eFaas Single-Sign On Integration

(Developer Portal)

Documentation Version 2.2

Last updated: 02nd November 2023

Contents

Introduction

eFaas is an openid connect single sign-on which provides a secure authentication process and a consent based mechanism to share the user data.

This document is intended to guide you through the process of eFaas integration.

Terminology

Relying Party (Client)	Application trying to authenticate the user
User	Anyone who is using eFaas to authenticate themselves to other applications
JWT(JSON Web Token)	A self-contained and digitally signed JSON string that contains information about the authentication event and user.
access_token	JWT that is used to grant access to protected resources
id_token	JWT that contains information about authentication event and can optionally contain user information
Code	Authorization code that can be exchanged for an id_token and access_token and/or refresh token
Discovery Document	The end point that returns urls and information associated with eFaas
Server side client	Relying parties that can keep a client secret confidentially and will maintain a session of their own after authorization (eg: MVC applications that use cookies to maintain session)
Non Server side client	Relying parties that cannot store a client secret confidentially and will depend entirely on eFaas for maintaining session (eg: Single page Application)

Getting Started

Important URLs

Developer Base URL	https://developer.gov.mv/efaas
Production Base URL	https://efaas.gov.mv
Discovery Document Url	{efaas_base_url}/.well-known/openid-configuration

Client Creation

When submitting client creation form, make sure you have provided the following information:

- Correct grant type
- Redirect URIs
- Post logout URI
- Backchannel or front channel logout URI (For server side applications only)
- Scopes required by your application
- If you require to refresh tokens

After submitting the client creation form, you will receive client credentials for both development and production environment. A client credential consists of:

Client Id	A unique identifier issued to the consumer to identify itself to eFaas
Client Secret	A shared secret established between the eFaas and consumer

eFaas Scopes

eFaas scopes are used to indicate what user information (user claims) will be made available from eFaas to the Relying Party (Client).

Relying Parties should only request the minimum number of scopes that would fulfil their application's requirements.

The following is a list of available scopes in efaas

NOTE: **openid** and **efaas.profile** will be assigned to all clients by default. Also the legacy **profile** scope will continue to work until we migrate all the current efaas clients to use the new scopes.

- openid
- efaas.profile
- efaas.email
- efaas.mobile
- efaas.birthdate
- efaas.photo
- efaas.work_permit_status
- efaas.passport_number
- efaas.country
- efaas.permanent_address

Sample User Claims

The following tables show details of claims associated with each eFaas scope.

Scope: efaas.openid

Claim name	Description	Type	Example
sub	Unique user key assigned to the user	string	178dedf2-581b-4b48-9d73-770f302751dc

Scope: efaas.profile

Claim name	Description	Type	Example
first_name	First name of the user	string	Mariyam
middle_name	Middle name of the user	string	Ahmed
last_name	Last name of the user	string	Rasheed
first_name_dhivehi	First name of the user in Dhivehi (Maldivians only)	string	ދަރިދަރު
middle_name_dhivehi	Middle name of the user in Dhivehi (Maldivians only)	string	އަޙްމަދު
last_name_dhivehi	Last name of the user in dhivehi (Maldivians only)	string	ރަޝީދު
gender	Gender of the user	string	M / F
idnumber	Identification number of the user • National ID number for Maldivians • Work permit number for work permit holders • Passport number for other foreigners	string	A000111 / WP941123 / LA110011
verified	Indicates if the user is verified	boolean	True / False
verification_type	Type of verification taken by the user	string	biometric / in-person
last_verified_date	The last date when the user was verified either using biometrics or by visiting an eFaas verification counter	date (M/dd/yyyy h:mm:ss tt)	6/26/2019 9:18:11 AM
user_type_description	Indicates the type of user	string	• Maldivian • Work Permit Holder • Foreigner
updated_at	The last date when the user information was updated	date (M/dd/yyyy h:mm:ss tt)	6/15/2023 2:12:38 PM

Scope: efaas.email

Claim name	Description	Type	Example
email	Email of the user	string	ahmed_ali@gmail.com

Scope: efaas.mobile

Claim name	Description	Type	Example
mobile	Mobile number of the user	string	9074512
country_dialing_code	Dialing code of the registered number	string	+960

Scope: efaas.birthdate

Claim name	Description	Type	Example
birthdate	Date of birth of the user	date M/dd/yyyy	12/20/1990

Scope: efaas.photo

Claim name	Description	Type	Example
photo	Photo of the user	url	https://efaas-api egov.mv/user/photo

Scope: efaas.work_permit_status

Claim name	Description	Type	Example
is_workpermit_active	Boolean indicating if the work permit is active (only applicable to work permit holders)	boolean	boolean

Scope: efaas.passport_number

Claim name	Description	Type	Example
passport	Passport number of the user	string	12/20/1990

Scope: efaas.country

Claim name	Description	Type	Example
country_name	Name of the country of the user	string	Maldives
country_code	ISO 3-digit code	int	462
country_code_alpha3	ISO alpha3 code	string	MDV
country_dialing_code	Dialing code of the country	string	+960

Scope: efaas.permanent_address

Claim name	Description	Type	Example
permanent_address	Permanent address of the user	String (JSON)	Given below

Note: JSON string escaping has been removed for readability


```
{
  "AddressLine1": "Blue Light",
  "AddressLine2": "",
  "Road": "Road Name",
  "AtollAbbreviation": "K",
  "AtollAbbreviationDhivehi": "ކ",
  "IslandName": "Male'",
  "IslandNameDhivehi": "މާލެ",
  "HomeNameDhivehi": "ބްލޫ ލައިޓް",
  "Ward": "Maafannu",
  "WardAbbreviationEnglish": "M",
  "WardAbbreviationDhivehi": "މ",
  "Country": "Maldives",
  "CountryISOThreeDigitCode": "462",
  "CountryISOThreeLetterCode": "MDV"
}
```

eFaas Integration Overview

The following functionalities should be implemented for a successful eFaas integration:

1. Authentication request
2. Token request
3. Validating tokens
4. Retrieving user info from userinfo endpoint
5. Refreshing tokens if required
6. Logging the user out
7. Single sign-out
8. eFaas One-Tap Login

Authorization Flows

Introduction

An authorization flow or a grant type describes the process by which a client obtains tokens from the authorization server(eFaas).

Currently we support two types of authorization flows in eFaas. They are:

- Hybrid (For server-side applications)
- Authorization Code + PKCE (For non-server side applications. eg: SPAs and mobile applications)

Hybrid Flow

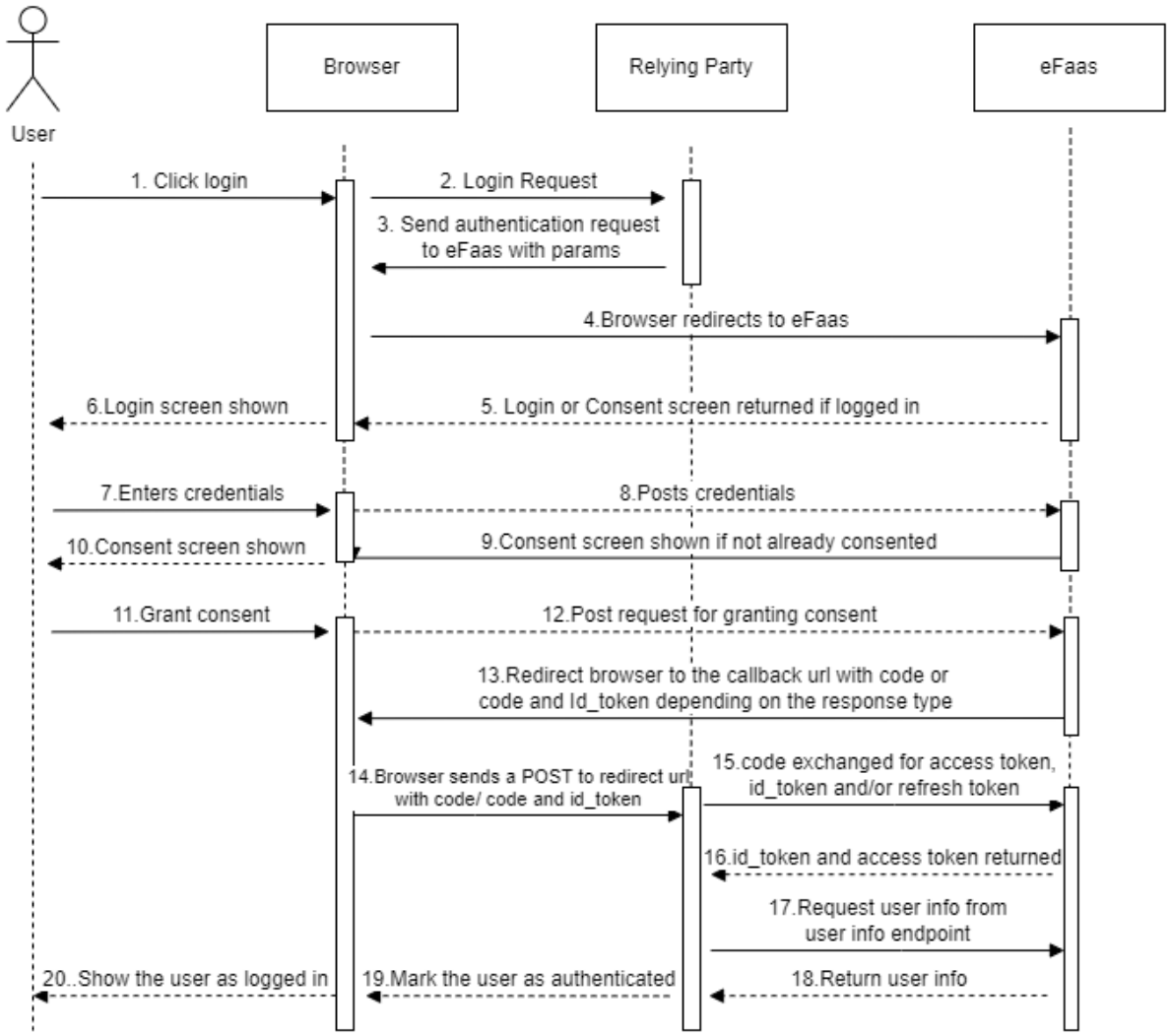
Hybrid flow is used by server-side applications that can keep a secret confidentially and maintain their own session.

Authorization Code Flow + PKCE

This flow is mainly used by browser based applications like SPAs and mobile applications that cannot keep a secret confidentially. However, Authorization Code + PKCE can be used by server side applications as well.

Previously **implicit flow** was used for non-server side applications, however OAuth 2.0 Best Practices now **recommend against** using this flow due to many risks such as returning access token in the URL.

Overall Authorization Flow



The authorization steps for both hybrid and authorization code + PKCE are same with only differences in parameters for authentication request and token requests. We will look at these requests in more detail in the next section.

IMPORTANT

In the following sections we have described how to manually create authentication requests and exchange authorization codes for tokens. However, **we strongly recommend** using openid connect or OAuth libraries developed for the respective frameworks. These libraries provide extension methods for logging in and logging out the users. They also handle PKCE challenges, token exchanges, generation and validation of state and nonce parameters and load user info as well.

Sample Requests For Hybrid Flow

Authentication Request

URL: {efaas_base_url}/connect/authorize

Method: GET

Query Parameters	Description
client_id	The client id provided during client registration
redirect_uri	The URI that is registered at eFaas as a callback uri
response_type	Space delimited values to indicate what to receive from eFaas as a reponse. For hybrid this should be one of • code id_token • code token • code id_token token
scope	The scopes that the relying party requires from eFaas. The values should be space delimited. **eg**: openid efaas.profile
nonce	A value that uniquely identifies the authorization request. It will be returned in the id_token.
state	This is a randomly generated string to prevent CSRF attacks and maintain state between eFaas and RP. The value of state parameter will be returned by eFaas during callback.

Sample Authentication Request

```
https://developer.gov.mv/efaas/connect/authorize?client_id=def7fc52-0761-4916-82e5-9b759d2f3589&redirect_uri=https://myapp.gov.mv/signin-oidc&response_type=code id_token&scope=openid profile&response_mode=form_post&nonce=nonce_123&state=state_abc
```

Note: URL encoding is removed for readability

A note about state parameter

Please do not send static values in the state parameter.

For example ……&state=adminLogin

State parameter must always be a random, unique and non-guessable. It should also be validated on eFaas callback to prevent CSRF attacks.

If your application requires additional context data to be maintained between authentication request and eFaas callback, you can store the information in session or the local storage of the browser with state parameter as the key. You may refer to this link for the implementation details https://auth0.com/docs/secure/attack-protection/state-parameters

Token Request

When you receive the callback after a successful authorization request, you can exchange the code for an access token and in some cases a refresh token using the token endpoint

Method: POST

URL: {efaas_base_url}/connect/token

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

Params	Description
client_id	The client id provided during client registration
client_secret	The client secret provided during client registration
grant_type	The grant type in this case is authorization_code
code	The authorization code received from eFaas
redirect_uri	The URI that is registered at eFaas as a callback uri

Sample Token Request

```
client_id=abc44ec3-aa7b-4eab-a50e-4d18f17c3f62&client_secret=9fz11cd8-7bb8-40fa-b3eb-bc5dc43439c3&grant_type=authorization_code&code=12b2478a5b71d175de8c50327fc33491bb0b519491608627c1cf0be46fb3610d&redirect_uri=https://myapp.gov.mv/signin-oidc
```

Sample Requests For Authorization Code + PKCE

This flow introduces additional parameters called code_verifier, code_challenge and code_challenge_method.

A code_verifier is a cryptographically random string that is generated and stored by the Relying Party. The code_challenge is the base64 encoded and SHA256 hashed value of the code_verifier.

The code_challenge is sent in the authentication request to the eFaas and eFaas will store this value. The code_verifier is sent to eFaas in the token request which will validated by eFaas. The token will only be issued if the hash of the code_verifier matches with the code_challenge.

Authentication Request

URL: {efaas_base_url}/connect/authorize

Method: GET

Query Parameters	Description
client_id	The client id provided during client registration
redirect_uri	The URI that is registered at eFaas as a callback uri
response_type	Space delimited values to indicate what to receive from eFaas as a reponse. For authorization code + PKCE it should be code.
scope	The scopes that the relying party requires from efaas. The values should be space delimited. eg: openid efaas.profile
code_challenge	This is the base64 encoded SHA256 hash of the code verifier
code_challenge_method	Method used to hash the code_verifier (SHA256)
state	This is a randomly generated string to prevent CSRF attacks and maintain state between eFaas and RP. The value of state parameter will be returned by eFaas during callback.

Sample Authentication Request

```
https://developer.gov.mv/efaas/connect/authorize?response_type=code&client_id=dc8311c9-6c42-449e-a080-0d031d2612ab&state=abc&scope=openid efaas.profile&redirect_uri=https://myapp.gov.mv/signin-oidc&code_challenge=K29soCkThVHYUT-r4uZtRMdTKb584oZLh83rd8MGqJk&code_challenge_method=S256
```

Token Request

When you receive the callback after a successful authorization request, you can exchange the code for an access token and, in some cases a refresh token, using the token endpoint.

For Authorization Code + PKCE, code_verifier must be included in the token request.

Method: POST

URL: {efaas_base_url}/connect/token

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

Params	Description
client_id	The client id provided during client registration
client_secret	The client secret provided during client registration. Client secret is **not required** for non-server side applications.
grant_type	The grant type in this case is authorization_code
code	The authorization code received from eFaas
redirect_uri	The URI that is registered at efaas as a callback uri
code_verifier	The unhashed code_challenge

Sample Token Request

```
client_id=abc44ec3-aa7b-4eab-a50e-4d18f17c3f62
&grant_type=authorization_code&code=12b2478a5b71d175de8c50327fc33491bb0b519491608627c1
cf0be46fb3610d&redirect_uri=https://myapp.gov.mv/signin-
oidc&code_verifier=eImN_fPyl2gbkUVrSVTrenoJYAIhTS3M-aaQ3Lx45Kbs
```

Token Validation

The Relying Party must validate the JWTs received from eFaas (id_token, access_token).

We strongly recommend using middleware provided by your application's framework or third-party libraries for validating tokens. You can find a list of libraries for common programming frameworks at the following link:

https://jwt.io/libraries

For further information, please refer to OpenId Specification at:

https://openid.net/specs/openid-connect-core-1_0.html

Retrieving User Info

Although user info is available in the id_token, it is recommended to use the userinfo endpoint provided by eFaas to retrieve the user info. Most of the openid connect libraries will do this automatically for you.

Method: GET

URL: {efaas_base_url}/connect/userinfo

Params	Description
Header: Authorization	Include the access token received after authentication

Retrieving User Photo

Applicable only if the client has **efaas.photo** scope.

Method: GET

URL: {photo_url_from_user_info_json}

Params	Description
Header: Authorization	Include the access token received after authentication

Tracking changes to User Information

Client applications are expected to keep a track of the **updated_at** user claim and update the user information accordingly.

Refreshing Tokens

Introduction

If you are accessing a RESTful API secured by eFaas, then you will require to refresh the access tokens, as access tokens expire quickly. This can be done by using the refresh_token received during the initial token request. When the refresh token request is successful, you will receive a new access_token and a new refresh_token.

Prerequisite

To be able to use refresh tokens, the Relying Party must be assigned offline_access scope during client registration. The Relying Party also needs to include offline_access scope in the authentication request.

Refresh Token Request

Method: POST

URL: {efaas_base_url}/connect/token

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

Params	Description
client_id	The client id provided during client registration
client_secret	The client secret provided during client registration. Client secret is **not required** for non-server side applications.
grant_type	The grant type in this case is refresh_token
refresh_token	The refresh_token received from initial token request

Sample Token Request

```
client_id=abc44ec3-aa7b-4eab-a50e-4d18f17c3f62&client_secret=9fz11cd8-7bb8-40fa-b3eb-bc5dc43439c3&grant_type=refresh_token&refresh_token=EFC5388CF55A8368DC0B69ECB82E4250F3359FD0F3EA23E52A4E502808A5AAS
```

Logging User Out

To log the user out of eFaas, send a GET request to endsession endpoint of eFaas with parameters in the table below.

After a successful logout, the user will be redirected to the registered post_logout_redirect_uri.

Method: GET

URL: {efaas_base_url}/connect/endsession

Params	Description
post_logout_redirect_uri	The post logout redirect uri that was registered at efaas
Id_token_hint	The id_token received during the authorization process
state	This is for round tripping state between the relying party and efaas

Sample Request

```
https://developer.gov.mv/efaas/connect/endsession?post_logout_redirect_uri=https://efaasapp.gov.mv/oidc/signout&id_token_hint=eyJhbGciOiJSUzI1NiIsImtpZCI6Ijc5M0E3NjB.eyJuYmYiOjE2MTQ1MDA2MTAsImV4cCI6MTYxNDUwMDkx&state=state_abc
```

Common Mistakes

- Providing access_token instead of id_token to the endsession endpoint.
- Incorrect post_logout_redirect_uri

The post_logout_redirect_uri must exactly match with the one configured for the client. No query parameters should be passed to the post_logout_redirect_uri.

For example, if the post_logout_redirect_uri is https://efaasapp.gov.mv/oidc/signout, only https://efaasapp.gov.mv/oidc/signout will be considered valid.

The following URLs will be considered **INVALID**:

- https://efaasapp.gov.mv/oidc/signout/
- https://efaasapp.gov.mv/oidc/signout?param=one

Single Sign Out

Introduction

A single sign-out is used to log out the user from all applications that's sharing the same eFaas session, if that session is terminated either by log out from one of the applications or log out from eFaas directly.

For server side applications, this is accomplished using back channel and front channel logout.

Prerequisite (For Server Side Applications Only)

Relying parties must register a back_channel_logout_uri or front_channel_logout_uri depending on how they plan to implement this type of logout. The relying party must also store the eFaas session id when persisting the user session

Back channel logout

A POST request will be sent from the eFaas server to the relying party's registered back_channel_logout_uri.

URL: { back_channel_logout_uri }

Method: POST

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

Content: logout_token= `eyJhbGciOiJSUzI1NiM0E3NjB.eyJuYmYiOj`

The relying party should validate the logout_token and retrieve the eFaas session id (sid) in the token. Remove the user session if sid matches with the sid in the user's session. Once the session is removed, the relying party must respond to eFaas with a 200 OK response.

Front channel logout

A GET request will be sent from the user's browser to the relying party's registered front_channel_logout_uri.

URL{ front_channel_logout_uri }

Method: GET

Query Parameter: logout_token= `eyJhbGciOiJSUzI1NiM0E3NjB.eyJuYmYiOj`

The relying party should validate the logout_token and retrieve the eFaas session id (sid) in the token. Remove the user session if sid matches with the sid in the user's session.

Browser-Based JavaScript Clients

To handle single sign-out, these applications must implement monitoring for check_session_iframe endpoint of eFaas. You can refer to the specifications at https://openid.net/specs/openid-connect-session-1_0.html for implementing this feature.

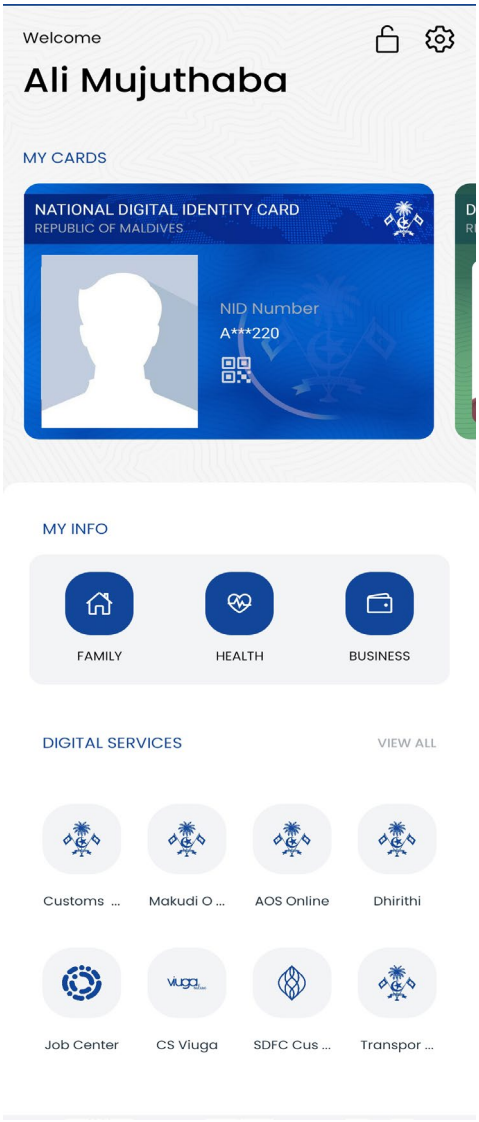
If you're are using **oidc-client-js** library (https://github.com/IdentityModel/oidc-client-js) this will be already implemented for you.

eFaas One-Tap Login - Third Party Apps Integration

Introduction

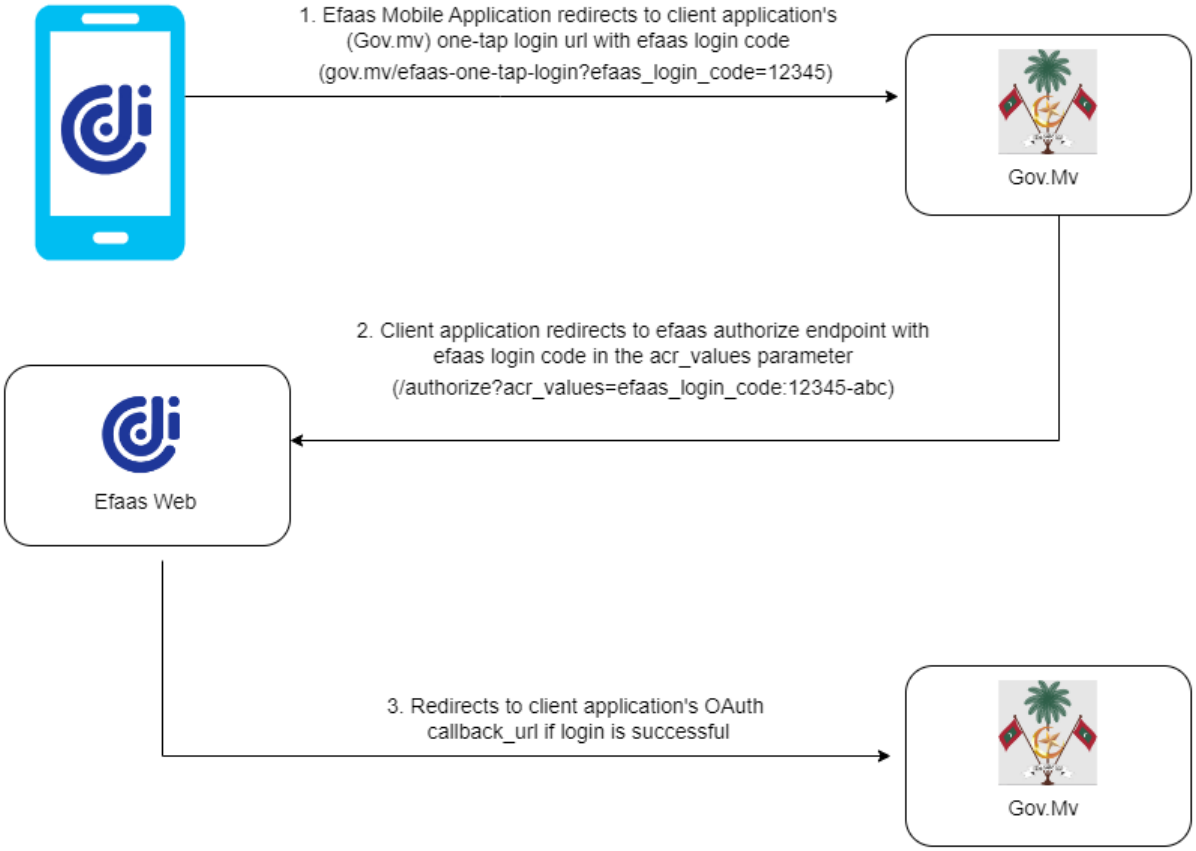
eFaas one-tap login allows users to log into third-party applications through eFaas Mobile Application, without having to re-enter their credentials.

The following is a screenshot of eFaas Mobile Application, with a list of eFaas integrated services displayed to the user. When a user clicks one of the service icons, they will be logged into the service application without having to re-enter their eFaas credentials.



Overall Flow

The following diagram shows the eFaas One-Tap login flow with Gov.Mv as the third-party application.



OneTap Login - Third Party Apps Integration

Pre-requisite

All third-party applications **must implement** the following endpoint which will be called by the eFaas Mobile Application for one-tap logins.

- **Endpoint**: {root_url}/efaas-one-tap-login
- **HTTP Method**: GET

Implementation Steps

1. eFaas Mobile application calls the one-tap endpoint of the third-party application with `efaas_login_code` included as a query parameter (eg: https://gov.mv/efaas-one-tap-login?efaas_login_code=a5d9a8ac-d583-41a7-8844-545dd608fad7)

2. The third-party application extracts the `efaas_login_code` from the endpoint.

3. Add the `efaas_login_code` to the `acr_values` parameter of efaas authorization url before redirecting to efaas for authentication

```
{
  client_id: CLIENT_ID,
  redirect_uri: REDIRECT_URL,
  response_type: RESPONSE_TYPE,
  scope: "openid profile",
  acr_values: "efaas_login_code:a5d9a8ac-d583-41a7-8844-545dd608fad7"
}
```

4. eFaas authenticates the user by validating the `efaas_login_code` and redirects to third-party application's callback url. The standard OAuth flow will continue from here.

TROUBLESHOOTING COMMON ERRORS

- After logout the user is not redirected back to the application
    - Check if the PostLogoutURL is correct
    - Check if the value passed to id_token_hint is the id_token and NOT access token

- 419 Page Expired error when redirecting to the application
    - Exclude the redirect URL from CSRF protection. For Laravel applications refer to this link https://laravel.com/docs/10.x/csrf#csrf-excluding-uris

BEFORE DEPLOYING TO PRODUCTION

- ✓ Make sure the following requests are working
  - Authorization request
  - Token request
  - Refresh token request
  - Logout
  - Post logout redirection
  - Backchannel or front channel logout
- ✓ Inform the production server IPs to be whitelisted, if hosting on a cloud based service.

FURTHER READING

- OAuth 2.0 RFC
  - https://datatracker.ietf.org/doc/html/rfc6749
- OpenID Connect Specifications
  - https://openid.net/specs/openid-connect-core-1_0.html
- OAuth 2.0 Threat Model and Security Considerations
  - https://datatracker.ietf.org/doc/html/rfc6819
- OAuth 2.0 Security Best Current Practice
  - https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics

END OF DOCUMENTATION