MICROSOFT ENTRA VERIFIED ID

IDV Partner Integration Guide

PrivacyKey Proof – Identity Document Verification — Government ID + Selfie

Version 1.1 | April 2026

Prepared by authID | www.authid.ai
support@authid.ai | +1 (516) 274-8700

Table of Contents

1 Overview

This guide provides step-by-step instructions for administrators to deploy the authID Proof integration with Microsoft Entra Verified ID. The reference implementation is a production-ready ASP.NET Core 8 web application that demonstrates the full end-to-end flow: a user proves their identity using a government-issued document and a real-time biometric selfie, and upon successful verification receives a Microsoft Entra Verified ID credential stored in Microsoft Authenticator.

A key architectural characteristic of this implementation is that it uses a polling-based status model — the application periodically polls the authID API to check verification status rather than relying on inbound webhooks. This simplifies deployment as no public webhook endpoint is required for the authID result.

Integration Summary

Parameter	Value
Solution Name	authID Proof
Reference App Framework	ASP.NET Core 8 / .NET 8 (MVC + Razor Views)
authID Status Pattern	Polling (GET /foreignOperations/documents/{operationId})
Verified ID Callback	POST /api/vc-issuance-callback (Microsoft → app, optional)
Verification Method	Government ID (PAD/IAD) + Biometric Liveness + Face Match
Supported Documents	Passport, Driver's License, National ID (190+ countries)
VC Credential Type	Woodgrove (configurable — FTECredentialType in appsettings.json)
Verified ID API	https://verifiedid.did.msidentity.com/v1.0/verifiableCredentials/
authID Environment	UAT: id-uat.authid.ai | Production: id.authid.ai

Scenarios Demonstrated

- Employee Onboarding — New user verifies government ID + selfie with authID, receives a Verified ID credential (Woodgrove FTE credential) in Microsoft Authenticator
- Alumni Verification — Returning user presents previously issued Woodgrove FTE credential to gain access, with optional Face Check

2 Prerequisites

Requirement	Details
Microsoft Entra ID Tenant	P1 or P2 license; Global Administrator or Verified ID Administrator access
Microsoft Entra Verified ID	Feature enabled in your tenant (free); DID Web configured
authID Enterprise Account	Active account with Proof module (GetForeignIDDocument) enabled
authID API Credentials	Username (GUID) and Password for Basic Auth token endpoint
Azure App Registration	Client ID + Secret for Verified ID API (scope: 3db474b9-.../.default)
Hosting Environment	.NET 8 runtime; Azure App Service recommended; HTTPS required
Microsoft Authenticator	iOS or Android; latest version for credential wallet support

Developer / Administrator Roles

- Global Administrator or Verified ID Administrator — configure Entra Verified ID, grant API permissions
- Application Administrator — register the app in Entra ID, manage client secrets
- authID Console Administrator — retrieve API credentials, configure the Proof workflow
- Azure Contributor — provision Azure App Service or hosting environment

Runtime Dependencies

- NuGet: Microsoft.Identity.Web, Microsoft.Identity.Web.UI, Azure.Identity, Azure.Security.KeyVault.Secrets, Newtonsoft.Json
- NuGet: VerifiedIDRequestService (Microsoft.Entra.VerifiedID) — handles Verified ID request lifecycle
- Frontend: authID Web Component (authid-web-component.js), QR code library (qrcode.min.js)

3 Architecture & Flow

The integration uses a three-service architecture: the authID Proof service for identity verification, an ASP.NET Core 8 application as the integration middleware and UI, and the Microsoft Entra Verified ID Request Service API for credential issuance and presentation.

Issuance Flow (Employee Onboarding)

AUTHID PROOF + VERIFIED ID ISSUANCE — POLLING MODEL

① User visits /onboarding/landing, enters name, clicks 'Verify with authID'
② Browser calls GET /authid-verification-data
③ App → authID Token Endpoint (Basic Auth) → receives Bearer access_token
④ App → POST /AdministrationServiceRest/v1/accounts → creates authID account
⑤ App → POST /AuthorizationServiceRest/v2/operations → creates Proof session
⑥ App returns { authIdUrl } = https://id-uat.authid.ai/?i={operationId}&s={oneTimeSecret}
⑦ Browser renders <authid-component> web component pointing at authIdUrl
⑧ User completes ID capture + selfie liveness inside the authID webcomponent
⑨ Browser polls GET /authid-status every 5 seconds
⑩ App polls GET /foreignOperations/documents/{operationId} → status == 1 = APPROVED
⑪ App → GET /v2/operations/{operationId}/result → extracts document data + selfie image
⑫ App → Entra token (client_credentials) → POST createIssuanceRequest → QR code + URL
⑬ Browser displays QR code (desktop) or deep link button (mobile)
⑭ User scans QR with Microsoft Authenticator → credential stored in wallet

Presentation Flow (Alumni / Relying Party)

VERIFIED ID CREDENTIAL PRESENTATION

① User visits /alumni/landing, clicks 'Get started'
② Browser calls GET /api/verifier/presentation-request
③ App → Entra token (client_credentials) → POST createPresentationRequest
④ Response: QR code + requestId stored in session
⑤ User scans QR → Microsoft Authenticator presents stored Woodgrove FTE credential

```
⑥ Entra Verified ID validates credential + calls app callback
(presentation_verified event)
⑦ App reads claims from verifiedCredentialsData, maps to session → grants
access
```

4 Step 1 — Obtain authID Credentials

The authID API uses HTTP Basic Authentication to obtain a Bearer access token. The credentials are a username (a GUID) and password provisioned by authID for your enterprise account.

1.1 Access the authID Console

1. Navigate to the authID Console at https://console.authid.ai and sign in as an administrator.
2. Navigate to Settings > API Credentials.
3. Note your Username (a GUID, e.g. 0e186a2b-c04f-4c6d-...) and generate or retrieve your API Password.

1.2 Retrieve Your API Endpoints

The reference implementation targets the UAT environment. For production, replace id-uat.authid.ai with id.authid.ai. The three base URLs used by the application are:

Service	Base URL
Identity / Token Service	https://id-uat.authid.ai/IDCompleteBackendEngine/IdentityService/v1
Administration Service	https://id-uat.authid.ai/IDCompleteBackendEngine/Default/AdministrationServiceRest
Transaction / Authorization Service	https://id-uat.authid.ai/IDCompleteBackendEngine/Default/AuthorizationServiceRest
Proof Web Component (user-facing)	https://id-uat.authid.ai/?i={operationId}&s={oneTimeSecret}

1.3 Obtain an Access Token

The application authenticates using HTTP Basic Auth (username:password, Base64-encoded) and POSTs to the token endpoint with no body:

```
POST https://id-uat.authid.ai/IDCompleteBackendEngine/IdentityService/v1/auth/token
Authorization: Basic Base64({username}:{password})
```

Response:

```
{
  "AccessToken": "<bearer token>",
  "RefreshToken": "<refresh token>"
```

IMPORTANT: Store the username and password in Azure Key Vault or ASP.NET Core User Secrets. Never commit credentials to source control. Tokens are short-lived — the application requests a new token for each operation sequence.

1.4 Verify Proof Module is Enabled

4. In the authID Console, navigate to Products > Proof.
5. Confirm that the GetForeignIDDocument operation type is active. This is the operation name used by the reference implementation (set as Name in AuthIdProofRequest).
6. The DocumentTypes array defaults to ["2"] in the reference implementation — confirm the document type codes for your target markets with your authID account team.

5 Step 2 — Configure authID Proof Workflow

The reference implementation uses the authID GetForeignIDDocument operation to initiate a document + biometric verification session. Configuration of the verification behavior (document types, liveness strictness, matching threshold) is managed through the authID Console and the operation payload.

2.1 Understanding the Proof Operation Request

When the user clicks 'Verify with authID', the application creates an account in authID (if it doesn't already exist) and then initiates a Proof operation. The key fields of the operation request are:

Field	Value in Reference App	Notes
Name	GetForeignIDDocument	Selects the government document + biometric workflow
AccountNumber	{email}	The user's email is used as the authID account identifier
Payload.DocumentTypes	["2"]	Document type code — 2 = all government IDs; confirm with authID
Timeout	3600	Session timeout in seconds (1 hour)
TransportType	0	Delivery mode; 0 = web component (iframe)

2.2 Create the authID Account

Before initiating a Proof session, the application creates (or reuses) an authID account keyed to the user's email address. A 409 Conflict response means the account already exists — the application handles this gracefully and proceeds with the existing account:

```
POST https://id-uat.authid.ai/.../AdministrationServiceRest/v1/accounts
Authorization: Bearer <access_token>
Content-Type: application/json
{
  "AccountNumber": "jane.smith@contoso.com",
  "DisplayName": "Jane Smith",
  "CustomDisplayName": "Jane Smith",
  "Email": "jane.smith@contoso.com",
  "Rules": 1,
  "Enabled": true
```

```
}
```

2.3 Initiate the Proof Session

```
POST https://id-uat.authid.ai/.../AuthorizationServiceRest/v2/operations
Authorization: Bearer <access_token>
Content-Type: application/json
{
  "AccountNumber": "jane.smith@contoso.com",
  "Name": "GetForeignIDDocument",
  "Timeout": 3600,
  "TransportType": 0,
  "Payload": {
    "DocumentTypes": ["2"],
    "Timestamp": "2026-04-08T12:00:00.000Z"
  }
}
```

Response:

```
{
  "OperationId": "<guid>",
  "OneTimeSecret": "<secret>"
```

> **NOTE:** If the operation endpoint returns 409 Conflict for reasons other than an account not found, the application retries with a fresh Timestamp in the Payload to make the request unique.

2.4 Launch the authID Web Component

The OperationId and OneTimeSecret are returned to the browser, which constructs the authID session URL and loads it in the <authid-component> custom HTML element (a full-screen iframe):

```
const authIdUrl = `https://id-uat.authid.ai/?i=${operationId}&s=${oneTimeSecret}`;
<authid-component data-url="{authIdUrl}" data-webauth="false"></authid-component>
```

The web component handles camera access, document capture, and liveness detection entirely within the iframe. No additional configuration in the application is required.

6 Step 3 — Register the Application in Microsoft Entra ID

Register the integration application in Microsoft Entra ID to obtain the credentials used to call the Verified ID Request Service API and, optionally, to sign in users with OpenID Connect.

3.1 Create the App Registration

7. Sign in to https://entra.microsoft.com as a Global Administrator or Application Administrator.
8. Navigate to Identity > Applications > App registrations > New registration.
9. Set the Name (e.g. authID Proof – Verified ID Integration), Supported account types: Single tenant, Redirect URI: leave blank (service-to-service only).
10. Click Register. Note the Application (client) ID and Directory (tenant) ID.

3.2 Create a Client Secret

11. Navigate to Certificates & secrets > Client secrets > New client secret.
12. Set a description (e.g. VerifiedID-Integration) and expiry (12 months recommended).
13. Copy the Value immediately — store it in Azure Key Vault or ASP.NET Core User Secrets.

3.3 Grant Verified ID API Permissions

14. Navigate to API permissions > Add a permission > APIs my organization uses.
15. Search for Verifiable Credentials Service Request and select it.
16. Select Application permissions > VerifiableCredential.Create.All.
17. Click Add permissions, then Grant admin consent for [Your Tenant]. Confirm.

> **NOTE:** The scope value used in the reference implementation is 3db474b9-6a0c-4840-96ac-1fceb342124f/.default — this is the Verified ID service's application ID. Set this as the VerifiedID:scope value in appsettings.json.

3.4 Configuration Values for appsettings.json

appsettings.json Key	Where to Find
VerifiedID:TenantId	Entra admin center > Overview > Tenant ID
VerifiedID:ClientId	App registration > Overview > Application (client) ID
VerifiedID:ClientSecret	Certificates & secrets (store in Key Vault)

appsettings.json Key	Where to Find
VerifiedID:scope	3db474b9-6a0c-4840-96ac-1fceb342124f/.default (fixed value)
VerifiedID:Authority	https://login.microsoftonline.com/{TenantId}/v2.0/
VerifiedID:DidAuthority	Entra admin center > Verified ID > Overview > DID

7 Step 4 — Configure Entra Verified ID Service & Credential Type

4.1 Onboard the Verified ID Service

18. In the Entra admin center, navigate to Verified ID.
19. If not yet onboarded, click Get started and complete the wizard:
    - Organization name: your organization's legal name
    - Trusted domain: a domain you control (e.g. verifiedid.yourdomain.com)
    - Key Vault: select or create an Azure Key Vault for signing key storage
20. After onboarding, note your DID (formatted as did:web:... — this becomes VerifiedID:DidAuthority).

4.2 Configure the Trusted Domain (DID Web)

21. Download the did.json document from the Verified ID onboarding page.
22. Host it at: https://yourdomain.com/.well-known/did.json over HTTPS.
23. Download did-configuration.json and host at: https://yourdomain.com/.well-known/did-configuration.json
24. In Verified ID, click Verify domain to confirm the DID document is reachable.

> **NOTE:** The reference implementation uses did:web:verifiedid.entra.microsoft.com:{tenantId}:{contractId} — your DID will follow the same pattern once configured. Confirm the exact value in the Entra admin center after onboarding.

4.3 Create the Credential Type

25. In Verified ID, navigate to Credentials > Add credential > Custom credential.
26. Configure the credential type used by the reference implementation:
    - Credential type name: Woodgrove (or your chosen type — set as VerifiedID:FTECredentialType)
    - Display name: Woodgrove Employee Credential
27. Define the credential schema. The reference implementation issues the following claims:

Claim Name	Display Name	Source
given_name	First Name	authID result — FirstName
family_name	Last Name	authID result — LastName / PrimaryIdentifier

Claim Name	Display Name	Source
display_name	Display Name	Concatenated given_name + family_name
mail	Email Address	User session (entered at onboarding)
photo	Photo (base64url)	authID result — CurrentFacialImage (liveness selfie)

28. After creating the credential, note the Credential Manifest URL — set this as VerifiedID:CredentialManifest in appsettings.json.
29. Set a validity period (e.g. 365 days) and configure revocation if required.

4.4 Claims Mapping Configuration

The reference implementation uses a claimsMapping configuration in appsettings.json to map claims from a presented Verified ID credential back to the application's internal model. For example, for credentials issued by your own tenant:

```
"VerifiedID:VerifiedCredentialExpert-{YourDID}-claimsMapping":
  "{\"firstName\":\"firstName\",\"lastName\":\"lastName\",\"photo\":\"photo\"}"
```

This maps the firstName, lastName, and photo claims from the presented credential into the WoodgroveCredential model used by the dashboard view.

8 Step 5 — Deploy the Integration Application (ASP.NET Core 8)

5.1 Project Structure

File / Folder	Purpose
Helpers/AuthIdService.cs	authID API client: token, account creation, proof session, status polling, result retrieval
Helpers/VerifiedIdIssuanceService.cs	Entra Verified ID API client: token acquisition (client_credentials), createIssuanceRequest
Controllers/HomeController.cs	MVC controller: all routes including /authid-verification-data, /authid-status, vc-issuance-callback
Models/AuthIdModels.cs	Strongly-typed models for authID API requests and responses
wwwroot/js/authid-web-component.js	authID <authid-component> custom HTML element (iframe wrapper)
appsettings.json	All configuration: authID credentials, Verified ID settings, credential manifest URL
Program.cs	DI registration: AddVerifiedIDRequestService(), AuthIdService, VerifiedIdIssuanceService

5.2 Service Registration (Program.cs)

The application registers three key services in the DI container:

```
builder.Services.AddVerifiedIDRequestService(); // Microsoft.Entra.VerifiedID
builder.Services.AddScoped<AuthIdService>();
builder.Services.AddScoped<VerifiedIdIssuanceService>();
```

5.3 Configure appsettings.json

The complete configuration block required for the integration (replace placeholder values with your own):

```
{
  "VerifiedID": {
    "ApiEndpoint":
"https://verifiedid.did.msidentity.com/v1.0/verifiableCredentials/",
    "TenantId": "<your-entra-tenant-id>",
    "Authority": "https://login.microsoftonline.com/<tenant-id>/v2.0/",
```

```
    "scope": "3db474b9-6a0c-4840-96ac-1fceb342124f/.default",
    "ManagedIdentity": false,
    "ClientId": "<your-app-registration-client-id>",
    "ClientSecret": "<store-in-key-vault>",
    "DidAuthority": "did:web:<your-verified-id-did>",
    "client_name": "Your Organization Name",
    "FTECredentialType": "Woodgrove",
    "CredentialType": "VerifiedCredentialExpert",
    "CredentialManifest": "<manifest-url-from-verified-id-portal>",
    "sourcePhotoClaimName": "photo",
    "matchConfidenceThreshold": 70,
    "acceptedIssuers": "<your-did>,<partner-did-if-any>",
    "staticClaims":
"{\"company\":\"Contoso\",\"department\":\"IT\",\"title\":\"Employee\"}"
  }
```

TIP: For production deployments, set ManagedIdentity: true and use Azure Managed Identity instead of a client secret. Remove ClientSecret and CertificateName from the configuration when using MSI.

5.4 Deploy to Azure App Service

30. Publish the application using Visual Studio (Publish > Azure > App Service) or the Azure CLI: dotnet publish -c Release followed by az webapp deploy.
31. Configure App Service application settings to match appsettings.json (or use Azure Key Vault references).
32. Ensure the App Service URL is HTTPS — required for session cookies and Verified ID callback delivery.
33. For Managed Identity: assign the App Service's system-assigned managed identity Get and List access to the Key Vault secrets.

9 Step 6 — authID Proof Session Flow (Polling, No Webhook)

Unlike webhook-based integrations, this implementation uses client-side polling to determine when the user has completed the authID verification. The browser repeatedly calls GET /authid-status, which in turn polls the authID status API.

6.1 The /authid-verification-data Endpoint

This endpoint is called once when the user initiates verification. It performs the account creation and proof session creation, stores the operationId and accessToken in server-side session, and returns the authID URL to the browser:

```
GET /authid-verification-data

// Controller action steps:
// 1. Read firstName/lastName from session
// 2. Call authIdService.GetAccessTokenAsync()  → Basic Auth → Bearer token
// 3. Call authIdService.CreateAccountAsync()   → POST /v1/accounts
// 4. Call authIdService.ProofUserAsync()       → POST /v2/operations
// 5. Store accessToken + operationId in HttpContext.Session
// 6. Return JSON:
   { "success": true, "authIdUrl": "https://id-uat.authid.ai/?i=...&s=..." }
```

6.2 The /authid-status Polling Endpoint

The browser polls this endpoint every 5 seconds after launching the authID web component. The endpoint reads the operationId from session and calls the authID status API:

```
GET /authid-status

// Status check: GET
/AdministrationServiceRest/foreignOperations/documents/{operationId}
// Response: { "Status": <int> }
//   Status == 1  → APPROVED (verification complete)
//   Status != 1  → still in progress

// On Status == 1:
//   1. Call GetDocumentDetailsAsync() → GET /v2/operations/{operationId}/result
//   2. Extract liveness selfie (Payload.Data.CurrentFacialImage.Data)
//   3. Call VerifiedIdIssuanceService.CreateIssuanceRequestAsync()
//   4. Return: { success: true, qrCode: "<base64>", issuanceUrl: "<deep link>" }
// On Status != 1: return { success: false, status: <n>, message: "in progress" }
```

6.3 Document Result Extraction

When verification completes, the application retrieves the full result from the authID transaction service and extracts the following data:

Data Element	JSON Path in authID Response
Liveness Selfie Image	Payload.Data.CurrentFacialImage.Data (base64)
Document Face Photo	Payload.Data.Document.FacialImage or .FacialImage.Data
Biometric Match Result	Payload.Data.BiometricVerificationResult.Verified + MatchProbability
Liveness Detection	Payload.Data.LivenessDetectionResult.IsLive + Probability
Document Fields (array)	Payload.Data.Document.Data[].Key / .Value
First Name	Key: FirstName
Last Name	Key: LastName or PrimaryIdentifier or primaryID
Date of Birth	Key: DateOfBirth or dateOfBirth
Document Number	Key: documentNumber or DocumentNumber
Expiry Date	Key: DateOfExpiry or dateOfExpiry
Issuing Country	Key: IssuingStateOrOrganization or issuer
Document Type	Key: DocumentType
Overall Decision	padResult==PASS + BarcodeSecurity==PASS + mismatchMrzOcr==false + biometric match + liveness

6.4 Overall Verification Decision Logic

The application evaluates the following conditions to determine a PASS result — all must be true:

- Liveness Detection: LivenessDetectionResult.IsLive == true
- Biometric Match: Payload.Data.Matched == true (or BiometricVerificationResult.Verified == true)
- Document Liveness (PAD): Document.Data["padResult"] == "PASS"
- Barcode Security: Document.Data["BarcodeSecurity"] == "PASS"
- MRZ/OCR Consistency: Document.Data["mismatchMrzOcr"] != "true"

10 Step 7 — Verified ID Credential Issuance

Once authID returns Status == 1, the application immediately calls the Microsoft Entra Verified ID Request Service API to create an issuance request. The resulting QR code is returned to the browser and displayed inline within Step 1 of the verification UI.

7.1 Acquire an Entra Access Token (Client Credentials)

The VerifiedIdIssuanceService acquires a token using the OAuth 2.0 client credentials flow — no user interaction required:

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

grant_type=client_credentials
&client_id={VerifiedID:ClientId}
&client_secret={VerifiedID:ClientSecret}
&scope=3db474b9-6a0c-4840-96ac-1fceb342124f/.default
```

The access_token from the response is used as a Bearer token for all Verified ID API calls.

7.2 Create the Issuance Request

```
POST
https://verifiedid.did.msidentity.com/v1.0/verifiableCredentials/createIssuanceRequ
est
Authorization: Bearer <entra_access_token>
Content-Type: application/json
{
  "includeQRCode": true,
  "authority": "did:web:<your-did>",
  "registration": { "clientName": "Woodgrove Helpdesk" },
  "type": "Woodgrove",
  "manifest":
"https://verifiedid.did.msidentity.com/v1.0/tenants/{tenantId}/...manifest",
  "callback": {
    "url": "https://<your-app>/api/vc-issuance-callback",
    "state": "<guid>",
    "headers": {}
  },
  "claims": {
    "given_name": "Jane",
    "family_name": "Smith",
    "display_name": "Jane Smith",
    "mail": "jane.smith@contoso.com",
    "photo": "<base64url-encoded liveness selfie>"
  }
}
```

> **NOTE:** The photo claim must be in base64url format (replace + with -, / with _, remove = padding). The reference implementation converts the base64 selfie from authID before including it in the VC claims.

7.3 Issuance Response — QR Code & Deep Link

The Verified ID API response contains:

```
{
  "requestId": "<guid>",
  "url": "openid-vc://?request_uri=...",   // deep link for mobile
  "expiry": 1714000000,                     // Unix timestamp
  "qrCode": "..."  // QR code image
```

- Desktop: Display the qrCode image for the user to scan with Microsoft Authenticator
- Mobile: Provide the url as a deep link button to open Microsoft Authenticator directly
- The request expires in approximately 15 minutes — the UI should indicate the time remaining

7.4 VC Issuance Callback (Microsoft → App)

The reference implementation includes a POST /api/vc-issuance-callback endpoint that Microsoft Verified ID calls when the user claims the credential in Authenticator. The application stores the result in IMemoryCache and exposes it via GET /api/vc-issuance-status for the browser to poll:

```
// Callback payload from Microsoft:
{
  "requestId": "<guid>",
  "requestStatus": "issuance_successful",
  ...
```

The browser polls GET /api/vc-issuance-status and when claimed == true, it unlocks Steps 2 and 3 of the onboarding UI to indicate the credential has been accepted.

11 Step 8 — Credential Presentation (Alumni / Relying Party)

After a user has claimed their Verified ID credential, subsequent logins to relying party applications can request presentation of that credential via Microsoft Authenticator — no re-scanning of the government ID is required.

8.1 Initiate a Presentation Request

The browser calls GET /api/verifier/presentation-request, which creates a presentation request via the Verified ID API. The requestId is stored in session for callback correlation:

```
POST
https://verifiedid.did.msidentity.com/v1.0/verifiableCredentials/createPresentation
Request
Authorization: Bearer <entra_access_token>
{
  "includeQRCode": true,
  "authority": "did:web:<your-did>",
  "registration": { "clientName": "Your Application" },
  "requestedCredentials": [{
    "type": "Woodgrove",
    "purpose": "To verify your Woodgrove employee credential",
    "acceptedIssuers": ["did:web:<your-did>"],
    "configuration": {
      "validation": { "allowRevoked": false, "validateLinkedDomain": true }
    }
  }]
}
```

8.2 Accepted Issuers Configuration

The acceptedIssuers list in the presentation request controls which credential issuers are trusted. The reference implementation supports multiple issuers — configured in appsettings.json as a comma-separated list:

```
"acceptedIssuers": "did:web:your-did,did:web:partner-did-1,did:web:partner-did-2"
// Also supports IDEMIA: did:web:eu.did.idemia.io, did:web:us.did.idemia.io
```

> **NOTE:** The alumni verification flow (referer contains /alumni/) uses the FTE credential type (Woodgrove) and accepts only your own DID as the issuer. The onboarding verification flow also accepts credentials from configured IDV partner issuers.

8.3 Claims Mapping from Presented Credential

When the presentation_verified callback is received, the application maps VC claims to the internal WoodgroveCredential model using the claimsMapping rules from appsettings.json. The photo claim is stored in base64url format and decoded when displayed:

```
// WoodgroveCredential maps to:
//   firstName, lastName  → from VC claims via claimsMapping
//   photo                → base64url-decoded for display
//   company, department, title → from staticClaims in appsettings.json
//   documentNumber    → randomly generated employee number (WG-XXXXXX)
```

12 Configuration Reference

Configuration Key	Description
VerifiedID:ApiEndpoint	Verified ID Request Service base URL (fixed value)
VerifiedID:TenantId	Your Entra tenant GUID
VerifiedID:Authority	Token endpoint base (login.microsoftonline.com/{tenantId}/v2.0/)
VerifiedID:scope	Fixed: 3db474b9-6a0c-4840-96ac-1fceb342124f/.default
VerifiedID:ManagedIdentity	true = use Azure MSI in production; false = use ClientId/Secret
VerifiedID:ClientId	App registration client ID
VerifiedID:ClientSecret	App registration secret (use Key Vault in production)
VerifiedID:DidAuthority	Your tenant's DID (did:web:...)
VerifiedID:client_name	Display name shown in Authenticator during issuance
VerifiedID:FTECredentialType	VC type to issue (e.g. Woodgrove)
VerifiedID:CredentialType	VC type to request during presentation (e.g. VerifiedCredentialExpert)
VerifiedID:CredentialManifest	Full manifest URL from Verified ID portal
VerifiedID:sourcePhotoClaimName	Claim name for Face Check photo (set to empty to disable Face Check)
VerifiedID:matchConfidenceThreshold	Face Check confidence threshold (70 = default)
VerifiedID:acceptedIssuers	Comma-separated list of trusted issuer DIDs for presentation
VerifiedID:staticClaims	JSON object with static claims added to every issued credential
VerifiedID:{type}-{issuer}-claimsMapping	JSON mapping from VC claim names to WoodgroveCredential properties

13 Troubleshooting

Issue	Likely Cause	Resolution
authID token request returns 401	Incorrect username/password Base64 encoding	Verify credentials are username:password (colon-separated) before Base64 encoding; confirm credentials with authID support
POST /v1/accounts returns 409 repeatedly	Account exists but Proof operation fails on retry	Check if mismatchMrzOcr or padResult caused an earlier failure; create a new account with a different AccountNumber
POST /v2/operations returns 409 (not found)	Account was not created before initiating proof	Ensure CreateAccountAsync() succeeds (status 200 or 409-exists) before calling ProofUserAsync()
GET /authid-status returns success=false indefinitely	User abandoned the authID iframe before completing	Add a timeout in the UI (e.g. 5 minutes); show a 'Try again' button; the session expires after Timeout seconds
Document result has null FirstName/LastName	Document key casing varies by document type	The app tries multiple key variants (FirstName, LastName, PrimaryIdentifier, primaryID) — check raw Payload.Data.Document.Data array for the actual key names
Liveness selfie is empty in VC photo claim	CurrentFacialImage.Data missing in result	Check authID Console logs for the operationId; confirm the Proof workflow has liveness enabled; a failed liveness check may result in no selfie
createIssuanceRequest returns 401	Entra access token missing or expired	Confirm client_credentials token is being requested with correct scope; check that admin consent was granted for VerifiableCredential.Create.All

Issue	Likely Cause	Resolution
createIssuanceRequest returns 400 (manifest)	CredentialManifest URL is incorrect	Copy the exact manifest URL from Verified ID portal > your credential type; it must match the DidAuthority tenant
QR code shown but credential not appearing in Authenticator	Issuance request expired (15 min TTL)	Implement re-issuance on user request; show a countdown timer; or automatically re-call /authid-status to get a fresh QR
claimsMapping returns null — dashboard empty	Credential type + issuer DID key not found in config	The key format is {CredentialType}-{issuerDID}-claimsMapping; verify DID exactly matches the issuer in the presented credential
Session lost between /authid-verification-data and /authid-status	Session timeout (default 5 min idle)	Increase IdleTimeout in Program.cs AddSession(); ensure session cookie is HttpOnly and Secure in production
vc-issuance-callback not called by Microsoft	App is not publicly reachable (localhost)	Deploy to a public HTTPS endpoint; Microsoft Verified ID cannot reach localhost; use Azure App Service or ngrok for testing

Diagnostic Tools

- authID operation result: GET /AuthorizationServiceRest/v2/operations/{operationId}/result — inspect Payload.Data structure directly
- Verified ID service health: Entra admin center > Verified ID > Overview
- Application logs: the reference app logs all authID responses to Console.WriteLine for debugging — review App Service log stream
- Credential manifest validation: fetch the CredentialManifest URL in a browser — it should return a valid JSON contract

14 Security Considerations

Credential Management

- Store all secrets (authID password, Entra client secret) in Azure Key Vault. Use Key Vault references in App Service configuration rather than appsettings.json values.
- Enable ManagedIdentity: true in production — this eliminates the Entra client secret entirely and uses the App Service's system-assigned managed identity.
- Rotate the authID API password and Entra client secret at least every 12 months. Plan for rotation with zero downtime using secret versioning.

Session Security

- The authID operationId and accessToken are stored in server-side session (not in the browser). Session cookies are configured as HttpOnly and Secure.
- Increase the session idle timeout from the default 5 minutes if the verification flow may take longer — users who pause mid-flow will lose their session.
- The vc-issuance-callback caches the claim result in IMemoryCache with a 10-minute TTL — consider persistent cache (Azure Cache for Redis) in multi-instance deployments.

Callback Security

- The POST /api/vc-issuance-callback endpoint is [AllowAnonymous] — it must be accessible to Microsoft's Verified ID service. Validate the requestId and requestStatus fields before acting on the callback.
- For additional security, include a shared secret in the callback headers configuration and validate it in the controller.

Data Privacy

- authID processes biometric data and government ID images on your behalf. Review authID's Data Processing Agreement and ensure your privacy policy covers biometric data collection.
- The application does not persistently store government ID images or raw biometric data — only extracted text claims and the liveness selfie (included in the VC photo claim) are retained.
- The liveness selfie is embedded in the Verified ID credential as a base64url-encoded claim. Ensure your credential retention and revocation policies account for this.

Identity Assurance

- The authID GetForeignIDDocument operation with liveness detection and biometric match meets NIST SP 800-63A IAL2 identity assurance requirements.
- The overall PASS decision requires all five checks: biometric liveness, face match, document PAD liveness, barcode security, and MRZ/OCR consistency. Do not issue a Verified ID credential unless all checks pass.

15 Support & Resources

authID Support

Resource	Details
Technical Support	support@authid.ai	+1 (516) 274-8700
Developer Documentation	https://developer.authid.ai
API Reference	API Reference Link
Console (UAT)	https://id-uat.authid.ai/portal
Status Page	https://status.authid.ai

Microsoft Entra Verified ID Resources

Resource	URL
Verified ID Overview	https://learn.microsoft.com/entra/verified-id/
IDV Partner Gallery	https://learn.microsoft.com/entra/verified-id/idv-partners
IDV Partner Submission Form	https://aka.ms/VIDCertifiedPartnerForm
Request Service API (Issuance)	https://learn.microsoft.com/entra/verified-id/issuance-request-api
Request Service API (Presentation)	https://learn.microsoft.com/entra/verified-id/presentation-request-api
Entra Admin Center	https://entra.microsoft.com
Microsoft Authenticator	https://aka.ms/authenticator
Sample Apps (Microsoft)	https://github.com/Azure-Samples/active-directory-verifiable-credentials

Need Integration Support?

The authID Solutions Engineering team provides hands-on support for Microsoft Entra Verified ID deployments, including reference implementation walkthroughs, sandbox provisioning, and production go-live assistance.