To generate an **Azure AD (Microsoft Entra) OAuth 2.0 token** for a **Function App** using a **client certificate** with curl, you’ll use the **Client Credentials flow**. The request goes to the **Azure AD token endpoint**.

**✅ Prerequisites**

1. You have an **Azure AD App registration** (for your Function App).
2. The app has a **client certificate** uploaded (public part in Azure AD).
3. The private key file (.pem or .pfx) is accessible to the curl command.
4. The Function App is protected by **Azure AD authentication**.
5. You know:
   * **Tenant ID**
   * **Client ID** (Application ID)
   * **Scope** (usually the Function App’s App ID URI + /.default, or api://<app-id>/.default)

**✅ Endpoint**

bash

CopyEdit

https://login.microsoftonline.com/<tenant\_id>/oauth2/v2.0/token

**✅ curl Command**

If you have:

* client\_cert.pem (public cert)
* client\_key.pem (private key)
* Tenant: TENANT\_ID
* Client ID: CLIENT\_ID
* Scope: api://<function-app-id>/.default

You can run:

bash

CopyEdit

curl -X POST \

-H "Content-Type: application/x-www-form-urlencoded" \

--cert client\_cert.pem \

--key client\_key.pem \

-d "grant\_type=client\_credentials" \

-d "client\_id=CLIENT\_ID" \

-d "scope=api://<function-app-id>/.default" \

"https://login.microsoftonline.com/TENANT\_ID/oauth2/v2.0/token"

**✅ Response**

The response will include:

json

CopyEdit

{

"token\_type": "Bearer",

"expires\_in": 3599,

"ext\_expires\_in": 3599,

"access\_token": "eyJ0eXAiOiJK..."

}

Use the access\_token as the Bearer token when calling your Function App:

bash

CopyEdit

curl -X GET \

-H "Authorization: Bearer <access\_token>" \

https://<your-function-app>.azurewebsites.net/api/<function-name>

⚠ **Important Notes**

* curl **does not sign JWTs**; Azure expects the client to authenticate with the certificate private key.
* For certificate-based auth, curl must provide the certificate and key (as above).
* If you have a **.pfx** file, convert it to PEM first:

bash

CopyEdit

openssl pkcs12 -in cert.pfx -out client\_cert.pem -clcerts -nokeys

openssl pkcs12 -in cert.pfx -out client\_key.pem -nocerts -nodes

Do you want me to **give you a full working example** for:  
✅ **Linux curl** or ✅ **Windows (PowerShell + curl equivalent)** with .pfx certificate?  
Or should I also show **how to include the certificate password**?

Ask ChatGPT

**You said:**

use the certificate path and certificate password in the curl command

**ChatGPT said:**

curl itself **does not natively handle .pfx (PKCS#12) with password** for Azure AD token requests, because the OAuth 2.0 Client Credentials flow with certificate requires a **signed JWT (client assertion)**, not just sending the cert file. Azure expects:

* client\_assertion = signed JWT using the certificate private key
* client\_assertion\_type = urn:ietf:params:oauth:client-assertion-type:jwt-bearer

curl alone cannot generate and sign this JWT — you need something like **OpenSSL + script** or **MSAL** to create the signed token first.

**✅ Correct Flow**

1. Generate **client assertion JWT** using your certificate private key (with password).
2. Use curl to POST the JWT to Azure AD to get the token.

**Step 1: Generate the JWT**

Use **OpenSSL** and a script (bash or PowerShell). Example (bash, using openssl):

bash

CopyEdit

HEADER=$(echo -n '{"alg":"RS256","typ":"JWT"}' | openssl base64 -e -A | tr '+/' '-\_' | tr -d '=')

PAYLOAD=$(echo -n '{"aud":"https://login.microsoftonline.com/<TENANT\_ID>/oauth2/v2.0/token","iss":"<CLIENT\_ID>","sub":"<CLIENT\_ID>","jti":"'"$(uuidgen)"'","exp":'"$(($(date +%s)+600))"'}' | openssl base64 -e -A | tr '+/' '-\_' | tr -d '=')

DATA="$HEADER.$PAYLOAD"

SIGNATURE=$(echo -n "$DATA" | openssl dgst -sha256 -sign client\_key.pem | openssl base64 -e -A | tr '+/' '-\_' | tr -d '=')

JWT="$DATA.$SIGNATURE"

Now $JWT is your **client assertion**.

**Step 2: Use curl with certificate password**

Your .pfx file has the cert + key with password, so first convert it:

bash

CopyEdit

openssl pkcs12 -in cert.pfx -out client\_cert.pem -clcerts -nokeys -password pass:<CERT\_PASSWORD>

openssl pkcs12 -in cert.pfx -out client\_key.pem -nocerts -nodes -password pass:<CERT\_PASSWORD>

**Step 3: Request the Token**

bash

CopyEdit

curl -X POST \

-H "Content-Type: application/x-www-form-urlencoded" \

-d "grant\_type=client\_credentials" \

-d "client\_id=<CLIENT\_ID>" \

-d "scope=api://<function-app-id>/.default" \

-d "client\_assertion\_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer" \

-d "client\_assertion=$JWT" \

"https://login.microsoftonline.com/<TENANT\_ID>/oauth2/v2.0/token"

**✅ Why can't you just pass --cert and --key with password in curl?**

Because Azure AD **does not do mutual TLS for app identity by default**; it requires a **signed JWT** using the private key, not a raw TLS handshake.

Do you want me to:  
✅ **write a full bash script** that does all steps (read PFX + password, build JWT, get token),  
or ✅ **give you a Windows PowerShell version** (with MSAL or manual JWT)?

Which one? **Linux** or **Windows**?

**You said:**

user the certificate .pfx file path and certificate password to generate above bearer token

**ChatGPT said:**

To generate an Azure AD access token using a **.pfx certificate** and **certificate password** entirely from the .pfx file, you **cannot rely on raw curl alone**, because:

* Azure AD **requires a signed JWT client assertion**, not just mutual TLS.
* curl cannot sign the JWT with the private key from the PFX.
* You need a script or utility to **extract the key and sign the JWT** before calling the token endpoint.

However, you can automate everything with **OpenSSL + bash** (Linux) or **PowerShell + MSAL** (Windows).

**✅ Linux Full Workflow using .pfx and password**

Here’s a step-by-step script that:  
✔ Reads .pfx with password  
✔ Creates a JWT signed with the private key  
✔ Calls Azure AD token endpoint with curl

**Bash Script**

bash

CopyEdit

#!/bin/bash

# Variables

TENANT\_ID="<your-tenant-id>"

CLIENT\_ID="<your-client-id>"

SCOPE="api://<function-app-id>/.default"

PFX\_FILE="/path/to/certificate.pfx"

PFX\_PASSWORD="<your-cert-password>"

TOKEN\_URL="https://login.microsoftonline.com/$TENANT\_ID/oauth2/v2.0/token"

# Extract private key and cert from PFX

openssl pkcs12 -in "$PFX\_FILE" -out cert.pem -clcerts -nokeys -password pass:$PFX\_PASSWORD -nodes

openssl pkcs12 -in "$PFX\_FILE" -out key.pem -nocerts -password pass:$PFX\_PASSWORD -nodes

# Prepare JWT header and payload

HEADER=$(echo -n '{"alg":"RS256","typ":"JWT"}' | openssl base64 -e -A | tr '+/' '-\_' | tr -d '=')

JTI=$(uuidgen)

EXP=$(($(date +%s)+600))

PAYLOAD=$(echo -n '{"aud":"'$TOKEN\_URL'","iss":"'$CLIENT\_ID'","sub":"'$CLIENT\_ID'","jti":"'$JTI'","exp":'$EXP'}' | openssl base64 -e -A | tr '+/' '-\_' | tr -d '=')

# Sign JWT

DATA="$HEADER.$PAYLOAD"

SIGNATURE=$(echo -n "$DATA" | openssl dgst -sha256 -sign key.pem | openssl base64 -e -A | tr '+/' '-\_' | tr -d '=')

JWT="$DATA.$SIGNATURE"

# Get token using curl

RESPONSE=$(curl -s -X POST \

-H "Content-Type: application/x-www-form-urlencoded" \

-d "grant\_type=client\_credentials" \

-d "client\_id=$CLIENT\_ID" \

-d "scope=$SCOPE" \

-d "client\_assertion\_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer" \

-d "client\_assertion=$JWT" \

"$TOKEN\_URL")

echo "Access Token Response: $RESPONSE"

**✅ Windows PowerShell (with .pfx and password)**

PowerShell can handle PFX + JWT easily using .NET and MSAL libraries.

powershell

CopyEdit

$TenantId = "<your-tenant-id>"

$ClientId = "<your-client-id>"

$Scope = "api://<function-app-id>/.default"

$PfxPath = "C:\path\to\cert.pfx"

$PfxPassword = "<your-cert-password>"

# Load cert

$SecurePassword = ConvertTo-SecureString -String $PfxPassword -Force -AsPlainText

$Cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2($PfxPath, $SecurePassword)

# Prepare JWT client assertion

$JwtHeader = @{ alg = "RS256"; typ = "JWT" }

$TokenUrl = "https://login.microsoftonline.com/$TenantId/oauth2/v2.0/token"

$Now = [DateTimeOffset]::UtcNow.ToUnixTimeSeconds()

$Exp = $Now + 600

$JwtPayload = @{

aud = $TokenUrl

iss = $ClientId

sub = $ClientId

jti = [guid]::NewGuid().ToString()

exp = $Exp

}

$HeaderEncoded = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes(($JwtHeader | ConvertTo-Json -Compress))) -replace '\+','-' -replace '/','\_' -replace '='

$PayloadEncoded = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes(($JwtPayload | ConvertTo-Json -Compress))) -replace '\+','-' -replace '/','\_' -replace '='

$DataToSign = "$HeaderEncoded.$PayloadEncoded"

$PrivateKey = $Cert.GetRSAPrivateKey()

$SignatureBytes = $PrivateKey.SignData([System.Text.Encoding]::UTF8.GetBytes($DataToSign), [Security.Cryptography.HashAlgorithmName]::SHA256, [Security.Cryptography.RSASignaturePadding]::Pkcs1)

$SignatureEncoded = [Convert]::ToBase64String($SignatureBytes) -replace '\+','-' -replace '/','\_' -replace '='

$ClientAssertion = "$DataToSign.$SignatureEncoded"

# Request token

$Body = @{

grant\_type = "client\_credentials"

client\_id = $ClientId

scope = $Scope

client\_assertion\_type = "urn:ietf:params:oauth:client-assertion-type:jwt-bearer"

client\_assertion = $ClientAssertion

}

$Response = Invoke-RestMethod -Uri $TokenUrl -Method Post -Body $Body

$Response.access\_token

**✅ Why this is required?**

Azure AD **does NOT accept just a PFX with password in curl** because:

* It expects a **signed JWT (client assertion)** using the cert’s private key.
* This JWT proves possession of the private key.

👉 Do you want me to **give you a single ready-to-run script (Linux)** or **PowerShell (Windows)** that:

* Accepts .pfx + password
* Generates JWT
* Gets token
* Calls your Function App with the token?

Which one should I prepare for you? **Linux (bash)** or **Windows (PowerShell)**? Or both?

**You said:**

powershell

**ChatGPT said:**

Here’s a **full PowerShell script** that:  
✔ Uses your **.pfx certificate and password**  
✔ Generates the **JWT client assertion**  
✔ Calls Azure AD to **get the access token**  
✔ Prints the **Bearer token**

**✅ PowerShell Script**

powershell

CopyEdit

# =========================

# Configurable Variables

# =========================

$TenantId = "<your-tenant-id>"

$ClientId = "<your-client-id>"

$Scope = "api://<function-app-id>/.default" # or your Function App App ID URI

$PfxPath = "C:\path\to\cert.pfx"

$PfxPassword = "<your-cert-password>"

$TokenUrl = "https://login.microsoftonline.com/$TenantId/oauth2/v2.0/token"

# =========================

# Load Certificate

# =========================

$SecurePassword = ConvertTo-SecureString -String $PfxPassword -Force -AsPlainText

$Cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2($PfxPath, $SecurePassword)

# =========================

# Create JWT Client Assertion

# =========================

$JwtHeader = @{ alg = "RS256"; typ = "JWT" }

$Now = [DateTimeOffset]::UtcNow.ToUnixTimeSeconds()

$Exp = $Now + 600 # token valid for 10 minutes

$JwtPayload = @{

aud = $TokenUrl

iss = $ClientId

sub = $ClientId

jti = [guid]::NewGuid().ToString()

exp = $Exp

}

# Function to Base64UrlEncode

function To-Base64Url {

param([string]$str)

$bytes = [System.Text.Encoding]::UTF8.GetBytes($str)

$encoded = [Convert]::ToBase64String($bytes)

return $encoded.TrimEnd('=') -replace '\+','-' -replace '/','\_'

}

# Encode Header and Payload

$HeaderEncoded = To-Base64Url (($JwtHeader | ConvertTo-Json -Compress))

$PayloadEncoded = To-Base64Url (($JwtPayload | ConvertTo-Json -Compress))

$DataToSign = "$HeaderEncoded.$PayloadEncoded"

# Sign Data with Private Key

$PrivateKey = $Cert.GetRSAPrivateKey()

$SignatureBytes = $PrivateKey.SignData([System.Text.Encoding]::UTF8.GetBytes($DataToSign), [Security.Cryptography.HashAlgorithmName]::SHA256, [Security.Cryptography.RSASignaturePadding]::Pkcs1)

$SignatureEncoded = [Convert]::ToBase64String($SignatureBytes).TrimEnd('=') -replace '\+','-' -replace '/','\_'

# Final JWT

$ClientAssertion = "$DataToSign.$SignatureEncoded"

# =========================

# Request Access Token

# =========================

$Body = @{

grant\_type = "client\_credentials"

client\_id = $ClientId

scope = $Scope

client\_assertion\_type = "urn:ietf:params:oauth:client-assertion-type:jwt-bearer"

client\_assertion = $ClientAssertion

}

Write-Host "Requesting token from Azure AD..."

$Response = Invoke-RestMethod -Uri $TokenUrl -Method Post -Body $Body -ContentType "application/x-www-form-urlencoded"

Write-Host "`nAccess Token:`n"

$Response.access\_token

**✅ How to Use**

1. Replace:
   * <your-tenant-id>
   * <your-client-id>
   * <function-app-id>
   * C:\path\to\cert.pfx
   * <your-cert-password>
2. Run in **PowerShell 7+** (or Windows PowerShell 5.1 with .NET support).
3. The script will print the **access token**.

**✅ Call Your Function App with Token**

After you get the token:

powershell

CopyEdit

$AccessToken = $Response.access\_token

$FunctionUrl = "https://<your-function-app>.azurewebsites.net/api/<function-name>"

Invoke-RestMethod -Uri $FunctionUrl -Headers @{ Authorization = "Bearer $AccessToken" }

Do you want me to **extend this script** so it:  
✔ Stores the token in a variable  
✔ Automatically calls your Function App endpoint with the token  
✔ Handles token expiration gracefully

**OR** keep it simple like above?