

Generic Web HSM Signing Protocol - Specification v1.0

Version: 1.0.3 **Date:** February 14, 2026

Table of Contents

1. [Introduction](#)
 2. [Architecture Overview](#)
 3. [Protocol Versioning](#)
 4. [Data Types](#)
 5. [Content Representation](#)
 6. [Object Structure](#)
 7. [Object Grouping](#)
 8. [Delivery & Callbacks](#)
 9. [Complete Request Schema](#)
 10. [Acknowledgment Response Schema](#)
 11. [Message Flow Diagrams](#)
 12. [Full Worked Examples](#)
 13. [Error Handling](#)
 14. [Security Requirements](#)
 15. [Glossary](#)
-

1. Introduction

1.1 Purpose

This document defines the **Generic Web HSM Signing Protocol v1.0**, a standardized communication protocol enabling web applications to request digital signatures from a Chrome browser extension, which in turn communicates with a native host application connected to hardware security modules (HSMs).

1.2 Scope

This protocol covers:

- Message formats between Web App ↔ Extension ↔ Native Host
- Content representation for various data types
- Delivery mechanisms for signed content
- Error handling and status reporting

1.3 Design Principles

1. **No Business Logic in Signing Stack:** The extension and native host are generic signing utilities. All business logic (what to sign, where to send) is defined by the calling web application.
2. **No Inline Binary:** PDF and binary content is NEVER transmitted inline. The native host fetches content from URLs provided by the caller.

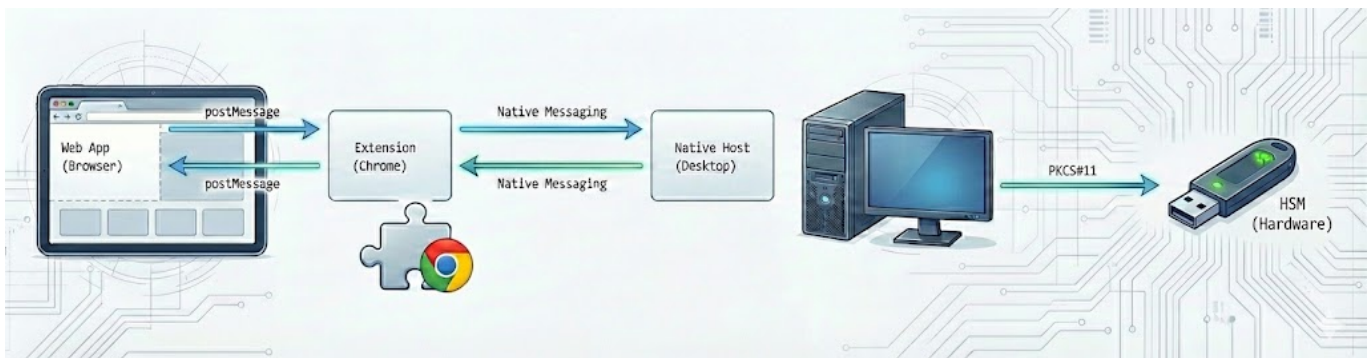
3. **Fire-and-Forget Acknowledgment:** The extension validates the request, forwards it to the native host, and immediately returns a synchronous acknowledgment ("**accepted**" or "**error**"). The web application does NOT wait for signing to complete.
4. **Callback-Based Result Delivery:** Signed content and status updates are delivered exclusively by the native host via HTTP callbacks (**onSuccess**, **onError**, **progress**) to the caller's backend. No signing results flow back through the extension.
5. **Hardware-Only Signing:** Only PKCS#11 hardware tokens are supported. No software certificates.

1.4 Roles

| Role | Description |
|-------------------------|--|
| Web Application | The calling application (e.g., a grades management system). Constructs signing requests and provides all URLs for content retrieval and result delivery. |
| Chrome Extension | Browser bridge. Receives requests via window.postMessage , validates origin, and forwards to native host via Chrome Native Messaging. |
| Native Host | Desktop application. Downloads content, performs HSM signing via PKCS#11, uploads results, and reports status. |
| HSM | Hardware Security Module containing the signing certificate and private key. |

2. Architecture Overview

2.1 Transport Mechanisms



2.2 Transport Details

| Leg | Transport | Format |
|-------------------------------|---------------------------------|----------------------------|
| Web App → Extension | window.postMessage() | JSON |
| Extension → Web App | window.postMessage() | JSON (acknowledgment only) |
| Extension → Native Host | Chrome Native Messaging (stdin) | JSON |
| Native Host → Content Server | HTTPS GET | Binary/Text |
| Native Host → Upload Server | HTTPS POST (raw bytes) | Binary/Text |
| Native Host → Callback Server | HTTPS POST | JSON |

Note (v1.0.3): The native host no longer sends a response back to the extension via stdout. Signing results are delivered exclusively through callbacks.

2.3 What Each Component Does

Web Application Responsibilities:

- Generate unique `requestId` (UUID)
- Construct complete signing request with all URLs
- Provide authentication headers for all external endpoints
- Handle acknowledgment from extension ("`accepted`" or "`error`")
- Implement callback endpoints (`onSuccess`, `onError`, optionally `progress`)
- Use callbacks as the sole source of signing results

Extension Responsibilities:

- Validate sender origin against allowlist
- Validate request schema
- Forward request to native host
- Return synchronous acknowledgment (accepted/error) to web app
- **Does NOT:** Wait for signing to complete, relay results, download content, modify payloads, or perform signing

Native Host Responsibilities:

- Download content from each provided `downloadUrl`
- Perform cryptographic signing via PKCS#11
- Upload signed content to provided `uploadUrl`
- POST status updates to callback endpoints (`onSuccess`, `onError`, optionally `progress`)
- Exit when processing is complete

3. Protocol Versioning

3.1 Version Field

Every request and response MUST include a `protocolVersion` field at the root level.

```
{
  "protocolVersion": "1.0",
  ...
}
```

3.2 Version Format

- Format: `MAJOR.MINOR` (e.g., "`1.0`", "`1.1`", "`2.0`")
- MAJOR: Incremented for breaking changes
- MINOR: Incremented for backward-compatible additions

3.3 Compatibility Rules

- Extension/Native Host MUST reject requests with unsupported `protocolVersion`
- Response MUST echo the same `protocolVersion` from the request

4. Data Types

4.1 Supported Types

The `dataType` field specifies the type of content being signed.

| dataType | Description | Content-Type | Allowed Modes |
|----------|-------------------------|--------------------------|--------------------|
| text | Plain UTF-8 text string | text/plain | inline*, remote |
| xml | UTF-8 XML document | application/xml | inline*, remote |
| json | UTF-8 JSON string | application/json | inline*, remote |
| pdf | PDF document (binary) | application/pdf | remote ONLY |
| binary | Opaque binary data | application/octet-stream | remote ONLY |

* only if the data to sign + JSON wrapper is under 1MB

4.2 Critical Rule: No Inline Binary

PDF and binary content MUST use mode: "remote" exclusively.

Inline Base64 encoding is **FORBIDDEN** because:

1. Base64 adds 33% overhead
2. Large payloads can exceed Chrome Native Messaging limits (~1MB)
3. `window.postMessage` with large payloads degrades browser performance

5. Content Representation

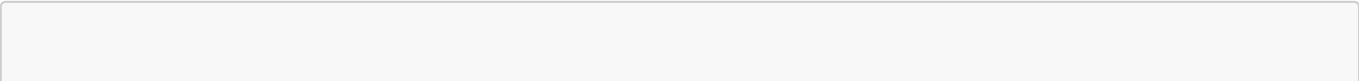
Note: This section describes the `content` object structure used in the **non-grouped objects array** (Section 6). For **grouped requests** using `objectGroups`, see Section 7 where `mode` is defined at the group level.

5.1 The `content` Object

When using the `objects` array (not `objectGroups`), every signable item has a `content` object describing how to access its data.

5.2 Inline Mode (Text/XML/JSON Only)

Use for small text content that can be embedded directly in the request.



```
{
  "mode": "inline",
  "encoding": "utf8",
  "content": "<the actual text content here>"
}
```

| Field | Type | Required | Description |
|----------|--------|----------|-------------------------|
| mode | string | YES | Must be "inline" |
| encoding | string | YES | Must be "utf8" |
| content | string | YES | The actual text content |

Example - Inline Text:

```
{
  "mode": "inline",
  "encoding": "utf8",
  "content": "185632|1500|2000|8|false|false"
}
```

Example - Inline XML:

```
{
  "mode": "inline",
  "encoding": "utf8",
  "content": "<?xml version=\"1.0\"?><root><data>value</data></root>"
}
```

5.3 Remote Mode (Required for PDF/Binary, Optional for Text)

Use for binary content or large text content fetched from a URL.

```
{
  "mode": "remote",
  "downloadUrl": "https://api.example.com/documents/123",
  "httpMethod": "GET",
  "headers": {
    "X-API-Key": "abc123",
    "Authorization": "Bearer token..."
  }
}
```

| Field | Type | Required | Description |
|-------|------|----------|-------------|
|-------|------|----------|-------------|

| Field | Type | Required | Description |
|-------------|--------|----------|---------------------------------|
| mode | string | YES | Must be "remote" |
| downloadUrl | string | YES | Full HTTPS URL to fetch content |
| httpMethod | string | NO | HTTP method (default: "GET") |
| headers | object | NO | HTTP headers for authentication |

5.4 Download URL Requirements

CRITICAL: The `downloadUrl` endpoint **MUST**:

- 1. Return **raw bytes** directly (not wrapped in JSON)
- 2. Set appropriate `Content-Type` header (e.g., `application/pdf`)
- 3. Support the specified `httpMethod` (usually GET)

CORRECT Implementation:

```
GET /api/documents/123
Authorization: Bearer token...

Response:
HTTP/1.1 200 OK
Content-Type: application/pdf
Content-Length: 45678

%PDF-1.4
... (raw PDF bytes)
```

INCORRECT Implementation (FORBIDDEN):

```
GET /api/documents/123

Response:
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 123,
  "value": "JVBERi0xLjQK..." <-- Base64 encoded - WRONG!
}
```

6. Object Structure

6.1 Single Object Format

Each item to be signed is represented as an "object" with the following structure:

```
{
  "id": "unique-identifier",
  "dataType": "...",
  "content": { ... },
  "pdfOptions": { ... },
  "xmlOptions": { ... },
  "upload": { ... },
  "callbacks": { ... }
}
```

6.2 Object Fields Reference

| Field | Type | Required | Description |
|------------|--------|----------|--|
| id | string | YES | Unique identifier for this object within the request |
| dataType | string | YES | One of: text, xml, json, pdf, binary |
| content | object | YES | Content definition (see Section 5) |
| pdfOptions | object | IF pdf | Required if dataType is "pdf" |
| xmlOptions | object | IF xml | Required if dataType is "xml" |
| upload | object | YES | Where to send signed content |
| callbacks | object | YES | Status notification endpoints |

6.3 PDF Options

Required when dataType is "pdf". Omit entirely for other types.

```
{
  "pdfOptions": {
    "label": "Student Grade Report"
  }
}
```

| Field | Type | Required | Description |
|-------|--------|----------|---|
| label | string | YES | Visible label text in the PDF signature |

6.4 XML Options

Required when dataType is "xml". Omit entirely for other types.

```
{
  "xmlOptions": {
    "xpath": "//Document/Signature",
    "idAttribute": "Id"
  }
}
```

| Field | Type | Required | Description |
|-------------|--------|----------|---------------------------------|
| xpath | string | YES | XPath to the signature location |
| idAttribute | string | NO | ID attribute name for reference |

7. Object Grouping

7.1 When to Use Grouping

MANDATORY when multiple objects share the same:

- `dataType`
- `callbacks` configuration
- `upload` configuration

Purpose: Reduce request size and enforce consistency.

7.2 Group Structure

7.2.1 Inline Mode (for text/xml/json)

When `mode` is `"inline"`, each object provides its own `content`:

```
{
  "objectGroups": [
    {
      "dataType": "text",
      "mode": "inline",
      "callbacks": { ... },
      "upload": { ... },
      "objects": [
        { "id": "grade-001", "content": { "encoding": "utf8", "value": "..." } },
        { "id": "grade-002", "content": { "encoding": "utf8", "value": "..." } }
      ]
    }
  ]
}
```

7.2.2 Remote Mode (REQUIRED for pdf/binary)

When `mode` is "remote", the `downloadUrl` is defined **once at the group level** with a **mandatory `<objectId>` placeholder**. Each object provides **only its `id`** — the native host constructs the final URL by substituting `<objectId>`.

```
{
  "objectGroups": [
    {
      "dataType": "pdf",
      "mode": "remote",
      "downloadUrl": "https://api.example.com/documents?id=<objectId>",
      "downloadHeaders": {
        "X-API-Key": "abc123"
      },
      "pdfOptions": { "label": "Official Document" },
      "callbacks": { ... },
      "upload": { ... },
      "objects": [
        { "id": "doc-001" },
        { "id": "doc-002" },
        { "id": "doc-003" }
      ]
    }
  ]
}
```

URL Construction Rule:

- The `<objectId>` placeholder in `downloadUrl` is **MANDATORY** when `mode` is "remote"
- The native host replaces `<objectId>` with each object's `id` value
- Example: `downloadUrl: "https://api.example.com/documents?id=<objectId>"` with `id: "doc-001"` becomes `https://api.example.com/documents?id=doc-001`

Why this design:

- Signing 500 PDFs no longer requires 500 separate URL definitions
- Reduces request payload size dramatically
- Enforces consistent URL patterns across a batch

7.3 Group Fields Reference

| Field | Type | Required | Description |
|--------------------------|--------|-----------|--|
| <code>dataType</code> | string | YES | Shared data type for all objects in group |
| <code>mode</code> | string | YES | "inline" or "remote" — determines how content is accessed |
| <code>downloadUrl</code> | string | IF remote | URL template with <code><objectId></code> placeholder (required when mode is remote) |

| Field | Type | Required | Description |
|-----------------|--------|----------|---|
| downloadHeaders | object | NO | HTTP headers for download requests (used when mode is remote) |
| pdfOptions | object | IF pdf | Shared PDF options (if dataType is pdf) |
| xmlOptions | object | IF xml | Shared XML options (if dataType is xml) |
| callbacks | object | YES | Shared callbacks configuration |
| upload | object | YES | Shared upload configuration |
| objects | array | YES | Array of objects (structure depends on mode — see 7.4) |

7.4 Object Within Group

The structure of each object depends on the group's mode:

When mode: "inline" (text/xml/json)

Each object provides id and content:

```
{
  "id": "grade-001",
  "content": {
    "encoding": "utf8",
    "value": "185632|1500|2000|8|false|false"
  }
}
```

| Field | Type | Required | Description |
|------------------|--------|----------|-----------------------------------|
| id | string | YES | Unique identifier for this object |
| content.encoding | string | YES | Must be "utf8" |
| content.value | string | YES | The actual text content to sign |

When mode: "remote" (pdf/binary, or large text)

Each object provides only id — the URL is constructed from the group's downloadUrl template:

```
{
  "id": "doc-001"
}
```

| Field | Type | Required | Description |
|-------|------|----------|-------------|
|-------|------|----------|-------------|

| Field | Type | Required | Description |
|-------|--------|----------|---|
| id | string | YES | Unique identifier — substituted into <objectId> placeholder |

7.5 Critical Rules

Rule 1: objects XOR objectGroups

A request MUST contain **either** `objects` OR `objectGroups`, **never both**.

| Scenario | Use |
|-----------------------------------|--|
| Single item | <code>objects</code> array with one element |
| Multiple items, different configs | <code>objects</code> array |
| Multiple items, same config | <code>objectGroups</code> array |
| Mixed configs | Multiple groups in <code>objectGroups</code> |

Rule 2: mode Determines Object Structure

| Group Mode | Object Contains | downloadUrl Location |
|------------|-----------------|---|
| "inline" | id + content | N/A |
| "remote" | id only | Group level (with <objectId> placeholder) |

Rule 3: PDF/Binary MUST Use Remote Mode

When `dataType` is "pdf" or "binary", `mode` MUST be "remote". Inline binary is forbidden.

8. Delivery and Callbacks

8.1 Upload Configuration

Defines where the native host sends signed content.

```
{
  "upload": {
    "uploadUrl": "https://api.example.com/signed-document?id=<objectId>",
    "httpMethod": "POST",
    "headers": {
      "X-API-Key": "abc123"
    },
    "signedContentType": "pdf"
  }
}
```

| Field | Type | Required | Description |
|-------------------|--------|----------|---|
| uploadUrl | string | YES | Full HTTPS URL for upload (with <objectId> placeholder) |
| httpMethod | string | NO | HTTP method (default: "POST") |
| headers | object | NO | HTTP headers for authentication |
| signedContentType | string | YES | Type of signed content (see below) |

URL Construction Rule:

- The <objectId> placeholder in uploadUrl is **MANDATORY**
- The native host replaces <objectId> with each object's id value
- Example: uploadUrl: "https://api.example.com/signed-document?id=<objectId>" with id: "report-001" becomes https://api.example.com/signed-document?id=report-001

8.2 Signed Content Types

| signedContentType | Description |
|-------------------|--------------------------------------|
| "string" | Signed hash/digest of text content |
| "pdf" | Signed PDF document bytes |
| "xml" | XML document with embedded signature |
| "binary" | Signed binary bytes |

8.3 Upload Request Format

The native host sends a **raw bytes POST** to the constructed URL:

```
POST /signed-document?id=report-001 HTTP/1.1
Host: api.example.com
Content-Type: application/pdf
X-API-Key: abc123
Content-Length: 125432

%PDF-1.4
... (signed PDF bytes)
```

Content-Type is determined by signedContentType:

| signedContentType | Content-Type Header |
|-------------------|--------------------------|
| "string" | text/plain |
| "pdf" | application/pdf |
| "xml" | application/xml |
| "binary" | application/octet-stream |

Backend Requirement: Your upload endpoint MUST:

- Accept the `id` query parameter to identify which object is being uploaded
- Accept raw bytes in the request body (not multipart)
- Return HTTP 2xx on success

8.4 Callback Configuration

Defines endpoints for status notifications. **Callbacks are the sole mechanism for delivering signing results** (v1.0.3). The extension returns only a synchronous acknowledgment; all actual outcomes (success, failure, progress) are reported by the native host directly to the caller's backend via these endpoints.

```
{
  "callbacks": {
    "onSuccess": "https://api.example.com/status/success",
    "onError": "https://api.example.com/status/error",
    "progress": "https://api.example.com/status/progress",
    "headers": {
      "X-API-Key": "abc123"
    }
  }
}
```

| Field | Type | Required | Description |
|------------------------|--------|----------|---|
| <code>onSuccess</code> | string | YES | URL called after successful signing + upload |
| <code>onError</code> | string | YES | URL called on any failure |
| <code>progress</code> | string | NO | URL for progress updates during signing. If omitted, the native host skips progress reporting with no impact on the signing workflow. |
| <code>headers</code> | object | NO | HTTP headers for callback requests |

Note: The request-level `metadata` object (see Section 9.4) is automatically echoed in all callback payloads. There is no separate metadata field in the callbacks configuration.

Note (v1.0.3 — `progress` is explicitly optional): Callers that do not need progress tracking MAY omit the `progress` field entirely. Implementing a stub endpoint is not required. The native host MUST NOT fail or alter its behavior when `progress` is absent — it simply skips progress reporting and proceeds normally.

8.5 Callback Payloads

Progress Callback (Native Host → Server)

```
POST /status/progress
Content-Type: application/json
X-API-Key: abc123
```

```
{
  "objectId": "report-001",
  "requestId": "550e8400-e29b-41d4-a716-446655440000",
  "status": "signing",
  "percentComplete": 50,
  "message": "Signing document...",
  "metadata": {
    "businessId": 12345,
    "courseCode": "CS101"
  }
}
```

| Field | Type | Description |
|-----------------|--------|--|
| objectId | string | ID of the object being processed |
| requestId | string | Request ID from original request |
| status | string | "signing" or "uploading" |
| percentComplete | number | 0-100 |
| message | string | Optional status message |
| metadata | object | Echoed from request-level metadata (Section 9.4) |

Success Callback (Native Host → Server)

```
POST /status/success
Content-Type: application/json
X-API-Key: abc123

{
  "objectId": "report-001",
  "requestId": "550e8400-e29b-41d4-a716-446655440000",
  "status": "completed",
  "uploadResult": {
    "statusCode": 200,
    "responseBody": "{\"message\":\"OK\"}"
  },
  "timestamp": "2026-01-20T10:30:00Z",
  "metadata": {
    "businessId": 12345,
    "courseCode": "CS101"
  }
}
```

| Field | Type | Description |
|----------|--------|----------------------------------|
| objectId | string | ID of the object that was signed |

| Field | Type | Description |
|--------------|--------|--|
| requestId | string | Request ID from original request |
| status | string | Always "completed" for success callbacks |
| uploadResult | object | Result from the upload endpoint (see below) |
| timestamp | string | ISO 8601 timestamp of completion |
| metadata | object | Echoed from request-level metadata (Section 9.4) |

uploadResult Object: The native host constructs this object by capturing the HTTP response from the upload endpoint.

| Field | Type | Description |
|--------------|--------|--|
| statusCode | number | HTTP status code returned by upload endpoint (e.g., 200, 201) |
| responseBody | string | Raw response body as string (JSON-stringified if the endpoint returned JSON) |

Error Callback (Native Host → Server)

```
POST /status/error
Content-Type: application/json
X-API-Key: abc123

{
  "objectId": "report-001",
  "requestId": "550e8400-e29b-41d4-a716-446655440000",
  "status": "failed",
  "error": {
    "code": "SIGN_FAILED",
    "message": "HSM communication timeout"
  },
  "timestamp": "2026-01-20T10:30:00Z",
  "metadata": {
    "businessId": 12345,
    "courseCode": "CS101"
  }
}
```

| Field | Type | Description |
|-----------|--------|-------------------------------------|
| objectId | string | ID of the object that failed |
| requestId | string | Request ID from original request |
| status | string | Always "failed" for error callbacks |
| error | object | Error details (see below) |

| Field | Type | Description |
|-----------|--------|--|
| timestamp | string | ISO 8601 timestamp of failure |
| metadata | object | Echoed from request-level metadata (Section 9.4) |

error Object: The native host constructs this object with details about what went wrong.

| Field | Type | Description |
|---------|--------|--|
| code | string | Error code from Section 13.1 (e.g., SIGN_FAILED, DOWNLOAD_FAILED, UPLOAD_FAILED) |
| message | string | Human-readable description of the error |

8.6 Callback Response Requirements

The callbacks endpoints MUST return HTTP 2xx. If the progress endpoint returns non-2xx, the native host:

- 1. Cancels signing for that object
- 2. Reports error with code PROGRESS_ENDPOINT_FAILED
- 3. Calls onError endpoint

9. Complete Request Schema

9.1 Request Structure (Web App → Extension)

```
{
  "protocolVersion": "1.0",
  "requestId": "<uuid>",
  "correlationId": "<optional-tracing-id>",
  "appId": "<caller-app-identifier>",
  "cert": {
    "certId": "<certificate-serial-or-thumbprint>",
    "label": "<optional-display-label>"
  },
  "metadata": {
    "<any>": "<caller-defined-context>"
  },
  "objects": [ ... ]
}
```

OR with grouping:

```
{
  "protocolVersion": "1.0",
  "requestId": "<uuid>",
  "correlationId": "<optional-tracing-id>",
  "appId": "<caller-app-identifier>",
```



```
"cert": {
  "certId": "<certificate-serial-or-thumbprint>",
  "label": "<optional-display-label>"
},
"metadata": {
  "<any>": "<caller-defined-context>"
},
"objectGroups": [ ... ]
}
```

9.2 Root Fields Reference

| Field | Type | Required | Description |
|-----------------|--------|----------|--|
| protocolVersion | string | YES | Must be "1.0" |
| requestId | string | YES | UUID generated by caller |
| correlationId | string | NO | Optional cross-system tracing ID |
| appId | string | YES | Identifier of the calling application |
| cert | object | YES | Certificate selection criteria |
| metadata | object | YES | Caller-defined context (can be empty {}) |
| objects | array | * | Array of objects (mutually exclusive with objectGroups) |
| objectGroups | array | * | Array of object groups (mutually exclusive with objects) |

* One of **objects** or **objectGroups** is required, but not both.

9.3 Certificate Selection

```
{
  "cert": {
    "certId": "ABC123456789",
    "label": "My Signing Certificate"
  }
}
```

| Field | Type | Required | Description |
|--------|--------|----------|---|
| certId | string | YES | Certificate serial number or thumbprint |
| label | string | NO | Display label for user prompts |

9.4 Metadata

Opaque object echoed in responses. Use for caller-specific context.

```
{
  "metadata": {
    "batchId": "batch-2026-001",
    "userId": "user@example.com",
    "department": "Engineering"
  }
}
```

10. Acknowledgment Response Schema

v1.0.3 Change: The extension now returns an immediate acknowledgment instead of waiting for signing to complete. Actual signing results are delivered exclusively through callbacks (Section 8).

10.1 Acknowledgment Structure (Extension → Web App)

```
{
  "protocolVersion": "1.0",
  "requestId": "<uuid-from-request>",
  "status": "accepted" | "error",
  "errors": [ ... ],
  "metadata": { ... }
}
```

10.2 Root Fields Reference

| Field | Type | Required | Description |
|-----------------|--------|----------|--|
| protocolVersion | string | YES | Echoed from request |
| requestId | string | YES | Echoed from request |
| status | string | YES | Acknowledgment status (see below) |
| errors | array | NO | Array of validation errors (only when status is "error") |
| metadata | object | YES | Echoed from request |

10.3 Status Values

| Status | Meaning |
|------------|---|
| "accepted" | Request is valid, forwarded to native host, and is being processed. Actual results will arrive via onSuccess / onError callbacks. |
| "error" | Request-level validation failure. The request was NOT forwarded to the native host. See errors array for details. |

10.4 Error Object

```
{
  "code": "BAD_REQUEST",
  "message": "Missing required field: protocolVersion"
}
```

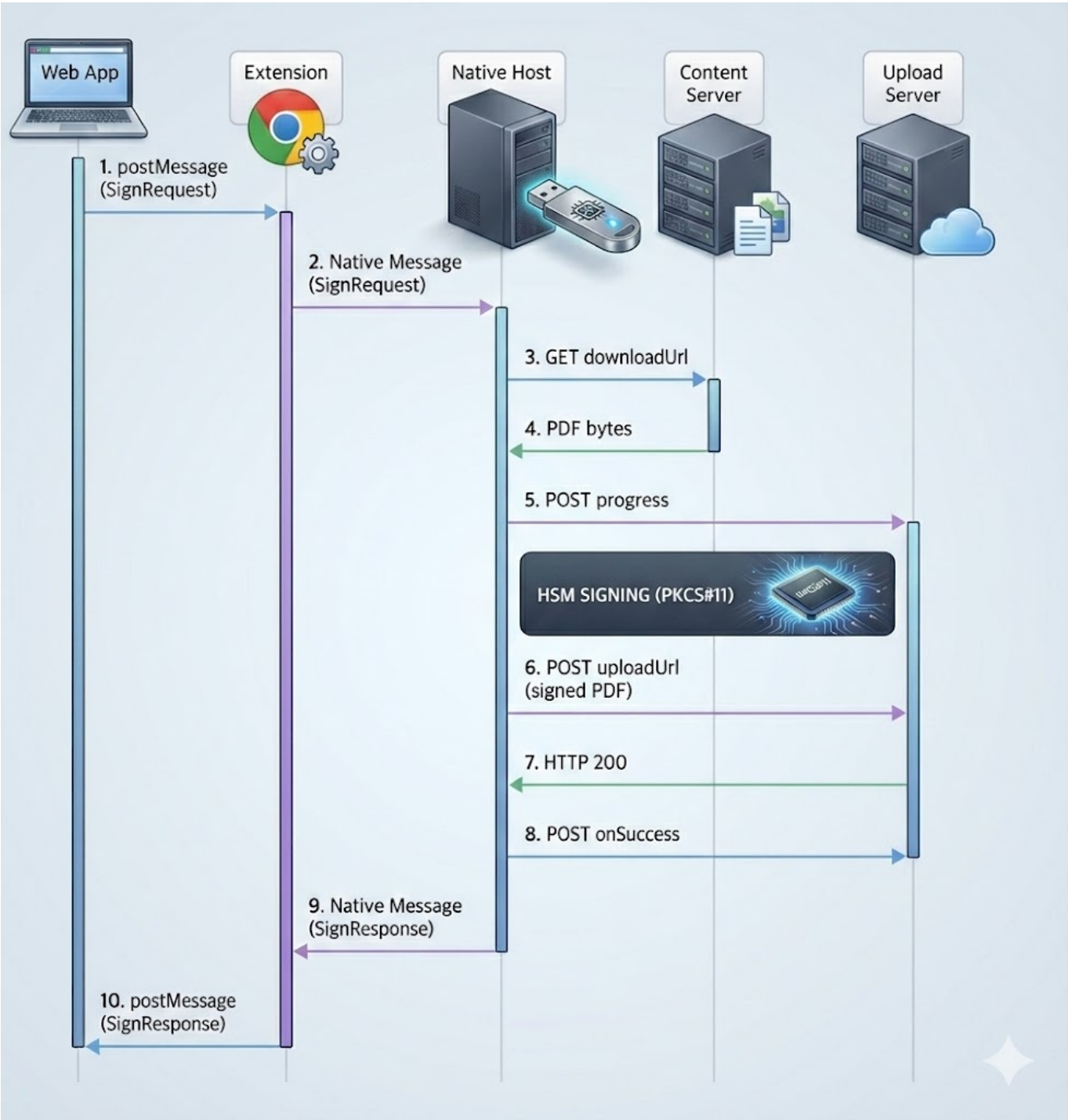
| Field | Type | Description |
|---------|--------|---|
| id | string | Object ID (omit for request-level errors) |
| code | string | Error code (see Section 13) |
| message | string | Human-readable description |

Important: Per-object errors (e.g., `DOWNLOAD_FAILED`, `SIGN_FAILED`, `UPLOAD_FAILED`) are no longer included in the acknowledgment. These are delivered by the native host to the caller's `onError` callback endpoint.

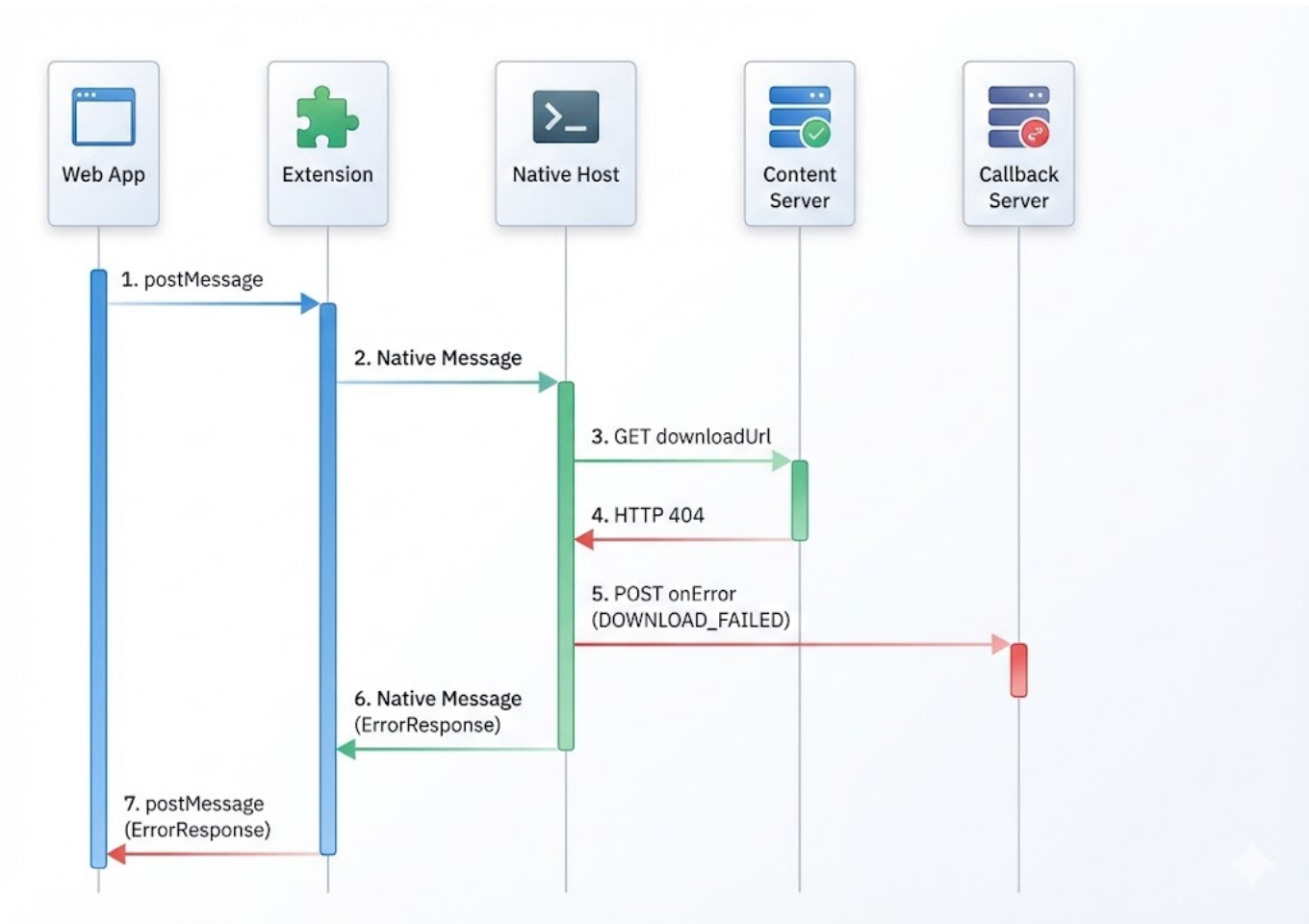
11. Message Flow Diagrams

v1.0.3 Change: The extension returns an acknowledgment immediately after forwarding the request. There is no response flow from native host → extension → web app. Signing results are delivered via callbacks.

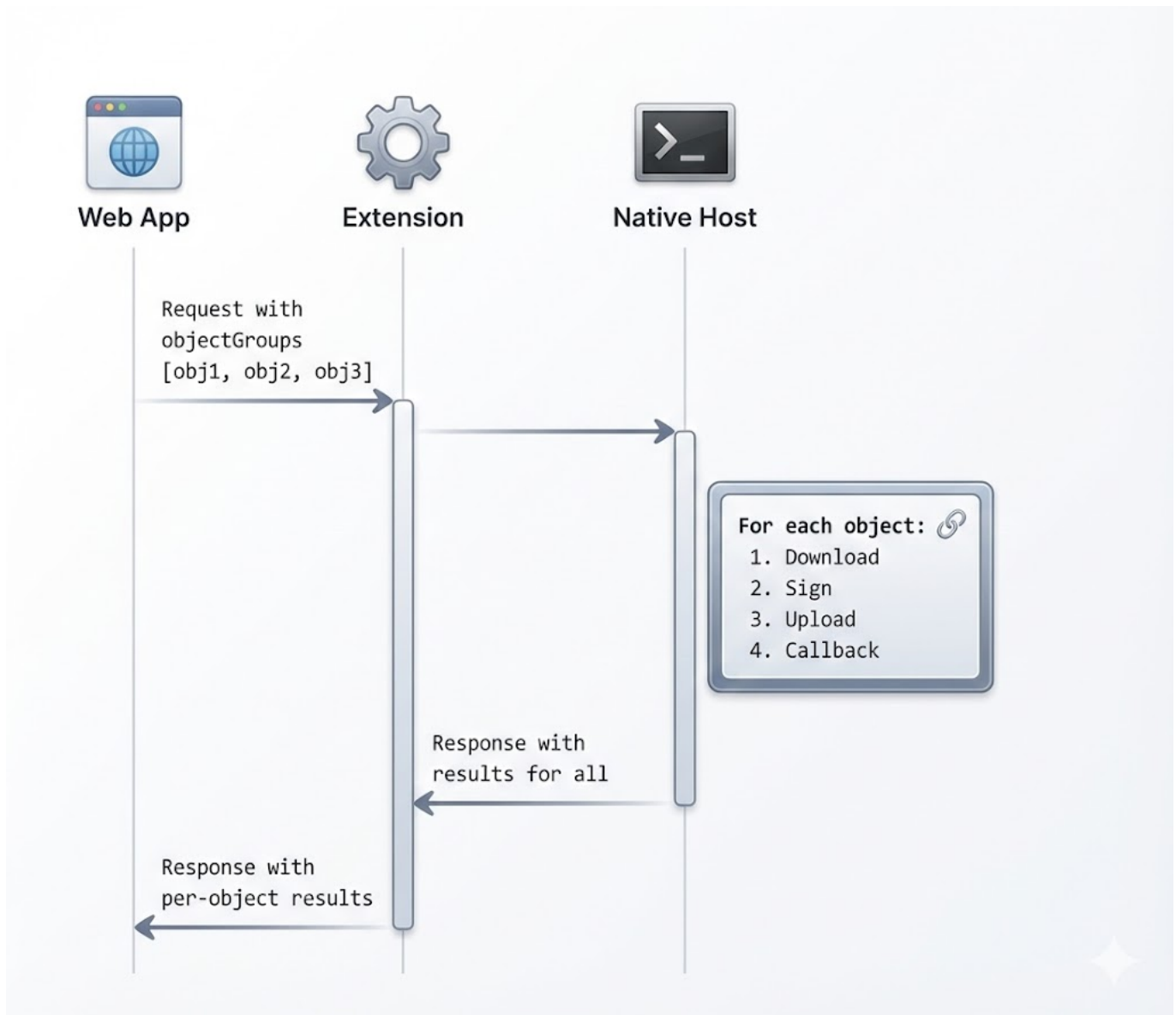
11.1 Successful Single PDF Signing Flow



11.2 Error Flow (Download Failed)



11.3 Batch Signing Flow (Multiple Objects)



12. Full Worked Examples

12.1 Example A: Sign a Single PDF Document

Scenario: A web application needs to sign a student report PDF.

Step 1: Web App Sends Request

```
// Web Application Code
const request = {
  protocolVersion: "1.0",
  requestId: "550e8400-e29b-41d4-a716-446655440000",
  appId: "student-portal",
  cert: {
    certId: "ABC123456789",
    label: "University Signing Certificate"
  },
  metadata: {
    studentId: "STU-2026-001",
```

```

    documentType: "grade-report"
  },
  objects: [
    {
      id: "report-001",
      dataType: "pdf",
      content: {
        mode: "remote",
        downloadUrl: "https://api.university.edu/reports/2026/STU-001/download",
        httpMethod: "GET",
        headers: {
          "X-API-Key": "university-api-key-12345",
          "Authorization": "Bearer eyJhbGciOiJIUzI1NiIs..."
        }
      },
      pdfOptions: {
        label: "Official Grade Report - University of Example"
      },
      upload: {
        uploadUrl: "https://api.university.edu/signed-document?id=<objectId>",
        httpMethod: "POST",
        headers: {
          "X-API-Key": "university-api-key-12345"
        },
        signedContentType: "pdf"
      },
      callbacks: {
        onSuccess: "https://api.university.edu/signing/status/success",
        onError: "https://api.university.edu/signing/status/error",
        progress: "https://api.university.edu/signing/status/progress",
        headers: {
          "X-API-Key": "university-api-key-12345"
        }
      }
    }
  ]
};

// Send to extension
window.postMessage({
  type: "HSM_SIGN_REQUEST",
  data: request
}, "*");

```

Step 2: Extension Validates and Returns Acknowledgment

The extension:

1. Checks `event.origin` against allowlist
2. Validates request schema
3. Forwards unchanged to native host via Chrome Native Messaging

4. **Immediately returns acknowledgment** ("accepted") to web app — does NOT wait for signing to complete

```
// Message sent to Native Host (stdin)
{
  "protocolVersion": "1.0",
  "requestId": "550e8400-e29b-41d4-a716-446655440000",
  "appId": "student-portal",
  "cert": {
    "certId": "ABC123456789",
    "label": "University Signing Certificate"
  },
  "metadata": {
    "studentId": "STU-2026-001",
    "documentType": "grade-report"
  },
  "objects": [
    {
      "id": "report-001",
      "dataType": "pdf",
      "content": {
        "mode": "remote",
        "downloadUrl": "https://api.university.edu/reports/2026/STU-001/download",
        "httpMethod": "GET",
        "headers": {
          "X-API-Key": "university-api-key-12345",
          "Authorization": "Bearer eyJhbGciOiJIUzI1NiIs..."
        }
      },
      "pdfOptions": {
        "label": "Official Grade Report - University of Example"
      },
      "upload": {
        "uploadUrl": "https://api.university.edu/signed-document?id=<objectId>",
        "httpMethod": "POST",
        "headers": {
          "X-API-Key": "university-api-key-12345"
        },
        "signedContentType": "pdf"
      },
      "callbacks": {
        "onSuccess": "https://api.university.edu/signing/status/success",
        "onError": "https://api.university.edu/signing/status/error",
        "progress": "https://api.university.edu/signing/status/progress",
        "headers": {
          "X-API-Key": "university-api-key-12345"
        }
      }
    }
  ]
}
```


Step 3: Native Host Downloads Content

```
GET /reports/2026/STU-001/download HTTP/1.1
Host: api.university.edu
X-API-Key: university-api-key-12345
Authorization: Bearer eyJhbGciOiJIUzI1NiIs...

---

HTTP/1.1 200 OK
Content-Type: application/pdf
Content-Length: 125432

%PDF-1.4
... (raw PDF bytes)
```

Step 4: Native Host Reports Progress

```
POST /signing/status/progress HTTP/1.1
Host: api.university.edu
Content-Type: application/json
X-API-Key: university-api-key-12345

{
  "objectId": "report-001",
  "requestId": "550e8400-e29b-41d4-a716-446655440000",
  "status": "signing",
  "percentComplete": 0,
  "message": "Starting signature process",
  "metadata": {
    "studentId": "STU-2026-001",
    "documentType": "grade-report"
  }
}

---

HTTP/1.1 200 OK
```

Step 5: Native Host Signs via PKCS#11

(Internal HSM operation - no network traffic)

Step 6: Native Host Uploads Signed Document

```
POST /signed-document?id=report-001 HTTP/1.1
Host: api.university.edu
```

```

Content-Type: application/pdf
X-API-Key: university-api-key-12345
Content-Length: 125432

%PDF-1.4
... (signed PDF bytes with embedded digital signature)

---

HTTP/1.1 200 OK
Content-Type: application/json

{"status": "received", "documentId": "DOC-2026-00123"}

```

Step 7: Native Host Calls Success Callback

```

POST /signing/status/success HTTP/1.1
Host: api.university.edu
Content-Type: application/json
X-API-Key: university-api-key-12345

{
  "objectId": "report-001",
  "requestId": "550e8400-e29b-41d4-a716-446655440000",
  "status": "completed",
  "uploadResult": {
    "statusCode": 200,
    "responseBody": "{\"status\": \"received\", \"documentId\": \"DOC-2026-00123\"}"
  },
  "timestamp": "2026-01-20T10:30:45Z",
  "metadata": {
    "studentId": "STU-2026-001",
    "documentType": "grade-report"
  }
}

```

Step 8: Extension Returns Acknowledgment to Web App

While steps 3-7 above are executing in the native host, the extension has **already** returned an acknowledgment to the web app (during Step 2):

```

{
  "protocolVersion": "1.0",
  "requestId": "550e8400-e29b-41d4-a716-446655440000",
  "status": "accepted",
  "metadata": {
    "studentId": "STU-2026-001",

```

```

    "documentType": "grade-report"
  }
}

```

Step 9: Web App Handles Acknowledgment

```

// Web App receives acknowledgment via postMessage listener
window.addEventListener("message", (event) => {
  if (event.data.type === "HSM_SIGN_RESPONSE") {
    const ack = event.data.data;

    if (ack.status === "accepted") {
      console.log("Request accepted – waiting for callbacks...");
      // The native host is processing asynchronously.
      // Actual results will arrive at the onSuccess/onError callback endpoints.
      // The web app's backend should handle callback payloads.
    } else if (ack.status === "error") {
      console.error("Request rejected:", ack.errors);
      // Request-level validation error (e.g., missing field, bad schema)
    }
  }
});

```

v1.0.3 Note: The web app does NOT receive signing results through `postMessage`. The success/error callback endpoints defined in the request are the sole delivery channel for results. This design ensures signing continues even if the user navigates away or refreshes the page.

12.2 Example B: Sign Multiple Text Items (Grades Batch)

Scenario: Sign 3 student grade strings in a single request.

Request (Web App → Extension)

```

{
  "protocolVersion": "1.0",
  "requestId": "661f9511-f3a0-42e5-b817-557766550001",
  "appId": "grades-system",
  "cert": {
    "certId": "CERT-GRADES-2026"
  },
  "metadata": {
    "batchId": "BATCH-2026-CS101-FINAL",
    "courseCode": "CS101",
    "semester": "2025-2026-S1",
    "professorId": "PROF-001"
  },
  "objectGroups": [

```

```

{
  "dataType": "text",
  "mode": "inline",
  "callbacks": {
    "onSuccess": "https://api.grades.edu/batches/BATCH-2026-CS101-FINAL/item-
success",
    "onError": "https://api.grades.edu/batches/BATCH-2026-CS101-FINAL/item-
error",
    "progress": "https://api.grades.edu/batches/BATCH-2026-CS101-FINAL/item-
progress",
    "headers": {
      "X-API-Key": "grades-api-key-67890",
      "X-Batch-Token": "batch-auth-token-xyz"
    }
  },
  "upload": {
    "uploadUrl": "https://api.grades.edu/signed-grade?id=<objectId>",
    "httpMethod": "POST",
    "headers": {
      "X-API-Key": "grades-api-key-67890",
      "X-Batch-Token": "batch-auth-token-xyz"
    },
    "signedContentType": "string"
  },
  "objects": [
    {
      "id": "grade-STU001",
      "content": {
        "encoding": "utf8",
        "value": "STU001|CS101|2026-S1|A|95|PROF-001|2026-01-15"
      }
    },
    {
      "id": "grade-STU002",
      "content": {
        "encoding": "utf8",
        "value": "STU002|CS101|2026-S1|B+|87|PROF-001|2026-01-15"
      }
    },
    {
      "id": "grade-STU003",
      "content": {
        "encoding": "utf8",
        "value": "STU003|CS101|2026-S1|A-|91|PROF-001|2026-01-15"
      }
    }
  ]
}

```

Key Points:

- `mode: "inline"` at group level — objects contain `id` + `content`
- All 3 grades share the same `callbacks` and `upload` configuration
- `uploadUrl` uses `<objectId>` placeholder — native host uploads to `https://api.grades.edu/signed-grade?id=grade-STU001`
- `signedContentType: "string"` indicates the upload will be a signature hash

Native Host Upload (Per Grade)

For grade-STU001:

```
POST /signed-grade?id=grade-STU001 HTTP/1.1
Host: api.grades.edu
Content-Type: text/plain
X-API-Key: grades-api-key-67890
X-Batch-Token: batch-auth-token-xyz

SHA256:a1b2c3d4e5f6...BASE64_SIGNATURE_HERE...
```

Response (Extension → Web App)

```
{
  "protocolVersion": "1.0",
  "requestId": "661f9511-f3a0-42e5-b817-557766550001",
  "status": "accepted",
  "metadata": {
    "batchId": "BATCH-2026-CS101-FINAL",
    "courseCode": "CS101",
    "semester": "2025-2026-S1",
    "professorId": "PROF-001"
  }
}
```

v1.0.3: The web app receives only the acknowledgment. Per-object results (`grade-STU001`, `grade-STU002`, `grade-STU003`) are delivered individually to the `onSuccess` callback endpoint as each grade is signed and uploaded.

12.3 Example C: Mixed Batch (PDFs + Text)

Scenario: Sign 2 PDF certificates and 3 text honor strings in one request.

```
{
  "protocolVersion": "1.0",
  "requestId": "772f0622-g4b1-53f6-c928-668877660002",
  "appId": "graduation-system",
  "cert": {
```

```

    "certId": "CERT-GRADUATION-2026"
  },
  "metadata": {
    "ceremony": "GRAD-2026-SPRING",
    "department": "Computer Science"
  },
  "objectGroups": [
    {
      "dataType": "pdf",
      "mode": "remote",
      "downloadUrl": "https://api.graduation.edu/certs/pdf?id=<objectId>",
      "downloadHeaders": {
        "X-API-Key": "grad-key-111"
      },
      "pdfOptions": {
        "label": "Official Graduation Certificate"
      },
      "callbacks": {
        "onSuccess": "https://api.graduation.edu/certs/signed",
        "onError": "https://api.graduation.edu/certs/error",
        "headers": { "X-API-Key": "grad-key-111" }
      },
      "upload": {
        "uploadUrl": "https://api.graduation.edu/signed-cert?id=<objectId>",
        "httpMethod": "POST",
        "headers": { "X-API-Key": "grad-key-111" },
        "signedContentType": "pdf"
      },
      "objects": [
        { "id": "cert-STU001" },
        { "id": "cert-STU002" }
      ]
    },
    {
      "dataType": "text",
      "mode": "inline",
      "callbacks": {
        "onSuccess": "https://api.graduation.edu/honors/signed",
        "onError": "https://api.graduation.edu/honors/error",
        "headers": { "X-API-Key": "grad-key-111" }
      },
      "upload": {
        "uploadUrl": "https://api.graduation.edu/signed-honors?id=<objectId>",
        "httpMethod": "POST",
        "headers": { "X-API-Key": "grad-key-111" },
        "signedContentType": "string"
      },
      "objects": [
        { "id": "honors-STU001", "content": { "encoding": "utf8", "value":
"STU001|SUMMA_CUM_LAUDE|3.95|2026" } },
        { "id": "honors-STU002", "content": { "encoding": "utf8", "value":
"STU002|MAGNA_CUM_LAUDE|3.82|2026" } },
        { "id": "honors-STU003", "content": { "encoding": "utf8", "value":
"STU003|CUM_LAUDE|3.65|2026" } }
      ]
    }
  ]
}

```

```

    ]
  }
]
}

```

Key Points:

- **PDF Group** (`mode: "remote"`):
 - `downloadUrl` at group level with `<objectId>` placeholder
 - Objects contain **only** `id` — no content definition needed
 - Native host constructs download URL: `https://api.graduation.edu/certs/pdf?id=cert-STU001`
 - Native host constructs upload URL: `https://api.graduation.edu/signed-cert?id=cert-STU001`
- **Text Group** (`mode: "inline"`):
 - Objects contain `id` + `content` with actual data
 - Native host constructs upload URL: `https://api.graduation.edu/signed-honors?id=honors-STU001`
- Each group has different `callbacks` and `upload` configurations
- `pdfOptions` only appears in the PDF group
- Request-level `metadata` (ceremony, department) is echoed in all callbacks

12.4 Example D: Partial Failure

Scenario: 3 documents requested, 1 fails to download.

Acknowledgment (Extension → Web App)

```

{
  "protocolVersion": "1.0",
  "requestId": "883g1733-h5c2-64g7-d039-779988770003",
  "status": "accepted",
  "metadata": { "batchId": "batch-123" }
}

```

Callbacks from Native Host

The native host calls `onSuccess` for `doc-001` and `doc-003`, and `onError` for `doc-002`:

```

POST /callback/error
{
  "objectId": "doc-002",
  "requestId": "883g1733-h5c2-64g7-d039-779988770003",
  "status": "failed",
  "error": {
    "code": "DOWNLOAD_FAILED",

```

```

    "message": "HTTP 404: Document not found at downloadUrl"
  },
  "timestamp": "2026-01-20T12:00:02Z",
  "metadata": { "batchId": "batch-123" }
}

```

v1.0.3: Partial failures are no longer reported in the acknowledgment ("**partial**" status is removed). Each object's outcome is reported individually through callbacks. The web app's backend is responsible for aggregating results and detecting partial failures.

12.5 Example E: Bulk PDF Signing (500 Documents)

Scenario: Sign 500 student report PDFs in a single efficient request.

```

{
  "protocolVersion": "1.0",
  "requestId": "994h2844-i6d3-75h8-e140-880099880004",
  "appId": "report-system",
  "cert": { "certId": "CERT-REPORTS-2026" },
  "metadata": {
    "batchId": "REPORTS-2026-SEMESTER1",
    "totalDocuments": 500
  },
  "objectGroups": [
    {
      "dataType": "pdf",
      "mode": "remote",
      "downloadUrl": "https://api.reports.edu/documents/download?id=<objectId>",
      "downloadHeaders": {
        "X-API-Key": "reports-api-key-99999",
        "Authorization": "Bearer batch-token-xyz"
      },
      "pdfOptions": {
        "label": "Official Student Report"
      },
      "callbacks": {
        "onSuccess": "https://api.reports.edu/batch/REPORTS-2026-SEMESTER1/success",
        "onError": "https://api.reports.edu/batch/REPORTS-2026-SEMESTER1/error",
        "progress": "https://api.reports.edu/batch/REPORTS-2026-SEMESTER1/progress",
        "headers": { "X-API-Key": "reports-api-key-99999" }
      },
      "upload": {
        "uploadUrl": "https://api.reports.edu/signed-document?id=<objectId>",
        "httpMethod": "POST",
        "headers": { "X-API-Key": "reports-api-key-99999" },
        "signedContentType": "pdf"
      },
      "objects": [

```



```
        { "id": "report-STU001" },
        { "id": "report-STU002" },
        { "id": "report-STU003" },
        { "id": "report-STU004" },
        { "id": "report-STU005" }
      ]
    }
  ]
}
```

Native Host Processing: For each object, the native host:

1. Constructs download URL: `https://api.reports.edu/documents/download?id=report-STU001`
2. Sends GET with `downloadHeaders`
3. Receives raw PDF bytes
4. Signs via PKCS#11
5. Constructs upload URL: `https://api.reports.edu/signed-document?id=report-STU001`
6. POSTs raw signed bytes with `Content-Type: application/pdf`
7. Calls appropriate callback

Backend API Requirements:

Your `/documents/download` endpoint MUST:

- Accept query parameter `id`
- Return raw PDF bytes (not JSON-wrapped)
- Set `Content-Type: application/pdf`

Your `/signed-document` endpoint MUST:

- Accept query parameter `id` to identify the document
- Accept POST with raw bytes in body (not multipart/form-data)
- Handle `Content-Type: application/pdf`

13. Error Handling

13.1 Error Codes Reference

| Code | HTTP Analog | Description | Recovery |
|----------------------------------|-------------|---------------------------|-----------------------|
| <code>BAD_REQUEST</code> | 400 | Invalid request schema | Fix request and retry |
| <code>UNSUPPORTED_VERSION</code> | 400 | Unknown protocolVersion | Use supported version |
| <code>UNSUPPORTED_TYPE</code> | 400 | Unknown dataType | Use valid dataType |
| <code>CERT_NOT_FOUND</code> | 404 | Certificate not available | Check certId |
| <code>DOWNLOAD_FAILED</code> | 502 | Failed to fetch content | Check downloadUrl |

| Code | HTTP Analog | Description | Recovery |
|--------------------------|-------------|----------------------------------|---------------------------|
| SIGN_FAILED | 500 | HSM signing operation failed | Check HSM status |
| UPLOAD_FAILED | 502 | Failed to upload signed content | Check uploadUrl |
| CALLBACK_FAILED | 502 | Failed to notify callback | Check callback URLs |
| PROGRESS_ENDPOINT_FAILED | 502 | Progress endpoint returned error | Check progress URL |
| TIMEOUT | 504 | Operation timed out | Retry or increase timeout |
| CANCELLED_BY_USER | 499 | User cancelled the operation | User action required |
| INTERNAL_ERROR | 500 | Unexpected internal error | Contact support |

13.2 Synchronous vs Asynchronous Errors (v1.0.3)

With the fire-and-forget model, errors are delivered through two channels:

Synchronous Errors (in acknowledgment)

Returned immediately by the extension in the `"error"` acknowledgment. These are **request-level validation failures** that prevent the request from being forwarded to the native host.

Applicable codes: `BAD_REQUEST`, `UNSUPPORTED_VERSION`, `INTERNAL_ERROR` (native host unavailable).

```
{
  "protocolVersion": "1.0",
  "requestId": "...",
  "status": "error",
  "errors": [
    {
      "code": "BAD_REQUEST",
      "message": "Missing required field: protocolVersion"
    }
  ],
  "metadata": {}
}
```

Asynchronous Errors (via callbacks)

Delivered by the native host to the `onError` callback endpoint during processing. These are **per-object runtime errors** encountered after the request was accepted.

Applicable codes: `CERT_NOT_FOUND`, `DOWNLOAD_FAILED`, `SIGN_FAILED`, `UPLOAD_FAILED`, `CALLBACK_FAILED`, `PROGRESS_ENDPOINT_FAILED`, `TIMEOUT`, `CANCELLED_BY_USER`.

```
POST /callback/error
{
  "objectId": "doc-002",
  "requestId": "...",
  "status": "failed",
  "error": {
    "code": "DOWNLOAD_FAILED",
    "message": "HTTP 404 from downloadUrl"
  },
  "timestamp": "2026-01-20T12:00:02Z",
  "metadata": { ... }
}
```

13.3 Request-Level vs Object-Level Errors

Request-Level Error (synchronous — returned in acknowledgment): The entire request is rejected before being forwarded to the native host.

```
{
  "status": "error",
  "errors": [
    {
      "code": "BAD_REQUEST",
      "message": "Missing required field: protocolVersion"
    }
  ]
}
```

Object-Level Error (asynchronous — delivered via `onError` callback): Individual objects fail during processing. Other objects in the same request may succeed.

```
POST /callback/error
{
  "objectId": "obj-2",
  "requestId": "...",
  "status": "failed",
  "error": {
    "code": "DOWNLOAD_FAILED",
    "message": "HTTP 404 from downloadUrl"
  },
  "timestamp": "...",
  "metadata": { ... }
}
```

14. Security Requirements

14.1 Transport Security

- **TLS Required:** All URLs (`downloadUrl`, `uploadUrl`, callbacks) MUST use HTTPS
- **Certificate Validation:** Native host MUST validate TLS certificates
- **No HTTP Fallback:** HTTP URLs MUST be rejected

14.2 Origin Validation

The extension MUST validate the sender origin:

```
// Extension content script
window.addEventListener("message", (event) => {
  const ALLOWED_ORIGINS = [
    "https://app.university.edu",
    "https://grades.university.edu"
  ];

  if (!ALLOWED_ORIGINS.includes(event.origin)) {
    console.error("Rejected request from unauthorized origin:", event.origin);
    return;
  }

  // Process request...
});
```

14.3 API Keys in Requests

Sending API keys via `postMessage` to the extension is **SAFE** because:

1. Chrome extension sandboxing isolates extension code
2. Malicious web content cannot intercept messages to extension background
3. Only the extension receives messages sent via `postMessage`

However, API keys MUST NOT be logged or included in error messages returned to the web app.

14.4 Content Integrity

- Content integrity is the **caller's responsibility**
- The native host does NOT verify content hashes
- If content integrity is critical, use signed URLs or implement application-level verification

14.5 Error Message Security

- Error messages returned to web app should be minimal
- Detailed errors should be logged locally by native host
- Never include secrets, tokens, or full URLs in error responses

15. Glossary

| Term | Definition |
|-------------------------|---|
| HSM | Hardware Security Module - physical device storing cryptographic keys |
| PKCS#11 | Cryptographic Token Interface Standard - API for HSM communication |
| Native Host | Desktop application communicating with Chrome via Native Messaging |
| Native Messaging | Chrome API for extensions to communicate with native applications |
| postMessage | Browser API for cross-origin communication between windows |
| Object | A single item to be signed (document, text string, etc.) |
| Object Group | Collection of objects sharing the same configuration |
| Callback | HTTP endpoint called by native host to report status |
| Upload | HTTP endpoint where native host sends signed content |
| Content | The actual data to be signed (accessed inline or via URL) |
| Remote Mode | Content is fetched from a URL by the native host |
| Inline Mode | Content is embedded directly in the request JSON |

Document History

| Version | Date | Author | Changes |
|---------|------------|------------|--|
| 1.0 | 2026-01-20 | Zeek Liviu | Initial release |
| 1.0.1 | 2026-01-22 | Zeek Liviu | Clarified Section 7: mode and downloadUrl at group level with <objectId> placeholder for efficient bulk PDF signing. Added Example E for 500-document batch. |
| 1.0.2 | 2026-01-23 | Zeek Liviu | Simplified Section 8: removed fieldName/fileName from upload, introduced <objectId> placeholder in uploadUrl , changed to raw bytes POST. Removed metadata from callbacks config (request-level metadata is now echoed in all callbacks). Added uploadResult and error object documentation. Removed redundant mimeType/fileName from content. Updated all examples. |
| 1.0.3 | 2026-02-14 | Zeek Liviu | Fire-and-forget acknowledgment model: Extension now returns an immediate "accepted" / "error" acknowledgment instead of waiting for signing to complete. Removed results , metrics , ResultObject , and "partial" / "ok" status from extension response. Signing results (success, error, progress) are delivered exclusively via callbacks. Reinforced progress callback as explicitly optional (no stub required). Updated Sections 1.3, 2.2, 2.3, 8.4, 10, 11, 12, 13. |

END OF SPECIFICATION