

API Authentication

KS Core API & KS Connect API



API Authentication

Contents

| | |
|--------------------------------------------------|----------|
| Introduction | 1 |
| OpenID Connect concepts | 1 |
| a. Application types | 1 |
| b. Tokens | 3 |
| c. Endpoints | 3 |
| i. Authorize Endpoint | 4 |
| ii. Token Endpoint | 4 |
| Integration types | 5 |
| a. KS Core API | 5 |
| i. Server to server | 5 |
| b. KS Connect API | 6 |
| i. Interactive (Browser and Mobile Applications) | 6 |
| ii. Non-Interactive (Server to server) | 8 |
| iii. Refresh tokens | 9 |

1. Introduction

Authentication is the act of challenging a party for legitimate credentials, providing the basis for creation of a security principle to be used for identity and access control. In simpler terms, it's the process of proving you are who you say you are. To be able to call the KS APIs, you first need to authenticate and for that you need to use our Identity Provider.

SALTO KS uses an Identity Provider that implements [OpenID Connect](#) which is an industry standard and is implemented on top of [OAuth 2.0](#). The purpose of an Identity Provider is to verify the identity of an end-user using different methods, which depend on the type of application trying to verify the user's identity.

Your application needs to obtain a Token before it can make API calls. There are different application types, as well as different tokens that have different uses.



2. OpenID Connect concepts

Before we explain how to obtain an Access Token that can be used to call the KS Connect API, we need to introduce some OpenID Connect concepts that will be mentioned later on.

a. Application types

The [OAuth 2.0 specification](#) (Section 2.1) defines two client types, based on their ability to maintain the confidentiality of their client credentials:

- **Confidential:** Clients capable of maintaining the confidentiality of their credentials or capable of secure client authentication using other means.
- **Public:** Clients that do not maintain the confidentiality of their credentials.

Going further with this clarification, we can recognize the following applications:

- ➔ **Web Applications with a Server component:** A web application is a confidential client running on a web server. The client credentials, as well as any access token issued to the client, are stored on the web server and are not exposed to or accessible by the resource owner.

Example: Web applications that generate HTML on the server.

- ➔ **User-agent based applications:** A user-agent-based application is a public client in which the client code is downloaded from a web server and executes within a web browser. Protocol data and credentials are easily accessible (and often visible) to the resource owner.

Example: SPA or Single Page Applications. Web applications that generate HTML in the client's browser and call a different API to send and receive the data.

- ➔ **Native applications:** A native application is a public client installed and executed on the device used by the resource owner. It is assumed that any client authentication credentials included in the application can be extracted. On the other hand, dynamically issued credentials such as access tokens or refresh tokens can receive an acceptable level of protection. At a minimum, these credentials are protected from hostile servers with which the application may interact.

Example: Mobile applications

This classification is important because different client applications might implement their authentication processes in a slightly different way. They will also have different features offered by the identity provider based on their security level.

Now let's move on to Tokens!

b. Tokens

It can sometimes be confusing to distinguish between different token types. They have different lifetimes and have different purposes. However, not all of them are available in all situations.

➔ **ID tokens** carry identity information encoded in the token itself, which must be a JWT (JSON Web Token).

➔ **Access tokens** are used to gain access to resources by using them as bearer tokens. A bearer token means that the bearer can access authorized resources without further identification. Because of this, it's important that bearer tokens are protected. If someone can get ahold of and "bear" your access token, they can masquerade as you. The lifetime of access tokens is usually one hour. While not mandatory by the standard, in our case this token is also a JWT.

➔ **Refresh tokens** exist solely to get more access tokens. It is not convenient to ask an end-user to enter their credentials every time the access tokens expire. On the other hand, from a security perspective, it should be possible to prevent access to someone without waiting weeks before their token expires. Refresh tokens are valid for multiple weeks and they can be revoked.

c. Endpoints

The protocol used to communicate with the Identity Provider is HTTP communication. It is important to distinguish Token and Authorize endpoints, because depending on the type of the client application, it could be that you will need to use only one of them.



i. Authorize endpoint

The Authorize endpoint can be used to request tokens or authorization codes via your browser. This process typically involves authentication of the end-user and optionally consent.

GET /connect/authorize?

client_id={{client}}&

scope={{space separated scopes}}&

response_type=id_token token&

redirect_uri={{redirect_uri}}&

state={{state}}&

nonce={{nonce}}

i. Token endpoint

The Token endpoint can be used to programmatically request tokens.

GET /connect/token?

client_id={{client_id}}&

client_secret={{secret}}&

grant_type=authorization_code&

redirect_uri={{redirect_uri}}&

code={{code}}

3. Integration types

When integrating with our SALTO KS APIs, you'll receive a client ID and, depending on the application type you are building, a client secret, which is configured for the specific OpenID Connect flow you will use. These flows are sometimes referred to as grant types. They are not specific to the SALTO KS platform, but rather an implementation of the public OpenID Connect protocol, which is also used by other systems on the internet. Libraries and frameworks that implement OpenID Connect protocol can also be used by partners integrating with the SALTO KS platform.

OpenID Foundation has a list of different OpenID Connect implementations, which can be found [here](#).

We recommend using a client library because they handle different flows automatically and will most likely be updated when the OpenID Connect protocol changes for any flow. This can happen if a security vulnerability is discovered and the protocol standards need to be modified.

a. KS Core API

i. Server to server


Clients integrating with our KS Core API are always configured under a server-to-server integration. For that, you will need a client ID and a client secret to retrieve the access token, which will then be used to call the API. This is known as a **client_credentials** flow and it consists of only one call to our Identity Provider:

POST /connect/token

Content-Type: application/x-www-form-urlencoded

Authorization: Basic Y2xpZW50X2lkOmNsaWVudF9pZF9zZWNYdGV0

grant_type=client_credentials&scope=some_api

This value represents the Base64 encoding of the client ID and client secret, joined by a single colon ':'. 

The KS Core API does not have the concept of an individual user, thus it is not needed for the `client_credentials` flow. Because only one call is needed and there is no back-and-forth communication between the server and the Identity Provider, return URLs are not necessary. Additionally, because this is a server-to-server communication, the assumption is that calling server is secured enough and that it is not operated by a human.

b. KS Connect API

Clients integrating with our KS Connect API can be configured under two types of integration: Interactive and Non-Interactive.

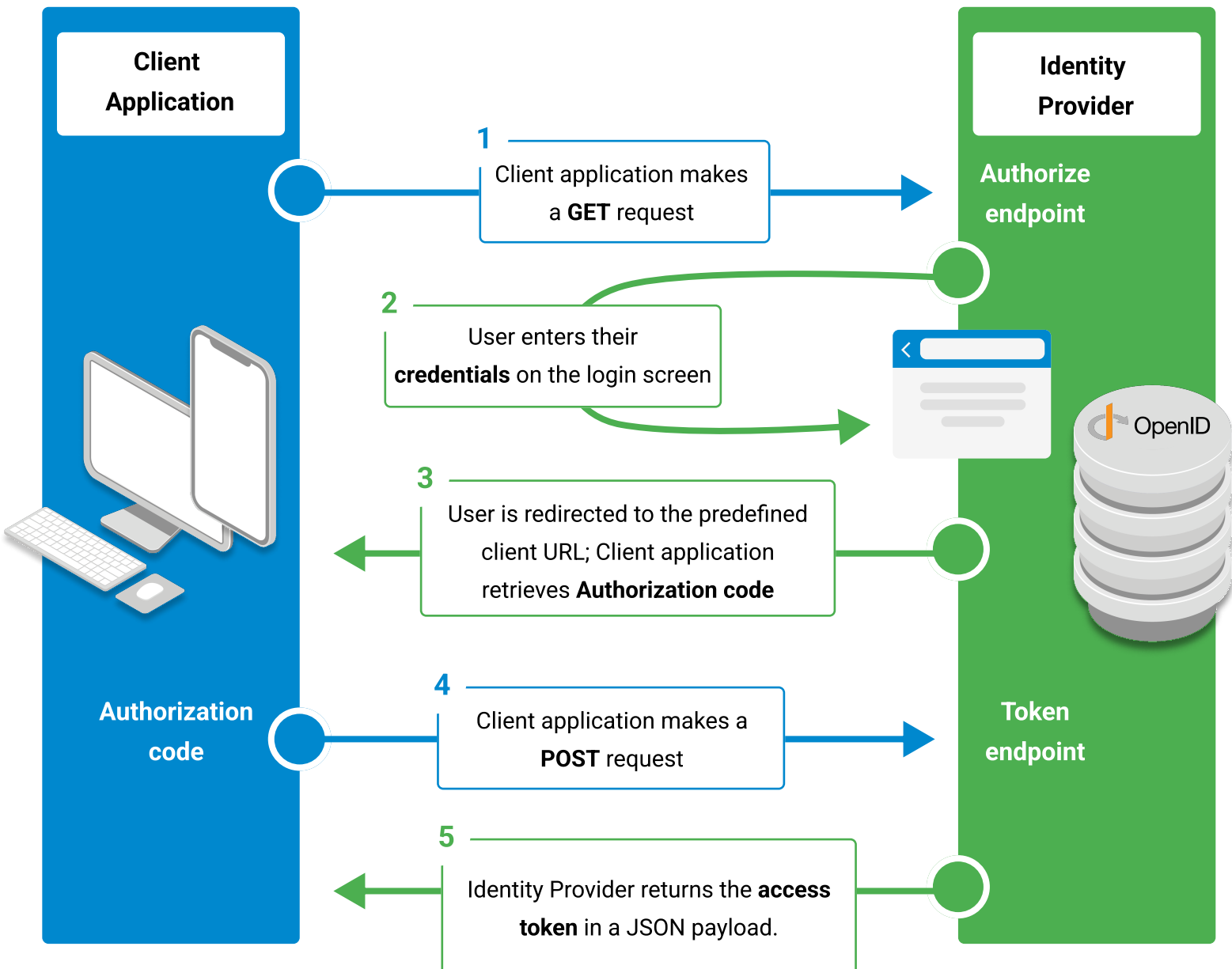
i. Interactive (Browser and Mobile Applications)

The Interactive integration applies to clients that are building Browser-based Applications, such as Single Page Applications (SPAs), as well as Mobile Applications. These applications have a User Interface (UI) component, through which clients interact with. In this case, clients are making use of an `authorization_code` flow. The authenticity of the client making the request is ensured with the client ID and the predetermined URLs that are registered in our system. Those return URLs are attached to your specific client ID and this is where the results of the requests are delivered. Only the owner of the systems on these URLs can receive the tokens.

The `authorization_code` flow consists of two components: a front-channel step (e.g. a login page) and a back-channel step. The operation performed in the front-channel generates an authorization code, which is then exchanged in the back-channel with the requested token.

Interactive clients can be either Public or Confidential. For both cases, the `authorization_code` flow is used together with a Proof Key for Code Exchange (PKCE), which is a security requirement to keep the flow secure. The difference is that for Confidential clients, a **client secret** is also required when providing the credentials, and the client secret must be known in order for the authorization code to be turned into an access token.

The `authorization_code` flow happens between the client's application and our Identity Provider, and it can be broken down into the following steps:



With the access token, the Client application can make a call to our KS Connect API. Furthermore, if the GET request in step 1 contains the scope `offline_access`, then the Identity Provider will return not only an access token, but also a refresh token in step 5.

```
https://identity-server-url/connect/authorize?response_type=code&scope=
user_api.full_access+offline_access&client_id={{client_id}}&
redirect_uri={{redirect_uri}}
```

To read more about Refresh tokens, please refer to section ‘iii. Refresh tokens’.

ii. Non-Interactive (Server to server)

The Non-interactive integration applies to clients that are not building a User Interface. In other words, it works as a Server-to-Server interaction. The KS Connect API always works with the concept of a user. For this reason, server-to-server interactions need a predefined user. For instance, a system user that needs to be previously added to all the sites the client wants to manage. This system user needs to be added under a Site owner role.

Non-interactive clients make use of a **password** flow. Once the environment is set up, clients must create a POST request to our Identity Provider to request the token, as well as provide a client ID and a client secret. Then, in the body of the request, the client must fill in the following values:

POST {{identity_server_url}}/connect/token

Content-Type: application/x-www-form-urlencoded

Authorization: Basic Y2xpZW50X2lkOmNsaWVudF9pZF9zZWNYdGV0

grant_type=password&

username=(insert KS email here)&

password=(insert KS password here)&

scope=user_api.full_access

This value represents the Base64 encoding of the client ID and client secret, joined by a single colon ‘:’.

The response body will contain the access token with which clients can make API requests.

iii. Refresh tokens

Refresh tokens have a much longer lifetime, so they can be used to obtain another access token with a POST call to the token endpoint. This flow works for both Interactive and Non-Interactive clients.

POST connect/token

Content-Type: application/x-www-form-urlencoded

Authorization: Bearer {{access_token}}

client_id=client&

client_secret=secret&

grant_type=refresh_token&

refresh_token=hdh922

...

When integrating with SALTO KS, every client application needs to go through an authentication process before making a call to our APIs. This is done so that our Identity Provider, which works as our security gateway, is able to verify the identity of the application that is trying to connect with us.

We hope this document has informed you of the different authentication flows that should be implemented depending on the type of application you're using. If you are seeking more information, don't hesitate to reach out to us through the SALTO Systems [Contact page](#).