed Change Log

12.3.1 Changes from Version 2.0 to 2.1

The following changes have been applied in version 2.1 relative to version 2.0:

Section	Change	Reason
3.5	Idempotency requirements have been weakened.	Following general web idempotency recommendations, the requirements on idempotency have been weakened, since strict idempotency is difficult to be handled. (CR116)
4.2.2	The attribute "path" has been re-named to "instance" for error messaging following [RFC7807] and explicitly extended for "status" to repeat the http response code on application level.	Erratum
4.2.2	A note was added, that the new standard [RFC9457] for http problem reports might be used instead of the older [RFC7807]. This is a new issue and might still be changed during the consultation of the document.	New standard for error reporting.
5.3	Rename TPP-Redirect-URI to Client-Redirect-URI Rename TPP-Nok-Redirect-URI to Client-Nok-URI	Make these parameters also usable for direct access models for corporates. Prepare the API Framework better for corporate direct access.
6.2	Clarifications and extensions to message signing: Adding the attributes typ and aud for the JWS protected header,	Secure parts of the path, which contains important references related to the transaction to be secured. (CR 112)

Section	Change	Reason
	clarifications on the alg attribute, adding api-contract-id to the sigD attribute. Extending existing examples	
6.2.2.2	Reference to certificate in TPP-Signature-Certificate deleted, since this header is not any more supported.	Erratum
8.3	Restricting the length of PSU related attributes in the http headers	Clarifications of string length supported for some http headers.
8.4.1, 9.4	Rename TPP-SCA-Preference to Client-SCA-Preference. Rename TPP-Explicit-Authorisation-Preferred to Client-Explicit-Authorisation-Preferred	Make these parameters also usable for direct access models for corporates. Prepare the API Framework better for corporate direct access.
8.4.2, 9.4,	Add the value ASPSP-CHANNEL to the ASPSP-SCA-APPROACH header.	Adding asynchronous SCA via ASPSP channels explicitly to the Framework resulting from new services, see below.
8.8.1	Adding "MAN:<resourceId>" as a scope standard for the OAuth2 based authorisation.	Preparation of the API Framework to support mandate authorisation.
New Section 8.9	Adding a section for introducing the ASPSP-Channel SCA approach (for premium services)	Resulting from new services (Mandate Services) where also an asynchronous authorisation by the PSU via ASPSP online channels is explicitly supported.
9.4.2	Add fee structures to the response as optional attributes.	Erratum.
9.4.3	Extend the response by the option to add directly an account list to select an IBAN from and a related hyperlink. This is an optimised solution to provide "No manual IBAN entry" for the embedded SCA approach as well potentially for the decoupled SCA approach. A remark was added that the status of the related payment resource	CR113

Section	Change	Reason
	will be transposed to "PNDG" to make clear on the resource level, that resource data needs to be pushed before further authorisation steps can be applied.	
9.7, 10.2	Rename tppMessages to apClientMessages.	Erratum
New Section 9.9	Add an access method, to post a debtor account to a payment related resource after the selection of an IBAN after having submitted a password e.g. in the embedded SCA approach, see above.	CR113
10.2	Add mandateResourceIds to the body of the signing basket as an additional attribute class, which can be referred to.	New Mandate API Service needs an extension to the signing basket definition.
10.3	Add mandateResourceIds to the response body of the retrieval of a signing basket resource. Correct all data types string to Max70Text	New Mandate API Service needs an extension to the signing basket definition.
11.6	Add mandateStatus as attribute for pushing status changes	New Mandate API Service needs an extension to the resource status notification function.
11.6	Change all data types of resource identification from String to Max70Text	Erratum