Got it 👍 — you want to **authenticate a Function App (client) with Azure AD using MSAL in PowerShell** by presenting a **client certificate** (instead of client secret) and a **Microsoft identity issuer authority**, then obtain a **Bearer token** that can be used to call another Function App (API).

Here’s a **step-by-step breakdown with working PowerShell code**:

**1. Prerequisites**

* Azure Function App registered in **Microsoft Entra ID (Azure AD)** as an **App Registration**.
* A **certificate** (self-signed or CA-issued) uploaded to the App Registration:
  + In Entra ID → **App Registrations** → Your App → **Certificates & secrets** → Upload certificate (public key).
* The API Function App (target) must also be registered and **expose a scope or App ID URI**.

**2. Collect Required Info**

* **Tenant ID** → from Azure AD overview.
* **Client ID** → of the Function App (the caller).
* **Thumbprint** of the uploaded certificate (on the Function App machine/Key Vault).
* **Authority (Issuer URL)**:
* https://login.microsoftonline.com/<tenant\_id>/
* **Scope or Resource** (depending on whether target exposes api://<appid>/.default or a standard Microsoft Graph scope).

**3. Install MSAL PowerShell Module**

Install-Module MSAL.PS -Scope CurrentUser

**4. PowerShell MSAL Script (Certificate Authentication)**

# Parameters

$tenantId = "<YOUR\_TENANT\_ID>"

$clientId = "<CLIENT\_APP\_ID>" # Calling Function App AppID

$certThumb = "<CERT\_THUMBPRINT>" # Cert installed in CurrentUser\My

$scope = "api://<TARGET\_APP\_ID>/.default" # Or "https://graph.microsoft.com/.default"

# Authority (issuer)

$authority = "https://login.microsoftonline.com/$tenantId"

# Load certificate from store

$cert = Get-ChildItem -Path Cert:\CurrentUser\My\$certThumb

if (-not $cert) {

throw "Certificate with thumbprint $certThumb not found in CurrentUser\My"

}

# Acquire token using MSAL and certificate

$tokenResponse = Get-MsalToken `

-ClientId $clientId `

-TenantId $tenantId `

-Scopes $scope `

-Authority $authority `

-ClientCertificate $cert

# Extract bearer token

$accessToken = $tokenResponse.AccessToken

Write-Output "Bearer token: $accessToken"

**5. Use Bearer Token in Function Call**

$functionUrl = "https://<target-function-app>.azurewebsites.net/api/<endpoint>"

$headers = @{ Authorization = "Bearer $accessToken" }

$response = Invoke-RestMethod -Uri $functionUrl -Headers $headers -Method GET

$response

✅ This way:

* **Certificate proves the Function App’s identity** to Azure AD.
* **Issuer authority** is Microsoft’s login endpoint (https://login.microsoftonline.com/<tenantid>).
* **MSAL handles token issuance**, and you pass it as a Bearer token when calling the target Function App.

Would you like me to also show you **how to make the Function App itself load the certificate from Key Vault instead of local cert store** (so it runs in a managed & secure way inside Azure)?

**You said:**

please generate the same powershell script that using client secret

**ChatGPT said:**

Perfect 👍 — using a **client secret** instead of a certificate is simpler because you don’t have to deal with cert storage. Here’s the **MSAL PowerShell version** for acquiring a **Bearer token** with client secret and Microsoft issuer authority:

**1. Prerequisites**

* App Registration created in **Microsoft Entra ID (Azure AD)** for the **calling Function App**.
* A **client secret** generated in **Certificates & secrets** (copy the value).
* The **target Function App** registered with an **App ID URI** or scope (e.g., api://<target-app-id>/.default).

**2. Install MSAL PowerShell Module**

Install-Module MSAL.PS -Scope CurrentUser

**3. PowerShell Script (Client Secret Authentication)**

# Parameters

$tenantId = "<YOUR\_TENANT\_ID>"

$clientId = "<CLIENT\_APP\_ID>" # Caller App (Function App)

$clientSecret = "<YOUR\_CLIENT\_SECRET>" # From App Registration

$scope = "api://<TARGET\_APP\_ID>/.default" # Or "https://graph.microsoft.com/.default"

# Authority (Microsoft issuer)

$authority = "https://login.microsoftonline.com/$tenantId"

# Secure the client secret

$secureSecret = ConvertTo-SecureString $clientSecret -AsPlainText -Force

# Acquire token with MSAL

$tokenResponse = Get-MsalToken `

-ClientId $clientId `

-TenantId $tenantId `

-ClientSecret $secureSecret `

-Scopes $scope `

-Authority $authority

# Extract Bearer token

$accessToken = $tokenResponse.AccessToken

Write-Output "Bearer token: $accessToken"

**4. Use Token in Function Call**

$functionUrl = "https://<target-function-app>.azurewebsites.net/api/<endpoint>"

$headers = @{ Authorization = "Bearer $accessToken" }

$response = Invoke-RestMethod -Uri $functionUrl -Headers $headers -Method GET

$response

✅ This uses the **client secret** instead of a certificate, but everything else (authority, scopes, function call) stays the same.  
⚠️ For **production**, using a **certificate** or **Managed Identity** is generally preferred over a client secret because secrets expire and must be rotated.

Do you want me to also show you the **Managed Identity** PowerShell approach (no secret/cert at all, just built-in identity of the Function App)? That’s often the cleanest way for Function Apps calling other Azure Functions securely.

Got it 👍 — you want to **generate an access token in PowerShell** for a **Function App** that requires **user impersonation** (i.e., the function expects to be called on behalf of a signed-in user with delegated permissions rather than app-only).

This typically means using **OAuth2 authorization code flow** (or device code flow) with **Microsoft Identity Platform (Azure AD / Entra ID)**.

**✅ Steps to Generate a Token with User Impersonation in PowerShell**

**1. Register an App in Entra ID**

* Go to **Entra ID (Azure AD) → App registrations**
* Register an app (e.g., FunctionAppUserImpersonationClient)
* Under **Authentication**:
  + Add a redirect URI (e.g., http://localhost)
* Under **API Permissions**:
  + Add your Function App’s **user\_impersonation scope** (this comes from exposing an API in Entra for your Function App).
* Grant **delegated permissions**.

**2. Use PowerShell + MSAL (Microsoft.Identity.Client) or Az CLI**

Here’s an example using **MSAL.PS** (community module wrapping MSAL):

# Install the module if not already installed

Install-Module MSAL.PS -Scope CurrentUser

# Define parameters

$clientId = "<CLIENT\_ID>" # from your app registration

$tenantId = "<TENANT\_ID>" # Directory (tenant) ID

$redirectUri = "http://localhost" # must match registered URI

$scope = "api://<YOUR-FUNCTION-APP-APPID-URI>/.default" # or user\_impersonation

# Acquire a token interactively (prompts sign-in)

$token = Get-MsalToken -ClientId $clientId -TenantId $tenantId -RedirectUri $redirectUri -Scopes $scope -Interactive

# Show the access token

$token.AccessToken

This will:

* Pop up a login window (or browser redirect).
* Authenticate the user.
* Return an **access token** with the delegated user\_impersonation scope.

**3. Call the Function App with the Token**

$headers = @{

"Authorization" = "Bearer $($token.AccessToken)"

}

Invoke-RestMethod -Uri "https://<your-function-app>.azurewebsites.net/api/<your-function>" -Headers $headers

**🔑 Key Notes**

* user\_impersonation scope only works with **delegated flows** (user login required). You can’t get it with client credentials (app-only).
* If you want **non-interactive service-to-service calls**, you’d use **client credentials flow** (but that won’t give you user\_impersonation).
* If you want a **device code flow** (no browser), you can do:

$token = Get-MsalToken -ClientId $clientId -TenantId $tenantId -Scopes $scope -DeviceCode

👉 Do you want me to also show you how to **expose your Function App API in Entra ID** so that the user\_impersonation scope exists and can be requested?

**You said:**

yes

**ChatGPT said:**

Perfect 👍 Let’s go step by step to **expose your Function App API in Entra ID** so that your user\_impersonation scope exists and can be requested in PowerShell (or by any client).

**🔹 Step 1: Register your Function App in Entra ID**

1. Go to **Entra ID (Azure AD)** → **App registrations** → **New registration**.
   * Name: MyFunctionApp
   * Supported account types: Choose as per need (usually "Accounts in this organizational directory only").
   * Redirect URI: leave empty for now.
   * Save.

This will give you:

* **Application (client) ID**
* **Directory (tenant) ID**

**🔹 Step 2: Expose the API**

1. Open your new **Function App registration**.
2. Go to **Expose an API**.
3. Click **Set** under *Application ID URI*.
   * Example: api://<client-id> (or a custom URI if you want).
4. Under **Scopes defined by this API**, click **Add a scope**:
   * Scope name: user\_impersonation
   * Who can consent? Admins and users
   * Admin consent display name: Access My Function App
   * Admin consent description: Allow the app to access My Function App on behalf of the signed-in user
   * Save.

Now your API exposes the user\_impersonation scope:  
👉 api://<client-id>/user\_impersonation

**🔹 Step 3: Register a Client App (to call the Function App)**

1. Create a **second app registration** (e.g., FunctionAppCaller).
2. Go to **Authentication** → add Redirect URI: http://localhost (for testing with PowerShell).
3. Go to **API Permissions** → **Add a permission** → **My APIs**.
   * Pick your MyFunctionApp.
   * Select **Delegated permissions** → check user\_impersonation.
   * Save & grant admin consent.

Now the client app is allowed to request tokens for user\_impersonation.

**🔹 Step 4: Request Token in PowerShell**

Now you can request the token for the FunctionAppCaller app:

# Install MSAL if not installed

Install-Module MSAL.PS -Scope CurrentUser

# Variables

$clientId = "<CLIENT\_ID\_OF\_FunctionAppCaller>"

$tenantId = "<YOUR\_TENANT\_ID>"

$redirectUri = "http://localhost"

$scope = "api://<CLIENT\_ID\_OF\_FunctionApp>/user\_impersonation"

# Get token interactively (user login required)

$token = Get-MsalToken -ClientId $clientId -TenantId $tenantId -RedirectUri $redirectUri -Scopes $scope -Interactive

# View token

$token.AccessToken

**🔹 Step 5: Call the Function App**

$headers = @{

"Authorization" = "Bearer $($token.AccessToken)"

}

Invoke-RestMethod -Uri "https://<your-function-app>.azurewebsites.net/api/<your-function>" -Headers $headers

✅ At this point:

* You have **Function App registered as an API** (exposes user\_impersonation).
* You have a **Client App** that can request delegated tokens.
* You can generate tokens via PowerShell for user impersonation.

# Parameters

$clientId = "<CLIENT\_ID\_OF\_ClientApp>"

$tenantId = "<YOUR\_TENANT\_ID>"

$scope = "api://<CLIENT\_ID\_OF\_FunctionApp>/user\_impersonation"

$redirectUri = "http://localhost:5000/" # must match app registration

# Start a simple HTTP listener to catch redirect

$listener = New-Object System.Net.HttpListener

$listener.Prefixes.Add($redirectUri)

$listener.Start()

Write-Host "Listening on $redirectUri ..."

# Build authorization request URL

$authUrl = "https://login.microsoftonline.com/$tenantId/oauth2/v2.0/authorize?" +

"client\_id=$clientId&response\_type=code&redirect\_uri=$([uri]::EscapeDataString($redirectUri))&response\_mode=query&scope=$([uri]::EscapeDataString($scope))&state=12345"

# Open browser for login

Start-Process $authUrl

# Wait for callback

$context = $listener.GetContext()

$response = $context.Response

$code = $context.Request.QueryString["code"]

# Respond to browser (so user sees something)

$buffer = [System.Text.Encoding]::UTF8.GetBytes("<html><body>Login complete. You can close this window.</body></html>")

$response.ContentLength64 = $buffer.Length

$response.OutputStream.Write($buffer,0,$buffer.Length)

$response.OutputStream.Close()

$listener.Stop()

Write-Host "Authorization Code received: $code"

# Exchange code for token

$tokenRequest = @{

client\_id = $clientId

scope = $scope

code = $code

redirect\_uri = $redirectUri

grant\_type = "authorization\_code"

}

$tokenResponse = Invoke-RestMethod -Method Post -Uri "https://login.microsoftonline.com/$tenantId/oauth2/v2.0/token" -Body $tokenRequest

Write-Host "Access Token: " $tokenResponse.access\_token