

PRISMS API Gateway – Authentication and Authorisation Overview

For external developers  
October 2023

Contents

[Document History 2](#_Toc997830350)

[1. Introduction 2](#_Toc1469765428)

[1.1 General 3](#_Toc843283202)

[1.2 Document Conventions 3](#_Toc1791912958)

[1.3 Intended Audience 3](#_Toc1465967280)

[1.4 References 3](#_Toc536823669)

[Official Standard 3](#_Toc2140235551)

[A Guide to OAuth 2.0 Grants 3](#_Toc609593289)

[1.5 Contact Details 3](#_Toc119221734)

[API Product Manager 4](#_Toc1863618489)

[1.6 Prerequisites for authentication 4](#_Toc1566901113)

[Prerequisites 4](#_Toc2088856548)

[Azure AD B2C 4](#_Toc1457893274)

[APIM Subscription Key 5](#_Toc1770055912)

[PRISMS 5](#_Toc288390087)

[2. Unattended flow 5](#_Toc1546405508)

[2.1 Step 1: Generate the client assertion using the X509 certificate 5](#_Toc408603087)

[2.2 Step 2: Obtain the access token from Azure AD B2C 7](#_Toc1396316285)

[2.3 Use access token to consume PRISMS APIs 9](#_Toc736378558)

[2.3.1 Get Australian States and Territories 9](#_Toc1928384659)

[3. Attended flow (MFA and non-MFA) 9](#_Toc1681251104)

[3.1 Step 1: Generate the PKCE pair 9](#_Toc1644038169)

[3.2 Step 2: Obtain the authorisation code with the challenge code 11](#_Toc526950509)

[3.3 Step 3: Use the authorisation code to obtain the access token with the verifier code 13](#_Toc1482466285)

[ Get courses by course code. 16](#_Toc306205708)

[4. Glossary 16](#_Toc348952832)

[Appendix A – Certificate and encoding specification 16](#_Toc2005364167)

[A.1 - Certificate specification 17](#_Toc1365025547)

[A.2 - Public Certificate extraction for B2C 17](#_Toc1801088682)

[A.3 - Using Microsoft's powershell commands to create and export a self-signed certificate 17](#_Toc1452790332)

[Token format 20](#_Toc1329220146)

[Authentication endpoint 22](#_Toc1666953645)

[Response 22](#_Toc1908097693)

[ClaimsApi endpoint (identity attributes) 23](#_Toc755534981)

[service\_identifier 23](#_Toc1232164916)

[object\_identifier 23](#_Toc1252229941)

[Response 23](#_Toc761481995)

[Appendix C – Authorisation service (B2C) data holdings 24](#_Toc1309401420)

[Appendix D – Rate limiting 26](#_Toc747008881)

# Document History

Table 1 - Document History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Version | Date | Author | Summary of Changes | Status | Authorised By |
| 0.1 | 8/09/2023 | J Ross | Create PRISMS version | Draft |  |

# 1. Introduction

## 1.1 General

The Department of Education (the department) has made PRISMS APIs available to Provider organisations via a Web API Gateway.

These APIs enable organisations to integrate their software solutions and associated workflows with PRISMS bypassing the need to use the PRISMS user interface.

This document describes the authentication requirements and provides the following:

* A descriptive overview.
* Prerequisites.
* Roles and responsibilities.

## 1.2 Document Conventions

The Scenario Synopsis sections of the document include sample request parameters and sample responses. The samples provided are not extensive; they are mock-ups and aim to highlight dependencies only. Please refer to the OAuth 2.0 specification where required.

## 1.3 Intended Audience

IT contacts of organisations subscribing to the Department’s PRISMS API Gateway to consume the PRISMS APIs.

## 1.4 References

### Official Standard

<https://tools.ietf.org/html/rfc6746>

### A Guide to OAuth 2.0 Grants

[A Guide To OAuth 2.0 Grants - Alex Bilbie](https://alexbilbie.github.io/guide-to-oauth-2-grants/)

This easily digestible article details the OAuth 2.0 conventions and grant types implemented by the API Gateway.

[RFC 6749 - The OAuth 2.0 Authorization Framework (ietf.org)](https://datatracker.ietf.org/doc/html/rfc6749)

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

## 1.5 Contact Details

### API Product Manager

Contact the API Product Manager at the following address:

[prismsapi@education.gov.au](mailto:prismsapi@education.gov.au)

## 1.6 Prerequisites for authentication

**Develop and Test the Application against the Integration Product**

**Fill out the Service Identity Access Form and email it to the API Management mailbox**

**Read the Authentication and Authorisation Guide for implementation details**

To gain access to the PRISMS APIs, your application must first obtain an access token from the Department's Azure AD B2C instance. This access token is then included in the Authorization header field of each request sent to the PRISMS API. Once the access token is validated, along with your APIM subscription key, the request will be forwarded to the PRISMS API endpoint and executed accordingly. This developer guide walks you through the required steps to obtain the access token.

There are two authentication flows to obtain the access token:

1. Unattended flow: this is a machine-to-machine flow and no user credential is required;
2. Attended flow: this flow requires the user to enter their PRISMS credential to obtain an access token.

## Prerequisites

Before proceeding, make sure you have the following information ready:

### Azure AD B2C

This information should be supplied in your welcome pack:

Domain name: edugovaunp.onmicrosoft.com

Tenant name: edugovaunp

Tenant ID: 87e1c7ba-7719-43f6-ae27-70666024fa0a

Your client app return URI: https://jwt.ms for example  
The thumbprint of the X509 certificate that you provided to DEWR

### APIM Subscription Key

Your subscription key: AbaAbBcaBcAaB1aAbBbab1ACCABbaaBC

**Note:**

1. Please refer to the APIM User Guide to create an APIM subscription key. Your API call will not be forwarded to the authentication service if it does not have a current and valid APIM Subscription Key.
2. Your APIM subscription key must be passed with a header titled **Ocp-Apim-Subscription-Key**.

### PRISMS

A valid PRISMS user credential (email and password). This will be created for you and should be in your welcome pack.

## 2. Unattended flow

Your client application obtains an access token using the unattended flow in two steps:

1. Generate the client assertion using the X509 certificate.
2. Obtain the access token from Azure AD B2C.

### 2.1 Step 1: Generate the client assertion using the X509 certificate

* Convert the Certificate thumbprint hex string to a base64 string using an [online converter](https://portal.api.np.prisms.education.gov.au/developer-guide-authentication) or the function Convert.ToBase64String().
* Remove the trailing = to use the final base64 string For example, the thumbprint 068AB1A370901FB879ACD90D36FAF05EDD43EC72 is converted to Boqxo3CQH7h5rNkNNvrwXt1D7HI.
* Create the client\_assertion JWT value by using the Node.js library jose or using the C# code below:

**Node.js**

const { SignJWT } = require("jose");  
const crypto = require("crypto");  
const uuid = require("uuid");  
const fs = require("fs");

async function main() {  
  const privateKeyPEM = crypto.createPrivateKey(fs.readFileSync("private.pem")); // Path to your private certificate  
  const jwt = await new SignJWT({})  
    .setProtectedHeader({  
      alg: "RS256",  
      typ: "JWT",  
      x5t: "Boqxo3CQH7h5rNkNNvrwXt1D7HI", // Thumbprint base64  
    })  
    .setIssuedAt()  
    .setIssuer("00000000-0000-0000-0000-000000000000") // The client ID your application in Azure AD B2C  
    .setSubject("00000000-0000-0000-0000-000000000000") // The client ID your application in Azure AD B2C  
    .setAudience(  
      "https://login.microsoftonline.com/{tenantId}/oauth2/v2.0/token"  
    ) // this is the Azure AD B2C tenant  
    .setExpirationTime("60m")  
    .setJti(uuid.v4())  
    .sign(privateKeyPEM);  
  console.log(jwt);  
} // main function  
main();

**C#**

private static string? GetAccessToken()  
{  
    var clientId = Configuration["Azure:ClientId"];  
    var tenantId = Configuration["Azure:TenantId"];  
    var requestedScope = Configuration["Azure:RequestedScope"];

    var certificate = new X509Certificate2(Path.Combine(Environment.CurrentDirectory, Configuration["Azure:CertificateP12File"]), Configuration["Azure:CertificateP12Password"]);

    var assertion = GetSignedClientAssertion(certificate, tenantId, clientId);

    IConfidentialClientApplication app = ConfidentialClientApplicationBuilder.Create($"{clientId}")  
        .WithAuthority($"https://login.microsoftonline.com/{tenantId}/v2.0/")  
        .WithClientAssertion(assertion)  
        //.WithCertificate(certificate)            
        .Build();

    var ar = app.AcquireTokenForClient(new string[] { requestedScope }).ExecuteAsync();

    return ar.Result.AccessToken;  
}

public static string GetSignedClientAssertion(X509Certificate2 certificate, string tenantId, string clientId)  
{  
    // Get the RSA with the private key, used for signing.  
    var rsa = certificate.GetRSAPrivateKey();

    //alg represents the desired signing algorithm, which is SHA-256 in this case  
    //x5t represents the certificate thumbprint base64 url encoded  
    var header = new Dictionary<string, string>()  
                {  
                    { "alg", "RS256"},  
                    { "typ", "JWT" },  
                    { "x5t", Base64UrlEncode(certificate.GetCertHash()) }  
                };  
    var claims = GetClaims(tenantId, clientId);  
    var headerBytes = JsonSerializer.SerializeToUtf8Bytes(header);  
    var claimsBytes = JsonSerializer.SerializeToUtf8Bytes(claims);  
    string token = Base64UrlEncode(headerBytes) + "." + Base64UrlEncode(claimsBytes);  
    string signature = Base64UrlEncode(rsa.SignData(Encoding.UTF8.GetBytes(token), HashAlgorithmName.SHA256, RSASignaturePadding.Pkcs1));  
    string signedClientAssertion = string.Concat(token, ".", signature);  
    return signedClientAssertion;  
}

static IDictionary<string, object> GetClaims(string tenantId, string clientId)  
{  
    string aud = $"https://login.microsoftonline.com/{tenantId}/v2.0";  
    string ConfidentialClientID = clientId; //client id 00000000-0000-0000-0000-000000000000  
    const uint JwtToAadLifetimeInSeconds = 60 \* 10; // Ten minutes  
    DateTimeOffset validFrom = DateTimeOffset.UtcNow;  
    DateTimeOffset validUntil = validFrom.AddSeconds(JwtToAadLifetimeInSeconds);  
    return new Dictionary<string, object>()  
        {  
            { "aud", aud },  
            { "exp", validUntil.ToUnixTimeSeconds() },  
            { "iss", ConfidentialClientID },  
            { "jti", Guid.NewGuid().ToString() },             
{ "nbf", validFrom.ToUnixTimeSeconds() },             
{ "sub", ConfidentialClientID }

};

}

### 2.2 Step 2: Obtain the access token from Azure AD B2C

The `client\_assertion` value is then used to obtain the access token from the following endpoint:

* https://login.microsoftonline.com/{tenantId}/oauth2/v2.0/token

The full request is:

POST https://login.microsoftonline.com/{tenantId}/oauth2/v2.0/token HTTP/1.1  
Content-Type: application/x-www-form-urlencoded

grant\_type=client\_credentials  
&client\_id=00000000-0000-0000-0000-000000000000  
&scope=https://edugovaudev.onmicrosoft.com/devprismsapiunattended/.default  
&client\_assertion\_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer  
&client\_assertion=truncated

The parameters are:

Table 2: access\_token request parameters

|  |  |
| --- | --- |
| Parameter | Value /Description |
| Grant\_type | Client credentials |
| Client\_id | The Client ID of the app registration in Azure AD B2C |
| Scope | Is the scope URI of the Unattended app registration in Azure AD B2C |
| Client\_assertion\_type | Urn: ietf: params: oauth:client-assertion-type:jwt - bearer |
| Client\_assertion | Generated from the step above |
|  |  |

The response from Azure AD B2C:

{  
  "token\_type": "Bearer",  
  "expires\_in": 3599,  
  "ext\_expires\_in": 3599,  
  "access\_token": "eyJ0eXAiOiJKV1QiLCJhbGci..."  
}

### **2.3 Use access token to consume PRISMS APIs**

Once the access token is obtained, it can be used along with your subscription key to consume the PRISMS APIs. As noted above, your APIM subscription key must be passed as a header parameter (Ocp-Apim-Subscription-Key). For example:

### 2.3.1 Get Australian States and Territories

GET [https://**gateway.api.prisms.education.gov.au**/staging/providers/v1/references/states](https://gateway.api.prisms.education.gov.au/staging/providers/v1/references/states) HTTP/1.1  
Ocp-Apim-Subscription-Key: AbaAbBcaBcAaB1aAbBbab1ACCABbaaBC  
Content-Type: application/json  
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGci...

**2.3.2 Validate the creation of a CoE to approve**

POST [https://**gateway.api.prisms.education.gov.au**/staging/providers/v1/providers/001234A/confirmation-of-enrolments/validation](https://gateway.api.prisms.education.gov.au/staging/providers/v1/providers/001234A/confirmation-of-enrolments/validation) HTTP/1.1  
Ocp-Apim-Subscription-Key: AbaAbBcaBcAaB1aAbBbab1ACCABbaaBC  
Content-Type: application/json  
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGci...

## 3. Attended flow (MFA and non-MFA)

In the attended flow, the user is required to authenticate with Azure AD B2C using their existing PRISMS account (email and password). The authentication flow follows the [OAuth 2.0 authorisation code flow in Azure AD B2C](https://portal.api.np.prisms.education.gov.au/developer-guide-authentication) with Proof Key for Code Exchange (PKCE). This is achieved using the following 3 steps:

* Generate the PKCE pair (challenge code and verifier code).
* Obtain the authorisation code with the challenge code.
* Use the authorisation code to obtain the access token with the verifier code.

### 3.1 Step 1: Generate the PKCE pair

The Proof Key for Code Exchange (PKCE) is an extension used in OAuth 2.0, to improve security for public clients. It ensures that the application that starts the authentication flow is the same one that finishes it.

The PKCE pair are named code\_verifier and code\_challenge. The code\_challenge is essentially a hashed value of the code\_verifier. The code\_challenge is sent to Azure AD B2C to obtain the authentication code. This authentication code, along with the code\_verifier, are sent to Azure AD B2C to obtain the access token.

**Important**: for **each** authentication request, the client app must generate a PKCE pair. The client app should store the code\_verifier in the User session to use it to obtain the access token.

**Using Javascript**

const { SignJWT } = require('jose');  
const crypto = require("crypto");  
const uuid = require("uuid");  
const fs = require('fs');

function main() {  
  var verifier = base64URLEncode(crypto.randomBytes(32));  
  var challenge = base64URLEncode(sha256(verifier));    
  console.log(`code\_verifier: ${verifier}`);  
  console.log(`code\_challenge: ${challenge}`);  
}

function base64URLEncode(str) {  
  return str.toString('base64')  
      .replace(/\+/g, '-')  
      .replace(/\//g, '\_')  
      .replace(/=/g, '');  
}

function sha256(buffer) {  
    return crypto.createHash('sha256').update(buffer).digest();  
}

main();

**Using C#**

using System.Security.Cryptography;

static void Main(string[] args)  
{  
    var codeVerifier = CryptoRandom.CreateUniqueId(32);  
    string codeChallenge;  
    using (var sha256 = SHA256.Create())  
    {  
        var challengeBytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(codeVerifier));  
        codeChallenge = Base64Url.Encode(challengeBytes);  
    }

    Console.WriteLine($"code\_verifier: {codeVerifier}");  
    Console.WriteLine($"code\_challenge: {codeChallenge}");

    Console.ReadLine();

}

In this section the following PKCE pair is used. **NOTE**: These are examples ONLY. Do not use these in your code.

code\_verifier: 6s7pmWJbs4Nry4SgTRkB11t5Rc2a59xrH0UqhIeIhg8  
code\_challenge: 8DPDtH6tQqICr8\_ofpAwAjnqq\_Rd6CGlCCqfCoZ9vjs

### 3.2 Step 2: Obtain the authorisation code with the challenge code

Open a new browser window on the user desktop and navigate to the /authorize endpoint of the custom policy.

https://{tenantName}.b2clogin.com/{tenantName}.onmicrosoft.com/B2C\_1A\_SISU\_PRISMS\_API\_IDP/oauth2/v2.0/authorize?  
client\_id=00000000-0000-0000-0000-000000000000  
&response\_type=code  
&redirect\_uri=https%3A%2F%2Fjwt.ms  
&response\_mode=query  
&scope=https%3A%2F%2F{tenantName}.onmicrosoft.com%2Fdevprismsapiattended%2Fprisms.user  
&state=random\_text  
&code\_challenge=8DPDtH6tQqICr8\_ofpAwAjnqq\_Rd6CGlCCqfCoZ9vjs  
&code\_challenge\_method=S256

The parameters are:

Table 3: PKCE authorisation challenge code parameters

|  |  |
| --- | --- |
| Parameter | Value /Description |
| Client\_id | The client ID of your app registration in Azure AD B2C |
| Response\_type: code | Code – indicates the request is for an authentication code |
| Redirect\_url | The endpoint of your client app that accepts the authentication code |
| Response\_mode: query |  |
| Scope | The permission URI of the resource that your app wants to consume |
| State | This can be a random text or a vlue defined in your client app |
| Code\_challenge | The challenge code generated in Step 1 |
| Code\_challenge\_method: S256 | The algorithm used to generate the challenge code |
|  |  |

You will be presented will a sign in screen as shown below:



Sign in with a valid PRISMS account and then you will be forwarded to the redirect URI of your client application with the authentication code in the query string parameter code. For example [https://jwt.ms](https://portal.api.np.prisms.education.gov.au/developer-guide-authentication):

https://jwt.ms/?state=random\_text&code=eyJ0eXAiOiJKV1QiLCJhbGci...

### 3.3 Step 3: Use the authorisation code to obtain the access token with the verifier code

You can use the authentication code obtained in Step 2 to get the access token by sending a POST request to the /token endpoint of the custom policy.

POST https://{tenantName}.b2clogin.com/{tenantName}.onmicrosoft.com/B2C\_1A\_SISU\_PRISMS\_API\_IDP/oauth2/v2.0/token HTTP/1.1  
Content-Type: application/x-www-form-urlencoded  
Cache-Control: no-cache  
  
grant\_type=authorization\_code  
&client\_id=00000000-0000-0000-0000-000000000000  
&scope=https%3A%2F%2F{tenantName}.onmicrosoft.com%2Fdevprismsapiattended%2Fprisms.user  
&redirect\_uri=https%3A%2F%2Fjwt.ms  
&code\_verifier=6s7pmWJbs4Nry4SgTRkB11t5Rc2a59xrH0UqhIeIhg8  
&code=eyJ0eXAiOiJKV1QiLCJhbGci...

**Table 4: Authorisation code parameters**

The parameters are:

|  |  |
| --- | --- |
| Grant type | Authorisation code |
| Client id | The client id of your app registration in Azure AD B2C |
| Redirect url | The endpoint of your client app that accepts the access token |
| Scope | The permission URI of the resource that your app wants to consume |
| Code verifier | The verifier code generated in Step 1 |
| Code | The authentication code obtained in Step 2 |
|  |  |

You should receive the access token in JWT format as below

{  
  "access\_token": "eyJ0eXAiOiJKV1QiLCJhbGci...",   
 "token\_type": "Bearer",  
  "not\_before": 1691372910,  
  "expires\_in": 3600,  
  "expires\_on": 1691376510,  
  "resource": "00000000-0000-0000-0000-000000000000",  
  "profile\_info": "truncated",  
  "scope": "https://{tenantName}.onmicrosoft.com/devprismsapiattended/prisms.write",  
  "refresh\_token": "eyJ0eXAiOiJKV1QiLCJhbGci...",  
  "refresh\_token\_expires\_in": 86400  
}

Once the access token is obtained, it can be used along with your subscription key to consume PRISMS APIs.

The client app can use the access token to access PRISMS APIs such as the one used in sections:

* **Get course by ID.**

# Get courses by course code.

# 4. Glossary

Table 5 – Glossary of Terms, Acronyms and Abbreviations

|  |  |
| --- | --- |
| Term/Acronym | Description |
| API | Application Programming Interface. A set of clearly defined methods of communication between various components. |
| Confidential Client | A (software) client that is capable of keeping a secret confidential to the world. For example, a web application. |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Secure HTTP. An encrypted version of HTTP using Transport Layer Security (TLS) or formerly SSL (Secure Sockets Layer) |
| JWT | JavaScript Object Notation (JSON) Web Token. A signed and self-contained standard for securely transmitting information between parties. |
| PRISMS | Provider Registration and International Student Management System |
| Public Client | A (software) client that is not capable of keeping a secret confidential to the world. For example, a desktop application. |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |

# 

# Appendix A – Certificate and encoding specification

## A.1 - Certificate specification

The Certificates generated by the Organisation must meet the following specification:

* The certificate **Subject Name** must be the same as the **Issuer Name**

## A.2 - Public Certificate extraction for B2C

The following information describes how to extract the public certificate and private key using OpenSSL from your M2M credential keystore. This is used as an example. Other software or methods can be used to extract the public certificate and private key.

1. Download and install OpenSSL <https://slproweb.com/products/Win32OpenSSL.html>.
2. Open the keystore xxxx.xml in a text editor.
3. Copy the public x.509 certificate and convert into xxx.cer by
   * copying **<publicCertificate>** base 64 encoded content into a new text editor
   * adding a line -----BEGIN CERTIFICATE----- to the top
   * adding a line -----END CERTIFICATE----- to the end
   * saving the file as xxx.p7b
   * opening the command prompt and \*cd to the directory
   * running the command: openssl pkcs7 -print\_certs -in xxx.p7b -out xxx.cer
   * opening the xxx.cer file using a text editor to view the public x.509 certificate (the first certificate in the list).

##### Extracting the private key

1. Copy the private key into an encrypted PKCS8 file xxx.key by
   * copying the **<protectedPrivateKey>** base 64 encoded content into a text editor
   * adding the line -----BEGIN ENCRYPTED PRIVATE KEY----- to the top
   * adding the line -----END ENCRYPTED PRIVATE KEY----- to the end
   * saving the file as xxx.key.
2. Convert xxx.cer and xxx.key into PKCS12 by
   * opening the command prompt and \*cd to the directory
   * running the command openssl pkcs12 -export -in xxx.cer -inkey xxx.key -out xxx.p12 -passin pass:<password> -passout pass:<password>.
3. Extract the private key from the PPKCS12 file xxx.p12 by
   * running the command openssl pkcs12 -info -in xxx.p12 -nodes -nocerts
   * entering the keystore password
   * viewing the private key, which will be shown between 'begin private key' and 'end private key'.

## A.3 - Using Microsoft's powershell commands to create and export a self-signed certificate

Microsoft’s powershell commands can be used for creating and exporting a self-signed certificate.

Provided below is an example powershell command to create a new self-signed certificate and extend the end date by 5 years:

Table 6 - Powershell command example - New-SelfSignedCertificate

|  |
| --- |
| New-SelfSignedCertificate -DnsName "sample.cer" -CertStoreLocation "cert:\LocalMachine\My" -NotAfter (Get-Date).AddYears(5) |

The following table contains the export certificate cmdlet to export a certificate store to a file:

Table 7 - Powershell cmdlet example - Export-Certificate

|  |
| --- |
| $cert = Get-ChildItem -Path cert:\LocalMachine\My\66581D94A90360884028B84208369D0B943D0C3D  Export-Certificate -Cert $cert -FilePath sample.cer |

The private key is not included in the export. If more than one certificate is being exported, then the default file format is SST. Otherwise, the default format is CERT. Use the Type parameter to change the file format.

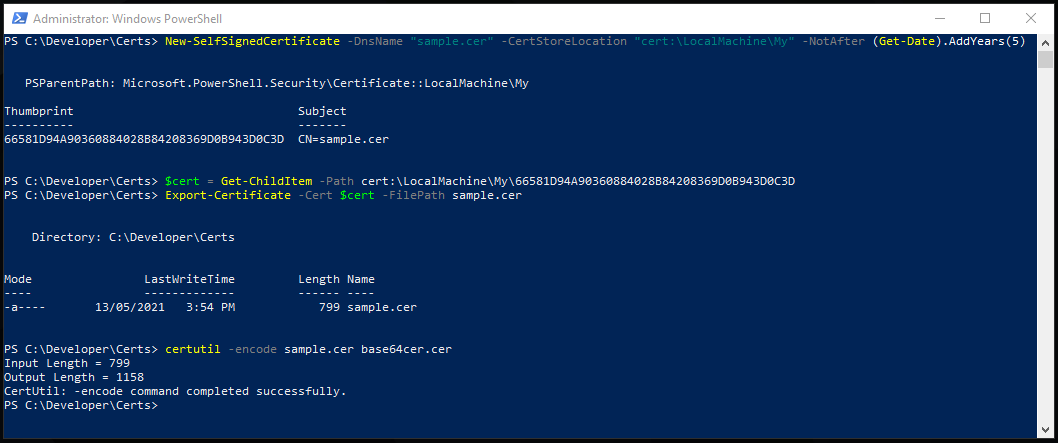
This example exports a certificate to the file system as a DER-encoded .cer file without its private key.

The following table contains the certutil command to convert the certificate to Base 64 format:

Table 8 - Powershell command example - certutil

|  |
| --- |
| certutil -encode sample.cer base64cer.cer |

See screenshot below:



|  |
| --- |
| using Microsoft.Identity.Client;  using Microsoft.Identity.Client;  using Microsoft.IdentityModel.Logging;  using System.Security.Cryptography.X509Certificates;  using System.Text.Json;  using System.Text;  using static System.Formats.Asn1.AsnWriter;  using System.Runtime.ConstrainedExecution;  using System.Collections.Generic;  using System.Collections.Generic;  using System.Security.Cryptography;  using System.Net.Http.Headers;  using System.Text.Json.Serialization;  using Newtonsoft.Json;  using JsonSerializer = System.Text.Json.JsonSerializer;  using System.Net;  using System.Text.Json.Nodes;  //https://entra.microsoft.com/#@edugovaudev.onmicrosoft.com  namespace m2m\_client\_assertion  {      internal class Program      {          static string client\_id = "8a765603-f96f-4fc7-b7ba-1104fadaba01";          static string tenantId = "edugovaudev.onmicrosoft.com";          static string requested\_scope = <https://edugovaudev.onmicrosoft.com/prisms/.default>;          static async Task Main(string[] args)          {              var now = DateTime.UtcNow;              var accessToken = GetAccessToken();              System.Console.WriteLine(accessToken);              //Use the access token to consume ResT API.                var apiClient = new HttpClient();              if (!string.IsNullOrEmpty(accessToken))              {                  var defaultRequestHeaders = apiClient.DefaultRequestHeaders;                  if (defaultRequestHeaders.Accept == null || !defaultRequestHeaders.Accept.Any(m => m.MediaType == "application/json"))                  {                      apiClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));                  }                  defaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);                  HttpResponseMessage response = await apiClient.GetAsync(<https://localhost:44372/api/whoami>);                  if (response.IsSuccessStatusCode)                  {                      string json = await response.Content.ReadAsStringAsync();                      JsonNode result = JsonNode.Parse(json);                      Console.ForegroundColor = ConsoleColor.Yellow;                      Console.WriteLine("WhoAmI result: \n");                      Console.WriteLine(result);                      Console.ForegroundColor = ConsoleColor.White;                  }                  else                  {                      Console.ForegroundColor = ConsoleColor.Red;                      Console.WriteLine($"Failed to call the web API: {response.StatusCode}");                      string content = await response.Content.ReadAsStringAsync();                      // Note that if you got reponse.Code == 403 and reponse.content.code == "Authorization\_RequestDenied"                      // this is because the tenant admin as not granted consent for the application to call the Web API                      Console.WriteLine($"Content: {content}");                  }                  Console.ResetColor();              }          }          private static string? GetAccessToken()          {              var certificate = new X509Certificate2("C:\\Developer\\PRISMS\\b2c-poc\\m2m-client-assertion\\funbring-full.p12", "Test123!!!");              var assertion = GetSignedClientAssertion(certificate, tenantId, client\_id);              IConfidentialClientApplication app = ConfidentialClientApplicationBuilder.Create($"{client\_id}")                  .WithAuthority($[https://login.microsoftonline.com/{tenantId}/v2.0/](https://login.microsoftonline.com/%7btenantId%7d/v2.0/))                  .WithCertificate(certificate)                  .Build();              var ar = app.AcquireTokenForClient(new string[] { requested\_scope }).ExecuteAsync();              return ar.Result.AccessToken;          }      }  } |

## Token format

The M2M Attended Token format is as follows:

A screenshot of a computer program

Description automatically generatedSample token string

|  |
| --- |
| eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6ImhBMnRLRmZLejZiM1IxRGI4enZqNlBTN3BpV0xfdWlKWjl5M21Fb3ZIbkkifQ..RHunUOBUvOV\_JBfuetL7jwyMy7\_TRf\_5UEcBDgyEJu0yk233e8pyy0yUsGY7AVh50yVnuHwIbv\_o223X1cnYgaUNYXidacUs2ZUZpvAH6p3973rvxjSfljVIYja1pLPU2QDI0krcxp1lSKPLnI4TUph0FFKF3OfaUZgiYmiDo7oq2-YmMqyroZM0UYHBED9KeBlfXDIAwaT3-ImG2Ay7TrkOpIFpZrocIS7CUl9B67LL09J6csOB-ffKMzjYUaLWjfXRrkv-7GLF6brP1wZtHWTBWtsrOuAC7sCqdp3PVmfi3t38IqjKEsuqYb5s6KXmfHYUxB76KleSPKthQo2xSg |

The M2M attended flow builds on top of the unattended flow. This flow consumes one additional claimsapi endpoint and the authentication endpoint. These are as follows:

## Authentication endpoint

/api/v1/login

The /login endpoint is the ReST interface into PRISMS legacy authentication process. This endpoint accepts the following request payload and performs username and password validation:

{

"username": "amit.chawla@dewr.gov.au",

"password": "1234"

}

## Response

HTTP 200 - Success

The JSON returned by the service endpoint is as follows:

|  |
| --- |
| {  "objectIdentifier": "ED8F9BB8-F80A-4A6F-83D3-68C13F4EB030"  } |

The objectIdentifier value is the unique identifier for the human user within the PRISMS identity database. The HTTP status code of 200 indicates that the user login has been successful.

HTTP 409 – Failure (ConflictResult)

In case the username/password validation is unsuccessful, the endpoint returns a ConflictResult – HTTP status code of 409.

The JSON returned by the service endpoint has the following format:

|  |
| --- |
| {  "version": "1.0.0",  "status": 409,  "code": "API12345",  "requestId": "50f0bd91-2ff4-4b8f-828f-00f170519ddb",  "userMessage": "Unable to process authentication request.",  "developerMessage": "Unable to process authentication request. Please check the credential configuration. The credential does not exist in the PRISMS context. Contact…",  "moreInfo": "https://restapi/error/API12345/moreinfo"  } |

The result payload is interpreted by the B2C custom policy engine. The engine logs and displays part of the payload, such as **developerMessage**, **version** and **requestId** only when the policy is running in the debug mode**.** The **userMessage** and the **moreInfo** attributes are used to provide further details about the message to the consuming end-user.

## ClaimsApi endpoint (identity attributes)

/api/v1/services/{**service\_identifier**}/users/{**object\_identifier**}/claims

The URI stem above contains the following parameters:

### service\_identifier

The service\_identifier is a GUID style identifier that uniquely identifies the service that the claims are meant for. This service identifier is specific to PRISMS API. Each service onboarded to the claims-subsystem will have its own identifier.

### object\_identifier

This is the identifier that uniquely identifies the user record within the PRISMS database. This is the GUID value returned by the /login ReST endpoint.

### Response

HTTP 200 - Success

The JSON returned by the service endpoint is as follows:

|  |
| --- |
| {  "objectIdentifier":"ED8F9BB8-F80A-4A6F-83D3-68C13F4EB030",  "displayName": "michael angelo",  "givenName": "michael",  "surname": "angelo",  "securitydeclarationaccepted": "1"  } |

objectIdentifier is the same GUID value as the input parameter. This GUID uniquely identifies the user record within the PRISMS database. This value is transformed to userapplicaitonaccountid in the eventual token string. This string is also persisted as the user’s upn on the B2C user database.

displayName is the user’s display name as set up within the PRISMS database.

givenName is the user’s first-name as set up within the PRISMS database.

surname is the user’s surname as set up within the PRISMS database.

HTTP 409 – Failure (ConflictResult)

In case the identity lookup is unsuccessful, the endpoint returns a ConflictResult – HTTP status code of 409.

The JSON returned by the service endpoint has the following format:

|  |
| --- |
| {  "version": "1.0.0",  "status": 409,  "code": "API12345",  "requestId": "50f0bd91-2ff4-4b8f-828f-00f170519ddb",  "userMessage": "Unable to process authentication request.",  "developerMessage": "Unable to process authentication request. Please check the credential configuration. The credential does not exist in the PRISMS context. Contact…",  "moreInfo": "https://restapi/error/API12345/moreinfo"  } |

The result payload is interpreted by the B2C custom policy engine. The engine logs and displays part of the payload, such as **developerMessage**, **version** and **requestId** only when the policy is running in the debug mode**.** The **userMessage** and the **moreInfo** attributes are used to provide further details about the message to the consuming end-user.

The attended authentication flow can be tested using the following test URL on the B2C policy – noting that this is a PoC, the system will create the same UPN for any sign-in user. The process will work within the prototype, if the sign-in/sign-up username is in the **email address format**.

# Appendix C – Authorisation service (B2C) data holdings

|  |  |
| --- | --- |
| Flow | Attributes |
| M2M Unattended | **Client\_id**. This identifier uniquely identifies the client software instance that has been registered within the authorisation service (B2C). |
| **x509 Certificate**. This certificate is generated by the vendor or provider administrator using the ATO RAM issued machine credential. Note that this is the **public portion** of the machine credential. |
| M2M Attended | **Redirect Uri.** This is the client software web application endpoint that receives the access, identity and refresh tokens from the authorisation service in the OIDC authorisation code grant flow. This is only required in the M2M Attended flow. |
| **Givenname –** given name of the prisms authorised user. |
| **Surname –** surname of the prisms authorised user. |
| **PRISMS User Id (legacy) –** User Id of the prisms authorised user as it exists in the prisms datastore. |
| **Email address –** surname of the prisms authorised user. |
| **TOTP Credential –** this is used to carry out the 2FA in the attended flow. |
| Service Management Portal | **Givenname –**  given name of the provider or vendor administrator user. This information is obtained from the digital identity ecosystem. |
| **Surname –**  surname of the provider or vendor administrator user. This information is obtained from the digital identity ecosystem. |
| **Digital Identity Unique Identifier (OID) –**  unique identifier of the provider or vendor administrator user account. This information is obtained from the digital identity ecosystem. |
| **Email address –**  email address of the provider or vendor administrator. This information is obtained from the digital identity ecosystem. |
| **ABN –**  ABN of the provider or vendor administrator. This information is obtained from the digital identity ecosystem. |

# Appendix D – Rate limiting

**Purpose**

The Rate Limiting pattern is used to control the number of requests a client can make to a service or API within a specified time window. This pattern serves several purposes:

• Prevents abuse of the service, ensuring fair usage.

• Protects the service from Distributed Denial of Service (DDoS) attacks.

• Optimizes resource allocation and maintains system stability.

• Ensures a consistent quality of service, even during peak usage.

**Components** The Rate Limiting pattern typically consists of the following components:

1. **Rate Limiter**: The rate limiter component enforces the rate limits by tracking the number of requests made by a client within a defined time window. It is responsible for allowing or rejecting requests based on predefined limits.
2. **Time Window**: The time window is a specified duration during which the rate limits are enforced. Common time windows include seconds, minutes, or hours.
3. **Limit**: The limit represents the maximum number of requests a client can make within the time window. It is typically expressed as requests per second (RPS) or requests per minute (RPM).
4. **Client Identification**: To enforce rate limits per client, some form of client identification is required. This may involve API keys, client IP addresses, or user tokens.
5. **Exceeding Limit Handling**: The rate limiting component must define how to handle requests that exceed the defined limits. Common approaches include returning an error response, delaying requests, or queuing requests for later processing.

**Usage**

To use the Rate Limiting pattern effectively, follow these steps:

1. **Identify Rate Limiting Requirements:** Determine the specific rate limiting requirements for your service or API. Consider factors such as the desired rate limits, time windows, and client identification methods.

2. **Implement Rate Limiter:** Develop or integrate a rate limiter component into your service. This component will track and enforce rate limits.

3. **Define Time Windows and Limits:** Specify the time windows and limits for different types of clients or API endpoints. For example, you might impose different rate limits for free and premium users or for different API endpoints.

4. **Client Identification:** Decide how clients will be identified for rate limiting purposes. This could involve API keys, OAuth tokens, IP addresses, or other identifiers.

5. **Error Handling:** Define how your service will respond when rate limits are exceeded. Ensure that the error responses are clear and informative, including information on when the client can make additional requests.

6. **Monitoring and Alerts**: Implement monitoring to track the usage and performance of the rate limiting component. Set up alerts to be notified of rate limit breaches or other issues.

7. **Documentation**: Document the rate limiting policies, including the rate limits, time windows, and how clients can identify themselves.

**Rate limiting in Azure API Management**

The rate limiting policies and usage guidelines for the PRISMS APIs in Azure API Management.

**Rate Limiting Policy**

• Policy Name: rate-limit-policy

• Scope: API level

• Rate Limit: [X requests per Y seconds/minutes/hours]

• Client Identification: Subscription key

**Rate Limiting Guidelines**

**Request Headers**

For clients to benefit from rate limiting, the Ocp-Apim-Subscription-Key header must be included in each API request.

**Rate Limit Exceeded**

When the rate limit is exceeded, the client will receive a 429 Too Many Requests response with the following details:

jsonCopy code

{ "error": "RateLimitExceeded", "message": "Rate limit for this API has been exceeded. Please try again later.", "retryAfter": [Number of seconds until the rate limit resets] }

The retryAfter field informs the client when they can make additional requests.

**Example**

Suppose the rate limit for the API is set to 100 requests per minute. If a client exceeds this limit, they will receive a 429 Too Many Requests response like the one below:

jsonCopy code

{ "error": "RateLimitExceeded", "message": "Rate limit for this API has been exceeded. Please try again in 30 seconds.", "retryAfter": 30 }