Sign in with Astropay Integration Guide

Version 1.0

Version History

Version	**Date**	**Description**
1.0	18/03/2021	Initial version with complete Authentication code flow.

Introduction

Sign in with Astropay (SWA) is a service by which Merchants can authenticate users on their own services using their previously created Astropay accounts, thus eliminating the need of signing them up first, boosting conversion and improving overall security and compliance.

To that end, Astropay makes available an Identity Provider that is Fully compliant with OpenID Connect 1.0 (OIDC), which is the current standard in federated authentication, is based on solid and proven standards such as oAuth 2.0 and JWT tokens and is used by the most popular identity services such as Google and Facebook. Therefore, integrating with Astropay for Sign in is just like integrating with any OIDC compliant provider, and all libraries and plugins built to that end should work without any friction.

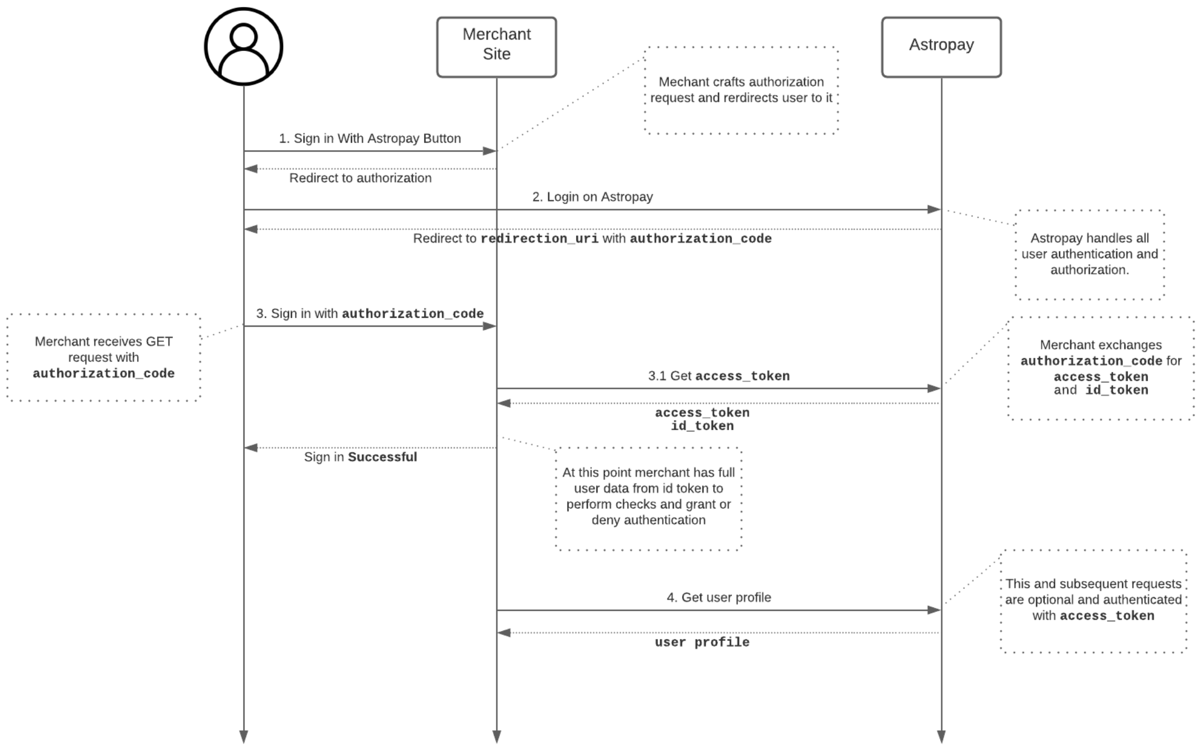
We strongly recommend using previously built and tested client libraries for integration with an OIDC provider, since this reduces greatly the integration effort. The OpenID Foundation provides a list of certified and uncertified libraries you can find in https://openid.net/developers/libraries/, but any working OIDC client library should work.

For more information on OIDC protocol, please visit https://openid.net/connect/

For integrating with SWA, merchants need to be configured on Astropay platform, and special credentials need to be used to ensure security. To start the integration process, merchants have to get in touch with their account manager or integration@astropay.com and ask for being registered for SWA integration, upon which they will receive client id and client secret, needed to integrate as will be seen below.

Authorization Flow description

OpenID Connect supports different mechanisms for authentication, called **flows**. Sign in with Astropay (SWA) supports the **Authentication (or Basic) Flow**, since it's the simple flow for secure sign in, and therefore is the flow described in the below diagram.


Merchant Site
Astropay
Mechant crafts authorization request and rerdirects user to it
1. Sign in With Astropay Button
Redirect to authorization
2. Login on Astropay
Redirect to redirection_uri with authorization_code
Astropay handles all user authentication and authorization.
3. Sign in with authorization_code
Merchant receives GET request with authorization_code
Merchant exchanges authorization_code for access_token and id_token
3.1 Get access_token
access_token id_token
Sign in Successful
At this point merchant has full user data from id token to perform checks and grant or deny authentication
This and subsequent requests are optional and authenticated with access_token
4. Get user profile
user profile

The Merchant (Relying party in OIDC terminology) must implement the following:

1. Sign in with Astropay (SWA) button

Side by side with regular login, merchant must present user with the button for triggering SWA authentication. Upon clicking, merchant must craft an *authorization request* and redirect the user to that endpoint, where his login and authorization will be handled by astropay. Since authorization request is done over GET method, redirection may be done with a standard HTTP redirection (see https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections), via HTML or even JS.

2. Authorization Request

Note: baseUrl depends on the environment being used for authentication. Below you may find a table with proper values with each environment.

Authorization endpoint on Astropay has the following signature and parameters, and it is consumed via GET method since it's the user on the browser who triggers it.

```
GET ${baseUrl}/cas/oidc/authorize
```

Parameter	Description	Mandatory
`client_id`	Identifies the client and must match the value preregistered in Astropay. It can be requested to integration support.	YES
`redirect_uri`	Callback location where the authorization code should be sent after user logs in on Astropay, therefore, it's the endpoint that is going to be designed to actually authenticate the user, as it will be explained below. It must match the uri previously registered on Astropay. Bear in mind that the URI must be URL encoded since it's treated as a parameter.	YES
`scope`	Scope of user attributes and actions for which the merchant is requesting access. Current version only supports the value `"openid+user.profile.read"`. Full scope listing can be found in below section.	YES
`response_type`	Flow type. Only `code` is supported in current version.	YES
`nonce`	Special parameter to prevent replay attacks. This same value will be present in the id token.	YES
`state`	A value to be returned in the token. The client application can use it to remember the state of its interaction with the end user at the time of the authentication call.	YES

The following example is a redirect URL crafted for the following values:

- CLIENT_ID as placeholder for the actual client id value.
- NONCE and STATE as placeholders for generated values that should be used.
- `redirect_uri` to handle login later: https://merchant.com/signIn/astropay
- `scope` and `response_type` with default values `openid+user.profile.read` and `code` respectively

```
https://idp-stg.astropay.com/cas/oidc/authorize?response_type=code&cl
ient_id=CLIENT_ID&scope=openid+user.profile.read&redirect_uri=https%3
A%2F%2Fmerchant.com%2FsignIn%2Fastropay&nonce=NONCE&state=STATE
```

When user goes to that URL, Astropay sign in will answer and get the user through the sign in process with his account configuration. Also, consent from the user to share his basic information with the merchant will be collected.

3. Sign in with Astropay on Merchant

3.1 Sign in handler on merchant service

After successful sign in in Astropay, user will be redirected to the address previously specified in `request_uri` with the following parameters, where a proper sign in handler must be implemented by the merchant.

Parameter	Description	Mandatory
`code`	The authorization code needed to complete authentication.	YES
`nonce`	Same value provided on authorization request	YES
`state`	Same value provided on authorization request	YES

Following the previous example, after successful authentication merchant will receive a GET request as follows:

```
GET
https://merchant.com/signIn/astropay?code=CODE&nonce=NONCE&state=STAT
E
```

This means the authentication was successful and is identified with the CODE. Also, the STATE value is the same that was previously used, and therefore it can be used to identify that this request comes from the same session initiated seconds ago, to tie the auth to an existing previous session if any, and to prevent replay attacks should anyone intercepts the CODE.

3.1. Get Access Token and complete Authentication.

On the sign in endpoint that receives the code, Merchant must take the value received (CODE) and use it to resolve the access token on the proper endpoint on Astropay:

```
POST ${baseURL}/cas/oidc/accessToken
```

Parameter	Description	Mandatory
`grant_type`	Grant type for the flow. Only *authorization_code* is supported in current version.	YES
`code`	The access code received that identifies the authentication request.	YES
`redirect_uri`	Same value provided on authorization request	YES

`client_id`	Client id for merchant identification	YES
`client_secret`	Client secret for merchant authentication	YES

Note that this is a direct B2B request between the merchant backend and Astropay's backend, and that it carries the additional client_secret, which must be present in order to authenticate the request is in behalf of the merchant and not a forged one made by someone who got an authorization code somewhere. The response carries the access token and id token and allows to complete the authentication.

Successful response is HTTP code 200, wrong client id or secret yields a 401 unauthorized and any other error yields a 400 Invalid request. Also, if successful, contents are encoded in JSON format.

Property	**Description**	**Type**
`access_token`	Access token that identifies the session on SWA. It can be used to authenticate subsequent requests to other endpoints	String
`token_type`	Type of the token to be used later. Always contains `bearer`.	`bearer`
`expires_in`	Token duration in seconds	int
`scope`	Allowed scope for the access token	String
`id_token`	JWT token with user information	JWT

Response contains `access_token` that can be used for subsequent request authentication and therefore must be tied to the local user session on the merchant, since its scope is for this single user and its validity is expressed by the `expires_in` value.

`id_token` contains a JWT token with user data. It can be parsed with any JWT library, and it contains basic user data such as phone number, country, identity level and other relevant information to know the customer.

Example request:
This example is simplified, any assigned cookies should be included to avoid being routed wrongly by load balances or blocked by security components.

```
POST /cas/oidc/accessToken HTTP/1.1
Host: idp-stg.astropay.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=CODE&redirect_uri=https://merchant
.com/signIn/astropay&client_id=CLIENT_ID&client_secret=CLIENT_SECRET
```

Example response:

```
{
   "access_token": "AT-5--KjKvRPuTyQCuejqPMbnNOayW1DnxWtl",
   "id_token":
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiJUR1QtOS1aVjNtcC05aUxiSjBlck14elY0T3BP
cWFWS0JRbHFlVHVxUkZxWVRoZTJnM3pncVk4QXFicGJVY0R0S2xLNjlVZjR3LXMtaWRwLTAxIiwic2lkIjoiN2
ViMzc0MTk4MmQ0MWNjNjJjODBmODUxOTU3YjAyODExMjBhMWU0OSIsImlzcyI6Imh0dHBzOi8vaWRwLXN0Zy5h
c3Ryb3BheS5jb20vY2FzL29pZGMiLCJhdWQiOiJjbGllbnQiLCJleHAiOjE2MTYxMjIwMDUsImlhdCI6MTYxNj
A5MzIwNSwibmJmIjoxNjE2MDkyOTA1LCJzdWIiOiI1OTggOTgyODcxMjYiLCJjbGllbnRfaWQiOiJjbGllbnQi
LCJhdXRoX3RpbWUiOjE2MTYwOTMyMDUsInN0YXRlIjoiZjk1OWQwZTBmMDMyIiwibm9uY2UiOiIxM2U0MDEyMD
lhN2FmIiwiYXRfaGFzaCI6IkRSOHIxaGRtMmZUVjdlcTNibk5NdmciLCJjYXRlZ29yeSI6dHJ1ZSwiY291bnRy
eSI6IlVZIiwiaWRlbnRpdHlfc3RhdHVzIjoiTk9ORSIsIm1vYmlsZV9udW1iZXIiOiI1OTggOTgyODcxMjYiLC
JwcmVmZXJyZWRfdXNlcm5hbWUiOiI1OTggOTgyODcxMjYifQ.f8aW7sQTywMiauwkAYmshxxFlCK7ihlzJdZgw
-_-Om5EvflZiFJHHBlmAEPBoea8W87E582D9mRPzxfkP7bE_XFLDedMbbl1t7gGNBoEX2_DFef8s58FseX9jT4
RlcyV7awP-FviNja2MFlT2zRaeJeKWuGI6vOdq7lZdts8MHRnhOwmZw3WXN4u8iAo3HSZgjgRVWrn0oWZF9YQC
_2_SCZg3cDbF1gd5o_9FQKusCSfnD3d7N7txPsFuw-X_pvTRNcYywrAv-7BgzfJHPYtTQWHdGjasmVArhbT9hQ
70pW2B_nqYa9XaOAbzEjfb0OB_Hi1cln3xPZcTh8IeWLdag",
   "token_type": "bearer",
   "expires_in": 28800,
   "scope": "openid user.profile.read"
}
```

At this point, merchant is guaranteed that the user identified by that particular information has undergone successful authentication on Astropay, and with the provided information can create a local session, therefore completing the SWA flow successfully.

4. Get user profile

If merchant doesn't want or is unable to interpret contents of the id token, or want to get user information later on, the user profile endpoint can be consumed easily to get user information.

```
GET ${baseUrl}/cas/oidc/profile
Authorization: Bearer <access_token>
```

Access token is used as authentication, and it must be attached as Bearer token on the Authorization header of the HTTP GET Request.

Response is profile in JSON format and contains the following relevant fields:

Property	Description	Type

`sub`	User id	String
`service`	Merchant service identification	URL
`auth_time`	The time the end user was authenticated, represented in Unix time (seconds).	long
`id`	User id	String
`client_id`	Merchant `client_id`	String
`attributes`	JSON node containing user attributes allowed by the current scope	JSON

User name used for logging in in Astropay, which is Mobile phone number, is present on id or sub properties, and the rest of the user information that Astropay knows about the user and the Merchant is allowed to access given the current scope is present under the attributes JSON node. Implementations should be flexible on accepting more fields here without breaking, since more attributes can be added by Astropay without notice.

Example request:
This example is simplified, any assigned cookies should be included to avoid being routed wrongly by load balances or blocked by security components.

```
GET /cas/oidc/profile HTTP/1.1
Host: idp-stg.astropay.com
Authorization: Bearer ACCESS_TOKEN
```

Example response:
```
{
   "sub": "598 98287126",
   "service": "https://localhost:9443/simple-web-app/openid_connect_login",
   "auth_time": 1616093205,
   "attributes": {
       "category": "1",
       "country": "UY",
       "identity_status": "NONE",
       "mobile_number": "598 98287126"
   },
   "id": "598 98287126",
   "client_id": "client"
}
```

Environments

Sandbox:
Base url - https://idp-stg.astropay.com
Authorization endpoint - https://idp-stg.astropay.com/cas/oidc/authorize
Access token endpoint - https://idp-stg.astropay.com/cas/oidc/accessToken
User profile endpoint - https://idp-stg.astropay.com/cas/oidc/profile

Production:
Under Construction

Authorization Scopes

Every scope enables merchant to access certain user attributes and/or methods and musst be used as described in the present document.

Scope	**Data**	**Methods**
`user.profile.read`	first_name last_name country document number mobile_number email address identity_status (kyc_status) category (BRONZE, SILVER,etc.)	
Future scopes		
`user.mobile.read`	mobile_number	
`user.email.read`	email	
`user.document.read`	document_number	
`user.purchase.history.read`	Purchase history for user	

References and Useful Links

https://openid.net/connect/
https://openid.net/developers/libraries/
https://oauth.net/2/
https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections
https://jwt.io/