Department Service Integration with e-Pramaan

OIDC Integration document

Centre for Development of Advanced Computing

Gulmohar Cross Road No. 9, Juhu, Mumbai, 400049,
Telephone: +91 22 2620 1606, +91 22 2620 1574,
Fax: +91 22 2621 0139, +91 22 2623 2195
Website: www.cdac.in

Content

Revision History

Version	Date	Reason for Change
1.0 (Draft)	10-05-22	
1.1	14-06-22	Updated URLs to epramaan.meripehchaan.gov.in
1.2	25-07-22	Updated process flow and parameter descriptions

Abbreviations

OIDC	Open ID Connect
SP	Service Provider (Department)
SSO	Single Sign On
AES	Advanced Encryption Standard
JWT	JSON Web Token

Intended Audience

The recommended audience for this document is the technical personnel responsible for e-Pramaan integration at Department end. This document may be useful for the Project manager/Department Head to assess the effort required for integration with e-Pramaan.

Prerequisite

The integrating person at Department end should be well versed in web application development and familiar with the technology and work flow of the Department Service.

1. Introduction

This document explains the steps involved in integrating Department services with e-Pramaan. It explains the workflow and the step-by-step process for integrating application with e-Pramaan.

2. Process Flow for SSO

Single Sign-On (SSO) is an access control mechanism across multiple independent software systems. This allows user to log in once and gain access to all related services, without being prompted for log in again at each of them. e-Pramaan allows the user to initiate SSO either from Department Service or from e-Pramaan portal. The OIDC protocol is used for SSO implementation.

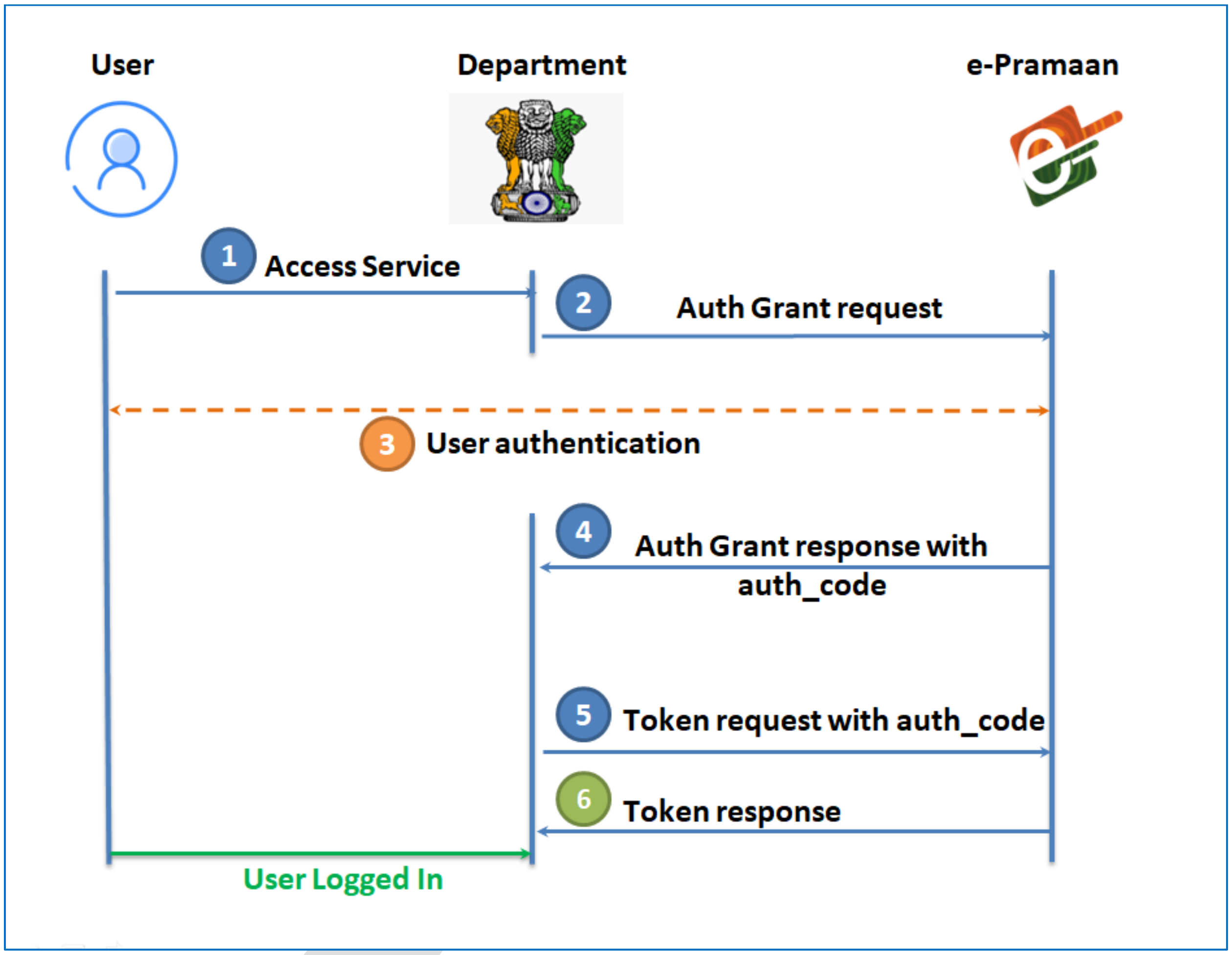

Figure 1: SSO initiated by Department Service

Steps involved in SSO initiated by Department Service are depicted in Figure 1:

- User at Department Service initiates SSO by clicking the option to "Login Using e-PramaanMeriPehchaan".
- Department Service then sends an *auth grant request* to e-Pramaan and forwards the user to e-Pramaan for authentication.
- User is authenticated by e-Pramaan using Challenge-Response mechanism.
- Once user is authenticated successfully on e-Pramaan, e-Pramaan sends the *auth grant response along with auth_code*to the service.
- Service sends a *token request* to e-Pramaan along with the *auth_code*.
- e-Pramaan checks for the *auth_code* and sends the *token response* to the service.
- Department Service consumes the token and allows user to login.
- If the user fails to authenticate himself / herself on e-Pramaan, the OIDC response returns failure in the auth grant response step.

3. Required steps for integration

The steps involved in integrating an application with e-Pramaan are listed below. Some sample JAVA code snippets are also provided in this document.

- Register the Department Service on e-Pramaan's Department portal.
- Provide link *"Login using e-PramaanMeriPehchaan"* on the login page of Department Service.
- Modify *onClick event* of *"Login using e-PramaanMeriPehchaan"* link to authenticate using e-Pramaan.
- Modify/implement logic to *consume SSO Token* sent by e-Pramaan.

The steps are explained in detail below.

Step 1: Register the Department service

- Whitelist your application's public IP and register On e-Pramaan's department portal - https://sp.epramaan.in:4003/
- Once the Department is registered, contact e-Pramaan team for activation of Department.
- Register the services as required.

Step 2: Add a link – *"Login using e-PramaanMeriPehchaan"*

- In your application, add a button/link named *'Login using e-PramaanMeriPehchaan'.*
- *onClick event* of *"Login using e-PramaanMeriPehchaan"* link, send Auth Grant request to e-Pramaan.

Step 3: Send Auth Grant request to e-Pramaan (HTTP Redirect POST)

Url: *https:// epstg.meripehchaan.gov.in/openid/jwt/processJwtAuthGrantRequest.do*

Following parameters required to be set while creating Authentication Request:

Parameter Name	Description
client_id	Service id sent to department email id at time of registration of service
scope	Must always be constant value **openid**
state	UUID (Must be unique for every request)
redirect_uri	Callback URL on which service wants to receive auth grant response.
request_uri	The url from which the request originates
response_type	Must always be constant value **code**
nonce	Create new randomly generated 16 characters string for every request
code_challenge	HMAC SHA256 of code_verifier (code_verifier is randomly generated string of 43 to 128 characters which needs to be created for each call, stored and sent during the token request call)
code-challenge_method	S256
apiHmac	HMAC SHA256 of queryString (serviceId+aesKey+stateID+nonce+redirectionURI+scope+codeChallenge) using Service aes_key.

Sample auth grant request: Method POST

```
https://_epstg.meripehchaan.gov.in/openid/jwt/processJwtAuthGrantRequest.do?
&scope=openid
&response_type=code
&redirect_uri=https:// epstg.meripehchaan.gov.in/OIDCClient/UDemo
&state=343fb7f4-b3dc-47b3-8f01-613a72eb022e
&code_challenge_method=S256
&nonce=W03PmTz97lpqMnsv43Kl1d5UzZLjJ55kNuh148t2Prs
&client_id=100000909
&code_challenge=DodV_hT3r-TVoNTbKY4rRN4xeMNIfbkVGmXnX3CMhrc
&request_uri=https:// epstg.meripehchaan.gov.in/opened/jwt/processJwtAuthGrantRequest.do
&apiHmac=rvSj7XibSYgb1Xzyi5PzP3eBDuR_O0e0i3J9oMfl55E=
```

Step 4: Create API for receiving the auth_code and state

Department Service needs to create an API which will receive the auth_code and state from e-Pramaan. In case of error, e-Pramaan will send the response with 3 error parameters i.e error, error_description and errorUri in the request parameters.

Step 5: Send token request (REST callmethod - POST)

After receiving the auth_code and state from e-Pramaan, Department Service will then send a Token grant request to e-Pramaan.

Url: *https:// epstg.meripehchaan.gov.in/openid/jwt/processJwtTokenRequest.do*

Following parameters required to be set while creating Token Request:

Parameter name	Description
code	Authorization code received in the auth grant request call
grant_type	Must always be constant value **authorization_code**
scope	Must always be constant value **openid**
redirect_uri	https://epramaan.meripehchaan.gov.in/openid/jwt/processJwtTokenRequest.do
request_uri	Request originating url (Service URL requesting the token)
code_verifier	UUID

Sample Token grant request (REST callmethod – POST):

```
{"code":["a2906a46-2315-4836-9df4-375afb1ee9b4"],
"grant_type":["authorization_code"],
"scope":["openid"],
"redirect_uri":["https://epstg.meripehchaan.gov.in/openid/jwt/processJwtTokenRequest.do"],
"code_verifier":["t2Hvc0I1An57kT5BoZu60Uvzv5VTf6kFE3cgjl-M5sY"],
"request_uri" : ["http:// epstg.meripehchaan.gov.in/UDemo"],
"client_id":["100000909"]}
```

Step 7:Consume Tokenresponse

After successful authentication at e-Pramaan the user is redirected to the Department service for which OIDC request was initiated. The user demographic information will be provided by e-Pramaan in the Token grant response. The service has to consume the response and decide the further logic for allowing the user to access desired service. The encrypted JSON Web *Token* (*JWT*) will have to be decrypted and the signature should be verified with the public key shared by e-Pramaan. For decryption and signature verification, appropriate library will have to be used (can be downloaded from https://jwt.io/libraries).

Snapshot of the possible values of JSON Web *Token* (*JWT*):

Sr. no.	Property Name	Description	Datatype	
1	sub	unique user Id	String	Mandatory
2	iat	Token Issued Time	String (datetime in long format)	Mandatory
3	exp	Token Expiry Time	String (datetime in long format)	Mandatory
4	jti	Token Identifier	String	Mandatory
5	name		String	Optional
6	email		String	Optional
7	mobile_number		String	Optional
8	dob	date of birth	String in dd/MM/yyyy	Optional
9	gender		String	Optional
10	house		String	Optional
11	locality		String	Optional
12	pincode		String	Optional
13	district		String	Optional
14	state		String	Optional
15	aadhaar_ref_no	Reference Number	String	Optional
16	sso_id	Same as sub	String	Mandatory
17	session_id		String	Optional

4. Appendix I: Pseudo code

4.1. Creation of Code verifier, code challenge and nonce

As per the technology of the Department Service, a suitable library needs to be used from the https://jwt.io site for OIDC request and response.

```
CodeVerifiercodeVerifier= new CodeVerifier();
Nonce nonce= new Nonce();

//Create Code Challenge with the code Verifier
CodeChallengecodeChallenge =
CodeChallenge.compute(CodeChallengeMethod.S256,codeVerifier);
```

4.2. Pseudo code for creation of API HMAC

```
public static String hashHMACHex(String hMACKey, String inputValue) {

byte[] keyByte = hMACKey.getBytes(StandardCharsets.US_ASCII);
byte[] messageBytes = inputValue.getBytes(StandardCharsets.US_ASCII);

  Mac sha256_HMAC = null;
                    sha256_HMAC = Mac.getInstance("HmacSHA256");

SecretKeySpecsecret_key = new SecretKeySpec(keyByte, "HmacSHA256");
                    sha256_HMAC.init(secret_key);
return Base64.getUrlEncoder()
                    .encodeToString(sha256_HMAC.doFinal(messageBytes));
}
```

Note: HMAC key will be AES key shared by e-Pramaan at the time of service registration on https://sp.epramaan.in:4003/ and input value will be string of "serviceId+aesKey+stateID+nonce+redirectionURI+scope+codeChallenge"

5. Appendix II: PYTHON code snippets

5.1. Creation of Code verifier, code challenge and nonce

- Required Module for codeverifier and codechallenge
  - Pip install pkce (run this command)
- Use imports
  - import uuid
  - import pkce

```
nonce = uuid.uuid4().hex  #Create new randomly generated 32 characters string for every request
code_verifier = pkce.generate_code_verifier(length=64) #Create new randomly generated 64
characters string for every request

code_challenge = pkce.get_code_challenge(code_verifier)
```

5.2. Creation of API HMAC

```
Client_id = “1xxxxxxx9”  # will be shared by e-Pramaan at the time of service registration
aesKey = "8xxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx8"
state = ''.join(secrets.choice(string.ascii_uppercase + string.ascii_lowercase) for i in range(16))    #
Must be unique and create new for each request

redirectionUri = “http://localhost:8000/views/processAuthCodeAndGetToken” # must be same as
the success url given at the time of service registration
scope = “openeid”
authGrantRequestUrl =
“https://epstg.meripehchaan.gov.in/openid/jwt/processJwtAuthGrantRequest.do”
certificate = "D:/python/epramaan.crt"  #  keep certificate in the application folder and give the path
accordingly


defhashHMAChex(key,value):
message = bytes(value, 'utf-8')
secret = bytes(key, 'utf-8')
hash = hmac.new(secret, message, hashlib.sha256)
hash.hexdigest()
var = base64.b64encode(hash.digest())
apihmac = str(var.decode('utf-8')).replace('/', '_').replace('+','-')
returnapihmac

authRequestUrl =
authGrantRequestUrl+"?scope="+scope+"&response_type="+response_type+"&redirect_uri="+redir
ectionUri+"&state="+state+"&code_challenge_method="+code_challenge_method+"&nonce="+non
```

```
ce+"&client_id="+client_id+"&code_challenge="+code_challenge+"&request_uri="+authGrantReque
stUrl+"&apiHmac="+apiHmac+""   #  final url for authcode request
```

Note:key will be AES key shared by e-Pramaan at the time of service registration and input value will be string of

"client_id+aesKey+state+nonce+redirectionURI+scope+codeChallenge"

5.3. Code for creation of Token Request and Token Decryption

- Commands must be executed to use JWT libraries
    - pip install pyjwt
    - pip install jwcrypto
- Required Modules to be import
    - import uuid
    - import http.client
    - import json
    - import hashlib
    - import base64
    - import hmac
    - import http
    - import secrets
    - import string
    - import pkce   # run command pip install pkce
    - from django.shortcuts import render, redirect
    - from django.views.decorators.csrf import csrf_exempt
    - import jwt   # run command pip install pyjwt
    - from jwcrypto import jwk, jwe   # run command pip install jwcrypto
    - from cryptography.x509 import load_pem_x509_certificate

Note :-write from django.views.decorators.csrf import csrf_exempt to exempt CSRF protection and give @csrf_exempt annotation on method

```
@csrf_exempt
defprocessAuthCodeAndGetToken(request):
code = request.GET['code']#  authcode received in the url


conn = http.client.HTTPSConnection("epstg.meripehchaan.gov.in")
payload = json.dumps({
        "code": [code ],
        "grant_type": [grant_type],
        "scope":[scope],
        "redirect_uri": [token_request_uri],
        "request_uri": [redirectionUri],
        "code_verifier": [code_verifier],
         "client_id": [client_id]
})

headers = {'Content-Type': 'application/json',}
conn.request("POST", "/openid/jwt/processJwtTokenRequest.do", payload, headers)
res = conn.getresponse()
data = res.read()
jweToken = data.decode("utf-8")

   base64urlencodedkey = base64.b64encode(hashlib.sha256(nonce.encode('utf-
8')).digest()).decode()
   finalbase64urlencodedkey = base64urlencodedkey.replace('+','-').replace('/','_').replace('=','')
#  key must be a jwk object to decrypt the jwe token
#  key can be converted into a jwk object by following steps
Startofkey= '{"kty":"oct","k":"'
endofkey='"}'
jwkobjectkey="%s%s%s"%(Startofkey,finalbase64urlencodedkey,endofkey)
finalKey = jwk.JWK.from_json(jwkobjectkey)    #   this finalKey will be used to decrypt the jwe token

jwe_token = jwe.JWE()
jwe_token.deserialize(jweToken)
jwe_token.decrypt(finalKey)
decrypted_payload = jwe_token.payload.decode()
certificateData = open(certificate, "r").read().encode()
cert = load_pem_x509_certificate(certificateData).public_key()
jsonData = jwt.decode(decrypted_payload, cert, algorithms=['RS256'], options={"verify_exp": False},)
# type: ignore
#  Token is decrypted and get the json data
```