STET PSD2 API

Documentation Part 1: Framework

Author: Robache Hervé

Date: 2022-10-03

Version: 1.6.3.1 (English)

Table of content

1. Introduction

1.1. Context

The revised Payment Service Directive (PSD2) points out some new roles providing services to a Payment Service User (PSU):

- Third Party Providers (TPP) which can be subdivided into three categories
  - Account Information Service Providers (AISP)
  - Payment Initiation Service Providers (PISP)
  - Card Based Payment Instrument Issuers (CBPII)
- Account Servicing Payment Service Providers (ASPSP).

Each Member Country has to transpose the PSD2, within its own national law.

The PSD2 is completed by a set of documents provided by the European Banking Authority (EBA). Among these documents, the Regulatory Technical Standards (RTS) for Strong Customer Authentication (SCA) details some requirements, for instance on security principles: traceability, strong customer authentication…

1.2. Mission

STET has been mandated by its shareholders in order to design and provide an open API (Aka STET PSD2 API) that would specify the different interactions between TPPs and ASPSPs for carrying out the different use cases of PSD2. This API could be extended to other (non-PSD2) use cases in the future but this extension is not part of the mandate.

As the RTS for SCA are now finalised, this version of the API and its documentation considers the new constraints and rules that have been introduced.

This version also includes

- Items that have been identified and studied in common with the BERLIN GROUP, in a strategy of convergence of the different European API initiatives.
- Evolvements linked to the change requests that have been received after first public releases of STET PSD2 API.

The STET PSD2 API does not cover:

- Interactions between PSUs and TPP
- Interactions between PSUs and ASPSP
- Registration information management

The technical characteristics of this API are provided within a SWAGGER 2.0 file. The present document purpose is to provide extra-information on this API and to give some interaction samples.

1.3. Legal framework

PSD2:

- http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32015L2366

EBA RTS on SCA and CSC:

- https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2018.069.01.0023.01.ENG&toc=OJ:L:2018:069:TOC

EBA Opinion on the implementation of the RTS on SCA and CSC:

- https://www.eba.europa.eu/documents/10180/2137845/Opinion+on+the+implementation+of+the+RTS+on+SCA+and+CSC+%28EBA-2018-Op-04%29.pdf

EIDAS:

- http://eur-lex.europa.eu/legal-content/FR/TXT/?uri=celex%3A32014R0910

1.4. Licence

This work has been coordinated by STET with the following contributors:

- BNP Paribas
- Le Groupe BPCE
- Le Groupe Crédit Agricole
- La Banque Fédérative du Crédit Mutuel – CIC
- La Banque Postale
- La Société Générale

- La Caisse des Dépôts et Consignations
- Le Crédit Mutuel - ARKEA
- HSBC France
- L'OCBF
- La Fédération Bancaire Française
- LUXHUB
- RAIFFEISEN LU

This release also takes into accounts the work of the Working Group of the French CNPS (Comité National des Paiements Scripturaux), co-chaired by:

- La Banque de France
- La Direction Générale du Trésor

Other attendees than banks to this Working Group were:

- L'ACPR (Autorité de Contrôle Prudentiel et de Résolution)
- La DINSIC (Direction Interministérielle des Systèmes d'Information et de Communication)
- L'AFEPAME (Association des Établissements de Paiement et de Monnaie Électronique)
- CGI Luxembourg S.A.
- MERCATEL
- La FEVAD (Fédération du e-commerce et de la vente à distance)
- L'ASF (Association française des Sociétés Financières)
- WORLDLINE
- BANKIN'
- LINXO
- BUDGET INSIGHT
- LYDIA
- LYRA NETWORK
- AMERICAN EXPRESS

2. Business Model

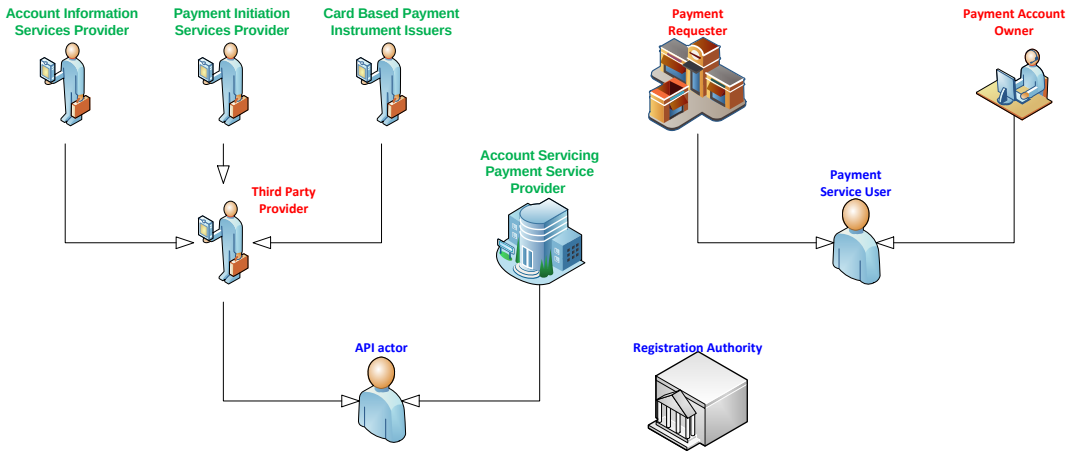
2.1. Actors and Roles

A PSD2 actor is either an entity or a physical person which can endorse one or several roles.

Most of the roles are defined in PSD2. However, some extra-roles have been specified for the purpose of the STET PSD2 API during the analysis phase of the project.

Within the following diagram:

- Actors have cyan-coloured labels
- Pure PSD2 roles have green-coloured labels
- Specific STET PSD2 API roles have red-coloured labels



2.1.1. Payment Service User (PSU)

PSUs are the end-users of the services provided by TPPs and ASPSPs.

They are either physical persons or entities (organisations, companies, administrations…).

They do not interact directly with the STET PSD2 API.

A given PSU endorses at least one of the following roles:

- Payment Account Owner (PAO) for one or several accounts held by one or several ASPSPs.
- Payment Requester (PR) asking either for a payment or a coverage check.

2.1.2. API actors

2.1.2.1. Account Servicing Payment Service Provider (ASPSP)

These are Payment Service Providers (PSPs) which are in charge of holding payment accounts for their customers (PSU).

2.1.2.2. Third Party Provider (TPP)

These actors can intermediate between PSUs and ASPSPs, acting on behalf of a PAO or a PR.

On one hand, a given PAO may contract with a TPP in order to use the services provided by this TPP:

- Account Information Services (AISP role) will allow the PAO to get information, through a single interface, about all of his/her accounts, whatever the ASPSP holding this account.
- Card Based Payment Instrument Issuers (CBPII role) that will check the coverage of a given payment amount by the PSU's account.

On the other hand, a PR may also contract with a TPP that will provide the following services:

- Payment Initiation Services for requesting a Payment Request approval by the PSU and requesting the subsequent execution through a Credit Transfer (PISP role).

2.1.3. Registration Authorities (RA)

RAs are in charge of registering and overviewing the PSD2 actors.

The registration information is the foundation on which each actor can rely in order to know:

- Who is a given actor?
  - Identity
  - Contacts (business, legal, operational…)
  - Insurance coverage
  - Authentication media
    - X.509 eIDAS certificates (https://eur-lex.europa.eu/eli/reg/2014/910/oj)
      - QWAC for TLS mutual authentication
      - QSEALC for content signature
    - Certification chain and services (revocation list, OCSP)
- For which roles this actor has been registered
  - AISP
  - PISP
  - CBPII
  - ASPSP
- Technical characteristics
  - APIs that are provided
  - URLs that are to be used, for test or live processing.

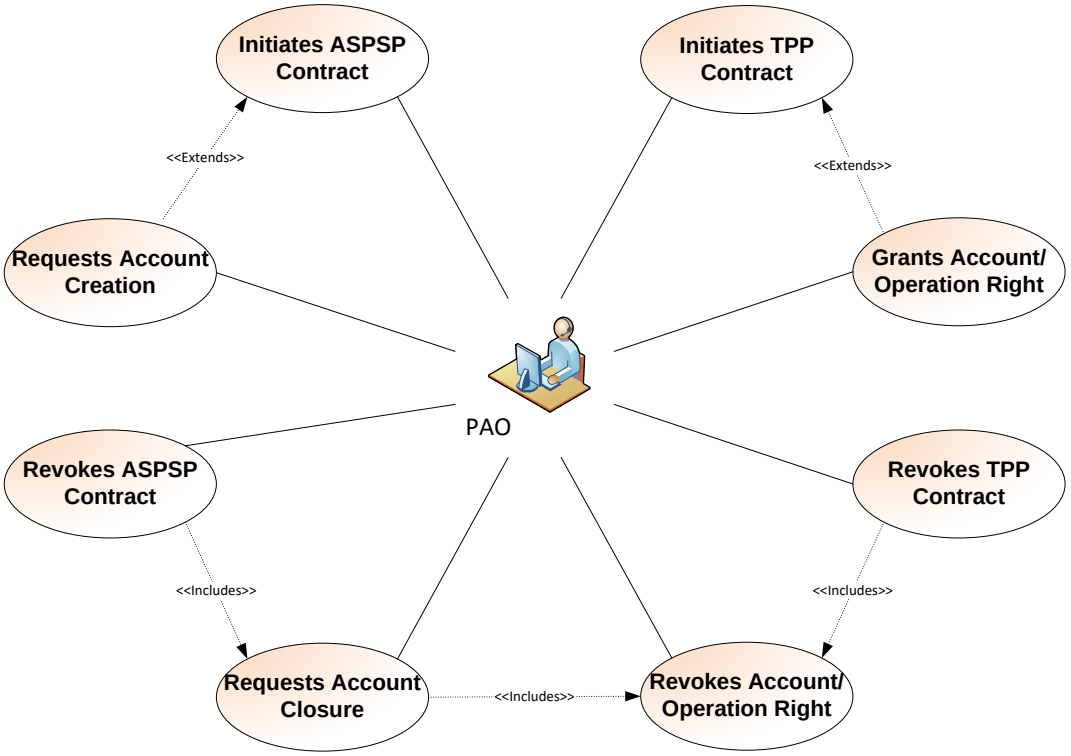
Registration Authorities must keep track of changes for each actor in order to recover the full history of the actor.

2.2. Use cases

Some of the use cases that are listed below are directly implemented by the STET PSD2 API, for they rely on interactions between TPPs and ASPSPs.

Other uses cases are tagged as “NON-API” and are only described for global understanding purpose.

2.2.1. PAO uses cases (NON-API)

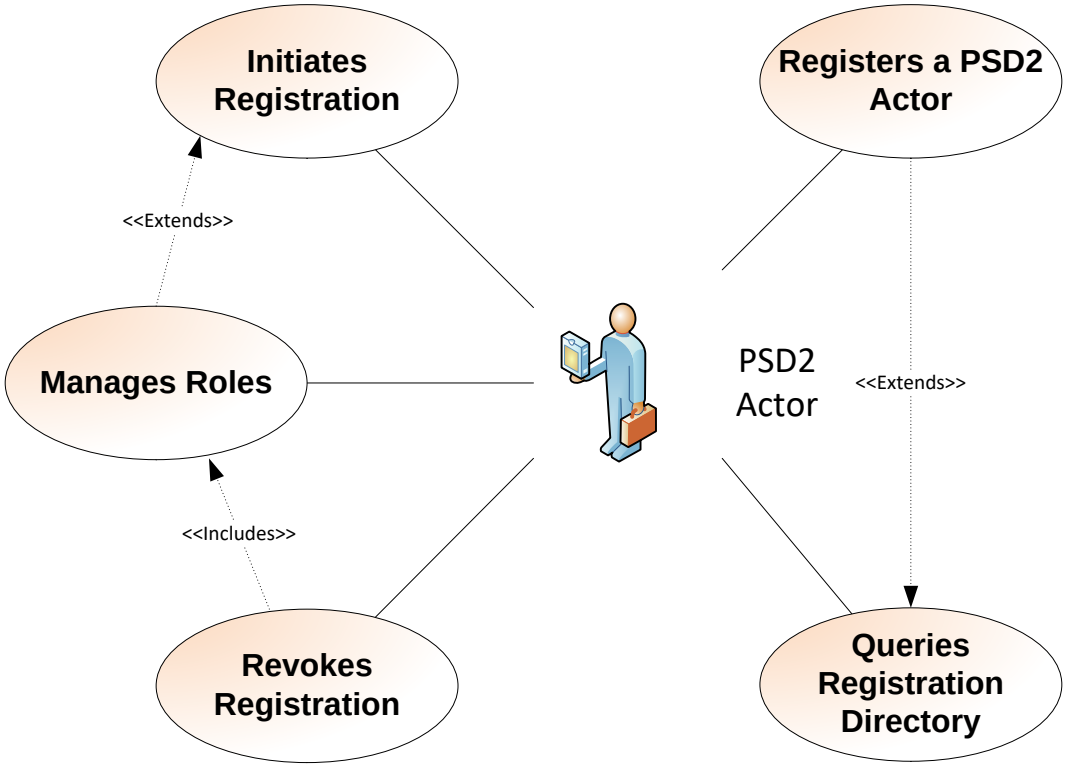


USE CASE (PAO)	DESCRIPTION	INTERACTIONS
Initiates ASPSP Contract	The user contracts with an ASPSP in order to use its services. This use case is likely extended by one or more occurrences of the “Requests Account Creation” use case	ASPSP
Requests Account Creation	The user asks the ASPSP to open a new payment account Requires a contract between the PAO and the ASPSP	ASPSP

USE CASE (PAO)	DESCRIPTION	INTERACTIONS
Requests Account Closure	The user asks the ASPSP to close an existing payment account This use case includes the “revokes Account/Operation Accreditation” use case for all operations on this account and for all granted TPP.	ASPSP TPP (indirectly)
Revokes ASPSP Contract	The user revokes the contract with the ASPSP This use case includes the “Requests Account Closure” use case for each account that is held by the ASPSP. This use case includes the “Revokes Account/Operation Accreditation” use case for all operations on each of these accounts and for all granted TPP.	ASPSP TPP (indirectly)
Initiates TPP Contract	The user contracts with a TPP having AISP and/or CBPII roles in order to use its service This use case is likely extended by one or more occurrences of the “Grants Account/Operation Accreditation” use case	TPP
Grants Account/Operation accreditation	The user allows the TPP to access a given set of operations on one of his/her payment accounts. Requires a contract between the PAO and the ASPSP, a contract between the PAO and the TPP and the registration of this PAO-TPP relationship by the ASPSP to enable the OAuth2 token management (cf. §3.4.2). Requires also that the capture and the execution of the accreditations are handled by the TPP (the further forwarding of these accreditations is an AISP use case and so out of scope of this use case: cf. §2.2.3).	ASPSP TPP
Revokes Account/Operation accreditation	The user asks the ASPSP to revoke the TPP access for a given set of operations on a given PAO account. Requires that the capture and the execution of the revocation are handled by the TPP.	ASPSP TPP

USE CASE (PAO)	DESCRIPTION	INTERACTIONS
Revokes TPP Contract	The user revokes the contract with the TPP. This use case includes the “Revokes Account/Operation Accreditation” for all grants given to the TPP, whatever the ASPSP. Since this cannot be automated, it is the PAO’s duty to initiate all the relevant revocations with each ASPSP.	TPP ASPSP

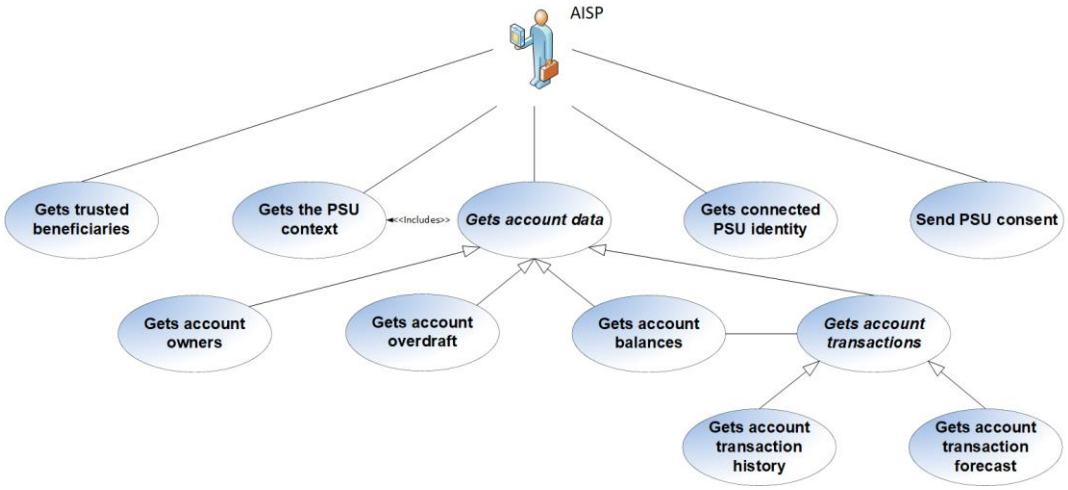
2.2.2. Registration use cases (NON-API)



USE CASE (PSD2 ACTOR)	DESCRIPTION	INTERACTIONS
Initiates Registration	The user asks the RA for registration. This use case is likely extended by one or more occurrences of the “Manages Roles” use cases	RA other actors (indirectly)
Manages Roles	The user asks the RA to be referenced for a given set of roles. This use case can be replayed in order to reference or dereference any role.	RA other actors (indirectly)

USE CASE (PSD2 ACTOR)	DESCRIPTION	INTERACTIONS
Revokes registration	The user informs the RA that its registration is to be cancelled	RA other actors (indirectly)
Queries Registration Directory	The user queries the RA directory in order to get data on other PSD2 actors: roles, certificates…	RA other actors (indirectly)
Registers a PSD2 actor	The user registers a given PSD2 actor into its own Directory	None

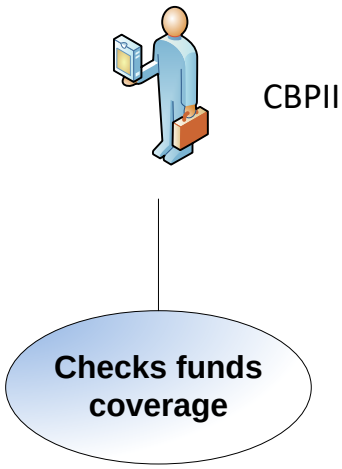
2.2.3. AISP use cases



USE CASE (AISP)	DESCRIPTION	INTERACTIONS
Gets the PSU Context	The user queries the ASPSP in order to get the payment accounts that are eligible for the relevant PSU.	ASPSP
Sends the PSU consent to the ASPSP	Having captured the consent choices from the PSU, the user sends them to the ASPSP	ASPSP
Gets Account Data	*This use case is abstract. Its purpose is to stress that the "Gets the PSU Context" is a prerequisite for all other use cases on a given account*	*none*
Gets Account Owners	The user queries the ASPSP in order to get the owners on one given account.	ASPSP
Get Account Overdrafts	The user queries the ASPSP in order to get the overdraft that applies on one given account.	ASPSP
Gets Account Balance	The user queries the ASPSP in order to get the balance on one given account. The ASPSP can provide several balance computing's (Instant Balance, Accounting Balance…), each balance type being specified with an explicit label.	ASPSP
Gets List of Transactions	This use case is abstract and can be seen as the common interface for the two following uses-cases.	ASPSP
Gets Account Transaction History	The user queries the ASPSP in order to get all the transactions that have been committed to one given PSU account within a given range of value dates.	ASPSP

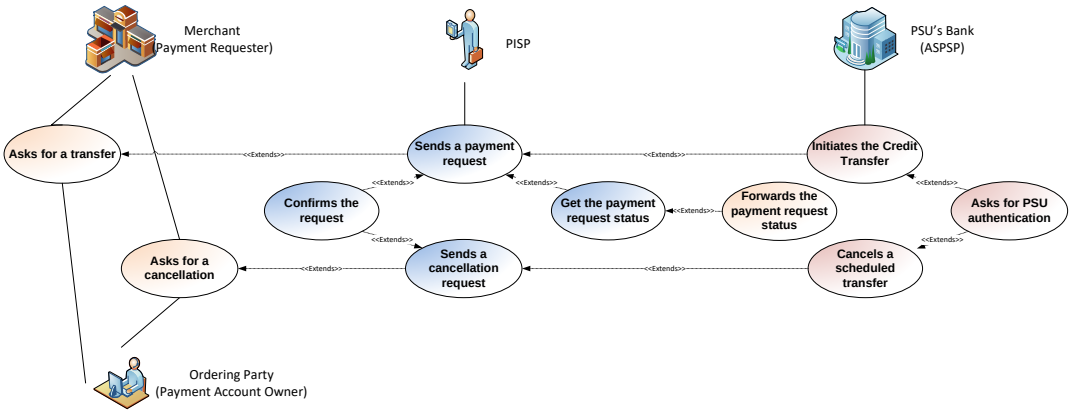
USE CASE (AISP)	DESCRIPTION	INTERACTIONS
Gets Account Transaction Forecast	The user queries the ASPSP in order to get all the transactions that are known by the ASPSP to be committed to a given PSU account	ASPSP
Gets connected PSU identity	The user queries the ASPSP in order to get the identity of the PSU on behalf of whom the AISP is connected	ASPSP
Gets trusted beneficiaries	The user queries the ASPSP in order to get all the beneficiaries that were registered as “trusted” par the PSU.	ASPSP

2.2.4. CBPII use cases



USE CASE (CBPII)	DESCRIPTION	INTERACTIONS
Checks Funds Coverage	The user queries the ASPSP in order to check if a given transaction amount can be covered by one given PSU account	ASPSP

2.2.5. PISP uses cases



USE CASE (PSU)	DESCRIPTION	INTERACTIONS
Asks for a transfer (Non-API)	Either the Merchant or the Payment Account Owner asks the PISP to initiate a transfer with one or several payment instructions or to set a standing order.	PISP
Asks for a cancellation (Non-API)	Either the Merchant or the Payment Account Owner asks the PISP to cancel: - all or part of the payment instructions that have been initiated - or a previously set standing order	PISP

USE CASE (PISP)	DESCRIPTION	INTERACTIONS
Sends a Payment Request	The user sends to the ASPSP all the information needed to initiate a Payment. The payment might have been requested either by the beneficiary (e.g. Merchant) or by the account owner him/herself. The payment may include one or several instructions, the maximum number of instructions can be specified by each ASPSP. Those instructions might have - Either a same requested execution date but multiple beneficiaries - Or a same beneficiary but different requested executions dates, those being either explicitly specified or scheduled through a given periodicity (standing orders)	ASPSP
Sends a cancellation request	The user sends to the ASPSP a request to cancel, from a previously posted payment request, one, several or all instructions provided that they have not yet been executed. Cancellations can be performed by sending the Payment request with modifications of the status and reason code at payment level and/or at instruction level.	ASPSP
Confirms the Request	The user confirms the Payment Request or the Cancellation Request to the ASPSP.	ASPSP

USE CASE (PISP)	DESCRIPTION	INTERACTIONS
Gets the Payment Request status	The user gets the status of the Payment Request from the ASPSP. This status embeds: - Information about the payment request and the execution of the subsequent Credit Transfers - Information about the effective booking of the payment instructions that are about to be executed - Information about the availability of funds for payment instructions that are about to be executed but are not effectively booked - Information about the trust given by the PSU to the beneficiary of the payment	ASPSP
Forwards the Payment Request status to the Creditor (Non-API)	The user informs the PR of the status of the Payment Request	PR (Creditor)

USE CASE (ASPSP)	DESCRIPTION	INTERACTIONS
Asks for PSU authentication (Non-API)	Provided the Payment Request is valid, the user asks the PAO in order to authenticate before execution of the relevant Payment Request or Cancellation Request	PSU(PAO)
Initiates the Credit Transfer (Non-API)	Provided the PAO has authenticated and the PISP has confirmed the payment request, the ASPSP initiates the relevant Credit Transfer.	Beneficiary's ASPSP (Creditor Agent)
Cancels a scheduled transfer (Non-API)	Provided the PAO has authenticated and the relevant transfers have not yet been executed, the ASPSP cancels the execution of the instructions that were specified by the PISP	None

3. Prerequisites and technical details

3.1. Actors registration

PSD2 actors must be registered by a registration authority. The information that has been collected must be accessible to other actors in order to provide trust and interoperability.

A non-registered actor cannot interact with another actor.

Each actor must be provided with at least one eIDAS certificate (QWAC), for TLS 1.2 purpose, delivered by a registered Qualified Trust Service Provider (QTSP).

The European Commission list of QTSPs can be retrieved at the following URL:

https://esignature.ec.europa.eu/efda/tl-browser/#/screen/home

3.2. Cross-Authentication and Data Encryption

The STET PSD2 API relies on TLS 1.2 protocol in order to get cross-authentication between actors. Moreover, this protocol also ensures data confidentiality during their transport on the network.

Whenever a TPP connects as a client to an ASPSP API service, it will check the ASPSP server certificate (QWAC) and present its own eIDAS certificate (QWAC) respecting the ETSI/TS119495 Technical Specification.

The Organisational Identification within the Subject Distinguished Name of the certificate should actually be regarded as an Authorization Number that will respect the following format rules:

- "PSD" as 3-character legal person identity type reference;
- 2-character ISO 3166 [7] country code representing the NCA country;
- hyphen-minus "-" (0x2D (ASCII), U+002D (UTF-8)); and
- 2-8-character NCA identifier (A-Z uppercase only, no separator);
- hyphen-minus "-" (0x2D (ASCII), U+002D (UTF-8)); and
- PSP identifier (authorization number as specified by the NCA).

In case of authentication failure, on one side or the other, the connection must be closed.

No additional encrypting or authenticating feature is required.

3.3. Customer Authentication Approaches

Three different approaches can be used by a TPP to allow the PSU authentication by the ASPSP. These approaches rely on a PSU identification that must be relevant to the ASPSP (National identifier or Bank customer identifier).

These approaches are implemented in different ways, depending on the relevant use case:

- either during the authorisation process (cf. §3.4.2), mostly for AISP and CBPII use cases,
- or during the consent confirmation process, for instance in case of a PISP submitting a Payment Request (cf. § 3.4.2).

3.3.1. Redirect Approach

Through the Redirect approach, the PSU authentication process is fully processed by the ASPSP.

In order to allow this, the TPP has to redirect the PSU to the ASPSP authentication service, meaning the PSU will leave temporarily the TPP interface for authenticating towards the ASPSP interface.

The TPP might have already captured a PSU identifier that can be handled by the ASPSP for unambiguously recognizing the PSU. In this case this identifier might be forwarded through the redirection, when the redirect protocol allows the forwarding of this identifier.

After finalisation of the authentication, the ASPSP redirects the PSU back to the TPP interface.

3.3.2. Decoupled approach

Through the Decoupled approach, the PSU authentication process is fully processed by the ASPSP.

In order to allow this the TPP has to capture a PSU identifier that can be handled by the ASPSP for unambiguously recognizing the PSU, and to forward this identifier to the ASPSP.

Based on this identifier, the ASPSP will trigger an authentication through a decoupled device or application, meaning that the PSU will not leave the TPP interface during the authentication process.

3.3.3. Exemptions to Strong Customer Authentication

Exemptions to Strong Customer Authentication are specified by the EBA RTS on SCA, especially for Payment Initiation Services.

In this context, the API allows the PISP to forward to the ASPSP any useful information. Moreover, the PISP may also hint the ASPSP on whether or not the relevant payment request could be subject to an exemption.

Eventually, the ASPSP keeps the final decision to apply or not this exemption.

3.4. Authorization

3.4.1. Levels of authorization

The following levels of authorization may be checked and combined in order to compute the effective rights granted to the TPP:

AUTHORIZATION LEVEL	DESCRIPTION
Authorization by TPP role	Once the TPP has been registered for a given role, it can call any of the PSD2 features provided by an ASPSP through the STET PSD2 API for this role.
Authorization by TPP-ASPSP agreement	The TPP can call any of the additional (non PSD2) features provided by an ASPSP through the STET PSD2 API, provided there is a bilateral agreement to use these features.
Authorization by TPP-PSU agreement	If the PSU has contracted with a TPP, he/she must - Give a list of the ASPSPs that he/she allows the TPP to access - Process an authentication against each of those relevant ASPSPs that will further allow the TPP to access the PSU data.
Authorization by PSU context	The PSU is able to specify his/her PSU context detailing, for each of its relevant accounts: - If this account will be accessible or not by the TPP - Which features can be used by the TPP The PSU can modify at any time his/her PSU context.

3.4.2. Technical basis

The TPP is authorized to access the ASPSP's API through an access token that can be retrieved through the OAuth2 Authorisation Framework (https://tools.ietf.org/html/rfc6749).

Different authorisation grants can be used, depending on the TPP's role and use case to be applied.

The TPP may need to handle multiple OAuth2 tokens provided by a given ASPSP on behalf of a given PSU. Actually, the request of a new OAuth2 token must not imply the revocation of a previous one.

3.4.2.1. TPP Identity matching

The OAuth2 protocol is enforced by checking the identity of the TPP during the OAuth2 procedures through the TPP's eIDAS certificate, based on MTLS (https://datatracker.ietf.org/doc/rfc8705/).

This enforcement is obtained by the mandatory provisioning by the TPP of a [client_id] field within all OAuth2 request. This [client_id] must match, directly or not, with the Authorisation Number located within the TPP's eIDAS certificate and this match must be checked by the ASPSP for each OAuth2 request.

Since MTLS is used, the use of the [client_secret] is not very useful but can nevertheless been required by some authorisation servers. Each implementation requiring the use of a [client_secret] must update its documentation on this topic.

Direct matching

The match can be obviously direct when the [client_id] is equal to the Authorisation Number.

In this case the ASPSP's API MANAGER might be able to check and accept "on the fly" the OAuth2 request.

Indirect matching

However, in some cases, especially when the API MANAGER is unable to process an "on the fly" registration, an OAuth2 technical setup should occur prior to any OAuth2 token request. This setup will result by the provisioning of a [client_id] value by the ASPSP to the TPP.

- The provisioning of multiple [client_id] values that could be used for different use cases by the TPP is possible through replaying the setup.
- Moreover, the setup allows the exchange of operational data between the TPP and the ASPSP for further use: logos, phone numbers, email addresses, certificates…

Eventually, this setup can be automated.

3.4.2.2. Automated OAuth2 technical setup

Principles

While most of the API managers provide an inline setup interface, this setup can also be automated.

The RFC 7591 specifies an interactive dynamic protocol that allows a client to provision some context metadata and get a [client_id] value. There is no restriction to provide several [client_id] values for the same client and context metadata. Nevertheless, the number of [client_id] requested by a same client must remain reasonable and motivated by a real need. As a complement, RFC 7592 specifies how to retrieve, modify or delete a previously posted context.

If several usage contexts are needed for a given API client, this client will have to reiterate the complete process to get as many [client_id] values as needed.

Actually, some TPPs might be client of an API on behalf of an agent (Article 4-38 of PSD2). Each agent should be considered as a specific usage context.

As this protocol is not mandatory, each API implementation will have to specify whether or not it is implemented.

Context metadata

The relevant metadata items to provide are listed below:

CLIENT METADATA NAME	CLIENT METADATA DESCRIPTION	REQUIREMENT	CHANGE CONTROLLER	REFERENCE
redirect_uris	Array of redirection URIs for use in redirect-based flows	Mandatory	IESG	[RFC7591]
software_statement	JSON Web Token (JWT) that asserts metadata values about the client software as a bundle	Optional	IETF	[RFC7591]
token_endpoint_auth_method	Requested authentication method for the token endpoint.	Mandatory According to the RFC8705 (cf. § 2.1.1), the value to be used will be "tls_client_auth" as soon as the draft will be promoted as an RFC.	IESG IETF	[RFC7591] [RFC8705]

CLIENT METADATA NAME	CLIENT METADATA DESCRIPTION	REQUIREMENT	CHANGE CONTROLLER	REFERENCE
tls_client_auth_subject_dn	Indicates the certificate subject value that the authorization server is to expect when authenticating the respective client.	Mandatory An [RFC4514] string representation of the expected subject distinguished name of the certificate, which the OAuth client will use in mutual-TLS authentication.	IETF	[RFC8705]
grant_types	Array of OAuth 2.0 grant types that the client may use	Mandatory Allowed values are: - "authorization_code" - “ciba” - "client_credentials" - "refresh_token"	IESG	[RFC7591]
response_types	Array of the OAuth 2.0 response types that the client may use	Optional “code” is the sole allowed value	IESG	[RFC7591]
client_name	Human-readable name of the client to be presented to the user	Mandatory Must specify the name of the agent or by default the name of the TPP.	IESG	[RFC7591]
client_uri	URL of a web page providing information about the client	Optional	IESG	[RFC7591]
logo_uri	URL that references a logo for the client	Optional	IESG	[RFC7591]
scope	Space-separated list of OAuth 2.0 scope values	Optional	IESG	[RFC7591]
contacts	Array of strings representing ways to contact people responsible for this client, typically email addresses	Mandatory At least one contact must be provided.	IESG	[RFC7591]
tos_uri	URL that points to a human-readable terms of service document for the client	Optional	IESG	[RFC7591]
policy_uri	URL that points to a human-readable policy document for the client	Optional	IESG	[RFC7591]
provider_legal_id	Authorization number of the TPP according to ETSI specification on eIDAS certificates for PSD2	Mandatory	STET (to be registered)	

CLIENT METADATA NAME	CLIENT METADATA DESCRIPTION	REQUIREMENT	CHANGE CONTROLLER	REFERENCE
client_legal_id	Authorization number of the agent see below)	Mandatory in case of an agent which is distinct from the TPP	STET (to be registered)	
logo	base64 encoded value of the logo	Optional	STET (to be registered)	
jwks	Client's JSON Web Key Set [RFC7517] document value, which contains the client's public keys.	Optional The value of this field MUST be a JSON object containing a valid JWK Set. These keys can be used by higher-level protocols that use signing or encryption.	IESG	[RFC7591]

In a similar way to the ETSI specification on the Authorization Number for TPPs, the agent Authorization Number must respect the following format:

- "AGT" as 3-character legal person identity type reference;
- 2-character ISO 3166 country code representing the NCA country;
- hyphen-minus "-" (0x2D (ASCII), U+002D (UTF-8)); and
- 2-8-character NCA identifier (A-Z uppercase only, no separator);
- hyphen-minus "-" (0x2D (ASCII), U+002D (UTF-8)); and
- Agent identifier (registration number as specified by the NCA).

Interactions

The TPP submits its context metadata through a

```
POST /register
```

In response, it gets this context metadata completed by

- the relevant [client_id]
- an optional [client_secret] that is not really useful since the authentication of the client is already done though MTLS.
- the [registration_client_uri] as an endpoint for configuration of the client
- the [registration_access_token] to be used for accessing the configuration of the client.
- its issuing timestamp.

RFC7591 allows the server to update some of the context metadata if needed.

At any time, the TPP can retrieve the context metadata through a

```
GET /register/{client_id}
```

Updating the context metadata can be done through a

```
PUT /register/{client_id}
```

And deleting the context metadata is possible through a

```
DELETE /register/{client_id}
```

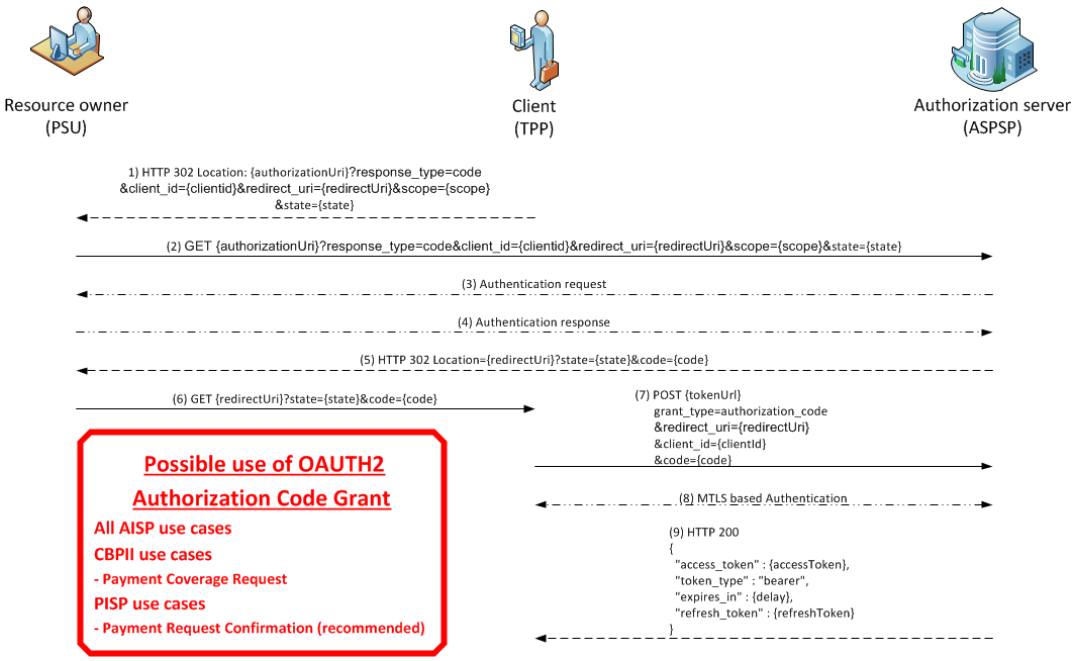
3.4.2.3. OAuth2 Authorization Code Grant

The authorisation process might rely on an OAuth2 sequence for obtaining an Authorization Code Grant token (cf. https://tools.ietf.org/html/rfc6749#section-4.1) and implements the REDIRECT approach.

This kind of token, depending on the ASPSP implementation:

- Can be used for all AISP use cases;
- Can be used for the CBPII use case;
- Can be used for the PISP confirmation use case.

The process can be summarized through the following steps.



At first, the PSU must specify to the TPP, the identity of one of its ASPSPs.

Optional Code verifier and challenge

Some authorization server may ask for implementation of RFC7636 (PKCE: Proof Key for Code Exchange) in order to enforce the OAuth2 Autorization Code grant.

Therefore, the client must as a first step create a code verifier and a code challenge.

Authorization Request

The TPP initiates the OAuth2 sequence by redirecting the PSU to the relevant ASPSP's authorization infrastructure, through the following URL pattern and parameters

Since this is done by a redirection of the PSU, the eIDAS of the TPP cannot be presented at this stage.

```
GET /authorize?response_type=code&client_id={clientId}&redirect_uri={redirectUri}&scope={scope}[&state={state}]
```

NAME		DATA	TYPE AND CONSTRAINS
response_type	[1..1]	Expected type of token	String[10] Must be valued with "code"
client_id	[1..1]	TPP identification	String[36] must be equal or linked to the OrganizationIdentifier part of the Distinguished Name of the eIDAS certificate, according to ETSI specification
redirect_uri	[0..1]	Call-back URL of the TPP	String[140]

scope	[0..1]	Specifies the generic accreditations that both the PSU and the TPP agreed on:
		- For AISP
		  ○ aisp
		  ○ extended_transaction_history
		- for CBPII
		  ○ cbpii
		- for PISP
		  ○ pisp
			String[140]
			Space delimited roles list.
			Mandatory

state	[0..1]	Internal state that can be used by the TPP for context management.	String[1024] Recommended
code_challenge	[0..1]	Code challenge as computed by the TPP according to RFC 7636	String[140] Required if implementation of RFC7636 is mandated by the Authorization Server
code_challenge_method	[0..1]	Code challenge method use by the TPP for computing the code challenge	"S256" or "plain" Default value equals to "plain"

Notice: The RFC 6749 does not specify the Authorization Code Grant to support the forwarding of the Resource Owner (PSU) user name or language preferences.

However, some OpenID Connect features might be used for these purposes even though the OpenID Connect specification is not fully applied (cf. § 3.4.2.4).

Moreover, this specification suggests a Token Introspection implementation (cf. § 3.4.2.7) whose parameters could be helpful to determine the PSU identity and usage context.

These additional parameters are summarized in the following table.

NAME	DATA	TYPE AND CONSTRAINS
login_hint_token	A token containing information identifying the end-user for whom the token was issued. This information can also include the specific usage context if needed. The particular details and security requirements for this element as well as how the end-user is identified by its content are specific to each ASPSP implementing this functionality. This token would have been retrieved through a token introspection request (cf. § 3.4.2.7)	String [2048]
ui_locales	End-User's preferred languages and scripts for the user interface.	String [140] End-User's preferred languages and scripts for the user interface, represented as a space-separated list [RFC 5646]
login_hint	Hint to the Authorization Server about the login identifier the End-User might use to log in (if necessary).	String[36]

The ASPSP

- Identifies and authenticates the PSU
- Computes the relevant TPP checks (roles, validity, non-revocation…)
- Checks the [redirect_uri] against the ones that might have been declared during the automated OAuth2 technical setup (cf. § 3.4.2.2). The provided [redirect_uri] must exactly match one those that have been registered.
- Registers the [code_challenge] and [code_challenge_method] with the code request when implementation of RFC 7636 is mandated.

Authorization Response

Afterwards, the ASPSP redirects the PSU to the TPP, using the previously given call-back URL (redirect_uri) and the following parameters:

NAME		DATA	TYPE AND CONSTRAINS
code	[1..1]	Short-time code to use in order to get the access token	String[36]
state	[0..1]	Internal state if provided by the TPP	String[1024] Recommended

The recommended lifetime of the authorization code as specified by the RFC 6749 is 10 minutes but it is up to the authorization server to set its own lifetime value.

Access Token Request

In order to get the access token, the TPP is now able to call, through a POST request, the ASPSP's authorization infrastructure with the following parameters.

```
POST /token HTTP/1.1
    Host: server.example.com

        grant_type=authorization_code
        &code={code}
        &redirect_uri={redirectUrl}
        &client_id={clientId}
```

NAME		DATA	TYPE AND CONSTRAINS
grant_type	[1..1]	Requested authorization type	String[36] Must be valued with "authorization_code"
code	[1..1]	Short-time code previously provided by the ASPSP	String[36]
redirect_uri	[0..1]	Call-back URL of the TPP	String[140] Must be equal to the one provided during the authorization code request
client_id	[1..1]	TPP identification.	String[36] must be equal or linked to the OrganizationIdentifier part of the Distinguished Name of the eIDAS certificate, according to ETSI specification
client_secret	[0..1]	The client secret	Since authentication of the client TPP is provided through MTLS, this parameter is not very useful.
code_verifier	[0..1]	RFC7636 code verifier that was initially set by the TPP	Required when implementation of RFC7636 is mandated by the Authorization Server

- The ASPSP
    - Identifies and authenticates the TPP through the presented eIDAS certificate (QWAC)
    - Checks the direct or indirect matching between the Authorization Number within the eIDAS certificate and the [client_id] value.
    - Computes the relevant TPP checks (roles, validity, non-revocation…)
    - Verifies the [code_verifier] value by recalculating the [code_challenge].

Access Token Response

- The ASPSP answers through a HTTP200 (OK) response that embeds the following data.

NAME		DATA	TYPE AND CONSTRAINS
access_token	[1..1]	Access token provided by the ASPSP to the TPP.	String[140]
token_type	[1..1]	Type of the provided access token (“Bearer” or “MAC”)	String[10] Must be valued with “Bearer”
expires_in	[0..1]	Token lifetime, in seconds. The token can be used several times as far as it is not expired.	Numeric
refresh_token	[0..1]	Refresh token that can be used for a future token renewal request.	String[140]

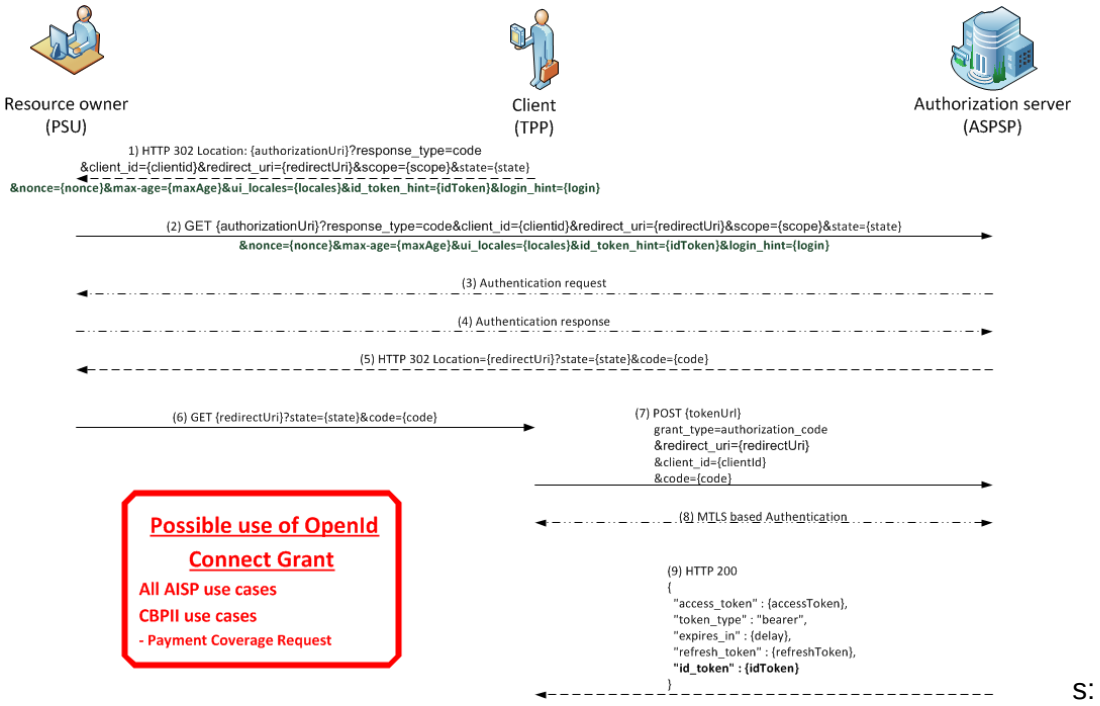
3.4.2.4. OpenID Connect extension to the OAuth2 Authorization Code Grant

As an optional feature, an authorization server may implement the « OpenID Connect Core 1.0 » specification on top of the OAuth2 “Authorization Code” flow.

The OpenID Connect protocol allows the API client (TPP) to get from the API server (ASPSP) an IdToken that will certify the identity of the PSU, once this PSU has been authenticated by the ASPSP.

Simple Authentication request

The Open Id Connect Authentication Request relies on the OAuth2 “Authorization Code” Authorization Request with some additional parameters, marked as bold in the following diagram)requirement.



s:

NAME		DATA	TYPE AND CONSTRAINS
response_type	[1..1]	Expected type of token	String[10] Must be valued with "code"
client_id	[1..1]	TPP identification	String[36] must be equal or linked to the OrganizationIdentifier part of the Distinguished Name of the eIDAS certificate, according to ETSI specification
redirect_uri	[0..1]	Call-back URL of the TPP	String[140]
Scope	[0..1]	Specifies the generic accreditations that both the PSU and the TPP agreed on: - For AISP o aisp o extended_transaction_history - for CBPII o cbpii. - **In any case** o **openid** o **offline_access**	String[140] Space delimited roles list. **additional scopes are required** - **openid for specifying the use of OpenID Connect** - **offline_access to allow the retrieval of a refresh token within the OpenID context.**
State	[0..1]	Internal state that can be used by the TPP for context management.	String[1024]
Nonce	**[0..1]**	**Association of a client session with an Id token used to mitigate replay attacks**	**String[36]**
max-age	**[0..1]**	**Maximum authentication age (in seconds)**	**String[15]**

NAME		DATA	TYPE AND CONSTRAINS
ui_locales	**[1..1]**	**End-User's preferred languages and scripts for the user interface.**	**String [140] End-User's preferred languages and scripts for the user interface, represented as a space-separated list [RFC 5646] Required by the API**
id_token_hint	**[0..1]**	**last known IdToken for the end-user (PSU), if any.**	**String [2048]**
login_hint	**[0..1]**	**Hint to the Authorization Server about the login identifier the End-User might use to log in (if necessary).**	**String[36]**
Login_hint_token	**[0..1]**	A token containing information identifying the end-user for whom the token was issued. This information can also include the specific usage context if needed. The particular details and security requirements for this element as well as how the end-user is identified by its content are specific to each ASPSP implementing this functionality. This token would have been retrieved through a token introspection request (cf. § 3.4.2.7)	**String [2048]**

The [id_token_hint] parameter is quite useful to ease a PSU authentication request renewal by forwarding his/her already known identification. For a first authentication request the [login_hint] parameter can be used by the TPP to forward the PSU identification, as known by the ASPSP.

As for the OpenID Connect Authentication Request is based on the OAuth2 Authorization Request, the latest is enhanced in the following way:

```
GET /authorize?
   response_type=code
   &client_id=s6BhdRkqt3
   &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
   &scope=openid%20%offline_access%20%aisp
   &nonce=n-0S6_WzA2Mj
   &state=af0ifjsldkj
HTTP/1.1
  Host: server.example.com
```

Signed Authentication Request

The OpenID Connect Authentication Request can also be passed as a Signed Request Object if the Authorisation Server allows at.

The structure of this Signed Request Object is a Json Web Token (JWT) whose data includes:

- the usual request parameters as specified above
- some additional parameters relating to the signature (see https://openid.net/specs/openid-connect-core-1_0.html#RequestObject for details)

It must be noted that although the Signed Request Object supersedes the usual request parameters, the latest may also be passed alongside.

Authentication Response

Case of a successful processing of the request, the server will return an authorization code through the redirection of the PSU towards the TPP.

```
HTTP/1.1 302 Found
 Location: https://client.example.org/cb?
  code=SplxlOBeZQQYbYS6WxSbIA
  &state=af0ifjsldkj
```

Token request

The TPP requests the exchange of the authorization code against an OAuth2 token.

```
POST /token HTTP/1.1
 Host: server.example.com
 Content-Type: application/x-www-form-urlencoded
grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

NB: The « Authorization » header is useless since authentication is provided through MTLS, based on the TPP eIDAS certificate (https://datatracker.ietf.org/doc/rfc8705/).

Token response

The Authorization server answers with:

- An OAuth2 access token
- An OAuth2 refresh token
- An IdToken

```
HTTP/1.1 200 OK
 Content-Type: application/json
 Cache-Control: no-store
```

```
Pragma: no-cache

{
 "access_token": "SlAV32hkKG",
 "token_type": "Bearer",
 "refresh_token": "8xLOxBtZp8",
 "expires_in": 3600,
 "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzc
  yI6ICJodHRwOi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5
  NzYxMDAxIiwKICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZ
  fV3pBMk1qIiwKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5Nz
  AKfQ.ggW8hZ1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6q
  Jp6IcmD3HP99Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJ
  NqeGpe-gccMg4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7Tpd
  QyHE5lcMiKPXfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoS
  K5hoDalrcvRYLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4
  XUVrWOLrLl0nx7RkKU8NXNHq-rvKMzqg"
}
```

IdToken structure

The structure of the IdToken is a Json Web Token (JWT).

In the previous example, the following data is included:

```
{
 alg: "RS256",
 kid: "1e9gdk7"
}.
{
 iss: ""'http://server.example.com'"",
 sub: "248289761001",
 aud: "s6BhdRkqt3",
 nonce: "n-0S6_WzA2Mj",
 exp: 1311281970,
 iat: 1311280970
}.
[signature]
```

The possible data items are described in the following table:

FIELD	REQUIREMENT	DESCRIPTION
iss	Mandatory	Token provider identifier
sub	Mandatory	Token subject identifier
aud	Mandatory	Token recipient [client_id]
nonce	Conditional	Mandatory retrieval of the [nonce] parameter if present in the initial Authentication Request
exp	Mandatory	IdToken expiration date [RFC3339]
iat	Mandatory	IdToken creation date [RFC3339]
auth_time	Conditional	End-user (PSU) authentication date and time ["https://tools.ietf.org/html/rfc3339"] when the [max_age] parameter is present in the initial Authentication Request

3.4.2.5. Client Initiated Backchannel Authentication Grant

This registration process is based on the OpenID FAPI Connect CIBA flow and implements the DECOUPLED approach. (https://openid.net/specs/openid-client-initiated-backchannel-authentication-core-1_0.html)

However, in order to avoid any preregistration step, this grant is supposed to use the polling mode as the sole way for getting the requested token.

This kind of token, depending on the ASPSP implementation:

- Can be used for all AISP use cases;
- Can be used for the CBPII use case;

Authorization Request

In order to get the access token, the TPP calls, through a POST request, the ASPSP's authorization infrastructure with the following parameters.

```
POST /bc_authorize HTTP/1.1
    Host: as.example.com

            client_id={client_id}
            &scope={scope}
            &login_hint_token={login_hint_token}
            &id_token_hint={id_token_hint}
            &login_hint={login_hint}
            &binding_message={binding_message}
            &ui_locales={ui_locales}
```

NAME		DATA	TYPE AND CONSTRAINS
client_id	[1..1]	TPP identification	String[36] must be equal or linked to the OrganizationIdentifier part of the Distinguished Name of the eIDAS certificate, according to ETSI specification
scope	[1..1]	Specifies the generic accreditations that both the PSU and the TPP agreed on: - For AISP o aisp o extended_transaction_history - for CBPII o cbpii	String[140] Space delimited roles list. Mandatory
login_hint_token	[0..1]	A token containing information identifying the end-user whom the token was issued. This information can also include the specific usage context if needed. The particular details and security requirements for this element as well as how the end-user is identified by its content are specific to each ASPSP implementing this functionality. This token would have been retrieved through a token introspection request (cf. § 3.4.2.7)	String [2048]
id_token_hint	[0..1]	An ID token previously issued to the Client in case the ASPSP has implemented The OpenID connect extension to OAuth2.	String[36]
login_hint	[0..1]	Hint to the Authorization Server about the login identifier the End-User might use to log in (if necessary).	String[36]
binding_message	[0..1]	A human readable identifier or message intended to be displayed to the end-user.	String [140]
ui_locales	[0..1]	End-User's preferred languages and scripts for the user interface.	String [140] End-User's preferred languages and scripts for the user interface, represented as a space-separated list [RFC 5646]

The ASPSP

- Identifies and authenticates the TPP through the presented eIDAS certificate (QWAC)
- Checks the direct or indirect matching between the Authorization Number within the eIDAS certificate and the [client_id] value provided by the JWT
- Computes the relevant TPP checks (roles, validity, non-revocation…)
- Identifies the PSU and checks if a decoupled channel can be used

Authorization Response

In case of a successful request validation, the ASPSP answers to the TPP through a HTTP200 (OK) response that embeds the following data.

NAME		DATA	TYPE AND CONSTRAINS
auth_req_id	[1..1]	Unique identifier for the authorization request.	String[36]
expires_in	[1..1]	Expiration time in seconds of the authorization request identifier	Number
interval	[0..1]	Minimum amount in seconds that the TPP must wait between polling requests.	Number

Meanwhile, the ASPSP contacts the PSU on the relevant decoupled channel, displays the authorization request content and asks for confirmation through an authentication.

Access Token Request

The TPP may then poll the token endpoint with the interval provided by the authorization response.

```
POST /token HTTP/1.1
    Host: as.example.com

        client_id={client_id}
        &grant_type={grant_type}
        &auth_req_id={auth_req_id}
```

NAME		DATA	TYPE AND CONSTRAINS
client_id	[1..1]	TPP identification	String[36] must be equal or linked to the OrganizationIdentifier part of the Distinguished Name of the eIDAS certificate, according to ETSI specification
client_secret	[0..1]	The client secret	Since authentication of the client TPP is provided through MTLS, this parameter is not very useful.
grant_type	[1..1]	Grant type	Must be equal to “urn:openid:params:grant-type:ciba”
auth_req_id	[1..1]	Authorization request unique identifier as provided through the Authorization response	String [36]

Successful token response

The ASPSP answers through a HTTP200 (OK) response that embeds the following data.

NAME		DATA	TYPE AND CONSTRAINS
access_token	[1..1]	Access token provided by the ASPSP to the TPP.	String[140]
token_type	[1..1]	Type of the provided access token (“Bearer” or “MAC”)	String[10] Must be valued with “Bearer”

NAME		DATA	TYPE AND CONSTRAINS
expires_in	[0..1]	Token lifetime, in seconds. The token can be used several times as far as it is not expired.	Numeric
refresh_token	[0..1]	Refresh token that can be used for a future token renewal request.	String[140]

Unsuccessful token response

In addition to the error codes that are already specified by RFC6749, the following values can also be used in the context of a HTTP400 response.

ERROR CODE	DESCRIPTION
authorization_pending	The authorization request is still pending as the end-user (PSU) has not yet been authenticated. In respect with the specified polling interval, the TPP will have to replay the token request.
slow_down	The authorization request is still pending as the end-user (PSU) has not yet been authenticated and the TPP will have to replay the token request. However, the polling interval must be increased by at least 5 seconds for this next request and all the subsequent ones.
expired_token	The authorization request identifier has expired. The TPP will need to make a new authorization request
access_denied	the end-user (PSU) denied the authorization request

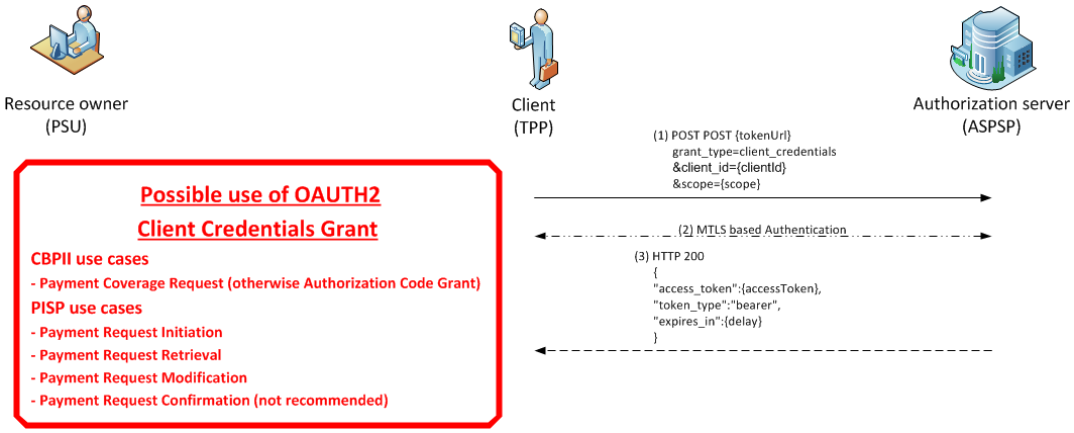
3.4.2.6. OAuth2 Client Credentials Flow

The registration of the TPP by the ASPSP relies on an OAuth2 sequence for obtaining a Client Credential grant token (cf. https://tools.ietf.org/html/rfc6749#section-4.4).

This kind of token, depending on the ASPSP implementation:

- Can be used for the CBPII use case ;
- Can be used for the PISP confirmation use case (basic REDIRECT Approach);
- Must be used for all others PISP use cases.

This procedure can be summarized through the following steps.



Access Token Request

The TPP sends directly, through a POST request, its access token request to the ASPSP authorization infrastructure with the following URL pattern and parameters

```
POST /token
        Host: authorization-server.com
        grant_type=client_credentials
        &scope={scope}
        &client_id={clientId}
```

NAME		DATA	TYPE AND CONSTRAINS
grant_type	[1..1]	Requested authorization type	String[36] Must be valued with "client_credentials"
scope	[0..1]	Specifies the generic accreditations that both the PSU and the TPP agreed on: PISP.	String[140] Space delimited roles list. Default value is "pisp"
client_id	[1..1]	TPP identification	String[36] must be equal or linked to the OrganizationIdentifier part of the Distinguished Name of the eIDAS certificate, according to ETSI specification
client_secret	[0..1]	The client secret	Since authentication of the client TPP is provided through MTLS, this parameter is not very useful.

The ASPSP

- Identifies and authenticates the TPP through the presented eIDAS certificate (QWAC)
- Checks the matching, direct or indirect, between the Authorization Number within the eIDAS certificate and the [client_id] value.
- Computes the relevant TPP checks (roles, validity, non-revocation…)

Access Token Response

- The ASPSP answers through a HTTP200 (OK) response that embeds the following data.

NAME		DATA	TYPE AND CONSTRAINS
access_token	[1..1]	Access token provided by the ASPSP to the TPP.	String[140]
token_type	[1..1]	Type of the provided access token (“Bearer” or “MAC”)	String[10] Must be valued with “Bearer”
expires_in	[0..1]	Token lifetime, in seconds. The token can be used several times as far as it is not expired.	Numeric

3.4.2.7. OAuth2 Token introspection

RFC 7662 (cf. https://tools.ietf.org/html/rfc7662) specifies how to provide meta-information about a given token.

It is up to each ASPSP to implement this functionality if needed.

Introspection Request

In order to get the meta-information about a given token, the TPP will call, through a POST request using its eIDAS certificate, the ASPSP’s authorization infrastructure with the following parameters.

```
POST /introspect HTTP/1.1
    Host: server.example.com

            token={tokenValue}
            &token_type_hint={tokenType}
```

NAME		DATA	TYPE AND CONSTRAINS
Token	[1..1]	String value of the token	String[144]
token_type_hint	[0..1]	Type of the token as defined	Must be valued with “refresh_token” or with “access_token”
client_id	[1..1]	TPP identification	String[36] must be equal or linked to the OrganizationIdentifier part of the Distinguished Name of the eIDAS certificate, according to ETSI specification
client_secret	[0..1]	The client secret	Since authentication of the client TPP is provided through MTLS, this parameter is not very useful.

- The ASPSP
    - Identifies and authenticates the TPP through the presented eIDAS certificate (QWAC)

- Checks the direct or indirect matching of the [token] value with the Authorisation Number that is located within the TPP's eIDAS certificate (QWAC).
- gets the relevant token and its meta-information in order to build the response.

Introspection Response

The ASPSP response in a JSON object with the suggested following elements as specified by the RFC:

NAME		DATA	TYPE AND CONSTRAINS
Active	[1..1]	Boolean indicator of whether or not the presented token is currently active.	"true" or "false"
Scope	[0..1]	Space-separated list of scopes associated with this token	String[140] Space delimited roles list.
client_id	[0..1]	Client identifier for the OAuth 2.0 client that requested this token.	String[36] must be equal or linked to the OrganizationIdentifier part of the Distinguished Name of the eIDAS certificate, according to ETSI specification
token_type	[0..1]	Type of the provided access token ("Bearer" or "MAC")	String[10] Must be valued with "Bearer" for access tokens. The value is irrelevant for refresh tokens.
Exp	[0..1]	Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token will expire.	Integer This field MUST be provided
Iat	[0..1]	Integer timestamp, measured in the number of seconds since January 1 1970 UTC, indicating when this token was originally issued	Integer This field can be provided

The STET API specification suggests the provision of another specific element specified as follows:

NAME	DATA	TYPE AND CONSTRAINS
login_hint_token	A token containing information identifying the end-user whom the token was issued. This information can also include the specific usage context if needed. The particular details and security requirements for this element as well as how the end-user is identified by its content are specific to each ASPSP implementing this functionality.	String [2048]

The [login_hint_token] can afterwards be used by the TPP in order to give back some clue about the PSU identity and usage context, especially:

- during a new OAuth2 Authorisation Code Grant (cf. § 3.4.2.3) or its OpenID connect equivalent (cf. § 3.4.2.4)
- during a new Client Initiated Backchannel Authentication (cf. § 3.4.2.5)
- during the submission of a Payment Request (through the supplementary data of the payload).

3.4.2.8. Use of the Access Token

The access token must be used within each request within the “Authorization” header, prefixed by the token type “Bearer”.

The [client_id] that is linked to the access token must directly or indirectly match with the Authorisation Number that is located within the TPP’s eIDAS certificate (QWAC).

If the access token is expired, the request will be rejected with HTTP401 with an error equal to “invalid_token” and the request can be replayed once the access token has been refreshed.

If the access token scope cannot cover the request (case of extended transaction history request for instance):

- The request will be rejected with HTTP403 with an error equal to “insufficient_scope”
- The refresh token will be revoked so the request could be replayed once a new token, having the right scope, would have been requested and provided.

3.4.2.9. Refreshing the Access Token

Refreshing the access token is only possible when the access token was granted through an OAuth2 “Autorization Code”, OpenID Connect or “Client Initiated Backchannel Authentication” Grants.

According to the RFC 6749 (cf. https://tools.ietf.org/html/rfc6749#section-6), the Refresh Token can be used by the TPP in order to get a refreshed Access Token by the following request.

```
POST /token HTTP/1.1
    Host: server.example.com

    grant_type=refresh_token
    &client_id={clientId}
    &refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
    &scope={scope}
```

NAME		DATA	TYPE AND CONSTRAINS
grant_type	[1..1]		Must be valued with “refresh_token”
refresh_token	[1..1]	Value of the provided refresh token	
client_id	[1..1]	TPP identification	String[36] must be equal or linked to the OrganizationIdentifier part of the Distinguished Name of the eIDAS certificate, according to ETSI specification
client_secret	[0..1]	The client secret	Since authentication of the client TPP is provided through MTLS, this parameter is not very useful.
scope	[0..1]	Specifies the generic accreditations that both the PSU and the TPP agreed on: “aisp” or “cbpii”. “extended_transaction_history” is not allowed in this case.	String[140] Space delimited roles list.

- The ASPSP
    - Identifies and authenticates the TPP through the presented eIDAS certificate (QWAC)
    - Checks the direct or indirect matching between the Authorization Number within the eIDAS certificate and the [client_id] value.
- The ASPSP answers through a HTTP200 (OK) response that embeds the following data.

NAME		DATA	TYPE AND CONSTRAINS
access_token	[1..1]	Access token provided by the ASPSP to the TPP.	String[140]
token_type	[1..1]	Type of the provided access token (“Bearer” or “MAC”)	String[10] Must be valued with “Bearer”
expires_in	[0..1]	Token lifetime, in seconds. The token can be used several times as far as it is not expired.	Numeric
refresh_token	[0..1]	Refresh token that can be replace the previous refresh token.	String[140]

If the refresh token has been revoked, the request will be rejected with HTTP400 and an error equal to “invalid grant”.

3.4.2.10. Refresh Token Revocation

The refresh token provided to an AISP is de facto revoked by the ASPSP

- After timeout of the by-law specified delay between two SCAs.
- After timeout of the ASPSP specified delay based on internal rules if any.
- After reject of a request for insufficient scope in order to allow the AISP to request another token with the desired scope.
- On request of a PSU wanting to revoke the TPP access on his/her account data.

The TPP is also able to ask for the revocation of the refresh token, according to RFC 7009 (cf. https://tools.ietf.org/html/rfc7009) through the following request.

```
POST /revoke HTTP/1.1
    Host: server.example.com

      token=45ghiukldjahdnhzdauz
      &token_type_hint=refresh_token
      &client_id={clientId}
```

NAME		DATA	TYPE AND CONSTRAINS
token	[1..1]	Token to be revoked by the ASPSP.	String[140]
token_type_hint	[0..1]	Information about the type of token to be revoked	Must be valued with “refresh_token”
client_id	[1..1]	TPP identification	String[36] must be equal or linked to the OrganizationIdentifier part of the Distinguished Name of the eIDAS certificate, according to ETSI specification
client_secret	[0..1]	The client secret	Since authentication of the client TPP is provided through MTLS, this parameter is not very useful.

- The ASPSP
    - Identifies and authenticates the TPP through the presented eIDAS certificate (QWAC)
    - Checks the direct or indirect matching of the [client_id] value with the Authorisation Number that is located within the TPP’s eIDAS certificate (QWAC).
    - Revokes the refresh token

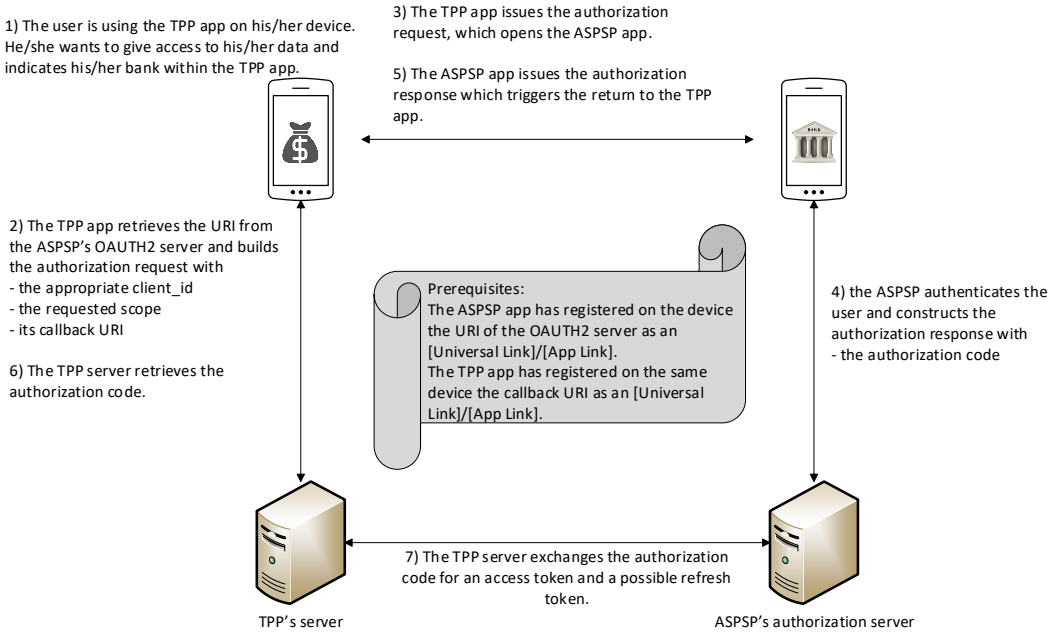
3.4.2.11. OAuth2 for native apps

The RFC 8252 (https://tools.ietf.org/html/rfc8252) extends the use of the OAuth Authorization request to applications that are installed on a given device (e.g. a smartphone).

Based on this RFC, one might consider having a straight through authorization process by using

- Universal Link (IOS based devices)
- App Link (Android based devices)

However, the API specification does not mandate this mechanism.



3.4.3. AISP authorization levels

Since a TPP is acting on behalf of a PSU being a PAO, the PSD2 use cases that are linked with AISP role require the following authorization levels:

- Authorization by Role
- Authorization by TPP-PSU agreement
- Authorization by PSU context

3.4.3.1. List of the relevant ASPSPs

When contracting with a TPP, the PSU will provide a list of the ASPSPs that it allows the TPP to access. This list may not be exhaustive and so may not include some of the PSU's ASPSPs.

3.4.3.2. Registration of the TPP-PSU agreement by each ASPSP

This registration is due to enable the further access of the TPP to the PSU's data that is hosted by a given ASPSP by providing the TPP with an OAuth2 access token.

The access token can be retrieved by one of the following Grants:

- OAuth2 Authorization Code grant (REDIRECT approach)
    - This grant can be enhanced with the following additional parameters borrowed from OpenID Connect:
        - [login_hint]
        - [ui_locales]
- OpenID Connect Grant (REDIRECT approach)
- Client Initiated Backchannel Authentication Grant (DECOUPLED approach)

Each ASPSP will have to document its own choice on this topic.

3.4.3.3. AISP OAuth2 Scopes

It is requested that AISP, CBPII or PISP roles will not be mixed within a single scope definition OAuth2 access token request.

The OAuth2 scope requested by an AISP can be one of the following values:

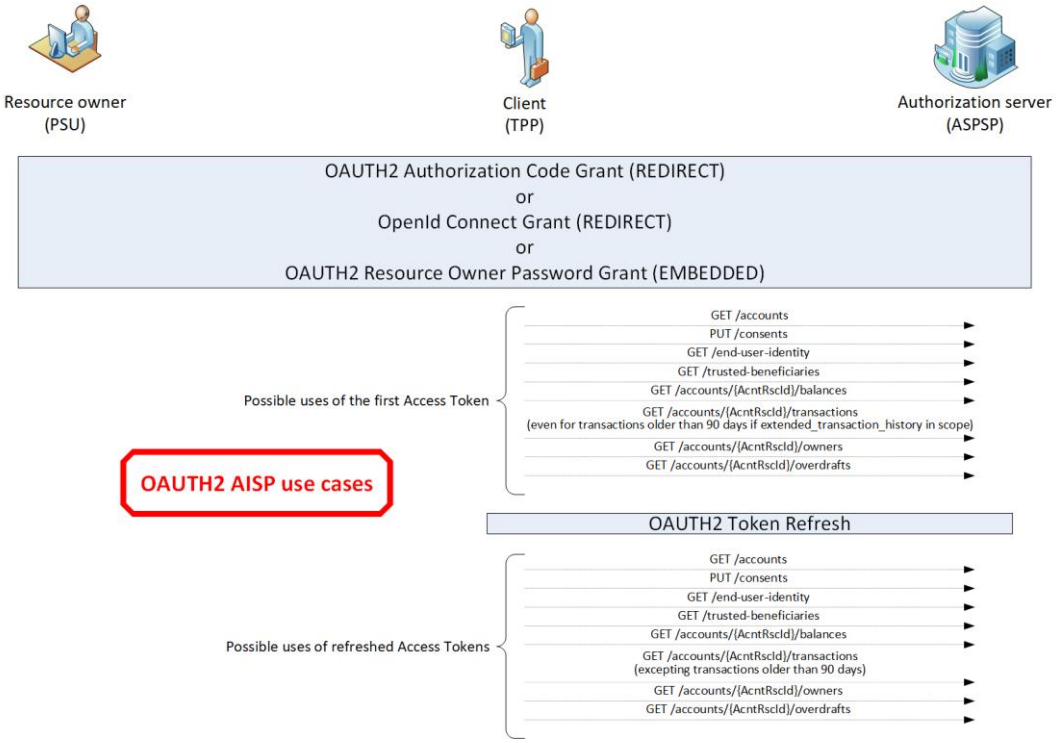
- “aisp”
- “aisp extended_transaction_history”

The first scope value allows the AISP accessing all accessible accounts and data allowed by the PSU until expiration of the by-law specified delay between two SCAs. However, the value does not allow requesting an extended transaction history, i.e. history including transactions older than 90 days.

The second scope value allows the AISP accessing all accessible accounts and data allowed by the PSU until expiration of the by-law specified delay between two SCAs. It also allows requesting an extended transaction history.

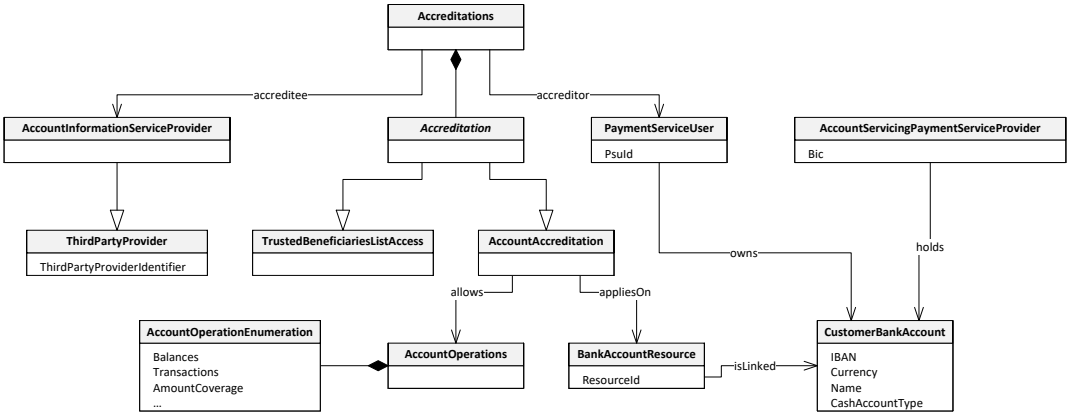
However, this “aisp extended_transaction_history” scope will be restricted to “aisp” by the ASPSP during the first token refresh. Thus:

- The AISP will be able to ask for an extended transaction history with the very first access token retrieved after a token request. So, in this case a single SCA will be required and used to get the token and to ask for an extended transaction history.
- Any further extended transaction history request will be considered as out of scope (cf. §3.4.2)



3.4.3.4. PSU detailed consent

The PSU detailed consent will specify which account or functionality will be accessible to the AISP. It can be seen as a collection of individual accreditations.



This collection is specific to a given PSU, a given TPP and a given ASPSP.

Each single accreditation relies on a specific account that is owned by the PSU and is held by the ASPSP. It specifies which pieces of data (transactions, balances) the TPP is allowed to carry out on this account.

The PSU manages this context with the AISP which is responsible of:

- The capture of the PSU choices:
    - The PSU specifies to the AISP which account and feature should be accessed or not.

- The execution of the PSU choices:
    - The AISP has the responsibility to respect the PSU choices and not to access any feature that it has not been granted for.

At any time, the PSU can edit his/her consent choices but this can only be done with the AISP.

Furthermore, the PSU consent may or may not be forwarded by the AISP to the ASPSP, according to one of the two following consent management models.

Full-AISP model (A1)

In this model, the ASPSP does not require to be informed of the details of the PSU consent.

Whatever the AISP request, the ASPSP will respond, being unable to check the compliance of the request against the PSU choices.

Actually, when getting the PSU context from the ASPSP (through the call [get /accounts]), the AISP will get all relevant HAL links and resource identifiers for each eligible account.

These HAL links will help the AISP to request the needed features on those accounts: balances and/or transactions.

Mixed model (A2)

In this model, the ASPSP does require to be informed of the details of the PSU consent. Therefore, the ASPSP has implemented an ad-hoc API entry-point that can be called by the AISP in order to forward the PSU choices.

Actually, when getting the PSU context from the ASPSP (through the call [get /accounts]), the AISP will get all eligible accounts but HAL links and resource identifiers will be provided only for the accounts on which consent was given by the PSU.

These HAL links will help the AISP to request the needed features on those accounts: balances and/or transactions.

Model choice

It is the charge of the ASPSP to implement or not the mixed model (A2). However, if this model has been implemented by the ASPSP, it is the charge of the AISP to forward the details of the PSU consent to the ASPSP whenever the PSU gives or edits this consent.

Once the details of the PSU consent has been received and saved by the ASPSP, the AISP, when getting the PSU context from the ASPSP (through the call [get /accounts]), will only get HAL links for authorized accounts and features.

3.4.4. CBPII authorization levels

Since a CBPII is acting on behalf of a PSU being a PAO, the PSD2 use cases that are linked with AISP and CBPII roles require the following authorization levels:

- Authorization by Role
- Authorization by TPP-PSU agreement
- Authorization by PSU context

However, in some cases, the CBPII might have been previously enrolled by the PSU to the relevant ASPSP (cf. §3.4.4.3).

3.4.4.1. List of the relevant ASPSPs

When contracting with a TPP, the PSU will provide a list of the ASPSPs that it allows the TPP to access. This list may not be exhaustive and so may not include some of the PSU's ASPSPs.

3.4.4.2. Registration of the TPP-PSU agreement by each ASPSP

This registration is due to enable the further access of the TPP to the PSU's data that is hosted by a given ASPSP by providing the TPP with an OAuth2 access token.

The access token can be retrieved:

- Either through an OAuth2 Authorization Code flow (REDIRECT approach)
- OpenID Connect Grant (REDIRECT approach)
- Client Initiated Backchannel Authentication Grant (DECOUPLED approach)

3.4.4.3. Pre-enrolled CBPII authorization level

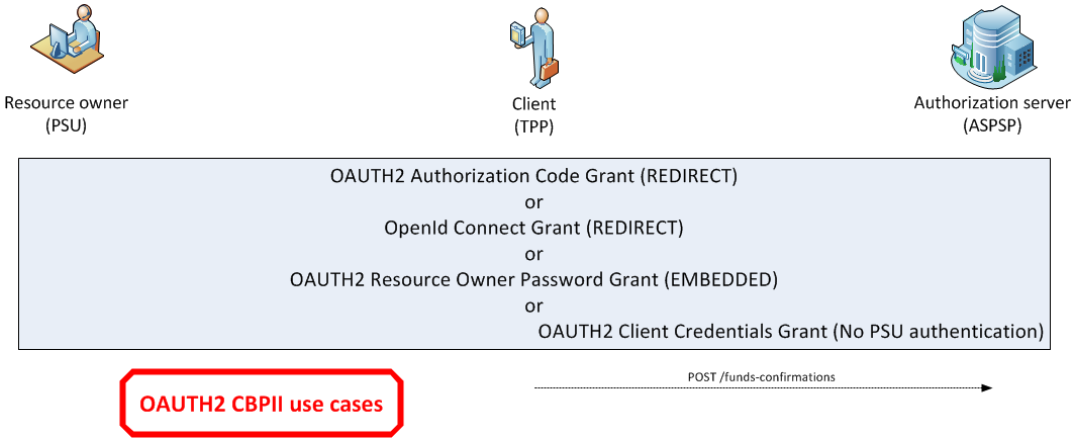
When the PSU has previously enrolled the CBPII to his/her relevant ASPSP, the latest may prefer to apply a simpler authorization scheme.

The access token can then be retrieved through an OAuth2 Client Credentials flow, aiming that PSU authentication is useless since the PSU consent was already captured.

3.4.4.4. CBPII scope

It is requested that AISP and CBPII roles will not be mixed within a single scope definition OAuth2 access token request.

The OAuth2 scope requested by a CBPII can only be “cbpii”.



3.4.5. PISP authorization levels and Fraud Management

3.4.5.1. Use cases

Posting and getting a Payment/Transfer Request

For posting or getting a Payment Request on behalf of a Merchant, or a Transfer Request on behalf of an Ordering Party, the PISP can use an access token that can be retrieved from the ASPSP through an OAuth2 Client Credentials flow.

However, the execution of the payment request requires a confirmation:

- From the PSU by a proper authentication
- From the PISP itself after completion of its fraud risk analysis.

In this perspective, during the posting of the Payment Request, the PISP will have to suggest the authentication approaches it supports. The ASPSP will then answer with the chosen authentication approach completed and the URL to be used for initiating the PSU authentication.

Confirmation of a Payment Request

This authentication shall be strong, unless exemption cases and can be performed through an ENFORCED REDIRECT or a DECOUPLED approach (cf. infra).

Once the PSU has confirmed the Payment Request to the ASPSP through this authentication, the PISP will get an access token. The PISP must use this access token for its own confirmation of the Payment Request after having checked, for instance, the absence of potential security flaw.

Cancellation of a Payment Request

In case the PISP has to cancel a payment request, the Access token to be used can be retrieved from the ASPSP through an OAuth2 Client Credentials flow.

However, the ASPSP may require a confirmation through an authentication of the PSU. This authentication can be performed through a SIMPLE REDIRECT or a DECOUPLED approach (cf. infra).

In this perspective, during the Cancellation Request, the PISP will have to suggest the authentication approaches it supports. The ASPSP will then answer:

- either with the decision of not processing the PSU authentication; the cancellation is then effective
- or with the chosen authentication approach completed with the URL to use for initiating the PSU authentication; the cancellation will be effective after this PSU authentication.

3.4.5.2. SIMPLE REDIRECT Approach

This approach can only be used for a Payment Request cancellation.

The PSU authentication is then processed through a simple redirection of the PSU to the ASPSP authentication server by using the URL that was initially provided by the ASPSP.

The ASPSP authenticates the PSU and then redirects the latest by using one of the call-back URLs that were provided by the PISP.

3.4.5.3. ENFORCED REDIRECT Approach

This approach is mandatory for a Payment Request confirmation in REDIRECT approach.

An Authorization Code token will be used for the confirmation. The PSU authentication is processed through the Authorization Code flow with the ASPSP authentication server.

Purpose and risk analysis

The payment initiation may indeed face some security issues in REDIRECT approach.

A first attack (session fixation) might happen, based on the fact that a given PSU will forward the redirection request to another PSU who can be in a situation to authenticate and pay the purchase made by the first PSU.

Moreover, even if the first attack is mitigated, the attacker might also try to simulate the redirection (fake redirect) to the TPP in order to induce the confirmation of the payment request to the ASPSP.

Session fixation protection

In order to avoid the session fixation attack, the PISP must ensure there is no “PSU-switch” during redirection. This can be done by managing a nonce that

- will be stored in the PSU user agent session before the redirection to the ASPSP and
- will be retrieved from the PSU user agent after the redirection.

In case the retrieval failed, the chances are good there was such an attack. The PISP should then cancel the payment request for fraud reason.

Otherwise, in case of successful nonce retrieval, the PISP can confirm the payment request to the ASPSP who is then able to trigger the relevant Credit Transfers.

Fake redirect protection

In order to post the confirmation, the PISP has to request an Authorization Code token from the ASPSP.

In response to the payment request, The ASPSP has provided the PISP with the URI of the Authorisation server. Some OAuth2 parameters must have been pre-valued:

- [response_type] valued with “code”
- [scope] valued with “pisp”
- [context] valued with a hint to the payment-request

The PISP will complete this URL with its own OAuth2 parameters

- [client_id]
- [state] if needed
- [redirect_uri] as call-back URL.

The OAuth2 Authorization Code grant can then complete (cf. §3.4.2.3).

After the retrieval of the Authorization Code through the redirection of the PSU back to the PISP, the latest must then ensure, by the nonce check mechanism, there was no “PSU-switch” during the redirection, as previously explained.

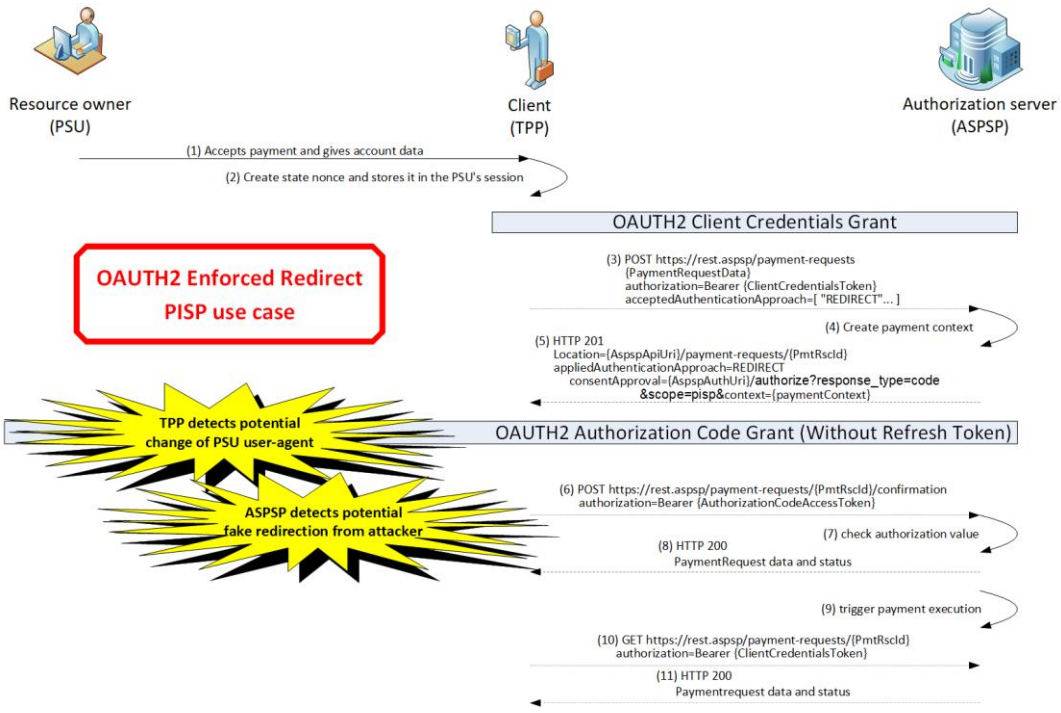
The PISP can then exchange the Authorization Code against the Access token.

- the lifetime of the access token is specified by the Authorization Server in order to limit the usability period.
- no refresh token has to be provided.

The confirmation is then posted, using this Access token.

In case of face redirect attack, the Access token could not have been retrieved by the PISP. Even in confirmation attempt, the ASPSP can detect the absence of the token and will then reject the payment request for FRAUD reason.

Otherwise, the confirmation sent by the PISP will lead to the normal triggering of the relevant Credit Transfers.



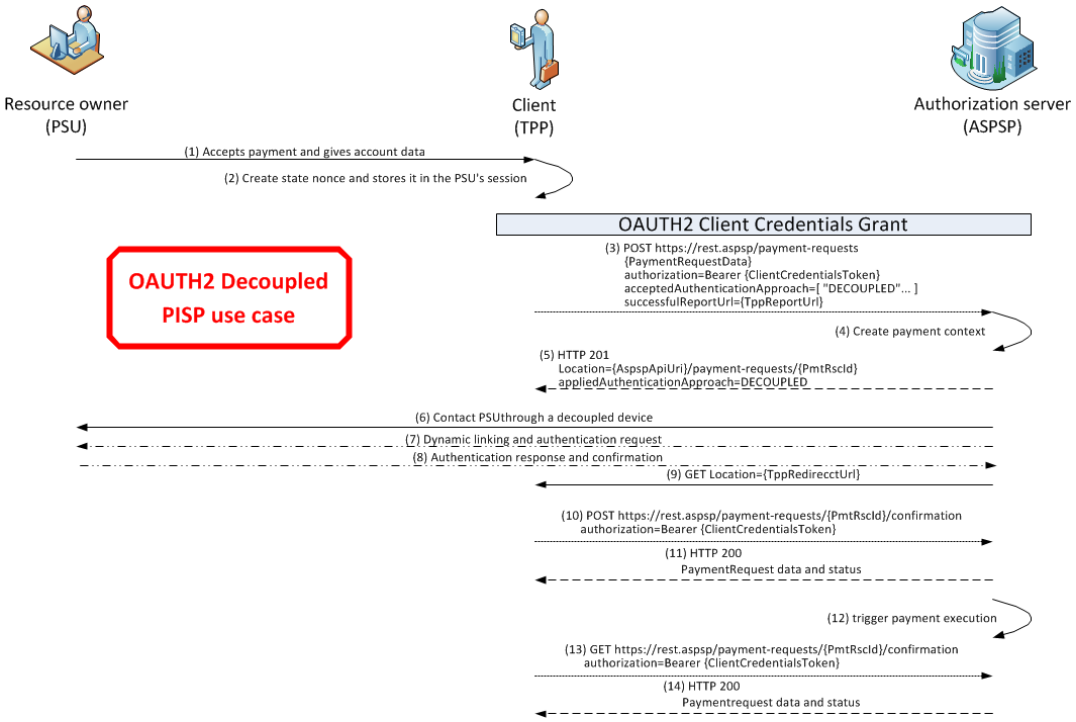
3.4.5.4. OAuth2 DECOUPLED Approach

In this approach, the Client Credential token can be used for all PISP use cases:

- Posting a payment request
- Getting the previously posted payment-request
- Modifying for cancellation the payment request
- Confirming the payment request

The PSU authentication is processed through a decoupled channel initiated by the ASPSP.

After PSU authentication, the PISP is informed by a direct call by the ASPSP. The PISP can then confirm the payment request that will lead to the normal triggering of the relevant Credit Transfers.



3.4.5.5. Recapitulative Table

	SIMPLE REDIRECT	ENFORCED REDIRECT	DECOUPLED
Nonce mechanism protection applied by PISP	The nonce must be computed by the PISP and stored within the PSU user-agent as far as the PISP accepts Simple REDIRECT or Enforced REDIRECT approaches when posting or cancelling a payment request.		

	SIMPLE REDIRECT	ENFORCED REDIRECT	DECOUPLED
Successful and Unsuccessful Report Uri provided by PISP	To be used by the ASPSP through PSU redirection		To be used directly by the ASPSP after PSU authentication
Accepted Authentication Approach set by PISP	Must include “REDIRECT”		Must include “DECOUPLED”
Applied Authentication Approach set by ASPSP	“REDIRECT”		“DECOUPLED”
PSU authentication	For confirmation of a cancellation	For confirmation of a Payment Request	

3.5. Applicative authentication

Each request sent by the TPP has to be signed and ASPSP might also apply sign their responses.

If the ASPSP notes that the signature is either absent or invalid for a given request, it shall reject this request with HTTP400.

If the TPP notes that the signature is invalid for a given response, it shall warn the relevant ASPSP.

3.5.1. Http-Signature Mechanism

Http-signature mechanism is specified by the following IETF draft-paper:

- https://datatracker.ietf.org/doc/draft-cavage-http-signatures/

One must notice that this IETF draft-paper has now been replaced by a new draft :

- https://datatracker.ietf.org/doc/draft-ietf-httpbis-message-signatures/

However, since OpenBankingEurope and ETSI, together with several PSD2 API inititiatives including STET, published a new signature standard, it is strongly advice to consider replacing http-signature by this new signature standard (cf. 3.5.2).

3.5.1.1. Signature computation

The way it should be implemented is the following

- Computing a SHA256 digest of the HTTP body and adding this digest as an extra HTTP header.

- Using a specific Qualified Certificate (QSealC), respecting the ETSI/TS119495 Technical Specification, in order to apply an RSA-SHA256 signature on
    - all the following headers that are present within the HTTP request sent by the TPP, including the previously computed digest
        - Date (if available)
        - Content-Type (when there is a payload)
        - Content-Length (when there is a payload)
        - X-Request-Id
        - All available "PSU"-prefixed Headers (cf. § 3.5.2)
        - the specific “(request-target)” pseudo-header which is specified by the IETF draft-paper

    - all the following headers that are present within the HTTP response given by the ASPSP, including the previously computed digest
        - Date (if available)
        - Content-Type (when there is a payload)
        - Content-Length (if available)
        - X-Request-Id

- Adding this signature within an extra HTTP header embedding
    - The key identifier which must specify the way to get the relevant qualified certificate (see below)
    - The algorithm that has been used
    - The list of headers that have been signed
    - The signature itself.

Since version #11 of the draft, two new pseudo-headers have been introduced in order to strengthen the signature: (created) and (expires). However, work is still going on this subject and the use of these two fields is not yet recommended.

EXTRA HTTP HEADER	DATA	COMMENT
Digest	Digest of the body	
Signature	http-signature of the request (cf. https://datatracker.ietf.org/doc/draft-cavage-http-signatures/)	

3.5.1.2. Value of key identifier

It is requested that this identifier is valued with:

- Either the keyId that has been assigned by the authorization server during the OAuth2 technical setup (cf. § 3.4.2.2).
- Either a URL aiming to provide the relevant Qualified Certificate.
    - In order to assure an easy discrimination of the certificate among others, it is requested that the last part of the URL to the certificate be suffixed by an underscore followed by the SHA-256 fingerprint of the certificate.
        - E.g.: https://path.to/myQsealCertificate_714f8154ec259ac40b8a9786c9908488b2582b68b17e865fede4636d726b709f
    - This URL could have been provided during the OAuth2 technical setup within the “5xu” field of the JKS provided by the TPP (cf. RFC7517).

3.5.2. JSON Web Signature Profile for Open Banking

This signature mechanism is specified by the following document:

- https://www.openbankingeurope.eu/media/2095/obe-json-web-signature-profile-for-open-banking.pdf

It relies on the use of a Json Web Signature (JWS as specified by RFC7515).

The way it should be implemented is the following

- Choosing a Qualified Certificate (QSealC), respecting the ETSI/TS119495 Technical Specification and belonging to the sender of the request or the response as signing certificate.
- Choosing a signature Algorithm among those listed in both RFC7518 and ETSI TS 119 312, although a regular review of potential weaknesses of these algorithms is highly recommended.
- Computing a SHA256 digest of the HTTP body and adding this digest as an extra HTTP header.
- Building a JWS protected header whose
    - [sigD/pars] field will list all the lower-case HTTP header fields to be signed (see below) and encoding it into Base64url.
    - [x5t#S256] field will be valued with the Base64url encoded hash of the previously chosen signing certificate
    - [alg] field will be valued with the signature algorithm name
- Building the HTTP header string as the list of all headers/values to be signed.
    - all the following headers that are present within the HTTP request sent by the TPP, including the previously computed digest
        - Date (if available)
        - Content-Type (when there is a payload)

        - Content-Length (when there is a payload)
        - X-Request-Id
        - All available "PSU"-prefixed Headers (cf. § 3.5.2)
        - the specific “(request-target)” pseudo-header which is specified by the IETF draft-paper
    - all the following headers that are present within the HTTP response given by the ASPSP, including the previously computed digest
        - Date (if available)
        - Content-Type (when there is a payload)
        - Content-Length (if available)
        - X-Request-Id
- Concatenating the Base64url encoded JWS protected header and the HTTP header string with a dot (“.”) as separator.
- Computing the signature, using the chosen certificate and algorithm, on the previous concatenation result.
- Concatenating the Base64url encoded JWS protected header and the resulting signature with a double-dot (“..”) as separator

Adding the previous concatenation result as an extra “x-jws-signature” HTTP header

3.6. Fraud-detection-oriented information

The following extra HTTP-headers must be used within the HTTP request sent by the TPP, provided the relevant pieces of data are available within the connection between the PSU and the TPP. This forwarding allows the ASPSP to integrate this information into its own fraud detection process.

Moreover, these headers can be considered as proof of the PSU being connected.

EXTRA HTTP HEADER	DATA	COMMENT
PSU-IP-Address	IP Address of the PSU terminal when connecting to the TPP	In regards with GDPR rules, this must be subject to PSU's consent
PSU-IP-Port	IP Port of the PSU terminal when connecting to the TPP	
PSU-HTTP-Method	HTTP Method used for the most relevant PSU's terminal request to the TTP	
PSU-Date	Timestamp of the most relevant PSU's terminal request to the TTP	
PSU-User-Agent	"User-Agent" header field sent by the PSU terminal when connecting to the TPP	
PSU-Referer	"Referer" header field sent by the PSU terminal when connecting to the TPP	
PSU-Accept	"Accept" header field sent by the PSU terminal when connecting to the TPP	
PSU-Accept-Charset	"Accept-Charset" header field sent by the PSU terminal when connecting to the TPP	
PSU-Accept-Encoding	"Accept-Encoding" header field sent by the PSU terminal when connecting to the TPP	
PSU-Accept-Language	"Accept-Language" header field sent by the PSU terminal when connecting to the TPP	
PSU-GEO-Location	The forwarded Geo Location of the corresponding HTTP request between PSU and TPP if available.	In regards with GDPR rules, this must be subject to PSU's consent
PSU-Device-ID	UUID (Universally Unique Identifier) for a device, which is used by the PSU, if available. UUID identifies either a device or a device dependant application installation. In case of installation identification this ID need to be unaltered until removal from device.	In regards with GDPR rules, this must be subject to PSU's consent

3.7. Other specific HTTP headers to be used

EXTRA HTTP HEADER	DATA	COMMENT
X-Request-ID	Correlation header to be set in a request and retrieved in the relevant response.	
Idempotency-Key	Idempotency key to be added by the API client for ensuring that a given request, when retried twice or more, will indeed only be executed once by the API server. This applies especially on POST requests.	The rules for uniqueness, validity and expiry must be set according to: https://datatracker.ietf.org/doc/draft-ietf-httpapi-idempotency-key-header/

3.8. Specific HTTP return codes and messages to be used

MESSAGE	HTTP CODE	SIGNIFIANCE
FORMAT_ERROR	400	Format of certain request fields are not matching the XS2A requirements. An explicit path to the corresponding field might be added in the return message.
RESOURCE_UNKNOWN	404	If resourceId in path
PERIOD_INVALID	400	Requested time period out of bound.

MESSAGE	HTTP CODE	SIGNIFIANCE
ACCESS_EXCEEDED	429	The access on the account has been exceeding the consented multiplicity per day.
REQUESTED_FORMATS _INVALID	406	The requested formats in the Accept header entry are not matching the formats offered by the ASPSP.

3.9. STET PSD2 API technical summary

TOPIC	CHOICE	COMMENT
Access network	Internet	
Network protocol	HTTP 1.1 (Minimum)	
Data encryption **Cross-authentication**	TLS 1.2	Could be enforced through STS and/or PFS
Authorization protocol	OAuth2	In respect of RFC 6749, 7009 One of the following token modes - Authorization Code Grant (AISP, CBPII and PISP) for REDIRECT approach - Client credential (PISP, CBPII) Based on MTLS, the identity of the TPP is provided by its eIDAS certificate during OAuth2 procedures. https://datatracker.ietf.org/doc/rfc8705/ The OpenId Connect extension can also be used in place Authorization Code Grant The Client Initiated Backchannel Authorization Grant can be used in order to implement a DECOUPLED approach
Applicative protocol	REST	In respect of the Richardson Maturity Model, on level three in order to provide HYPERMEDIA links.
Applicative authentication	http-signature	Notice this is actually an IETF draft, waiting for approval and so subject to some modifications. https://datatracker.ietf.org/doc/draft-cavage-http-signatures/
PSU Strong Customer Authentication approaches	REDIRECT, DECOUPLED	
Data format	JSON/UTF8	With use of ISO20022 based data structures
Technical documentation	SWAGGER 2.0	Date/Time format must respect ISO8601 and RFC3339 in accordance with OpenApi specifications. The creator of a Date/Time shall choose - Any time-zone format although UTC format is recommended. - Any second fraction format, including no second fraction. A simple date can be specified as an UTC date/time with a time part equal to “00:00:00Z”. The recipient of a Date/Time must be able to interpret its value as far as it is compliant with ISO8601 and RFC3339.