● LIVE ON BASE MAINNET · OPEN STANDARD · MIT

DCS Verify

Cryptographic credentials anyone can verify — without trusting the issuer's servers.

Technical whitepaper v1.0. Covers the R+2 receipt format, canonicalization and content addressing, Ed25519 signing, Bitstring status lists, Base mainnet anchoring, the four-state verification algorithm, inbound W3C Verifiable Credentials and Open Badges 3.0, the trust registry, and complete usage guides for verifiers, issuers, and developers — including how to re-verify everything independently with no DCS account.

CONTENTS

1 · Design principles & system overview

Most credential systems answer “is this real?” with “trust our server.” DCS Verify is built so the answer is a mathematical proof the holder carries with them. The issuing service can disappear and every credential it ever issued remains verifiable.

- **Self-contained proof**
  Every credential is an R+2 receipt that carries its own content ID, public key, and Ed25519 signature. Verification needs the receipt — not the issuer's cooperation.
- **Verify is free, public, account-free**
  No login, no API key, no rate-wall for verification. Trust that costs money to check is not trust.
- **Honest by construction**
  Network statistics come from the database or display as absent. Unsupported formats return “not yet verifiable,” never a fake “valid.” Advisory signals (AI, fraud) are never blended into the cryptographic verdict.
- **Open standard**
  R+2 is published under MIT with conformance vectors (github.com/DCS-LabsAI/r2-standard). The reference verifier runs in any browser or Node runtime.

SYSTEM OVERVIEW

```
  ISSUER                       DCS VERIFY API                      ANYONE
  ------                       --------------                      ------
  POST /v1/credentials/issue   sign (Ed25519, server-side)         receipt / QR / link
        subject JSON   ----->  cid = sha256(jcs(unsigned))  ----->   |
                               store verbatim (jsonb)                | POST /v1/credentials/verify
                               anchor on Base (optional)             | or recompute locally:
                               status-list slot (optional)           |   jcs -> sha256 -> ed25519.verify
                                                                     v
                               public status list <----------- read bit, check anchor on Basescan
```

Components: a Node.js (ESM) API on Railway at api.dcslabs.ai; Postgres (Supabase) storing receipts verbatim; Cloudflare Pages serving the public site and PWA at verify.dcslabs.ai; and the Base mainnet for anchors. The only secret material is the issuer signing key and a gas-only anchor wallet key, both held as server environment variables and never present in the repository, the database, or any response.

2 · The R+2 receipt: format, canonicalization, content ID

The receipt is the atomic unit. It is a JSON object in two parts: the unsigned form (everything that is signed) and the signature materials over it.

RECEIPT STRUCTURE

```
{
  "id": "<uuid>",                          // receipt id (lookup key)
  "cid": "sha256:<64 hex>",                // content id over the UNSIGNED form
  "issuer": "dcslabs",                     // registry issuer id
  "issuer_did": "did:web:dcslabs.ai",
  "credential_type": "certificate",        // certificate|result|license|document|preservation_receipt
  "subject": { ... },                      // the claims being attested
  "validity": { "valid_from": "...", "valid_until": "..."|null },
  "status": { "status_list_id": "...", "status_list_index": N,
              "status_purpose": "revocation" } | null,
  "previous_credential": "sha256:..."|null, // hash-chain to the version it renews
  "render_template": "...", "issued_at": "<ISO 8601>",
  "pubkey": "<base64 raw Ed25519>",         // key the signature verifies against
  "signature": "<base64>",                  // Ed25519 over utf8(jcs(unsigned))
  "canonical_alg": "RFC8785-JCS",
  "base_tx": "0x..."|null, "anchored_at": "..."|null
}
```

CANONICALIZATION (RFC 8785 JCS)

JSON has no single byte representation — key order, whitespace, and number formatting all vary. Before hashing or signing, the unsigned form is canonicalized with the JSON Canonicalization Scheme: object keys sorted lexicographically at every depth, no insignificant whitespace, canonical number forms, and keys with undefined values removed:

```
Object.keys(obj).filter(k => obj[k] !== undefined).sort()   // applied recursively
```

CONTENT ID (CID)

The CID binds the credential to its exact content. The unsigned form is the receipt minus the fields signature, signatures, and proof (and the cid itself is computed before signing):

```
cid = "sha256:" + hex( SHA-256( UTF-8( JCS( unsigned_form ) ) ) )
```

Any change to any signed field — a name, a date, a status pointer — produces a different CID and breaks the signature. The CID doubles as the public lookup key and the value anchored on-chain.

HASH-CHAINED VERSIONS

A renewed credential carries previous_credential = the CID of the version it replaces. The chain is tamper-evident: insertion, deletion, reordering, or substitution breaks it. Verification of an old version reports validity_status “renewed” and surfaces the latest CID — an old certificate can never impersonate the current one.

---

3 · Keys, signing & issuer identity

- **Algorithm**
  Ed25519 (RFC 8032) via WebCrypto, server-side only. Sign input is utf8(jcs(unsigned)) — never a re-serialized variant.
- **Key custody**
  The signing key exists in exactly one place: the ISSUER_ED25519_SK environment variable on the API host. Generated locally by the operator (gen-issuer-key.mjs), never committed, never transmitted in chat or email, never logged.
- **Issuer identity**
  Issuers are identified by DID. DCS publishes did:web:dcslabs.ai; the DID document at https://dcslabs.ai/.well-known/did.json lists the active public key (and predecessors after rotation) as Multikey entries.
- **Rotation**
  gen-rotate-issuer-key.mjs generates a fresh keypair and prints the updated DID document. Old receipts remain verifiable forever because each receipt carries the public key it was signed with; the DID document proves that key belonged to the issuer.
- **API keys (issuers only)**
  Issuance requires a bearer key: "dcsl_" + base64url(24 random bytes). The server stores only its SHA-256 hash. Verification requires no key at all.

---

4 · Status lists: revocation, suspension, renewal

Validity changes over time; the signature does not. Status lives outside the receipt in W3C Bitstring Status List v1.1 semantics: each status-enabled credential owns one bit in a compressed, publicly readable bitstring.

```
state        bit   reversible   meaning
-----        ---   ----------   -------
valid         0    —            active, not flagged
suspended     1    yes          temporarily not valid (reactivate clears)
revoked       1    NO           permanently invalid — revocation cannot be undone
```

- **Privacy**

The public list is an anonymous bitstring. Reading it reveals nothing about any subject — only that bit N of list L is set. Only someone holding the credential knows which bit is theirs.

- **Rule enforced in code**
  reactivate() only clears a suspension; no code path clears a revocation. This is a deliberate, audited invariant.
- **Public read**
  GET /v1/status/{listId} returns the compressed list; GET /v1/status/cid/{cid} resolves one credential's live state directly.

5 · On-chain anchoring (Base mainnet)

Anchoring gives a credential an existence proof independent of DCS infrastructure: the CID is written into the calldata of a 0-value transaction from the DCS anchor wallet to itself on Base mainnet.

```
tx.to    = anchor wallet (self)        tx.value = 0
tx.data  = utf8 bytes of the CID       cost     = a fraction of a cent on Base
receipt.base_tx = the tx hash          check    = open basescan.org/tx/<hash>, read input data
```

Properties: the anchor proves the CID existed no later than the block timestamp; it proves the anchor wallet (controlled by DCS) attested it; and it survives DCS entirely — the chain is the record. The anchor wallet holds only gas money and cannot sign credentials; compromising it cannot forge anything.

6 · Verification: the four-state algorithm

Verification is deterministic and runs the same way on the server, in the browser reference verifier, or by hand:

```
INPUT: receipt (or cid -> fetch receipt), optional original document

 1. unsigned  = receipt minus {signature, signatures, proof}
 2. cid'      = sha256:hex(SHA-256(utf8(jcs(unsigned))))      MUST equal receipt.cid
 3. sig_ok    = Ed25519.verify(receipt.pubkey, receipt.signature, utf8(jcs(unsigned)))
 4. key_ok    = receipt.pubkey appears in issuer's DID document (did:web)
 5. hash_ok   = if document supplied: its hash matches the subject's document hash (else null)
 6. time      = now within [valid_from, valid_until]
 7. status    = read bit from the public status list (suspended/revoked when set)
 8. renewal   = walk previous_credential chain forward; newer version -> 'renewed' + latest_cid
 9. anchor    = base_tx present and confirmed on Base (anchor_present)

OUTPUT validity_status:  valid | expired | suspended-revoked | renewed
       + checks: signature_valid, hash_match, anchor_present, latest_cid
```

The verdict is purely cryptographic. Advisory signals (portal risk, AI authenticity, fraud intelligence) are reported in separate fields, always labelled advisory, and never change validity_status.

7 · How to verify — four ways

7.1 WEB (NO ACCOUNT) — verify.dcslabs.ai/verify.html

Paste a CID (sha256:…) or the full receipt JSON; optionally paste the original document JSON to prove the file matches. The page shows the verdict, the three checks, and the raw materials (CID, public key, canonicalization algorithm, Basescan link, status-list link) so you can re-check everything yourself. Deep link: verify.dcslabs.ai/verify.html?cid=sha256:…

7.2 APP (PWA) — verify.dcslabs.ai/app.html

Installable on any phone (Add to Home Screen). Scan a credential QR with the camera (Chrome/Android via BarcodeDetector, iPhone via the built-in jsQR fallback), or paste a CID/receipt. Same verification path, same raw materials. Works offline as a shell; verification itself always hits the network — verdicts are never served from cache.

7.3 API (ONE CALL, NO KEY)

```
curl -s -X POST https://api.dcslabs.ai/v1/credentials/verify \
  -H 'Content-Type: application/json' \
  -d '{"cid":"sha256:..."}'                      # or {"receipt":{...},"document":{...}}

-> { "validity_status": "valid", "signature_valid": true,
     "hash_match": null, "anchor_present": true, "latest_cid": null }
```

7.4 FULLY INDEPENDENT (ZERO DCS INFRASTRUCTURE)

This is the design claim: a skeptic can verify with no DCS account, server, or goodwill.

```
# 1. obtain the receipt (from the holder, or GET /v1/credentials/<cid> while DCS exists)
# 2. recompute the CID (Node):
node -e '
  const jcs=o=>JSON.stringify(sort(o));const sort=v=>Array.isArray(v)?v.map(sort):
   (v&&typeof v==="object")?Object.keys(v).filter(k=>v[k]!==undefined).sort()
   .reduce((a,k)=>(a[k]=sort(v[k]),a),{}):v;
  const r=require("./receipt.json");const u={...r};
  delete u.signature;delete u.signatures;delete u.proof;
  const h=require("crypto").createHash("sha256").update(Buffer.from(jcs(u))).digest("hex");
  console.log("sha256:"+h===r.cid ? "CID OK" : "CID MISMATCH");'
# 3. verify the signature with any Ed25519 library against receipt.pubkey
# 4. confirm receipt.pubkey in https://dcslabs.ai/.well-known/did.json
# 5. read the status bit:   GET /v1/status/<list_id>   (public bitstring)
# 6. confirm the anchor:    basescan.org/tx/<base_tx>  (calldata = the CID)
```

Steps 2–4 and 6 work even if every DCS server is offline. The MIT reference verifier (github.com/DCS-LabsAI/r2-standard) implements all of it.

---

8 · Universal Verify: W3C VC & Open Badges inbound

POST /v1/verify/universal accepts any credential payload, detects the format, and verifies what it genuinely can:

```
format          status     what happens
------          ------     ------------
dcs-r2          live       full four-state verify (section 6)
w3c-vc          live       DCS envelope -> full receipt verify; external VC -> Data Integrity proof
open-badges     live       OB 3.0 = VC profile; same Data Integrity path (eddsa-jcs-2022 only)
pdf-signature   scaffold   honest refusal: 'not yet verifiable; do not treat as valid'
scitt           scaffold   honest refusal
```

EXTERNAL VC VERIFICATION (eddsa-jcs-2022)

```
docHash  = SHA-256( utf8( JCS( vc minus proof ) ) )
cfgHash  = SHA-256( utf8( JCS( proof minus proofValue, + vc's @context ) ) )
verified = Ed25519.verify( pubkey, base58btc(proofValue), cfgHash || docHash )
key resolution:  did:key (offline decode)  ·  did:web (HTTPS did.json, 5s timeout)
```

If the external VC declares a BitstringStatusListEntry, the foreign status list credential is fetched, the encodedList is base64url-decoded and gunzipped, and the exact bit is read (MSB-first). A set bit flips the verdict to not verified. Reported honestly alongside: status_list_proof_checked:false — the foreign list's own signature is not yet verified, so the response says exactly that.

---

9 · Trust infrastructure: registry, score, explorer

Cryptography proves a credential is intact and from a key; the trust layer helps relying parties judge the issuer behind the key. Everything in it is public, account-free, and reproducible.

TRUST SCORE (0-100, OPEN FORMULA)

```
component     weight   input (all public)
---------     ------   ------------------
reliability     30     revocation rate (0 issued -> neutral 0.5)
longevity       20     days since first issuance (full at 730)
volume          20     verification events, log-scaled (full at 100k)
audit           20     none=0 · in_progress=0.5 · verified=1
federation      10     cross-issuer reputation (neutral 0.5 until network data)
```

Every /v1/registry/{issuer} response includes the inputs, the per-component sub-scores, and the reasons — anyone can recompute the number. There is no pay-to-rank and no hidden weighting. A young issuer scoring in the 30s is the honest design working, not a failure: scores are earned by real activity.

The Explorer (verify.dcslabs.ai/explorer.html) shows live network counts (credentials, active issuers, anchors, verification events) read from the database, flagged early_network while small. The Registry page renders any issuer profile at registry.html?issuer=<id>.

10 · How to issue

10.1 GET A KEY (FREE)

Self-serve at verify.dcslabs.ai/join.html (or POST /v1/auth/signup with {"email":"..."}). The key is shown once. It immediately allows test issuance under the sandbox flow and API exploration.

10.2 REQUEST YOUR ISSUER ID

```
curl -s -X POST https://api.dcslabs.ai/v1/issuers/request \
  -H "Authorization: Bearer $KEY" -H 'Content-Type: application/json' \
  -d '{"issuer_id":"acme-academy","display_name":"Acme Academy",
       "contact_email":"registrar@acme.org","profile":"r2-education"}'
```

The request lands as pending; DCS approves after an accountability check (the registry only lists real organisations). Once approved the issuer appears in the public registry automatically and may issue under its own ID.

10.3 ISSUE A CREDENTIAL

```
curl -s -X POST https://api.dcslabs.ai/v1/credentials/issue \
  -H "Authorization: Bearer $KEY" -H 'Content-Type: application/json' \
  -d '{
    "issuer": "acme-academy",
    "credential_type": "certificate",
    "profile": "r2-education",                    // optional: enforces sector claims
    "subject": { "subject_name": "Asha Verma",
                 "issuer_institution": "Acme Academy",
                 "credential_title": "Full-Stack Diploma",
                 "issued_date": "2026-06-04" },
    "status_enabled": true,                       // revocable (recommended)
    "anchor": true                                // Base mainnet anchor
  }'
-> 201 + the signed receipt (cid, signature, pubkey, status slot, base_tx)
   ?format=w3c also returns the W3C VC envelope
```

Give the holder the receipt JSON (or just the verify link / QR: verify.dcslabs.ai/verify.html?cid=<cid>). Manage lifecycle with POST /v1/credentials/{cid}/revoke | suspend | reactivate. Sector profiles (r2-education, r2-gov, r2-health, r2-finance) enforce required claims at issue time; education is permanent-by-default and pairs naturally with results from DCS Rank.

11 · Integrations & distribution

VERIFY BADGE — ONE LINE, VERIFIED EVERYWHERE

```
<script src="https://verify.dcslabs.ai/badge.js" data-cid="sha256:..."></script>
```

The embeddable badge renders the credential's LIVE status (it queries the public verify endpoint — never a cached verdict) and links to the public verify page. Like an SSL padlock for credentials: the trust mark travels with the credential wherever it is displayed, and clicking it gives the viewer the full raw-materials verification, no account required.

QR + DEEP LINKS

Every credential is addressable as verify.dcslabs.ai/verify.html?cid=sha256:... — printable as a QR on certificates, ID cards, or documents. The PWA app scans these with the phone camera (BarcodeDetector on Android/Chrome, jsQR fallback on iPhone Safari) and verifies in one step.

SPRINT PRESERVATION RECEIPTS (P3)

DCS Sprint's build-preservation pipeline issues a signed preservation_receipt per preserved build (POST /v1/sprint/preservation). Each receipt hash-chains into the same R+2 receipt chain and may be anchored on Base — connecting long-term storage mirrors to cryptographic proof of what was preserved and when. 12/12 conformance tests green.

WALLET BRIDGE — OID4VP / OID4VCI (P4)

Credentials are wallet-ready by standard: the OpenID for Verifiable Presentations (OID4VP) request/session endpoints and the OID4VCI issuance flow are implemented and conformance-tested (21/21) against the W3C VC envelope every receipt can be exported to (?format=w3c). HONEST STATUS: the session layer is live as spec-conformant stubs; a full round-trip against a real external wallet (Google / Apple / EU reference wallet) is the remaining external test before claiming wallet compatibility outright.

ANTI-CLONE PORTAL CHECK (P5)

POST /v1/portal/check assesses any URL for issuer-impersonation signals (lookalike domains, suspicious TLDs, brand-in-untrusted-domain) and returns a risk level with guidance. Strictly advisory — the real defence is structural: authenticity comes from the signed receipt, which verifies anywhere, so a cloned portal cannot mint validity.

---

12 · Security model & honest limitations

WHAT COMPROMISE COSTS

asset	if stolen	blast radius / recovery
issuer signing key	forged credentials until rotation	rotate + DID doc update; old receipts unaffected
anchor wallet key	small ETH (gas only)	cannot forge anything; refill new wallet
an issuer API key	issuance under that issuer	revoke key (hash-stored, instant)
the database	read receipts (already verifiable)	cannot forge: no signing material stored
verify.dcslabs.ai	phishing surface only	receipts verify anywhere; portal-check advisory

CURRENT LIMITATIONS (DISCLOSED BY THE API ITSELF)

- **Foreign status-list proofs**
  External Bitstring lists: bit checked, list's own signature not yet verified (status_list_proof_checked:false).
- **Format coverage**
  PDF/PAdES signatures and SCITT receipts are honest scaffolds — they refuse rather than guess.
- **Audit status**
  audit:none until an independent review publishes; the audit pack (threat model, evidence, reproduction steps) is prepared.
- **No production ZK**
  R+4 selective disclosure is gated on a multi-party trusted-setup ceremony, by design.
- **Privacy choice**
  Verification events are counted, but verifier identity is never recorded; reputation is aggregate-only; status lists are anonymous bitstrings.

13 · API reference (api.dcslabs.ai)

```
PUBLIC — no key required
  POST /v1/credentials/verify          {cid|receipt, document?} -> four-state result
  GET  /v1/credentials/{id|cid}        receipt (?format=w3c for VC envelope)
  GET  /v1/status/{listId}             compressed public bitstring
  GET  /v1/status/cid/{cid}            live status for one credential
  POST /v1/verify/universal            any format -> verify or honest refusal
  GET  /v1/verify/universal/formats    format support matrix
  GET  /v1/registry                    all issuers + reproducible trust scores
  GET  /v1/registry/{issuer}           one issuer profile + score breakdown
  GET  /v1/explorer/stats              real network counts
  GET  /v1/reputation/{issuer}         aggregate-only reputation (network-gated)
  POST /v1/portal/check                {url} -> anti-clone advisory
  POST /v1/advisory/ai | /fraud        advisory scaffolds (never in the verdict)
  POST /v1/auth/signup                 {email} -> free API key (rate-limited)

ISSUER — Bearer dcsl_ key
  POST /v1/credentials/issue           sign + store + optional anchor/status
  POST /v1/credentials/{cid}/revoke    permanent
  POST /v1/credentials/{cid}/suspend   reversible
  POST /v1/credentials/{cid}/reactivate  clears suspension only
  POST /v1/issuers/request             request your own issuer ID
  POST /v1/sprint/preservation         preservation receipt (hash-chained)

CORS: configured via CORS_ORIGINS. Verification endpoints are public by design.
```

DCS Verify is live at verify.dcslabs.ai · API at api.dcslabs.ai · Standard, conformance vectors, and the MIT reference verifier at github.com/DCS-LabsAI/r2-standard. Verify it yourself — don't take our word for it.