

EMREX 2.0 Protocol: Technical Specification

Table of Contents

1. Introduction

2. Architecture

3. Protocol Flow

4. Security Mechanisms

5. Data Formats

6. Endpoints

7. Error Handling

8. Compliance

9. Impact of Migration to EMREX 2.0 for EMPs and EMCs

1. Institution Selection

2. Authorization Request with PKCE (RFC 7636)

3. Client Verification (RFC 7523)

4. User Authentication & Consent

5. Token Exchange (RFC 6749)

6. Data Retrieval with Signed Responses

Introduction

EMREX 2.0 is a **secure, OAuth2-based protocol** for exchanging educational results (e.g., grades, diplomas) between institutions. It replaces the legacy EMREX 1.0 POST/POST flow with a **modern Authorization Code Flow + PKCE**, ensuring:

- **End-to-end security** (RFC 6749, RFC 7636, RFC 7523).
- **Data integrity** via signed responses (RFC 7515).
- **GDPR compliance** through granular consent.
- **Interoperability** via standardized data formats (ELMO XML/JSON, PDF).

Key Components

Component	Role	Standards Used
EMC	EMREX Client (home institution). Initiates the flow.	OAuth2, PKCE, JWT
EMP	EMREX Provider (host institution). Hosts the results.	OAuth2, JWT, RFC 7515 (signatures)
EMREG	Central registry. Stores EMP/EMC metadata and public keys.	JSON, HTTPS
Student	End-user. Selects results and grants consent.	OAuth2 Redirect Flow

Architecture

EMREX 2.0 follows a **decentralized architecture** with three core components:

1. EMC (EMREX Client)

Frontend: UI for institution selection and result review.

Backend: Handles OAuth2 flow, token exchange, and signature verification.

New in 2.0:

PKCE (RFC 7636) for code interception protection.

JWT client authentication (RFC 7523) instead of shared secrets.

Signature verification for responses.
2. EMP (EMREX Provider)

Frontend: Student login and result selection UI.

Backend: OAuth2 Authorization Server + Resource Server.

Resource Server: Provides signed results in multiple formats.

New in 2.0:

Dynamic client registration via EMREG.

Signed responses (RFC 7515) for all data exchanges.
3. EMREG (Registry)

Central directory for:

EMP/EMC metadata (e.g., `redirect_uris`, public keys).

Supported data formats and endpoints.

Accessible via REST API (JSON responses).

Protocol Flow

1. Institution Selection

Purpose: Student selects the host institution (EMP) from a list.

Sequence:

1. Student logs into **EMC Frontend**.
2. **EMC Backend** fetches available EMPs from **EMREG**:

```
GET /emp-list HTTP/1.1
Host: emreg.eu
```

3. **EMREG** responds with a list of EMPs (JSON):

```
{
  "emp_list": [
    {
      "acronym": "DUO-NL",
      "country": "NL",
      "institutions": ["University of Amsterdam", "TU Delft"],
      "authorization_url": "https://emp.nl/oauth2/authorize",
      "token_url": "https://emp.nl/oauth2/token",
      "public_key": "-----BEGIN PUBLIC KEY-----",
      "supportedDataFormats": [
        ...
      ]
    }
  ]
}
```

4. **EMC Frontend** displays the list; student selects an EMP.

Standards:

- **RFC 8259**: JSON format for EMP metadata.
- **HTTPS**: Mandatory for all EMREG communications.

2. Authorization Request with PKCE (RFC 7636)

Purpose: Initiate a secure OAuth2 flow with PKCE to prevent code interception.

Sequence:

1. **EMC Backend** generates a **PKCE code verifier** and **challenge**:

```
code_verifier = BASE64URL(SHA256(random(32))); // RFC 7636
code_challenge = BASE64URL(SHA256(code_verifier));
```

- `code_verifier` is stored in the session.
- `code_challenge` is sent to the EMP.

2. **EMC Backend** prepares the authorization URL with:

- OAuth2 parameters (RFC 6749):
 - `response_type=code`
 - `client_id` (e.g., `emc-nl-uva`).
 - `redirect_uri` (must match EMREG registration).
 - `state` (CSRF protection).
 - `scope` (eg `user_info elmo`) - Optional
- PKCE parameters (RFC 7636):
 - `code_challenge` (SHA-256).
 - `code_challenge_method=S256`.

3. **EMC Frontend** redirects the student to the **EMP Authorization Endpoint**:

```
HTTP/1.1 302 Found
Location: https://emp.nl/oauth2/authorize?
  response_type=code &
  client_id=emc-nl-uva &
  redirect_uri=https%3A%2F%2Femc.uva.nl%2Fcallback &
  state=xYz123abc456def789 &
  code_challenge=E9Me1v6jU2Fj0q4A2Tes0YsX9jpwcKQV86Z5HnXKhtI &
  code_challenge_method=S256
```

Standards:

- **RFC 6749**: OAuth2 Authorization Code Flow.
- **RFC 7636**: PKCE for public clients.
- **RFC 6750**: state parameter for CSRF protection.

3. Client Verification (RFC 7523)

Purpose: Verify the EMC's identity using JWT assertions.

Sequence:

1. **EMP Backend** fetches the **EMC's metadata** from **EMREG**:

```
GET /clients/emc-nl-uva HTTP/1.1
Host: emreg.eu
Authorization: Bearer <EMP_API_TOKEN>
```

2. **EMREG** responds with:
 - EMC's **public key** (for JWT validation).
 - Registered **redirect_uris**. Example:

```
{
  "client_id": "emc-nl-uva",
  "redirect_uris": ["https://emc.uva.nl/callback"],
  "public_key": "-----BEGIN PUBLIC KEY-----..."
}
```

3. **EMP Backend** validates:
 - The **redirect_uri** matches the registered URI.
 - The EMC's identity via **client_id**.

Standards:

- **RFC 7523**: JWT-based client authentication.
- **RFC 7515**: JWT signature validation.

4. User Authentication & Consent

Purpose: Authenticate the student and obtain explicit consent for data sharing.

Sequence:

1. **EMP Frontend** prompts the student to log in (e.g., SAML, local credentials).
2. After authentication, **EMP Backend** retrieves the student's results.
3. **EMP Frontend** displays results for selection (e.g., courses, grades).
4. Student selects results and grants consent.
5. **EMP Backend** stores the selection with a **reference ID** (**ref_id**):

```
{
  "ref_id": "a1b2c3d4-5678-90ef-ghij-klmnopqrstuv",
  "student_id": "s123456",
  "selected_results": ["course1", "course2"],
  "consent_granted": true,
  "consent_timestamp": "2026-02-18T12:00:00Z"
}
```

6. **EMP Backend** generates an **authorization code** bound to:
 - **ref_id**, **client_id**, **redirect_uri**, and **code_challenge**.
7. **EMP Frontend** redirects the student back to the **EMC** with the code:

```
HTTP/1.1 302 Found
Location: https://emc.uva.nl/callback?
code=AUTH_CODE_123 &
state=xYz123abc456def789
```

5. Token Exchange (RFC 6749)

Purpose: Exchange the authorization code for an access token using JWT client authentication.

Sequence:

1. **EMC Backend** validates the callback (`state` , `code`).
2. Generates a **signed JWT** (RFC 7523) for client authentication:
 - **Header:**

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "emc-key-1"
}
```

- **Payload:**

```
{
  "iss": "emc-nl-uva",
  "sub": "emc-nl-uva",
  "aud": "https://emp.nl/oauth2/token",
  "jti": "unique-id-123",
  "iat": 1672531200,
  "exp": 1672531500,
  "nbf": 1672531200
}
```

3. Sends a **token request** to the **EMP Token Endpoint**:

```
POST /oauth2/token HTTP/1.1
Host: emp.nl
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code &
code=AUTH_CODE_123 &
redirect_uri=https%3A%2F%2Femc.uva.nl%2Fcallback &
code_verifier=CODE_VERIFIER &
client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer &
client_assertion=SIGNED_JWT
```

4. **EMP Backend**:
 - Validates the JWT signature (using EMC's public key from EMREG).
 - Validates `code_verifier` against `code_challenge` (PKCE).
 - Returns an **access token** (short-lived, ≤10 minutes):

```
{
  "access_token": "ACCESS_TOKEN_456",
  "token_type": "Bearer",
  "expires_in": 300,
  "scope": "elmo userinfo"
}
```

Standards:

- **RFC 6749**: OAuth2 Token Exchange.
- **RFC 7523**: JWT client authentication.
- **RFC 7636**: PKCE validation.

6. Data Retrieval with Signed Responses

Purpose: Fetch results in a secure, signed format.

Sequence:

1. **EMC Backend** requests data using the `access_token` :

```
GET /results?data_format=elmo&data_format_version=2.1 HTTP/1.1
Host: emp.nl
Authorization: Bearer ACCESS_TOKEN_456
X-Request-ID: req-12345
```

Supported `data_format` values:

Format	Content-Type	Description
elmo	application/xml	ELMO XML (default, based on EN 15981/15982).
elm	application/json	JSON alternative to ELMO.
userinfo	application/xml	Student identification (ELMO Learner).
eidas	application/json	eIDAS-compliant user info.
pdf_metadata	application/json	List of available PDFs (e.g., diplomas).
pdf	application/pdf	PDF document (requires <code>pdf_id</code>).

2. **EMP Resource Server:**

- Validates the `access_token` (scope, expiration, `ref_id` binding).
- Signs the response with its private key:

```
HTTP/1.1 200 OK
Content-Type: application/xml
X-Signature: EMP_SIGNATURE_ABC123
X-Signature-Algorithm: RS256
X-Request-ID: req-12345

<elmo>
  <learner>
    <id>s123456</id>
    <name>Jane Doe</name>
  </learner>
  ...
</elmo>
```

- Returns `HTTP 204 No Content` if no results are available.

3. **EMC Backend:**

- Verifies the `X-Signature` using the EMP's public key (from EMREG).
- Stores the results and notifies the student.

Standards:

- **RFC 7515:** JSON Web Signature (JWS) for response signing.
- **EN 15981/15982:** ELMO XML format.

Security Mechanisms

Mechanism	Purpose	Standard
OAuth2 Authorization Code Flow	Prevents token exposure in the frontend.	RFC 6749
PKCE	Mitigates authorization code interception (RFC 7636 §4.6).	RFC 7636
JWT Client Authentication	Secures server-to-server communication (replaces client secrets).	RFC 7523
Short-lived Tokens	Access tokens expire in ≤10 minutes.	RFC 6749 §5.1
Signed Responses	Ensures data integrity and authenticity (RFC 7515 §3).	RFC 7515

GDPR Protection (state) Mechanism	Purpose	Standard
	Prevents cross-site request forgery (RFC 6749 §10.12).	RFC 6749
EMREG Validation	EMP verifies EMC's <code>redirect_uri</code> and public key.	HTTPS + JSON
Granular Consent	Student selects specific results to share (GDPR compliance).	GDPR Art. 6

Example Data Formats

ELMO XML

- **Schema:** [ELMO XSD \(v2.1.2\)](#)
- **Description:** XML format based on **EN 15981/15982** for educational records (e.g., courses, grades, diplomas).
- **Headers:**

Header	Value Example	Description
<code>Content-Type</code>	<code>application/xml</code>	MIME type.
<code>X-Signature</code>	<code>EMP_SIGNATURE_ABC123</code>	Base64-encoded signature (RFC 7515).
<code>X-Signature-Alg</code>	<code>RS256</code>	Signature algorithm.

ELM

- **Schemas:** [ELM Schemas](#)
- **Description:** The European Learning Model (ELM) is a Data Model for Interoperability of Learning Opportunities, Qualifications, Accreditation and Credentials in Europe, developed by the European Commission.
- **Headers:**

Header	Value Example	Description
<code>Content-Type</code>	<code>application/json</code>	MIME type.
<code>X-Signature</code>	<code>EMP_SIGNATURE_ABC123</code>	Base64-encoded signature (RFC 7515).
<code>X-Signature-Alg</code>	<code>RS256</code>	Signature algorithm.

PDF Metadata

- **Schemas:** TBD
- **Description:** JSON list of available PDF documents (e.g., diplomas, transcripts) with metadata such as title, issue date, type, size, and checksum.
- **@TODO:** Define pdf-metadata schema
- **Headers:**

Header	Value Example	Description
<code>Content-Type</code>	<code>application/json</code>	MIME type.
<code>X-Signature</code>	<code>EMP_SIGNATURE_ABC123</code>	Base64-encoded signature (RFC 7515).
<code>X-Signature-Alg</code>	<code>RS256</code>	Signature algorithm.

- **Example:**

```
{
  "pdf_list": [
    {
      "pdf_id": "123-456-789",
      "title": "Bachelor's Diploma in Computer Science",
      "issue_date": "2023-07-15",
      "type": "diploma",
      "size_bytes": 102400,
      "checksum": "sha256:abc123..."
    }
  ]
}
```

PDF Document

- **Description:** Binary PDF file (e.g., diploma, transcript) with a detached signature in the `X-Signature` header.
- **Headers:**

Header	Value Example	Description
Content-Type	application/pdf	MIME type.
X-Signature	EMP_SIGNATURE_ABC123	Base64-encoded signature (RFC 7515).
X-Signature-Alg	RS256	Signature algorithm.

- **Example Request:**

```
GET /results?data_format=pdf&pdf_id=123-456-789 HTTP/1.1
Host: emp.com
Authorization: Bearer ACCESS_TOKEN_456
```

Userinfo

- **Description:** XML format based on **ELMO Learner** for student identification.
- **@TODO** Define a **seperate schema for userinfo**. **Json/xml or both?**
- **Schema:** [ELMO Learner XSD](#)
- **Headers:**

Header	Value Example	Description
Content-Type	application/xml	MIME type.
X-Signature	EMP_SIGNATURE_ABC123	Base64-encoded signature (RFC 7515).
X-Signature-Alg	RS256	Signature algorithm.

- **Example:**

```
<learner>
  <person>
    <family_name>Doe</family_name>
    <given_name>John</given_name>
    <birthdate>2000-01-01</birthdate>
  </person>
</learner>
```

EIDAS UserInfo

- **Description:** JSON format compliant with **eIDAS Regulation** for electronic identification, including attributes such as `sub`, `family_name`, `given_name`, `birthdate`, and eIDAS-specific metadata.
- **@TODO** Define our own EIDAS schema (all attributes from EIDAS schema, but only `familiy_name`, `given_name` and `birthdate` required.)
- **Schema:** [eIDAS Attributes](#)
- **Headers:**

Header	Value Example	Description

Content-Type Header	application/json Value Example	MIME type. Description
X-Signature	EMP_SIGNATURE_ABC123	Base64-encoded signature (RFC 7515).
X-Signature-Alg	RS256	Signature algorithm.

- **Example:**

```
{
  "sub": "eu.eidas.naturalperson.123456789",
  "family_name": "Doe",
  "given_name": "Jane",
  "birthdate": "1995-05-15",
  "person_identifier": "NL/BSN/123456789",
  "eidas": {
    "level_of_assurance": "substantial",
    "issuer": "https://eidas.nl/identity-provider"
  }
}
```

Extensions

In addition to the standard `data_formats` that an EMP registers in Emreg, an EMP can also register a URL to an OpenAPI specification (see the `extensionOpenApiSpec` field in Emreg). This OpenAPI specification defines additional resources that are specific to this EMP and are not part of the standardized Emreg `data_formats`. These extra resources can be accessed using the previously obtained Access Token.

Endpoints

Component	Endpoint	Method	Description	Authentication
EMREG	<code>/emp-list</code>	GET	List of available EMPs.	Bearer Token (EMC)
EMREG	<code>/clients/{client_id}</code>	GET	EMC metadata (public key, redirect URIs).	Bearer Token (EMP)
EMP	<code>/oauth2/authorize</code>	GET	Authorization endpoint (OAuth2).	None (user login required)
EMP	<code>/oauth2/token</code>	POST	Token endpoint (OAuth2).	JWT Assertion (RFC 7523)
EMP Resource Server	<code>/results</code>	GET	Fetch results in selected format.	Bearer Token (access_token)

Error Handling

Scenario	HTTP Status	Response Body	Recovery
Invalid <code>client_id</code>	400	<code>{"error": "invalid_client"}</code>	Register EMC in EMREG.
Mismatched <code>redirect_uri</code>	400	<code>{"error": "invalid_request", "error_description": "redirect_uri mismatch"}</code>	Update <code>redirect_uri</code> in EMREG.
Expired <code>code</code>	400	<code>{"error": "invalid_grant", "error_description": "code expired"}</code>	Restart the flow.
Invalid <code>code_verifier</code> (PKCE)	400	<code>{"error": "invalid_grant", "error_description": "PKCE verification failed"}</code>	Restart the flow.
Invalid JWT signature	401	<code>{"error": "invalid_client", "error_description": "JWT signature invalid"}</code>	Regenerate JWT with correct private key.
Unsupported		<code>{"error": "invalid_request", "error_description":</code>	See Emreg for EMP supported

data_format/version Scenario No results available	400 HTTP Status 204	"unsupported format/version"} Response Body (Empty body)	data formats Recovery Notify the student.
Token expired	401	{"error": "invalid_token", "error_description": "token expired"}	Request a new token.
Signature verification failed	403	{"error": "invalid_signature"}	Verify EMP's public key in EMREG.

Compliance

EMREX 2.0 complies with the following standards:

Category	Standard	Implementation Details
Authentication	RFC 6749 (OAuth2)	Authorization Code Flow + PKCE.
	RFC 7523 (JWT)	JWT client authentication (RS256).
	RFC 7636 (PKCE)	code_challenge with S256.
Security	RFC 7515 (JWS)	Signed responses (RS256).
	RFC 6749 §10.12	CSRF protection via state parameter.
	RFC 8259 (JSON)	ELM JSON and metadata responses.
Data Formats	EN 15981/15982	ELMO XML format.
Privacy	GDPR Article 6	Granular consent + explicit user approval.
Interoperability	HTTPS (RFC 2818)	Mandatory for all communications.
eIDAS	eIDAS Regulation	Support for eIDAS-compliant user identification (data_format=eidas).

Impact

9. Impact of Migration to EMREX 2.0 for EMPs and EMCs

The transition from **EMREX 1.0** to **EMREX 2.0** introduces significant architectural, security, and operational changes. This chapter outlines the key impacts for **EMPs** (**EMREX Providers**) and **EMCs** (**EMREX Clients**), including technical adjustments, security enhancements, and compliance considerations.

9.1. Key Changes in EMREX 2.0

Aspect	EMREX 1.0	EMREX 2.0	Impact
Authentication Flow	POST/POST with signed Elmo	OAuth2 + PKCE + JWT (RFC 6749, 7523, 7636)	High: Requires OAuth2 Authorization Server implementation.
Client Authentication	None	JWT Assertions (RFC 7523)	High: EMCs must generate and sign JWTs; EMPs must validate them. EMCs must register
Data Retrieval	Single ELMO XML response	Multiple formats (ELMO, ELM, PDF, EIDAS) + signed responses (RFC 7515)	Medium: EMPs must support new formats and signing; EMCs must verify signatures.
Security	HTTPS + ELMO signing	OAuth2 PKCE + JWT + Signed Responses	High: New security mechanisms require updates to token handling and validation logic.

EMREG Aspect	Static metadata (EMREX 1.0, acronym, URL, key)	Dynamic metadata (endpoints, data formats, OpenAPI extensions) (EMREX 2.0)	Medium Impact: EMPs/EMCs must register new fields (e.g., <code>authorizationUrl</code> , <code>extensionOpenapiSpec</code>).
Consent Management	Implicit (via session)	Explicit + granular (GDPR-compliant)	Medium: EMPs must implement consent storage and validation.
Error Handling	Custom error codes (e.g., <code>NCP_ERROR</code>)	Standardized OAuth2 error responses	Low: Update error handling to comply with RFC 6749.
Token Lifespan	NA	Short-lived access tokens (5 minutes)	Low: See token expiration

9.2. Impact for EMPs (EMREX Providers)

Technical Adjustments

1. OAuth2 Authorization Server:

- Implement **OAuth2 Authorization Code Flow + PKCE** (RFC 6749, RFC 7636).
- Support **JWT-based client authentication** (RFC 7523) to replace shared secrets.
- Example endpoints to implement:
 - `/oauth2/authorize` (for student login and consent).
 - `/oauth2/token` (for token exchange).
 - `/results` (resource server with signed responses).

2. Dynamic Registration in EMREG:

- Register new fields in EMREG:
 - `authorizationUrl`, `tokenUrl`, `resourceUrl`.
 - `supportedDataFormats` (e.g., `elmo`, `elm`, `pdf`).
 - `extensionOpenapiSpec` (for custom APIs).
- Example EMREG registration snippet:

```
{
  "emreg2Props": {
    "authorizationUrl": "https://emp.example.com/oauth2/authorize",
    "tokenUrl": "https://emp.example.com/oauth2/token",
    "resourceUrl": "https://emp.example.com/results",
    "supportedDataFormats": [
      { "name": "elmo", "version": "2.1" },
      { "name": "pdf", "version": "1.0" }
    ]
  }
}
```

3. Signed Responses:

- Sign all responses (ELMO, PDF, etc.) with the EMP's private key (RFC 7515).
- Include headers:
 - `X-Signature`: Base64-encoded signature.
 - `X-Signature-Algorithm`: e.g., `RS256`.

4. Consent Management:

- Store student consent explicitly (e.g., `ref_id`, selected results, timestamp).
- Bind consent to the `authorization_code` and `access_token`.

5. Legacy Support (Optional):

- Maintain a **dual-stack** (EMREX 1.0 + 2.0) during migration.
- Use EMREG to advertise support for both versions.

Security Enhancements

- Eliminate Shared Secrets**: Replace with JWT assertions (RFC 7523).
- PKCE Protection**: Mitigate authorization code interception.
- Short-Lived Tokens**: Reduce risk of token leakage (max 10-minute lifespan).
- Signature Verification**: Ensure data integrity for all responses.

Operational Impact

- Training**: Staff must be trained on OAuth2 flows and JWT handling.
- Monitoring**: New metrics for token usage, consent validation, and signature errors.
- Compliance**: Align with GDPR (granular consent) and eIDAS (trust levels).

9.3. Impact for EMCs (EMREX Clients)

Technical Adjustments

1. OAuth2 Client Implementation:

- Integrate PKCE (RFC 7636) for authorization requests.
- Generate and sign JWT assertions (RFC 7523) for token requests.
- Example token request:

```
POST /oauth2/token HTTP/1.1
Host: emp.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code &
code=AUTH_CODE_123 &
redirect_uri=https%3A%2F%2Femp.example.com%2Fcallback &
code_verifier=CODE_VERIFIER &
client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer &
client_assertion=SIGNED_JWT
```

2. Dynamic EMP Discovery:

- Fetch EMP metadata from EMREG (e.g., `authorizationUrl`, `supportedDataFormats`).
- Handle new data formats (e.g., `elm`, `eidas`) alongside ELMO.

3. Signature Verification:

- Verify X-Signature headers in EMP responses using the EMP’s public key (from EMREG).
- Example verification logic (pseudocode):

```
public boolean verifySignature(String response, String signature, String publicKey) {
    // Decode signature and verify using RS256
    return JWT.verify(response, signature, publicKey, "RS256");
}
```

4. Token Management:

- Handle short-lived tokens (≤ 10 minutes) with retry logic.
- Cache EMP public keys to avoid repeated EMREG lookups.

5. Legacy Fallback (Optional):

- Support both EMREX 1.0 (POST/POST) and 2.0 (OAuth2) during migration.
- Use feature flags to toggle between versions.

Security Enhancements

- **No Shared Secrets:** Use JWT assertions instead.
- **PKCE:** Protect against code interception attacks.
- **Signature Validation:** Ensure responses are tamper-proof.

Operational Impact

- **User Experience:** Students may notice a more explicit consent step.
- **Error Handling:** Update UI/UX for OAuth2 error responses (e.g., `invalid_grant`).
- **Logging:** Track token exchanges, signature failures, and consent issues.

9.4. Migration Checklist

For EMPs

Task	Status	Notes
Implement OAuth2 Authorization Server	☐	Use libraries like Spring Security OAuth2.
Register new endpoints in EMREG	☐	Update <code>emreg2Props</code> .
Add support for signed responses	☐	Use RFC 7515 (JWS).
Implement consent storage	☐	Store <code>ref_id</code> , selected results, timestamp.

Test with a pilot EMC Task	🔄 Status	Validate token exchange and data retrieval. Notes
Update documentation for students/staff	🔄	Explain new consent and login flows.

For EMCs

Task	Status	Notes
Integrate PKCE and JWT assertions	🔄	Use libraries like <code>oauthlib</code> (Python) or <code>nimbus-jose-jwt</code> (Java).
Update EMREG metadata fetch logic	🔄	Handle new fields (e.g., <code>authorizationUrl</code>).
Implement signature verification	🔄	Verify <code>X-Signature</code> headers.
Handle short-lived tokens	🔄	Add retry logic for expired tokens.
Test with a pilot EMP	🔄	Validate end-to-end flow.
Update student-facing instructions	🔄	Explain new consent and redirect steps.

9.5. Common Challenges and Mitigations

Challenge	Impact	Mitigation
OAuth2 Complexity	High development effort	Use existing libraries (e.g., Spring Security, <code>oauthlib</code>) and reference implementations.
Short-Lived Tokens	Increased API calls	Implement token caching and automatic retry logic.
Signature Verification Overhead	Performance impact	Cache EMP public keys and pre-validate signatures.
Consent Management	GDPR compliance risk	Store consent with timestamps and allow revocation.
Dual-Stack During Migration	Operational complexity	Use feature flags to toggle between EMREX 1.0 and 2.0.
EMREG Metadata Updates	Downtime risk	Schedule updates during low-traffic periods using <code>activeByDateTimeUtc</code> .

9.6. Timeline and Recommendations

Suggested Migration Timeline

1. Phase 1: Preparation (1-2 months)
 - Audit current EMREX 1.0 implementation.
 - Train teams on OAuth2, JWT, and PKCE.
 - Set up a staging environment with EMREX 2.0.
2. Phase 2: Development (2-3 months)
 - Implement OAuth2 flows (EMPs) or client logic (EMCs).
 - Integrate with EMREG for dynamic metadata.
 - Add signature verification and consent management.
3. Phase 3: Testing (1 month)
 - Test with pilot partners (e.g., 1 EMP + 1 EMC).
 - Validate end-to-end flows (authorization, token exchange, data retrieval).
 - Perform load testing for token refresh scenarios.
4. Phase 4: Go-Live (1 month)
 - Gradually roll out EMREX 2.0 alongside 1.0 (dual-stack).
 - Monitor for errors (e.g., token expiration, signature failures).

- Deprecate EMREX 1.0 after 6 months.

Recommendations

- **For EMPs:**
 - Start with a **minimal viable implementation** (e.g., support only ELMO + PKCE).
 - Use **OpenAPI** to document custom extensions (via `extensionOpenapiSpec`).
 - Monitor token usage and consent revocations for GDPR compliance.
 - **For EMCs:**
 - Prioritize **JWT assertion generation** and **signature verification**.
 - Cache EMP metadata (e.g., public keys, endpoints) to reduce EMREG calls.
 - Provide clear student instructions for the new consent flow.
 - **For Both:**
 - Join the **EMREX community** for shared tools and best practices.
 - Use the **EMREX 2.0 test suite** to validate compliance.
-

9.7. Backward Compatibility

EMREX 2.0 is **not fully backward-compatible** with EMREX 1.0 due to:

- **Protocol Changes:** POST/POST → OAuth2 + PKCE.
- **Security:** Shared secrets → JWT assertions.
- **Data Formats:** Single ELMO → Multiple formats + signing.