Generic Web HSM Signing Protocol (Draft)

A versioned, extensible protocol for web apps to request digital signatures from a Chrome extension and native host that talk to physical HSMs. It keeps app business logic outside the signing stack while supporting varied payload types (strings, PDFs, XML, binary) and large remote assets.

1. Roles & Transport

- **Web App → Extension**: JSON via `window.postMessage` only; the extension itself is the bridge. Origin allowlist enforced by the extension.
- **Extension → Native Host**: Chrome Native Messaging (stdin/stdout JSON). The extension validates/normalizes and forwards; it does not download content.
- **Native Host → HSM**: PKCS#11 only (hardware-backed certificates; no CSP/CNG, no software certs).

2. Versioning & Identifiers

- `protocolVersion` (e.g., `1.0`), required in every request and response.
- `requestId` (UUID) provided by the caller; echoed in all hops.
- `correlationId` (optional) for cross-system tracing.

3. Data Types

`dataType` is required per payload item (per field/document to sign):

- `text` (UTF-8 string)
- `xml` (UTF-8 XML)
- `json` (UTF-8 JSON string)
- `pdf` (application/pdf binary)
- `binary` (opaque bytes)

4. Content Representation

Each payload item describes how to access its content.

- **Text / XML / JSON**: Can be `inline` or `remote`.
  - Inline: `{ "mode": "inline", "encoding": "utf8", "content": "..." }`
- **PDF / Binary**: MUST use `mode: "remote"`. **Inline Base64 is forbidden** to minimize payloads and prevent overflows.
  - Remote fetch: `{ "mode": "remote", "downloadUrl": "https://...", "httpMethod": "GET", "headers": {"Authorization": "Bearer ...", "X-API-Key": "..."}, "mimeType": "application/pdf", "fileName": "report.pdf" }`

Notes:

- **No JSON wrapping**: The `downloadUrl` must return the **raw binary bytes** (e.g., `Content-Type: application/pdf`). Do not return a JSON object containing a Base64 string.
- API keys or auth tokens travel in `headers`; choose the header name that matches the protected endpoint.

- Content integrity is the responsibility of the caller; no hash verification is performed by the extension or native host.

5. Signing Options per Object

Signing profile (algorithms, formats) is fixed by the platform; callers cannot choose algorithms. Signing hints are provided **only if applicable**:

- `pdfOptions` (required if and only if `dataType: "pdf"`): `{ "label": "<visible label text>" }` (single signature, no page/rect selection exposed).
- `xmlOptions` (required if and only if `dataType: "xml"`): `{ "xpath": "//Root/Node", "idAttribute": "Id" }`.

Omit these fields entirely if the dataType does not require them.

6. Delivery & Callback (Mandatory)

Signed output is never returned inline. Each object **must** specify delivery endpoints. **Optimization Rule**: If multiple objects share the same callback/upload configuration, you **MUST** use the **Object Grouping** mechanism (Section 7) instead of repeating definitions.

- `callback`: `{ "onSuccess": "https://...", "onError": "https://...", "progress": "https://...", "payload": {...} }`
  - `progress`: route where the native host POSTs periodic updates during signing (see Section 6a).
  - `onSuccess`: route where the native host POSTs when signing **and** upload complete successfully.
  - `onError`: route where the native host POSTs if signing or upload fails at any point.
  - `payload`: arbitrary metadata provided by the caller; the native host includes it in all callbacks unchanged.
- `upload`: `{ "uploadUrl": "https://...", "httpMethod": "POST", "fieldName": "file", "fileName": "signed.pdf", "signedContentType": "pdf" }`
  - `fieldName`: form field name for the multipart upload (e.g., `"file"`).
  - `fileName`: file name to send in the multipart `Content-Disposition` header (e.g., `"report-1-signed.pdf"`). Can include `<objectId>` placeholder for grouping scenarios.
  - `signedContentType` (required; caller responsibility): indicates what will be sent:
    - `"string"`: hash digest of the signed text
    - `"pdf"`: signed PDF bytes
    - `"xml"`: modified XML structure with signature
    - `"binary"`: signed binary bytes
  - **Headers**: Use the `headers` object from the `content` object of each object; the native host includes them in the upload multipart request.

The native host streams results to `upload` and reports lifecycle via `callback`; no inline `signedContent` is returned.

6a. Progress Endpoint Specification

The native host POSTs to the `progress` endpoint periodically while signing:

```
```

```
{
  "payloadId": "<objectId>",
  "requestId": "<requestId>",
  "status": "signing" | "uploading",
  "percentComplete": 0-100,
  "message": "<optional status message>"
}
```

- **Headers**: Include the same `headers` from the object's `content` object.
- **Expected Response**: HTTP 200 (or 2xx). If the native host receives anything else, it cancels the signing process for that payload with `PROGRESS_ENDPOINT_FAILED` error and POSTs to `onError`.
- **Timing**: Progress is sent at least once at the start (0%) and end (100%), and optionally at intermediate points.
- **No guarantee of delivery**: The native host does not retry failed progress POSTs; it only cancels if a non-2xx response is received.

7. Object Grouping (Required for Bulk)

To reduce request size and enforce protocol efficiency, requests with multiple items sharing configuration **MUST** use `objectGroups`:

```
"objectGroups": [
  {
    "dataType": "text",
    "callback": { "onSuccess": "...", "onError": "...", "progress": "...",
"payload": {...} },
    "upload": { "uploadUrl": "...", "httpMethod": "POST", "fieldName": "file",
"fileName": "<objectId>", "signedContentType": "string" },
    "objects": [
      { "id": "p1", "content": { "mode": "inline", "encoding": "utf8", "content":
"..." } },
      { "id": "p2", "content": { "mode": "inline", "encoding": "utf8", "content":
"..." } }
    ]
  }
]
```

- `dataType` (and `pdfOptions`/`xmlOptions` if applicable) are shared across all items in the group.
- `callback` and `upload` are shared across all items in the group.
- Each item in `objects` has:
  - `id`: Unique identifier for the item (replaces `payloadId`).
  - `content`: content definition.
- `fileName` in `upload` should use `<objectId>` placeholder to generate unique filenames.

Note: A request must contain **either** `objects` (for distinct configs) **or** `objectGroups` (for shared configs), never both.

8. Metadata

`metadata` is a free-form object for caller-specific context. The extension/native app treats it as opaque but echoes it back in responses. Recommended keys: `appId`, `businessContext`, `userDisplayHint`.

9. Request Schema (Web App → Extension)

```
{
  "protocolVersion": "1.0",
  "requestId": "<uuid>",
  "correlationId": "<optional>",
  "appId": "<caller app id>",
  "cert": { "certId": "<serial or thumbprint>", "label": "<optional UI label>" },
  "metadata": { "any": "context provided by caller" },
  "objects": [
    {
      "id": "p1",
      "dataType": "pdf",
      "content": { "mode": "remote", "downloadUrl": "https://.../report/1.pdf",
"headers": {"X-API-Key": "<key>"}, "mimeType": "application/pdf" },
      "pdfOptions": { "label": "Student Report" },
      "upload": { "uploadUrl": "https://.../report/1/signed", "httpMethod":
"POST", "fieldName": "file", "fileName": "report-1-signed.pdf",
"signedContentType": "pdf" },
      "callback": { "onSuccess": "https://.../report/1/status", "onError":
"https://.../report/1/error", "progress": "https://.../report/1/progress",
"payload": { "businessType": 101 } }
    }
  ]
}
```

Notes on Request Structure:

- `appId` and `metadata` are **required** (metadata may be empty `{}`).
- A request must contain **either** `objects` **or** `objectGroups`, never both.
- `pdfOptions` is required if `dataType: "pdf"`, omitted otherwise.
- `xmlOptions` is required if `dataType: "xml"`, omitted otherwise.
- All headers sent to endpoints (progress, upload, callback) come from the `headers` object in the `content` property of each object.

10. Extension → Native Host Message

The extension validates the sender origin (see Security) and forwards a normalized message (no downloading performed by the extension):

```
{
  "protocolVersion": "1.0",
  "requestId": "<uuid>",
  "appId": "<caller app id>",
  "cert": { "certId": "..." },
  "metadata": { "any": "context provided by caller" },
  "objects": [
    {
      "id": "p1",
      "dataType": "pdf",
      "content": { "mode": "remote", "downloadUrl": "https://...", "headers": {"X-
API-Key": "<key>"}, "mimeType": "application/pdf" },
      "pdfOptions": { "label": "Student Report" },
      "upload": { "uploadUrl": "https://...", "httpMethod": "POST", "fieldName":
"file", "fileName": "report-1-signed.pdf", "signedContentType": "pdf" },
      "callback": { "onSuccess": "https://...", "onError": "https://...",
"progress": "https://...", "payload": { "businessType": 101 } }
    }
  ]
}
```

11. Native Host → Extension Response

```
{
  "protocolVersion": "1.0",
  "requestId": "<uuid>",
  "status": "ok" | "error" | "partial",
  "results": [
    {
      "id": "p1",
      "status": "ok",
      "uploadResult": { "statusCode": 200, "responseBody": "..." },
      "callbackResult": { "status": "sent", "endpoint": "onSuccess", "timestamp":
"2025-12-10T10:00:00Z" }
    }
  ],
  "errors": [ { "code": "SIGN_FAILED", "message": "...", "id": "p1" } ],
  "metrics": { "totalMs": 1234 }
}
```

12. Extension → Web App Response

The extension returns the native response (plus any extension-level errors) to the caller. Signed payload bytes are never included inline; delivery happens via the `upload` and `callback` endpoints.

13. Error Codes (suggested)

- `BAD_REQUEST` (validation), `UNSUPPORTED_TYPE`, `CERT_NOT_FOUND`, `DOWNLOAD_FAILED`, `HASH_MISMATCH`, `SIGN_FAILED`, `UPLOAD_FAILED`, `TIMEOUT`, `CANCELLED_BY_USER`, `INTERNAL_ERROR`.

14. Security Guidelines

- **Origin Allowlist**: Enforce origin allowlist in the extension; content scripts must check `event.origin` because manifest permissions alone do not block `window.postMessage` senders.
- **TLS Requirement**: Require TLS for all `downloadUrl`/`uploadUrl`.
- **API Keys in Requests**: Sending API keys in `postMessage` to the extension is **safe**. The Chrome Web Store isolates the extension from malicious web content; a content script cannot inspect or intercept messages sent to the extension background. The extension sandbox ensures only the extension receives those messages.
- **Content Integrity**: Content integrity is the caller's responsibility. The extension and native host assume that provided content is correct; no hash verification is performed.
- **Error Handling**: Do not echo secrets in errors; keep error details minimal to the web app, richer logs locally.
- **User Prompts**: Optionally prompt the user with a summary (who is asking, how many items, types).

15. Examples

A. Sign two inline strings (Grouped)

```
{
  "protocolVersion": "1.0",
  "requestId": "req-123",
  "appId": "simur3",
  "cert": { "certId": "SERIAL-ABC" },
  "metadata": { "batchId": "batch-001" },
  "objectGroups": [
    {
      "dataType": "text",
      "callback": {
        "onSuccess": "https://example.com/ok",
        "onError": "https://example.com/err",
        "progress": "https://example.com/prog",
        "payload": {"id": 1}
      },
      "upload": { "uploadUrl": "https://example.com/upload/1", "httpMethod": "POST", "fieldName": "file", "fileName": "<objectId>", "signedContentType": "string" },
      "objects": [
        { "id": "g1", "content": { "mode": "inline", "encoding": "utf8", "content": "185632|1500|2000|8|false|false" } },
        { "id": "g2", "content": { "mode": "inline", "encoding": "utf8", "content": "165612|2500|4000|0|true|false" } }
      ]
    }
  ]
}
```

B. Sign one PDF via remote download and upload result

```
{
  "protocolVersion": "1.0",
  "requestId": "req-456",
  "appId": "simur3",
  "cert": { "certId": "SERIAL-ABC" },
  "metadata": { "documentType": "report", "courseId": "CS101" },
  "objects": [
    {
      "id": "r1",
      "dataType": "pdf",
      "content": {
        "mode": "remote",
        "downloadUrl": "https://api.example.com/reports/1",
        "httpMethod": "GET",
        "headers": {"X-API-Key": "<key>"},
        "mimeType": "application/pdf",
        "fileName": "report-1.pdf"
      },
      "pdfOptions": { "label": "Report 1" },
      "upload": {
        "uploadUrl": "https://api.example.com/reports/1/signed",
        "httpMethod": "POST",
        "fieldName": "file",
        "fileName": "report-1-signed.pdf",
        "signedContentType": "pdf"
      },
      "callback": { "onSuccess": "https://api.example.com/reports/1/status",
"onError": "https://api.example.com/reports/1/error", "progress":
"https://api.example.com/reports/1/progress", "payload": { "businessType": 101 } }
    }
  ]
}
```

C. Sign multiple grades (grouped objects)

Instead of duplicating objects with identical options:

```
{
  "protocolVersion": "1.0",
  "requestId": "req-789",
  "appId": "simur3",
  "cert": { "certId": "SERIAL-XYZ" },
  "metadata": { "semester": "2025-S1", "department": "Engineering" },
  "objectGroups": [
    {
      "dataType": "text",
      "callback": {
        "onSuccess": "https://api.example.com/grades/signed",
        "onError": "https://api.example.com/grades/error",
        "progress": "https://api.example.com/grades/progress",
        "payload": { "batchId": "batch-001", "courseId": "CS101" }
      },
      "upload": {
        "uploadUrl": "https://api.example.com/grades/upload",
        "httpMethod": "POST",
        "headers": {"X-API-Key": "<key>"},
        "fieldName": "file",
        "fileName": "<objectId>",
        "signedContentType": "string"
      },
      "objects": [
        {
          "id": "grade-101",
          "content": { "mode": "inline", "encoding": "utf8", "content":
"185632|1500|2000|8|false|false" }
        },
        {
          "id": "grade-102",
          "content": { "mode": "inline", "encoding": "utf8", "content":
"165612|2500|4000|0|true|false" }
        },
        {
          "id": "grade-103",
          "content": { "mode": "inline", "encoding": "utf8", "content":
"175490|1800|3500|2|false|true" }
        }
      ]
    }
  ]
}
```