ETSI TS 103 407 V1.1.1 (2016-04)

Cross Platform Authentication for limited input hybrid consumer equipment

Reference
DTS/JTC-033

Keywords
authentication, authorization, protocol

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights 5
Foreword 5
Modal verbs terminology 5
Executive summary 6
Introduction 6
1 Scope 7
1.1 General 7
2 References 7
2.1 Normative references 7
2.2 Informative references 7
3 Definitions and abbreviations 8
3.1 Definitions 8
3.2 Abbreviations 8
4 Protocol overview 9
4.1 Introduction 9
4.2 Modes of association 10
4.2.1 Overview 10
4.2.2 Authenticated association ('user mode') 10
4.2.3 Unauthenticated association ('client mode') 10
5 Core concepts 10
5.1 Bearer token 10
5.2 Domain 11
6 Roles 11
6.1 Client 11
6.2 Service provider 11
6.3 Authorization provider 11
7 The device flow 12
7.1 Protocol version 12
7.2 Basic principles 12
7.2.1 HTTPS 12
7.2.2 JSON 12
7.2.3 Integrating applications with the CPA protocol 12
7.2.4 Example of authenticated association using the CPA device flow 12
7.3 User mode 13
7.4 Client mode 14
7.5 Automatic provisioning of tokens 16
7.5.1 Overview 16
7.5.2 User signs in and enters a pairing code 16
7.5.3 User signs in and gives confirmation 17
7.5.4 Token is automatically granted 19
7.5.5 Refreshing an expired access token 20
7.6 Deleting the association between a client and an authorization provider 21
7.6.1 Overview 21
7.6.2 Deleting the association on the client side 21
7.6.3 Deleting the association on the authorization provider side 22
8. Client/Authorization Provider API 22
8.1 Overview 22
8.2 /register - Register a client 23
8.2.1 /register request 23
8.2.2 /register response 23
8.3 /associate - Associate a client with a user account 24

8.3.1 /associate request 24
8.3.2 /associate response 24
8.3.2.1 Pairing the client with a user account 24
8.3.2.2 Confirmation required before granting access to a new service 25
8.3.2.3 Access automatically granted to a new service 26
8.4 /token - Obtain a bearer token 27
8.4.1 /token request 27
8.4.1.1 Client mode 27
8.4.1.2 User mode 27
8.4.1.3 Refreshing an expired access token 28
8.4.2 /token response 28
8.5 Verification endpoint 30
8.5.1 Verification endpoint request 30
8.5.2 User permission request 30
8.5.3 Redirection response 30
9 Service Provider/Authorization Provider API 31
9.1 Overview 31
9.2 Establishing trust between the service provider and authorization provider 31
9.3 /authorized - Access token verification endpoint 31
9.3.1 /authorized request 31
9.3.2 /authorized response 31
Annex A (informative): How to integrate applications with the CPA protocol 33
A.1 Introduction 33
A.2 Authentication challenge 33
A.3 Accessing a protected resource 34
Annex B (informative): Bibliography 35
B.1 Reference implementation 35
B.2 Related documents 35
Annex C (informative): Change History 36
History 37

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

NOTE: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has active members in about 60 countries in the European broadcasting area; its headquarters is in Geneva.

European Broadcasting Union
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Executive summary

The Cross Platform Authentication (CPA) protocol defines how to securely associate an IP-connected media device (such as a hybrid radio, set top box or Smart TV) with the online account of a user of a set of web services delivered to that device.

Introduction

The present document specifies version 1.0 of the Cross Platform Authentication (CPA) protocol.

The CPA protocol is specifically designed for devices with limited input and display capabilities that are not addressed by existing standards and to cater for companies that share a back-end authorization provider for managing identities but implement services separately.

The CPA protocol defines a clear separation of responsibilities between the service provider and the authorization provider. This enables the protocol to be applied in a variety of business configurations typical of the broadcast industry. For example, the CPA protocol can be used in the following scenarios:

- the service provider and the authorization provider are both managed by the same company;
- an umbrella company manages the authorization provider (so centralizing identity management) but keeps its service providers separate from one another;
- service providers from different companies use the same central authorization provider managed by a separate company (e.g. a national federated identity service).

1 Scope

1.1 General

The protocol described in the present document is intended for devices with limited input capabilities, such as hybrid radios, IP-connected set top boxes and Smart TVs, that can communicate with web services over HTTPS.

The protocol specifies two APIs:

- the API between a client device and an authorization provider by which a client obtains a bearer token;
- the API between a service provider and an authorization provider by which a service provider verifies an access token.

The present document gives an overview of the protocol (clause 4), covering the core concepts (clause 5) and roles (clause 6) used in CPA and how the device flow works (clause 7).

The CPA APIs are specified in the present document in clauses 8, Client/Authorization Provider API and clause 9, Service Provider/Authorization Provider API.

An informative annex A describes how service providers can tell clients that the option to authenticate using the CPA protocol is available, and how the bearer token obtained via CPA should be used to access protected resources. Although this clause is not normative, it is strongly recommended these conventions are followed where possible to maximize interoperability.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document:

[1] IETF RFC 2818: "HTTP Over TLS".

[2] IETF RFC 7159: "The JavaScript Object Notation (JSON) Data Interchange Format".

[3] IETF RFC 4122: "A Universally Unique Identifier (UUID) URN Namespace".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] IETF RFC 6749: "The OAuth 2.0 Authorization Framework".

[i.2] IETF RFC 6750: "The OAuth 2.0 Authorization Framework: Bearer Token Usage".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

access token: bearer token used by a client as authority to access a service provider

authenticated association: association in which the client identity is associated with a user identity

authorization: granting of permission based on authenticated identification

authorization provider: service that manages client identities and the association between client identities and authenticated user identities

bearer token: security token the possession of which entitles the bearer to all rights associated with it

client: application managed entity uniquely identified by an authorization provider

client mode: authorization mode where the client relation is not associated with a user account

device: IP-connected hardware intended for user interaction

device flow: series of exchanges between the client and the authorization provider by which the client obtains a bearer token

identity provider: service that manages identity information on behalf of users and services

limited display device: device with the ability to display at least two rows of sixteen alphanumeric characters from the ISO-646 Invariant Code Set (loosely, 7-bit ASCII)

limited input device: device that has at least the ability to accept or cancel a proposed action

registration: process by which a client obtains a client identity from an authorization provider

service provider: service requiring authorization to access its protected resources

token: unique opaque string used to represent a specific granting of authorization to use a service

NOTE: See also access token and bearer token.

unauthenticated association: association in which the client identity is not associated with a user identity

user: person or agent using an application

user mode: authorization mode where the client relation is associated with a user account

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

API	Application Programming Interface
CPA	Cross Platform Authentication

NOTE The subject of the present document.

HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
OAuth	Open Authorization
URI	Uniform Resource Identifier

4 Protocol overview

4.1 Introduction

The present document defines the protocol between a client device and an authorization provider by which a client device acquires an authorization token it may use to access protected resources and by which the service provider can identify the client device and any associated user account.

Cross Platform Association (CPA) is based on OAuth 2.0 IETF RFC 6749 [i.1] and the bearer tokens used in CPA are compatible with OAuth 2.0 bearer tokens IETF RFC 6750 [i.2]. While CPA can be used more generally, the focus of the present document is on IP-connected media devices with limited input and display. When such a device cannot run a fully capable HTTP User Agent such as a browser, the user needs access to another device which can run such software to complete the authorization.

In the typical CPA device flow described below, the application-specific steps are prefixed with (App) and the CPA-specific steps with (CPA):

- (App): An application on the device requests a protected resource from the service provider.
- (App): The service provider checks with the authorization provider to see if the device is authorized.
- (App): The authorization provider indicates that the device is not authorized.
- (App): The service provider response to the device includes the endpoint where the device may initiate the device flow and which modes of association the service provider supports.
- (CPA): The device registers itself with the authorization provider and receives a `client_id` and `client_secret` in return.
- (CPA): The device makes a request to the authorization provider to associate the device.
- (CPA): The authorization provider returns a verification URI and one-time `user_code` which the device displays to the user. It also returns a corresponding `device_code` which is not displayed.
- (CPA): The device polls the authorization provider using the `device_code` as identification.
- (App): In the meantime, the user uses a browser on another computer to open the supplied verification URI to enter the code. As part of this process, the user will be required to authenticate their user identity, typically by logging in via their identity provider.
- (CPA): Once the user has verified the association by entering their code, the authorization provider responds to the polling device by returning a bearer token.
- (App): The application running on the device can then reissue the request for the protected resource, this time including the supplied bearer token.

In the device flow outlined above, CPA specifies the protocols between a) the client device and the authorization provider and b) the service provider and the authorization provider.

The client/authorization provider API (see clause 8) consists of the following endpoints supplied by the authorization provider:

- /register to obtain a client identity consisting of a `client_id` and `client_secret;`
- /associate to initiate the association between a client and a user account;
- /token to acquire a token;
- the verification endpoint to authenticate the user's identity and verify the association.

The service provider/authorization provider API (see clause 9) consists of one endpoint supplied by the authorization provider:

- /authorized to verify a token

All other interactions between the participants in the CPA protocol are outside the scope of the present document. However, the present document includes illustrative examples to show how CPA may be integrated into an application-specific flow. These integration points include:

- how the client discovers the authorization endpoint (see clause A.2);
- how the service provider and authorization provider establish a trusted relationship (see clause 9.2);
- how a user verifies the association between their client and their user account (see clause 8.5).

4.2 Modes of association

4.2.1 Overview

CPA defines two modes of association: authenticated association ('user mode') which links a device to a user account, and unauthenticated association ('client mode'), which provides a client identity to a device without linking it to a user account.

In both cases, the client is issued a persistent client ID so that it can be identified across requests to service providers that use the same authorization provider.

Authorization providers may offer the following forms of association:

- client mode only;
- user mode only;
- client mode and user mode.

4.2.2 Authenticated association ('user mode')

If the authorization provider offers user mode, then the client ID may be linked with a user account so that requests from the device can be linked to that user. This linking of device and user is called 'authenticated association', or user mode (see clause 7.3).

Once an authenticated association has been set up, a service provider can provide a unified personalized service across a range of devices.

4.2.3 Unauthenticated association ('client mode')

The present document also defines a way to create an unauthenticated association between a device and an authorization provider, known as *client mode* (see clause 7.4), and a means of migrating from this unauthenticated association to an authenticated association.

The purpose of unauthenticated association is to support applications that require service-side persistence but do not require association with an online user account.

By using the `client_id` as an identifier, a service provider can identify a client from session to session and hence provide personalization limited to a specific device.

5 Core concepts

5.1 Bearer token

The CPA device flow is built around the concept of a *bearer token*, possession of which grants access to protected services IETF RFC 6750 [i.2].

The core function of the CPA protocol is to securely get this bearer token onto the device.

A *token* is a unique string that is intended to be unforgeable. The token is a shared secret between the client, the service provider and the authorization provider.

A *bearer* token is one for which simple possession confers the corresponding authority to its bearer.

In CPA, the token also serves as the key for looking up the association between a client and a user account. It is not a permanent identifier as it may expire and be replaced with a new token.

5.2 Domain

Tokens are valid only within the scope of the hostname of the service provider for which they are granted. This is to prevent service providers enabling the issuing of tokens for domains for which they are not responsible.

Tokens are valid for all service provider endpoints in that domain. For example, a token granted for the domain ***api.example.com*** is valid for the endpoints ***https://api.example.com/tag*** and ***https://api.example.com/epg***.

The domain is specified by the client when it requests the token. The authorization provider decides whether the client is issued a token for the requested domain.

6 Roles

6.1 Client

The client represents the user of a service on a device.

When a client registers with an authorization provider, it is given a unique `client_id` and a `client_secret`.

The `client_id` is known to the client, the service provider and the authorization provider. This client identity is shared between all service providers that use the same authorization provider.

A `client_id` shall be unique within an authorization provider and shall be valid only for that authorization provider. Different authorization providers shall not share client identities.

The `client_secret` shall be known only to the client and the authorization provider. All requests from the client to the authorization provider shall include the `client_id` and `client_secret.` The `client_secret` shall be used only in requests from the client to the authorization provider.

6.2 Service provider

The service provider provides one or more web services under the same domain.

Each bearer token is valid only for the service provider domain for which it was issued.

The exact details of how a service provider indicates to a client which authorization provider to use and which protected resources are accessible with a CPA bearer token are application-specific and not defined by the present document.

One way is for the service provider to return a `WWW-Authenticate` header in response to a request for a protected resource. This header tells the client the URI of the authorization provider and which association modes are available. For a fuller description of this approach, see clause A.2.

6.3 Authorization provider

An authorization provider provides client identities to one or more clients and maintains the link between the client identity and the user identity.

An authorization provider may answer authorization requests for one or more service providers.

Service providers may be grouped within an authorization provider to share an authorization domain. This enables the automatic provisioning of tokens for service providers within the same group once a client has been verified for one service provider within that group.

A secure trusted connection should be used between the service provider and the authorization provider and the authorization provider should verify the identity of the service provider.

7 The device flow

7.1 Protocol version

This present document describes version 1.0 of the CPA protocol. The method that service providers should use to indicate the version in use is application specific, for example, see clause A.2.

7.2 Basic principles

7.2.1 HTTPS

All communication between the client and the service provider, between the client and the authorization provider, and between the service provider and the authorization provider shall be conducted over HTTPS as specified in IETF RFC 2818 [1]. Clients shall verify the certificates used by the service provider and authorization provider.

7.2.2 JSON

All requests made by the client to the authorization provider shall use JSON IETF RFC 7159 [2] in the request body to specify parameters. All responses from the authorization provider shall use JSON in the response body to return data to the client.

Error responses (4xx status codes) shall contain a body of the form:

```
{
    "error": "..."
}
```

where `"..."` indicates a string describing the specific error that has occurred.

7.2.3 Integrating applications with the CPA protocol

How the client discovers the URI of the authorization provider and how it specifies the bearer token to the service provider are application specific and so outside the scope of the present document. There are, however, some conventions that have arisen in practice which should be followed where applicable:

- The service provider should issue a `WWW-Authenticate` challenge (see clause A.2) in response to a request for a protected resource.
- The client should include the bearer token in the header in requests for protected resources (see clause A.3).

Within the CPA protocol itself, all participants shall use JSON payloads to communicate.

7.2.4 Example of authenticated association using the CPA device flow

The following is an example of authenticated association using the CPA device flow from the user's point of view:

- Alice turns on her radio and it starts playing her favourite channel. An application running on the radio detects that a service is available for this channel that offers Alice the ability to bookmark the currently played track by pressing the 'tag' button. Alice presses the 'tag' button.
- The radio calls the service provider and receives a reply containing an authorization challenge specifying the authorization provider to use. The radio *registers* with the authorization provider (`/register`) and receives a client identity and secret. The radio can now offer Alice the ability to *associate* her radio with her online account with the broadcaster. Alice decides to do this.
- The radio calls the authorization provider to initiate the association (`/associate`). The authorization provider returns a URI to visit and a code to enter there. The radio displays these to Alice who then visits the address specified by the URI in her browser, signs in (out of scope of the present document) and enters the code.

- In the meantime, the radio polls the `/token` endpoint to determine whether Alice has completed the process. When Alice has successfully entered the code, the request to `/token` returns a new bearer token authorizing access to the requested service.
- The radio may now use this bearer token in future calls to the service provider. The service provider asks the authorization provider whether the token is authorized. The authorization provider verifies the token and returns the identifier for Alice's account and the `client_id` of the radio client.

7.3 User mode

In user mode, interactions between the client and the service provider are associated with a user account. The process by which the client obtains an access token to access protected resources at the service provider is shown in figure 1:

1) The client first requests a resource from the service provider, either without supplying a bearer token, or using an invalid access token. The service provider checks the validity of the access token using the authorization provider's `/authorized` endpoint. If the token is missing or invalid, the authorization provider returns `404 Not Found` to the service provider. The service provider then returns `401 Unauthorized` to the client, including in the response an authentication challenge using the `WWW-Authenticate` HTTP header that specifies the authentication provider URI and the modes supported by the service provider, as described in clause A.2.

2) If the client has not already registered with this authorization provider, the client POSTs to the authorization provider's `/register` endpoint. The authorization provider registers the client and returns a `client_id` and `client_secret` with HTTP status `201 Created`. The client should store this information keyed by the authorization provider's domain name to be able to use the `client_id` with another service provider using the same authorization provider.

3) The client initiates the process of associating with a user account by POSTing its `client_id`, `client_secret` and the domain of the service provider to the authorization provider's `/associate` endpoint. The authorization provider returns a `verification_uri` and `user_code` to display to the user and a `device_code` to be used by the client in the next step.

4) Using the `device_code`, the client starts polling the authorization provider's `/token` endpoint to check if the association has been validated by the user. The authorization provider returns a `202 Accepted` response, indicating that the user has not verified the association using the `user_code` yet.

5) The user visits the `verification_uri`, signs in using a secure method (out of scope of the present document) and enters the `user_code`. The authorization provider then creates the relationship between the `client_id` and the user identity.

   As an alternative to requiring the user to manually enter the `user_code`, a client on a device capable of presenting a web browser to the user, such as a smartphone or tablet, may instead use the following method:

   i) The client opens a web browser at the `verification_uri`, providing the `user_code` and a `redirect_uri` as HTTP GET parameters.

   ii) The user signs in as above and explicitly confirms the association between the client and the service provider.

   iii) The website at the `verification_uri` then redirects the browser to the `redirect_uri`. In the case of a mobile application (e.g., on certain mobile operating systems), the act of navigating to the `redirect_uri` can be used to cause the client to switch back to the application.

6) The client polls the authorization provider's `/token` endpoint and receives a `200 OK` response containing the `access_token` and the `user_name`.

7) The client is now able to request the protected resource from the service provider, using the `access_token`. The service provider checks if the `access_token` is valid using a POST request to the authorization provider's `/authorized` endpoint. If it is valid, the authorization provider returns `200 OK` to the service provider including the `client_id` and `user_id`. The service provider returns the protected resource to the client, possibly personalized using the `user_id`.

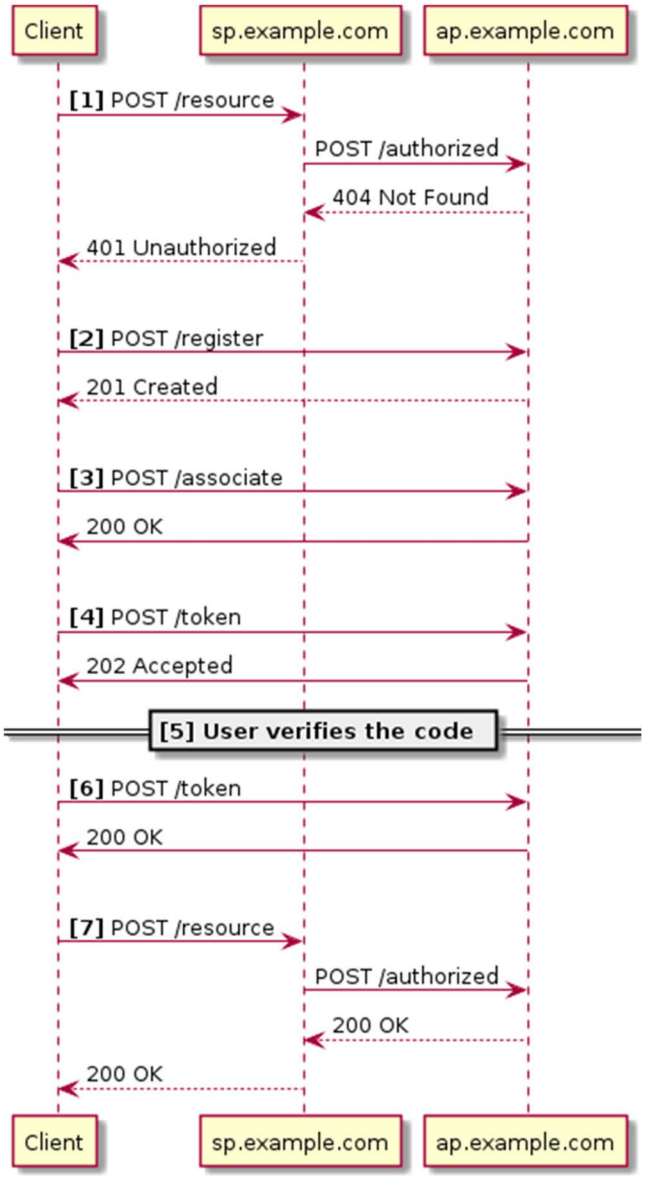


Figure 1: Overview of CPA user mode device flow

7.4 Client mode

In client mode, interactions between the client and the service provider are not associated with a user account. The authorization provider assigns a unique identifier to the client, so the service provider can store information and personalize the interaction related to this device.

The process by which the client obtains an access token to access protected resources at the service provider is shown in figure 2:

1) The client first requests a resource from the service provider, either without supplying a bearer token, or using an invalid access token. The service provider checks the validity of the access token using the authorization provider's `/authorized` endpoint. If the token is missing or invalid, the authorization provider returns `404 Not Found` to the service provider. The service provider then returns `401 Unauthorized` to the client, including in the response an authentication challenge using the `WWW-Authenticate` HTTP header that specifies the authentication provider URI and the modes supported by the service provider, as described in clause A.2.

2) If the client has not already registered with this authorization provider, the client POSTs to the authorization provider's `/register` endpoint. The authorization provider registers the client and returns a `client_id` and `client_secret` with HTTP status `201 Created`. The client stores this information, linked with the authorization provider's domain name in order to be able to use the `client_id` with another service provider using the same authorization provider.

3) The client POSTs to the authorization provider's `/token` endpoint, including in the request its `client_id` and `client_secret`, as well as the domain of the service provider. The authorization provider returns a `200 OK` response containing an `access_token` for the requested service provider.

4) The client can now request the protected resource from the service provider, using the `access_token`. The service provider checks if the `access_token` is valid using a POST request to the authorization provider's `/authorized` endpoint. If it is valid, the authorization provider returns `200 OK` to the service provider including the `client_id`. The service provider returns the protected resource to the client, possibly personalized using the `client_id`.

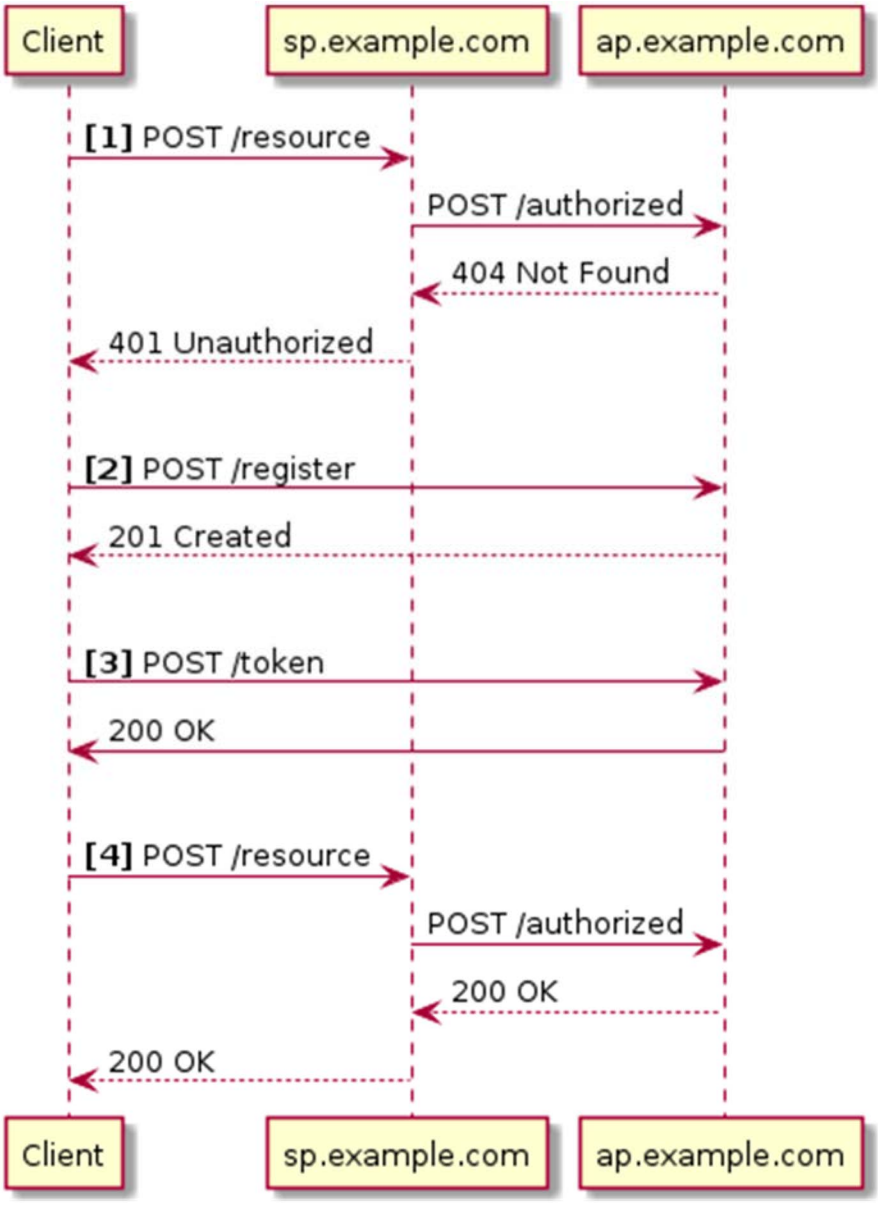


Figure 2: Overview of CPA client mode device flow

7.5 Automatic provisioning of tokens

7.5.1 Overview

Service providers may be grouped within an authorization provider to share an authorization domain. This enables the automatic provisioning of tokens for service providers within the same group once a client has been verified for one service provider within that group.

Once a client has established an association in user mode with a particular service provider, because this association has already been verified, the authorization provider may in future, instead of requesting the user to verify a code, automatically authorize the client to request a token for another service provider's domain within the same group.

The authorization provider should apply internal business rules reflecting the relationship between service providers in deciding whether to automatically issue a token.

With automatic provisioning of tokens for service providers within an authorization provider, a user can sign in only once and interact with multiple service providers, potentially across multiple organizations.

The `client_id` at the authorization provider can be used to determine whether the client has a user account already associated with it. Once a user account has been identified in this way any interaction with other service providers can be automatically associated with this user account or another account belonging to the same user, depending on business requirements:

1) The authorization provider may require the user to sign in and enter a pairing code.

2) The authorization provider may require the user to sign in and give confirmation, without entering a pairing code.

3) The authorization provider may automatically grant an access token, without requiring any user action.

Clauses 7.5.2 to 7.5.3 describe these three cases.

7.5.2 User signs in and enters a pairing code

The authorization provider may require the user to sign in and enter a pairing code before granting access to a new service provider. The process by which the client obtains an access token for this service provider is shown in figure 3. The process in this case is similar to that described in clause 7.3.

1) The client requests a protected resource from a service provider without having a valid token for the service provider's domain. The service provider returns a `401 Unauthorized` response, including a `WWW-Authenticate` header that specifies the authorization provider to use, and indicating that user mode is supported.

2) The client has already registered with this authorization provider, so already has a valid `client_id` and `client_secret`. The client POSTs to the authorization provider's `/associate` endpoint, passing the `client_id`, `client_secret`, and service provider's domain as parameters. The authorization provider returns a `device_code` to the client, with a `user_code` to be displayed by the device to the user, and a `verification_uri`.

3) Using the `device_code`, the client polls the authorization provider's `/token` endpoint to check if the association has been verified by the user. The authorization provider returns a `202 Accepted` response to indicate that the user has not verified the association using the `user_code` yet.

4) The user visits the `verification_uri`, signs in, and enters the `user_code`. The authorization provider can then create the relationship between the `client_id` and the user identity.

5) The client polls the authorization provider's `/token` endpoint and receives a `200 OK` response containing the `access_token` and the `user_name`.

6) The client may now request the protected resource from the service provider, using the `access_token`. The service provider checks if the `access_token` is valid using a POST request to the authorization provider's `/authorized` endpoint. If it is valid, the authorization provider returns `200 OK` to the service provider including the `client_id` and `user_id`. The service provider returns the protected resource to the client, possibly personalized using the `user_id`.

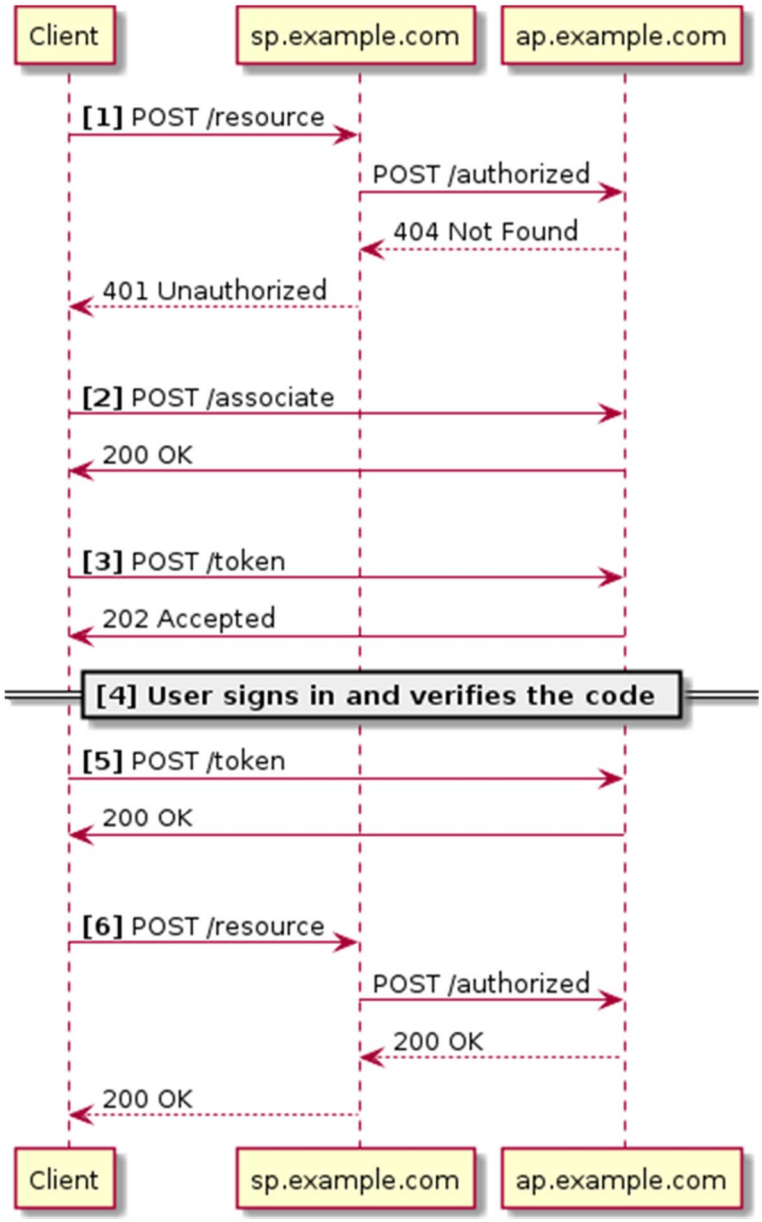


Figure 3: Provision of token for an existing client with a new service provider with pairing code input

7.5.3 User signs in and gives confirmation

The authorization provider may require the user to sign in and give confirmation, without entering a pairing code, before granting access to a new service provider. The process by which the client obtains an access token for this service provider is shown in figure 4.

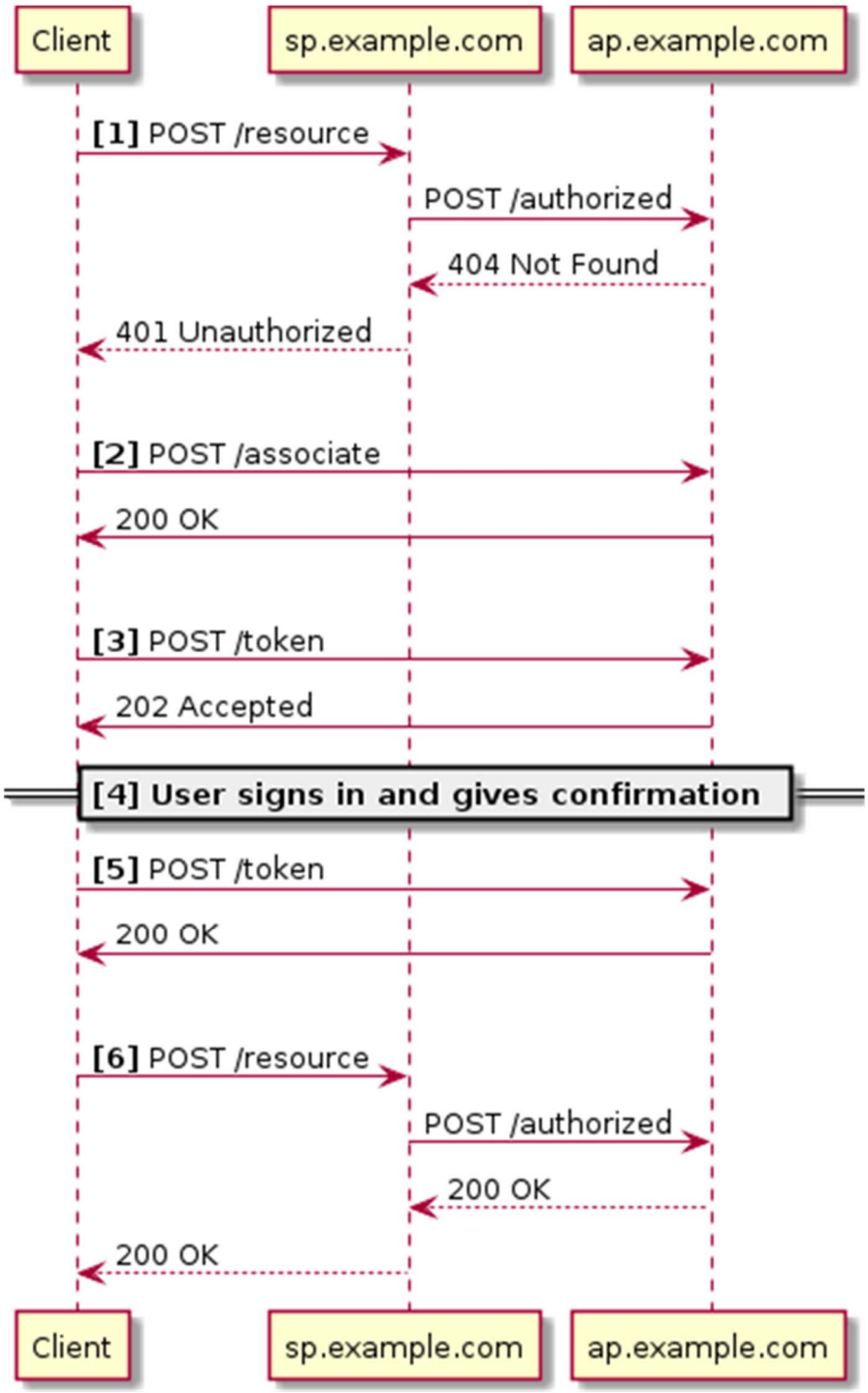


Figure 4: Provision of token for an existing client with a new service provider with user confirmation

1) The client requests a protected resource from a service provider without having a valid token for the service provider's domain. The service provider returns a `401 Unauthorized` response, including a `WWW-Authenticate` header that specifies the authorization provider to use, and indicating that user mode is supported.

2) The client has already registered with this authorization provider, so has a `client_id` and `client_secret`. The client POSTs to the authorization provider's `/associate` endpoint, passing the `client_id`, `client_secret`, and service provider's domain as parameters. The authorization provider returns a `device_code` to the client. Note that no `user_code` is returned.

3) While the user is signing in and before giving confirmation, the client repeatedly POSTs to the authorization provider's `/token` endpoint, passing the `client_id`, `client_secret`, and `device_code`. The authorization provider returns a `202 Accepted` response, which indicates that confirmation has not yet been given by the user.

4) The user visits the `verification_uri,` signs in via the identity provider and confirms the association between their device and the new service provider (how this is done is out of scope of the present document).

5) The client POSTs to the authorization provider's `/token` endpoint, passing the `client_id`, `client_secret`, and `device_code`. The authorization provider returns a `200 OK` response containing an `access_token` for the requested service provider.

6) The client can now request the protected resource from the service provider, using the `access_token`. The service provider checks if the `access_token` is valid using a POST request to the authorization provider's `/authorized` endpoint. If it is valid, the authorization provider returns `200 OK` to the service provider including the `client_id` and `user_id`. The service provider returns the protected resource to the client, possibly personalized using the `user_id`.

7.5.4 Token is automatically granted

The authorization provider may automatically grant an access token for the new service provider, without requiring any user action.

The process by which the client obtains an access token for this service provider is shown in figure 5:

1) The client requests a protected resource from a service provider without having a valid token for the service provider's domain. The service provider returns a `401 Unauthorized` response, including a `WWW-Authenticate` header that specifies the authorization provider to use, and indicating that user mode is supported.

2) The client has already registered with this authorization provider, so has an existing `client_id` and `client_secret`. The client POSTs to the authorization provider's `/associate` endpoint, passing the `client_id`, `client_secret`, and service provider's domain as parameters. The authorization provider returns a `device_code` to the client.

3) The client POSTs to the authorization provider's `/token` endpoint, passing the `client_id`, `client_secret`, and `device_code`. The authorization provider returns a `200 OK` response containing an `access_token` for the requested service provider.

4) The client is now able to request the protected resource from the service provider, using the `access_token`. The service provider checks if the `access_token` is valid using a POST request to the authorization provider's `/authorized` endpoint. If it is valid, the authorization provider returns `200 OK` to the service provider including the `client_id` and `user_id`. The service provider returns the protected resource to the client, possibly personalized using the `user_id`.

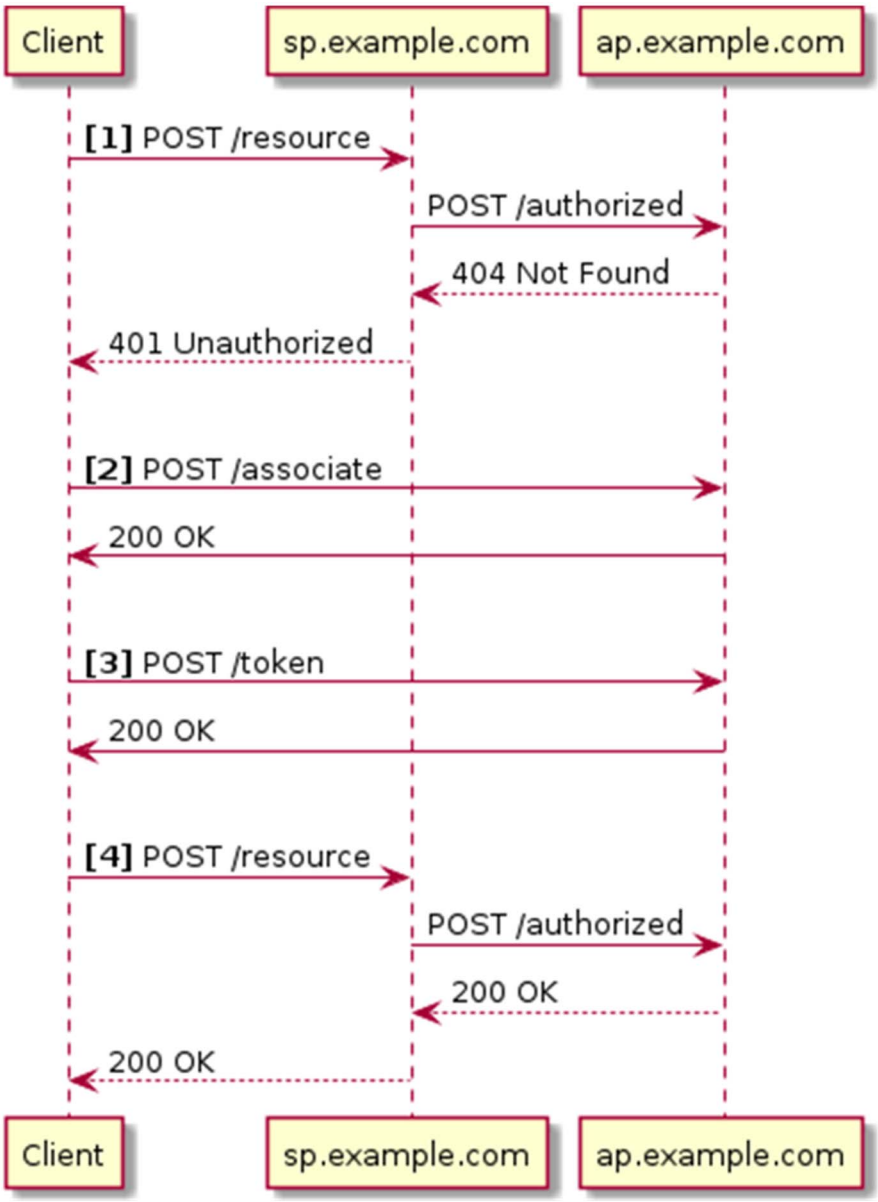


Figure 5: Automatic provision of token for an existing client

7.5.5 Refreshing an expired access token

Authorization providers MAY issue access tokens that expire after a time interval. A client that uses an expired access token to access a protected resource follows the steps shown in figure 6 to obtain a new access token:

1) The client requests a protected resource at the service provider, supplying its existing access token. By querying the authorization provider, as described in clause 9 the service provider determines that the access token has expired. The service provider then returns an HTTP `401 Unauthorized` response to the client, including in the response a `WWW-Authenticate` header, as described in clause A.2.

2) The client makes a request to the authorization provider's `/token` endpoint, and passes its `client_id` and `client_secret`, and the service provider's domain. After verifying that the `client_id` and `client_secret` are valid for that domain, the authorization provider issues the client a new token for that client, which replaces any existing tokens previously issued for that client.

3) The client repeats its request for the protected resource using the new access token, and the request now succeeds.

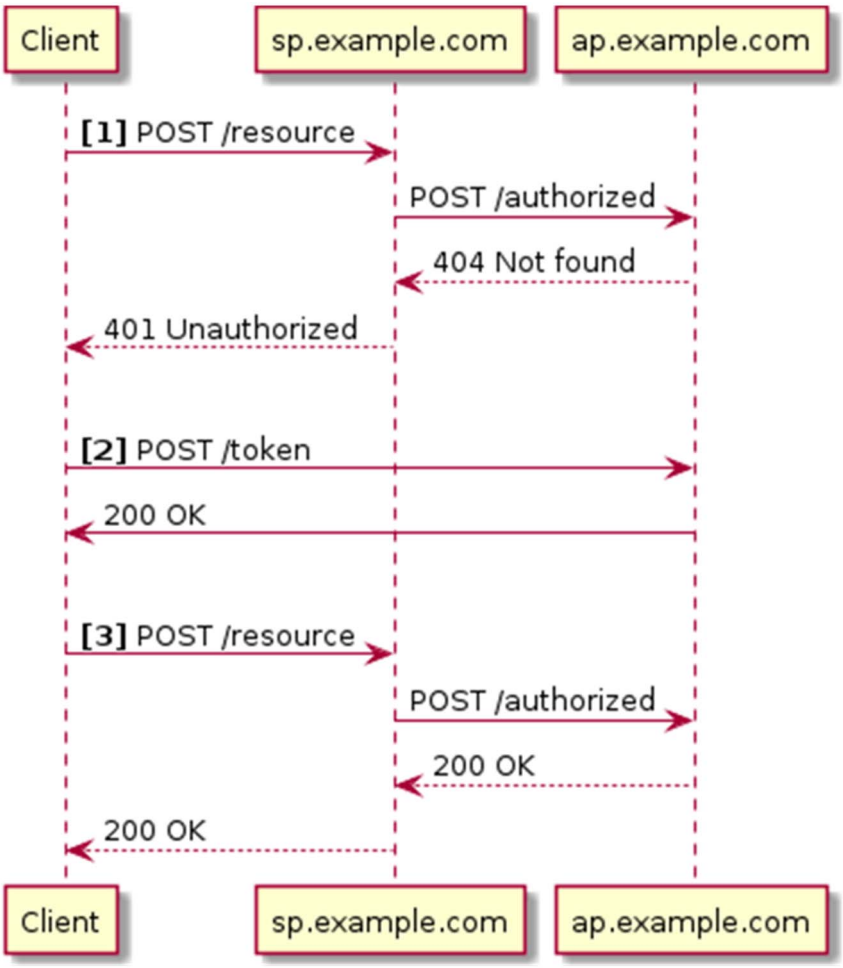


Figure 6: Refreshing an expired access token

7.6 Deleting the association between a client and an authorization provider

7.6.1 Overview

A user can delete the association between a client and an authorization provider either from the client side or the authorization provider side. The specific methods to do so are not part of the CPA specification - there is no defined API endpoint to delete an association. What follows is a brief description of how the deletion of associations can be managed by the client and authorization provider.

7.6.2 Deleting the association on the client side

To delete the association between a client and authorization provider on the client side, the client should delete the `client_id` and `client_secret` for an authorization provider and all tokens associated with that authorization provider.

Without the token, the client will be issued an authentication challenge on the next call to the service provider. If the client retained the `client_id` and `client_secret`, then it could automatically acquire another token for the association (if the authorization provider has implemented automatically provisioned tokens).

Without the `client_id`, the client will appear as if it were an entirely unknown client, and so the service provider will initiate the association flow as if for a new client (which in effect, it now is).

7.6.3 Deleting the association on the authorization provider side

To delete the association on the authorization provider side, the authorization provider should delete all tokens associated with the `client_id`, all mappings between the `client_id` and a `user_id` and the client record containing the `client_id` and `client_secret` themselves, thereby invalidating the `client_id` by removing it.

8. Client/Authorization Provider API

8.1 Overview

This clause defines the API offered by the authorization provider to the client. It consists of the following endpoints:

- `/register` - Register a client
- `/associate` - Associate a client with a user account
- `/token` - Obtain a bearer token
- verification endpoint - Authenticate the user's identity and verify the association

To register with the authorization provider, the client makes a request to the authorization provider's registration endpoint, /register. In response, the authorization provider assigns a unique client identifier (`client_id`) and an associated client secret (`client_secret`). See clause 8.2.

To associate a client with a user account, the client first makes a request to the authorization provider's association endpoint, `/associate`. In response, the authorization provider assigns a user verification code and returns this to the client together with a URI the user should visit to verify the association. See clause 8.3.

There are three possible successful outcomes, as described in clause 7.5:

1) The client is not yet associated with a user identity.

2) The client is already associated with a user identity but the authorization provider requires user confirmation before granting access to a new service provider.

3) The client is already associated with a user identity and the authorization provider can automatically grant access to a new service provider without requiring user confirmation.

In each case, to prevent caching of the device code, the authorization provider shall include the following HTTP headers in the response: `Cache-Control`: `no-store` and `Pragma`: `no-cache`.

See clause 8.3.2.

To obtain an access token, the client makes a request to the authorization provider's token endpoint, `/token`. The specific parameters and their contents differ depending on whether the client is in *client mode* or *user mode*.

In *client mode*, since the authorization provider does not require any further action on the part of the user, the authorization provider can automatically issue a token.

In *user mode*, the client polls the authorization provider repeatedly until the user inputs the correct user verification code and either grants or denies access to the client, or the user code expires.

The client makes the following request at an arbitrary but reasonable interval which shall not exceed the minimum interval specified by the authorization provider in the `/associate` response.

The token endpoint is specified in clause 8.4.

The verification endpoint does not have a fixed URI path. It is a dynamically discovered value returned by the /associate response. See clause 8.5.

8.2 /register - Register a client

8.2.1 /register request

The client makes a HTTP POST request to the authorization provider's registration endpoint and shall include the following parameters in a JSON object in the request body:

client_name

The human-readable name of the client.

software_id

An identifier for the client software application. This value should not change when the software version changes. The authorization provider shall treat this field as self-asserted by the client and shall not make any trusted decisions based on the value of this field alone.

software_version

A version identifier for the software that comprises a client. This value should change on any update to the client software application. The authorization provider shall treat this field as self-asserted by the client and shall not make any trusted decisions based on the value of this field alone.

Example request

```
POST /register HTTP/1.1
Host: ap.example.com
Content-Type: application/json

{
    "client_name": "Test client",
    "software_id": "cpa-test-client",
    "software_version": "1.0.0"
}
```

8.2.2 /register response

If registration is successful, the authorization provider generates a new client identifier for the client. The authorization provider responds with HTTP status 201 and shall include the following information in a JSON object in the response body:

client_id

A string containing a unique identifier issued to the client during the registration process. This client identifier shall be unique at the server and shall not be in use by any other client of that authorization provider.

client_secret

A string containing the client secret, which shall be unique for each `client_id`. This value shall be included by the client in all subsequent requests to the authorization provider. The client shall ensure that the `client_secret` is kept secret between the client and the authorization provider.

Example response

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "client_id": "1234",
    "client_secret": "sdalfqealskdfnk13984r2n23klndvs"
}
```

If an error occurs, the authorization provider shall respond with HTTP status 400 and shall include the following information in a JSON object in the response body:

error

The following error value:

- `invalid_request` - One or more required parameter values is missing or invalid.

Example error response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "error": "invalid_request"
}
```

8.3 /associate - Associate a client with a user account

8.3.1 /associate request

The client makes a HTTP POST request to the authorization provider's association endpoint and shall include the following parameters in a JSON object in the request body:

client_id

The identifier issued to the client when it registered with the authorization provider. See clause 8.2.

client_secret

The secret shared with the client when it registered with the authorization provider. See clause 8.2.

domain

The domain name of the service provider (which may include a port number). The access token returned is valid only for the domain specified here. Clients shall not send an access token to a domain other than the domain for which it was issued.

Example request

```
POST /associate HTTP/1.1
Host: ap.example.com
Content-Type: application/json

{
    "client_id": "1234",
    "client_secret": "sdalfqealskdfnk13984r2n23klndvs",
    "domain": "sp.example.com"
}
```

8.3.2 /associate response

8.3.2.1 Pairing the client with a user account

If registration is successful, the authorization provider generates a new device code and user code for the client. The authorization provider responds with HTTP status 200 and shall include the following information in a JSON object in the response body:

device_code

The temporary device verification code in UUID format as specified in IETF RFC 4122 [3].

user_code

The temporary user verification code. This is an 8 character string. The value shall only use alphanumeric characters from the ISO-646 Invariant Code Set. This shall be unique at the authorization provider for the duration of the pending associate request.

verification_uri

The end-user verification URI on the authorization provider. For devices with limited input capability, the URI should be displayed by the client to the user. The URI should be short, so it can be displayed by the client device and because the user will be asked to manually type it into their user-agent. For the case of a device capable of presenting a web browser to the user, such as smartphone or tablet, see clause 8.5.1.

interval

An integer value indicating the minimum time in seconds the client should wait between making requests to the `/token` endpoint.

expires_in

An integer indicating the maximum time in seconds from the time the response was issued that the `device_code` and `user_code` will remain valid.

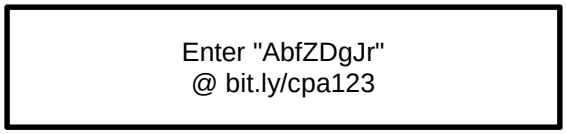
Example response

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

{
    "device_code": "197bf88c-749a-42e2-93f0-e206bac2252f",
    "user_code": "AbfZDgJr",
    "verification_uri": "https://ap.example.com/verify",
    "interval": 5,
    "expires_in": 1800
}
```

The client device should use the information in this response to display a message such as:

or



How the verification process is managed is out of scope of the present document. However, when the user visits the URI specified by `verification_uri`, they should have to sign in as part of the process of entering the `user_code` to verify their user identity).

8.3.2.2 Confirmation required before granting access to a new service

If the client is already associated with a user account, and that account has been granted access to a service provider that is a member of a group of service providers at the authorization provider, then when the same client wants to obtain a token to access another service provider in the same group, the authorization provider may require the user to confirm granting access before issuing a token. This may be used, for example, to ask the user to accept the service provider's terms and conditions of use. In this case the authorization provider responds with HTTP status 200 and shall include the following information in a JSON object in the response body:

device_code

The temporary device verification code in UUID format as specified in IETF RFC 4122 [3]. This shall be unique at the authorization provider for the duration of the pending associate request.

verification_uri

The end-user verification URI on the authorization provider. The URI should be displayed by the client to the user.

The URI should be short and easy to remember as end-users will be asked to manually type it into their user-agent.

interval

An integer value indicating the minimum time in seconds the client should wait between making requests to the `/token` endpoint.

expires_in

An integer indicating the maximum time in seconds from the time the response was issued that the `device_code` and `user_code` will remain valid.

Example response

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

{
    "device_code": "197bf88c-749a-42e2-93f0-e206bac2252f",
    "verification_uri": "https://ap.example.com/verify",
    "interval": 5,
    "expires_in": 1800
}
```

8.3.2.3 Access automatically granted to a new service

If the client is already associated with a user account, then in order to provide single sign-on for the user, the authorization provider may automatically grant access to the new service provider without requiring user confirmation. In this case the authorization provider responds with HTTP status 200 and shall include the following information in a JSON object in the response body:

device_code

The temporary device verification code in UUID format as specified in IETF RFC 4122 [3]. This shall be unique at the authorization provider for the duration of the pending associate request.

expires_in

An integer indicating the maximum time in seconds from the time the response was issued that the `device_code` shall remain valid.

Example response

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

{
    "device_code": "197bf88c-749a-42e2-93f0-e206bac2252f",
    "expires_in": 1800
}
```

If an error occurs, the authorization provider shall respond with HTTP status 400 and shall include the following information in a JSON object in the response body:

error

One of the following error values:

- `invalid_request` - One or more required parameter values is missing or invalid.
- `invalid_client` - The `client_id` and/or `client_secret` is invalid.

Example error response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "error": "invalid_client"
}
```

8.4 /token - Obtain a bearer token

8.4.1 /token request

8.4.1.1 Client mode

The client makes a HTTP POST request to the authorization provider's token endpoint and shall include the following parameters in a JSON object in the request body:

grant_type

The literal string "`http://tech.ebu.ch/cpa/1.0/client_credentials`".

client_id

The client identifier issued to the client when it registered with the authorization provider. See clause 8.2.

client_secret

The client secret issued to the client when it registered with the authorization provider. See clause 8.2.

domain

The domain name of the service provider (which may include a port number).

Example request

```
POST /token HTTP/1.1
Host: ap.example.com
Content-Type: application/json

{
    "grant_type": "http://tech.ebu.ch/cpa/1.0/client_credentials",
    "client_id": "1234",
    "client_secret": "sdalfqealskdfnk13984r2n23klndvs",
    "domain": "sp.example.com"
}
```

8.4.1.2 User mode

The client makes a HTTP POST request to the authorization provider's token endpoint at an arbitrary but reasonable interval which shall not exceed the minimum interval specified by the authorization provider in the `/associate` response. The request shall include the following parameters in a JSON object in the request body:

grant_type

The literal string "`http://tech.ebu.ch/cpa/1.0/device_code`".

device_code

The temporary device verification code returned in the response to `/associate`. The client should discard the `device_code` after a successful response to this call. The authorization provider shall invalidate the `device_code` after it has been exchanged for an access token.

client_id

The client identifier issued to the client when it registered with the authorization provider. See clause 8.2.

client_secret

The client secret issued to the client when it registered with the authorization provider. See clause 8.2.

domain

The domain name of the service provider (which may include a port number).

Example request

```
POST /token HTTP/1.1
Host: ap.example.com
Content-Type: application/json

{
    "grant_type": "http://tech.ebu.ch/cpa/1.0/device_code",
    "device_code": "197bf88c-749a-42e2-93f0-e206bac2252f",
    "client_id": "1234",
    "client_secret": "sdalfqealskdfnk13984r2n23klndvs",
    "domain": "sp.example.com"
}
```

8.4.1.3 Refreshing an expired access token

To replace an expired token with a new access token, the client makes a HTTP POST request to the authorization provider's /token endpoint and shall include the following parameters in a JSON object in the request body:

grant_type

The literal string "`http://tech.ebu.ch/cpa/1.0/client_credentials`".

client_id

The client identifier issued to the client when it registered with the authorization provider. See clause 8.2.

client_secret

The client secret issued to the client when it registered with the authorization provider. See clause 8.2.

domain

The domain name of the service provider (which may include a port number).

Example request

```
POST /token HTTP/1.1
Host: ap.example.com
Content-Type: application/json

{
    "grant_type": "http://tech.ebu.ch/cpa/1.0/client_credentials",
    "client_id": "1234",
    "client_secret": "sdalfqealskdfnk13984r2n23klndvs",
    "domain": "sp.example.com"
}
```

8.4.2 /token response

If successful, the authorization provider shall generate a new access token for the client, and shall invalidate any existing token for that client for the given service provider domain. The authorization provider shall respond with HTTP status 200 and shall include the following information in a JSON object in the response body:

access_token

The access token.

token_type

The literal string "`bearer`".

domain_name

A string containing the name of the service provider suitable for display on the client device.

In user mode the response shall contain:

user_name

This value is intended for display on the client device to help the user identify the originating account.

It should contain a string the user recognizes as representing their account at the service provider. This should be a string the user themselves have entered via their authenticated web account. If a name is not available, the authorization provider shall return an empty string.

If the client is registered in client mode, this field shall be omitted as the client is not associated with a user account.

Additionally, the response should include:

expires_in

If present, an integer value in JSON number format that indicates the lifetime of the access token, in seconds from the time the response was issued. If not present, the access token does not expire.

To prevent caching of the access token, the authorization provider shall include the following HTTP headers in the response: `Cache-Control`: `no-store` and `Pragma`: `no-cache`.

Example response

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

{
    "user_name": "Alice",
    "access_token": "28b8caec68ae4a8c89dffaa37d131295",
    "token_type": "bearer",
    "domain_name": "Channel 1"
}
```

If the user has not yet correctly input the user verification code and/or granted access, and the user verification code has not expired, the authorization provider shall respond with HTTP status 202 and shall include the following information in a JSON object in the response body:

reason

The literal string "`authorization_pending`". This indicates that the authorization request is still pending as the end-user has not yet visited the authorization provider and entered their verification code.

Example response

```
HTTP/1.1 202 Accepted
Content-Type: application/json

{
    "reason": "authorization_pending"
}
```

If an error occurs, the authorization provider shall respond with HTTP status 400 and shall include the following information in a JSON object in the response body:

error

One of the following error values:

- `invalid_request` - One or more required parameter values is missing or invalid.
- `invalid_client` - The `client_id` and/or `client_secret` is invalid.
- `slow_down` - The client has exceeded the polling limit specified by the interval parameter of the `/associate` response. The authorization provider may include a `retry_in` field in the response to indicate the amount of time the client shall wait before retrying the request (see below).
- `expired` - The time allowed for the user to verify the association has expired.

- `cancelled` - The authorization provider wishes to cancel the pairing process. This can occur, For example, the user declined to associate the client with their user account, e.g., by not accepting terms and conditions presented to them by the authorization provider.

The response may include:

retry_in

If the error value is slow_down, this field may be included in the response. If included, the value shall be an integer number of seconds from the time the response was issued that the client should wait before retrying the request.

Example error response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "error": "slow_down",
    "retry_in": 10
}
```

8.5 Verification endpoint

8.5.1 Verification endpoint request

In response to an HTTP GET request to the verification endpoint, the authorization provider should authenticate the user's identity and ask the user to enter the `user_code`. How this verification process is managed is out of scope of the present document.

For a device capable of presenting a web browser to the user (such as a smartphone or tablet), the client can simplify the user interaction by automatically passing the `user_code` to the authorization provider. See step 5, clause 7.3. In this case the client opens the browser at the `verification_uri`, including the following query parameters in the URI:

user_code

The `user_code` provided by the authorization provider. Providing this parameter in a request to the `verification_uri` allows the `user_code` field to be auto-filled.

redirect_uri

Once the user has signed in the resource at the `verification_uri` may redirect the browser to the `redirect_uri`. In the case that the client is a mobile application the `redirect_uri` could be used to return the user back to the application which initiated the association, depending on the functionality of the client, as described in clause 8.5.3.

8.5.2 User permission request

The resource at the `verification_uri` shall request permission from the user prior to associating a device even if the user is already signed in. This prevents a client from associating a user's account without their explicit consent.

8.5.3 Redirection response

The redirection response shall have HTTP status 302 and shall include a `result` query parameter as described below.

result

The outcome of the user authentication and user permission request. This shall be one of the following values:

- `success` - The user has been authenticated, and the association approved.
- `cancelled` - The authorization provider wishes to cancel the pairing process. This can occur, for example, if the user declines to associate the client with their user account, e.g. by not accepting terms and conditions presented to them by the authorization provider.

9 Service Provider/Authorization Provider API

9.1 Overview

This clause defines the API offered by the authorization provider to the service provider. It consists of a single endpoint:

- `/authorized` - Verify an access token and return client and user identities

The access token verification endpoint `/authorized` allows the service provider to verify an access token with the authorization provider. See clause 9.3.

9.2 Establishing trust between the service provider and authorization provider

The service provider and authorization provider should establish a mutually trusting relationship in which the authorization provider assigns a unique access token to the service provider, such that the service provider can make authenticated HTTPS requests to the authorization provider.

The means by which this relationship is established, and how the access token is communicated to the service provider, is outside the scope of the present document.

9.3 /authorized - Access token verification endpoint

9.3.1 /authorized request

The service provider makes a HTTP POST request to the authorization provider's access token verification endpoint and shall include the following parameters in a JSON object in the request body:

access_token

The access token to verify.

domain

The domain name of the service provider (which may include a port number).

To authenticate itself with the authorization provider, the service provider shall include the access token issued to it by the authorization provider in an `Authorization` header in the request, as described in IETF RFC 6750 [i.2], clause 2.1.

Example request

```
POST /authorized HTTP/1.1
Host: ap.example.com
Content-Type: application/json
Authorization: Bearer 3028bad800a94789a4b54202123d500a

{
    "access_token": "28b8caec68ae4a8c89dffaa37d131295",
    "domain": "sp.example.com"
}
```

9.3.2 /authorized response

If the access token belongs to a registered client, the authorization provider shall respond with HTTP status 200 and shall include the following information in a JSON object in the response body:

client_id

A string containing the unique identifier of the client who owns the given access token.

The response may include:

user_id

If the client is associated with a user account, the authorization provider shall return a string containing the unique identifier for the user. If the client is not associated with a user account, the authorization provider shall omit the `user_id` field from the response.

Example response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "client_id": "1234",
    "user_id": "1"
}
```

If one or more required parameter values is missing or invalid, the authorization provider shall respond with HTTP status 400 and shall include the following information in a JSON object in the response body:

error

The literal string "`invalid_request`".

Example error response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "error": "invalid_request"
}
```

If the token is unknown to the authorization provider, is not associated with the service provider domain specified in the domain parameter given in the request, or if the token has expired, the authorization provider shall respond with HTTP status 404 and shall include the following information in a JSON object in the response body:

error

The literal string "`not_found`"

Example error response

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "error": "not_found"
}
```

If the service provider is not authorized to make requests to the authorization provider, the authorization provider shall respond with HTTP status 401 and shall include the following information in a JSON object in the response body:

error

The literal string "`unauthorized`"

Example error response

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json

{
    "error": "unauthorized"
}
```

Annex A (informative): How to integrate applications with the CPA protocol

A.1 Introduction

Annex A describes how service providers can inform clients that the option to authenticate using the CPA protocol is available, and how the bearer token obtained via CPA should be used to access protected resources. This clause is not normative. However, it is strongly recommended these conventions are followed where possible to maximize interoperability.

A.2 Authentication challenge

If a client makes a request for a protected resource without a valid bearer token for a service provider, the service provider may either reject the request (with a 4xx response) or return the unauthorized version of the resource (with a 2xx response). In either case, the response should include a `WWW-Authenticate` header as follows (see IETF RFC 6750 [i.2], clause 3):

```
WWW-Authenticate: CPA version="1.0"
                  name="Example Authorization Provider"
                  uri="https://ap.example.com/cpa"
                  modes="client,user"
```

The header includes the following values:

version

The version of the CPA protocol in use by the server. This may be a comma-separated list of versions, if the server supports multiple protocol versions. See clause 7.1.

name

The display name of the authorization provider. Clients should store their `client_id` and `client_secret` against the domain, not name.

uri

The URI of the authorization provider. The client appends to this URI the path to the various endpoints defined in the present document, such as `register`, `associate`, and `token`.

NOTE: An authorization provider that changes its domain value will be considered by clients as a new authorization provider, requiring a new client ID and client secret.

modes

If the client did not supply a valid bearer token in the request, this value should be a comma-separated list of authorization modes (`client` and/or `user`) supported by the authorization provider.

If the client supplied a valid *client mode* token in the request, and the authorization provider also supports *user mode*, this value should be "`user`", to indicate to the client that the option to elevate to *user mode* is available.

The client then uses the CPA protocol to request a token from the authorization provider.

A.3 Accessing a protected resource

Once the client has acquired a bearer token for a service provider, it should use that token in all future requests to protected resources at the service provider.

The "bearer" token type is utilized by including the access token in an Authorization header in the request, as described in IETF RFC 6750 [i.2], clause 2.1.

```
GET /resource HTTP/1.1
Host: sp.example.com
Authorization: Bearer 28b8caec68ae4a8c89dffaa37d131295
```

Annex B (informative): Bibliography

B.1 Reference implementation

- Authorization provider: https://tech.ebu.ch/code/cpa/authorization-provider.
- Service provider: https://tech.ebu.ch/code/cpa/service-provider.
- JavaScript client: https://tech.ebu.ch/code/cpa/javascript-client.
- iOS client: https://tech.ebu.ch/code/cpa/ios-client.
- Android client: https://tech.ebu.ch/code/cpa/android-client.

B.2 Related documents

IETF draft-ietf-oauth-dyn-reg-14: "OAuth 2.0 Dynamic Client Registration Protocol", 29 July 2013.

IETF draft-recordon-oauth-v2-device-00: "OAuth 2.0 Device Profile", July 2010.

Annex C (informative): Change History

Date	Version	Information about changes
December 2015	1.1.1	initial draft

History

Document history		
V1.1.1	April 2016	Publication