# **C2PA Sandbox — Product & Technical Spec (v2)**

## **1) Goals & Non-Goals**

**Goals**

* One-page, lightweight app to:
  + Generate EC keypair client-side and CSR
  + Request a C2PA certificate (server proxies your dev API)
  + Edit/replace the C2PA manifest (JSON editor + presets)
  + Sign an image using the obtained cert + private key
  + Embed TSA timestamp (staging TSA)
  + Verify the signed asset and show a human-readable report
  + Download: private key (PKCS#8), certificate (PEM), signed asset, manifest JSON
* Deployable on **DigitalOcean App Platform** or a **Droplet** with zero-ops setup.
* Friendly, branded, “preview/sandbox” feel; clear disclaimers.
* Footer with API documentation for

**Non-Goals (for v2)**

* Multi-tenant accounts, RBAC, audit trails, or billing.
* Long-term storage of user assets/keys on our servers.
* Heavy load / perf-testing (this is a preview).

## **2) User Stories (Happy Paths)**

1. **Issue & Sign**
   1. I drag an image (PNG/JPEG) into the app.
   2. I hit **Generate Keys & CSR** → my browser creates EC keypair + CSR.
   3. I hit **Request Certificate** → server proxy creates a cert (using your dev API), returns PEM.
   4. I open **Manifest Editor**, use a preset, tweak fields, ensure ta\_url points to staging TSA.
   5. I click **Sign Image** → browser embeds C2PA manifest with my cert and TSA timestamp.
   6. I see **Verification** pass and download my signed image, cert, private key, and manifest.
2. **Verify Only**
   1. I drag in a previously signed file.
   2. I click **Verify** → see claims, timestamp source, trust chain, warnings if any.
3. **Replace Manifest**
   1. I import an existing manifest JSON, tweak, and re-sign the original image.

## **3) UX / Branding**

**Single-page layout:** three stacked cards with tabs for advanced options.

* **Header**: SSL.com logo + “C2PA Preview Sandbox” badge. Short explainer/disclaimer.
* **Card A — Keys & Certificate**
  + Buttons: *Generate Keys & CSR*, *Request Certificate*, *Download Private Key*, *Download Cert*
  + Status pill (e.g., “Keys in memory only • not uploaded”)
* **Card B — Image & Manifest**
  + Drag-and-drop image zone (with small preview)
  + *Manifest Editor* (JSON textarea with live schema hints, formatting, presets dropdown)
  + *Sign Image* button
* **Card C — Verify**
  + *Verify* button (auto-runs post-sign)
  + Result panel: green/amber/red summary + expandable details (assertions, TSA, chain, warnings)
* **Footer**: Links to staging Trust Bundles (ECC/RSA) and TSA endpoints; “Preview environment — expect changes”.

**Visuals**

* Clean light theme; Tailwind + shadcn/ui; accessible contrast; minimal SSL.com styling (blue accents, rounded 2xl cards, subtle shadows).
* Copy tone: helpful, transparent, concise.

## **4) Architecture Overview**

**Front end**: Next.js (App Router) + React, Tailwind, shadcn/ui.  
 **Crypto & C2PA**:

* **WebCrypto** for EC P-256 keypair generation and PKCS#8 export (download).
* **CSR generation** in browser (pkijs or node-forge via WASM/bundled build).
* **C2PA sign/verify** via a browser-compatible C2PA library (e.g., c2pa-js + WASM).
  + Configure manifest’s ta\_url to the staging TSA.
  + If CORS blocks TSA, fall back to server proxy /api/tsa/\*.

**Backend (thin)**

* Next.js API routes:
  + /api/cert-requests → server-side call to your **dev C2PA issuance API** using **server-only** X-Account-ID + Authorization (kept in DO Secrets). Returns certificate PEM to client.
  + /api/tsa/\* (optional) → proxy TSA requests if CORS requires it; return raw bytes; no persistence.

**Storage**

* Keys and manifest live **in-memory** on the client. Optional: encrypted localStorage toggle (off by default). No server DB.

**Observability**

* Console + toast logs client-side.
* Server logs to stdout (captured by DO).
* Lightweight metrics (request counts, errors) using a tiny logger.

## **5) Security & Privacy**

* **Keys never leave the browser.** Private key is generated with WebCrypto and only exported for user download.
* **Server secrets** (Account ID, Bearer token, API base) live only in DO Secrets/Env. Never shipped to client.
* **No uploads stored** on our server; files processed in memory; optional DO Spaces can be enabled later.
* **CSP & Headers**: strict CSP, X-Content-Type-Options, Referrer-Policy, Permissions-Policy, SameSite cookies (if any).
* **Rate Limiting**: IP-based rate limit on /api/cert-requests and /api/tsa/\* (e.g., 30/minute) to protect preview infra.
* **Clear Disclaimers**: Preview only; do not use for production content; subject to change.

## **6) API Integrations**

### **6.1 Certificate Request (Server → Your Dev API)**

* **POST** /api/cert-requests
  + **Body (from client):** { csr: string (PEM), profileId?: string, conformingProductId?: string, subject?: { CN, O, C } }
  + **Server action:** Transform to your dev API schema; inject X-Account-ID, Authorization from env; POST to <https://api.c2patool.io/api/v1/certificate-requests>.
  + **Response to client:** { certificatePem: string, requestId?: string, meta?: any }
* **Notes**
  + Mask secrets in logs.
  + Handle 4xx/5xx with user-friendly messages.

### **6.2 TSA Timestamp (Optional Proxy)**

* **POST** /api/tsa/timestamp → proxies to <https://api.staging.c2pa.ssl.com/v1/timestamp> (ECC default).
  + Pass through content-type and binary body.
  + Strict size limits (e.g., 10 MB).

## **6.3 Developer & Sandbox Documentation**

### **Purpose**

This section enables developers, partners, and testers to experiment with SSL.com’s C2PA preview environment independently.  
 It outlines core endpoints, authentication requirements, and example payloads.

### **Base URLs**

|  |  |
| --- | --- |
| **Environment** | **Base URL** |
| **Preview (staging)** | <https://api.staging.c2pa.ssl.com/v1/> |
| **Production (future)** | <https://api.c2pa.ssl.com/v1/> |

### **Core Endpoints**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Method** | **Endpoint** | **Description** |
| **Create Certificate Request** | POST | /certificate-requests | Generates a new C2PA certificate request using a supplied CSR. |
| **Time Stamping (ECC)** | POST | /timestamp/ecc | Submits a signed manifest for ECC timestamping. |
| **Time Stamping (RSA)** | POST | /timestamp/rsa | Submits a signed manifest for RSA timestamping. |
| **Trust Bundle (ECC)** | GET | /repository/C2PA-ECC-TRUST-BUNDLE.pem | Retrieves the ECC trust anchor bundle. |
| **Trust Bundle (RSA)** | GET | /repository/C2PA-RSA-TRUST-BUNDLE.pem | Retrieves the RSA trust anchor bundle. |
| **Manifest Signing (via Tool)** | POST | /sign *(to be proxied through your sandbox tool)* | Signs uploaded image + manifest using issued certificate and TSA endpoint. |
| **Verify Manifest** | POST | /verify *(optional endpoint for downstream validation)* | Verifies that a manifest and its associated asset meet C2PA conformance. |

### **Authentication**

All requests must include:

Authorization: Bearer <API\_TOKEN>  
X-Account-ID: <ACCOUNT\_UUID>  
Content-Type: application/json

⚠️ **Do not** hard-code tokens in the frontend build; read them from environment variables or a secure proxy.

### **Example cURL – Certificate Issuance**

curl --location '<https://api.c2patool.io/api/v1/certificate-requests>' \  
--header 'X-Account-ID: fc23c1e2-f186-43f0-99ff-43b621dcff6e' \  
--header 'Authorization: Bearer <token>' \  
--header 'Content-Type: application/json' \  
--data '{  
 "certificate\_profile\_id": "6ba3b70c-38fe-44c3-803f-910c5873d1d6",  
 "certificate\_signing\_request": "<PEM\_CSR>",  
 "conforming\_product\_id": "f5ac57ef-428e-4a82-8852-7bde10b33060",  
 "experimental": {  
 "CN": "Demo Certificate CN",  
 "O": "SSL.com Corporation",  
 "C": "US"  
 }  
}'

### **Suggested Sandbox Flow**

1. **Upload Image → Generate CSR → Request Certificate**
2. **Edit or Replace Manifest Assertion**
3. **Sign Asset → Timestamp via ECC/RSA Endpoint**
4. **Download Signed Asset, Manifest, and Certificate**
5. **Optionally Verify** signature and manifest integrity.

### **Developer Notes**

* Respect preview-environment rate limits and traffic guidance.
* Use ECC endpoints for lighter payloads; RSA for compatibility testing.
* Validate signed outputs against the C2PA reference verifier.
* Report anomalies or interoperability issues via the SSL.com feedback channel.

## **7) Client Flows & State**

### **7.1 Generate Keys & CSR (Client)**

1. WebCrypto: ECDSA P-256, extractable private key for download.
2. Build CSR (pkijs/forge), Subject defaults:
   1. CN: “Demo Certificate CN”
   2. O: “SSL.com Corporation”
   3. C: “US”
3. Show success + enable **Request Certificate**.

### **7.2 Request Certificate (Client → Server)**

1. POST /api/cert-requests with CSR + optional profileId, conformingProductId, subject override.
2. Server talks to dev API; returns PEM.
3. Store PEM in client state; enable **Sign Image**.

### **7.3 Manifest Editing**

* JSON editor with:
  + **Presets**: “Basic Photo”, “Newsroom Photo”, “Marketing Asset”
  + Auto-insert "ta\_url": "<https://api.staging.c2pa.ssl.com/v1/timestamp>" (editable)
  + Schema hints + validate/format button
  + Import/Export manifest JSON

### **7.4 Sign Image (Client)**

1. User drags image (PNG/JPEG).
2. Use c2pa-js (or equivalent) to:
   1. Build C2PA claim from user manifest JSON.
   2. Use private key + cert PEM for signing.
   3. Include TSA timestamp via ta\_url (direct or via /api/tsa/timestamp if needed).
3. Return signed asset (same file type) for download.
4. Auto-trigger **Verify**.

### **7.5 Verify (Client)**

* Run verification on the signed asset.
* Show:
  + Signature OK / warnings
  + Claims list (title, creator, date, edits)
  + Timestamp status (TSA URL; time)
  + Chain / trust bundle reference (link to ECC/RSA bundles in staging)

## **8) Data Model (Client State)**

type KeyMaterial = {  
 privateKeyCrypto: CryptoKey; // in-memory  
 publicKeyCrypto: CryptoKey;  
 privateKeyPkcs8?: ArrayBuffer; // only for download; not stored by default  
};  
  
type Certificate = {  
 pem: string; // returned from server  
};  
  
type Manifest = {  
 json: any; // validated object  
};  
  
type Asset = {  
 original: File | ArrayBuffer;  
 signed?: ArrayBuffer;  
};  
  
type Verification = {  
 status: 'ok' | 'warning' | 'error';  
 details: any;  
};

## **9) Validation, Errors, and Edge Cases**

* **CSR errors**: show inline tips (CN/O/C missing, unsupported curve, malformed PEM).
* **Cert issuance errors**: surface upstream message + link “Try again later”.
* **Manifest JSON invalid**: block “Sign” until valid; “Format JSON” helper.
* **Large images**: client-side size limit (e.g., 15 MB); compress tip if exceeded.
* **TSA unreachable/CORS**: retry with proxy; if still fails, proceed without timestamp *only if* user explicitly accepts a warning (default: block).
* **Verification mismatch**: show red banner; offer to download raw report for debugging.

## **10) Deployment (DigitalOcean)**

### **Option A — App Platform (recommended)**

* Repo: single Next.js app.
* **Environment Variables (Build & Run)**:
  + C2PA\_API\_BASE=https://api.c2patool.io
  + C2PA\_ACCOUNT\_ID=... *(Secret)*
  + C2PA\_BEARER\_TOKEN=... *(Secret)*
  + C2PA\_TSA\_BASE=https://api.staging.c2pa.ssl.com/v1
  + C2PA\_TSA\_URL\_DEFAULT=https://api.staging.c2pa.ssl.com/v1/timestamp
  + NODE\_OPTIONS=--max\_old\_space\_size=1024
* Build command: next build
* Run command: next start
* Auto-HTTPS via DO.
* Set **scaling** to 1–3 instances; enable DO rate limiting if available.

### **Option B — Droplet**

* Ubuntu LTS, Node 20.x, PM2 or systemd service.
* Nginx reverse proxy with gzip + HTTP/2 + TLS.
* Same env vars via /etc/environment or systemd unit.

**Optional: DO Spaces** (disabled by default)

* Bucket for temporary file caching if needed; auto-purge job; never store private keys.

## **11) Testing & Acceptance Criteria**

### **1. Functional Testing**

|  |  |  |
| --- | --- | --- |
| **Area** | **Description** | **Acceptance Criteria** |
| **API Connectivity** | Verify frontend successfully connects to the C2PA Issuance and Signing APIs. | - All API endpoints return valid HTTP 2xx responses. - Certificate issuance, signing, and manifest modification requests execute successfully. |
| **File Handling** | Test image uploads (PNG, JPG) and manifest editing. | - User can upload images up to 20MB. - Manifest data can be viewed, edited, and resubmitted. - Signed image downloads without corruption. |
| **Certificate Management** | Verify certificate generation and download flow. | - Users can generate new certificates. - Private key and cert files download correctly. - No sensitive data persists after session end. |
| **Signing Process** | Validate that signing produces a C2PA-compliant signed image and manifest. | - Signed file includes expected C2PA metadata block. - Verification confirms provenance integrity. |
| **Time Stamping** | Confirm integration with SSL.com’s C2PA TSA endpoints (ECC & RSA). | - Timestamp requests complete successfully. - Returned manifests include ta\_url and valid timestamp signature. |

### **2. UI/UX Testing**

|  |  |  |
| --- | --- | --- |
| **Area** | **Description** | **Acceptance Criteria** |
| **Visual Branding** | Verify adherence to SSL.com design system (color palette, logo placement, typography). | - UI uses SSL.com primary blue, secondary silver/gray, and white backgrounds. - Fonts and logo placement conform to brand guide. - No unstyled elements present. |
| **Responsiveness** | Ensure layout functions on desktop, tablet, and mobile. | - App scales gracefully across breakpoints. - Buttons, modals, and text inputs remain legible and aligned. |
| **User Guidance** | Validate user flow clarity for upload → sign → verify → download. | - Each stage provides clear instructions and feedback. - Errors display inline and are human-readable. |
| **Accessibility** | Confirm keyboard navigation and ARIA labeling. | - All actionable elements are focusable. - Screen readers identify input labels and buttons. |

### **3. Performance & Reliability**

|  |  |  |
| --- | --- | --- |
| **Area** | **Description** | **Acceptance Criteria** |
| **API Latency** | Measure request times for issuance, signing, and TSA. | - Average round trip ≤ 3s for signing/issuance. - No request timeouts under normal conditions. |
| **Error Handling** | Validate resilience under invalid inputs and server errors. | - Clear user messages for 4xx/5xx responses. - System does not crash or hang. - Errors logged to console for debug mode. |
| **Deployment Readiness** | Verify compatibility with DigitalOcean droplet setup. | - Build scripts run with npm run build or yarn build. - Environment variables configurable via .env. - Reverse proxy (Nginx) correctly routes API requests. |

### **4. Security and Compliance**

|  |  |  |
| --- | --- | --- |
| **Area** | **Description** | **Acceptance Criteria** |
| **API Keys & Auth** | Ensure secure handling of bearer tokens and account IDs. | - No hardcoded keys in client bundle. - Tokens read from environment variables. - Frontend never exposes signing credentials. |
| **Data Privacy** | Validate temporary storage of uploaded assets. | - Uploaded files stored only in browser memory (not persisted). - Session cleared on reload/logout. |
| **C2PA Compliance** | Verify manifest conformance and trust chain validation. | - Output passes validation against C2PA reference verifier. - TSA URLs resolve to SSL.com endpoints. - Trust bundle validation passes. |

### **5. Acceptance for Staging Release**

* All critical paths (upload, sign, verify, download) pass functional testing.
* UI matches SSL.com branding and loads in <3s.
* DigitalOcean deployment verified with HTTPS endpoint.
* API responses verified with both ECC and RSA timestamp endpoints.
* Core security checklist (API key handling, CORS, CSP) validated.
* Documentation updated with endpoint usage and setup steps.
* QA signoff and demo to CTO/CEO for preview launch.