Prisma DIDs – Cardano Integration Plan

Document Type: Technical Integration Specification

Version: 1.0

Date: December 2025

Authors: Prisma Team

Status: Active

Specification

1. Cardano Standards Compliance

CIP-10: Metadata Label Registry

Prisma DIDs uses registered CIP-10 labels for all on-chain data:

Label	Decimal	Purpose	Status	Scope
`L_DID`	199674	DID lifecycle events (create/update/revoke)	Reserved	Global (all DIDs)
`L_VC`	199675	VC anchoring events (issue/validate/revoke)	Reserved	Prisma/ALJ

Note: Labels are in the application-specific range (1-65535). Formal CIP-10 registration to be submitted post-MVP.

Custom Organization Labels:

Organizations deploying their own VC Indexer can register additional CIP-10 labels for their credential types. The label schema must be compatible with the base `VCEventPayload` structure.

CIP-20: Transaction Metadata

All Prisma DIDs data is stored as CIP-20-compliant transaction metadata:

```
{
  "199674": {
    "id": "did:cardano:stake1u9...",
    "ipfs": "bafybeig...",
    "action": "create",
    "v": 1,
    "prev": null,
    "payloadSig": "{\"key\":\"...\",\"sig\":\"...\",\"address\":\"...\"}"
  }
}
```

Metadata Constraints:

- Strings limited to 64 bytes (chunked with `_0`, `_1` suffixes if needed)
- Total metadata size limited to ~16KB per transaction
- JSON must be valid UTF-8
- Null values must be converted to empty strings (`""`) for Cardano metadata compatibility

CIP-30: Wallet dApp Connection

Prisma DIDs uses the following CIP-30 methods:

Method	Purpose
`getRewardAddresses()`	Extract stake address for DID derivation
`signData(address, payload)`	Sign DID/VC events with stake key
`getUsedAddresses()`	Get payment address for transactions
`signTx(tx)`	Sign metadata transactions
`submitTx(tx)`	Submit to Cardano network

Wallet Compatibility:

- Eternl
- Lace
- Nami
- Flint
- Typhon

2. DID Method: `did:cardano`

Identifier Format

```
did:cardano:<stake-address>
```

Where `<stake-address>` is the bech32-encoded stake address from `getRewardAddresses()[0]`.

Example:

```
did:cardano:stake1u9rqg7a3skqzx8jxzvm9k2w3c5k6h7j8l9m0n1p2q3r4s5
```

DID Document Storage

DID Documents are stored on IPFS and referenced by CID in metadata:

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ],
  "id": "did:cardano:stake1u9...",
  "controller": "did:cardano:stake1u9...",
  "verificationMethod": [
    {
      "id": "did:cardano:stake1u9...#key-0",
      "type": "Ed25519VerificationKey2020",
      "controller": "did:cardano:stake1u9...",
      "publicKeyMultibase": "z6Mk..."
    }
  ],
  "authentication": ["did:cardano:stake1u9...#key-0"],
  "assertionMethod": ["did:cardano:stake1u9...#key-0"]
}
```

On-Chain Event Schema

DID Event Payload:

```
interface DIDEventPayload {
  id: string; // did:cardano:stake1...
  ipfs: string; // IPFS CID of DID Document
  action: 'create' | 'update' | 'revoke';
  v: number; // Version (1, 2, 3, ...)
  prev: string | null; // Previous tx hash (null for create)
  payloadSig: string; // JSON-stringified PrismaPayloadSig
}
```

Signature Wrapper:

```
interface PrismaPayloadSig {
  key: string; // Ed25519 public key (hex)
  sig: string; // Ed25519 signature (hex)
  address: string; // Signing address (bech32)
}
```

3. VC Anchoring

VC Event Schema

```
interface VCEventPayload {
  event: 'issue' | 'validate' | 'revoke';
  issuerDid: string;
  holderDid: string;
  vcHash: string; // jti for SD-JWT, SHA-256 for Ed25519
  vcType: string; // e.g., "ContributionCredential"
  vcFormat: 'sd-jwt' | 'ed25519' | 'bbs'; // REQUIRED
  validatorDid?: string; // For validate events
  reason?: string; // For revoke events
}
```

On-Chain Example:

```
{
  "199675": {
    "event": "issue",
    "issuerDid": "did:cardano:stake1u9...",
    "holderDid": "did:cardano:stake1ux...",
    "vcHash": "urn:uuid:550e8400-e29b-41d4-a716-446655440000",
    "vcType": "ContributionCredential",
    "vcFormat": "sd-jwt"
  }
}
```

vcHash Strategy

For **SD-JWT credentials**, `vcHash` uses the `jti` (JWT ID) claim to preserve selective disclosure. For **Ed25519 credentials**, `vcHash` is the SHA-256 hash of the canonicalized credential. See TECHNICAL_DESIGN.md §7.4 and §8.2 for full specification.

4. Transaction Structure

Typical DID Create Transaction

```
Inputs:
  - UTxO from wallet (payment address)

Outputs:
  - Change to wallet (minus fee)

Metadata:
  199674: {
    "id": "did:cardano:stake1u9...",
    "ipfs": "bafybeig...",
    "action": "create",
    "v": 1,
    "prev": null,
    "payloadSig": "{...}"
  }

Fee: ~0.17-0.25 ADA
```

Cost Analysis

Operation	Metadata Size	Estimated Fee
DID Create	~500 bytes	0.17-0.20 ADA
DID Update	~550 bytes	0.18-0.22 ADA
DID Revoke	~500 bytes	0.17-0.20 ADA
VC Anchor	~300 bytes	0.17-0.20 ADA

Monthly Cost Estimate (100 DIDs, 500 VCs):

- DID operations: ~25 ADA
- VC operations: ~90 ADA
- **Total: ~115 ADA (~$70 at $0.60/ADA)**

5. Resolution Process

DID Resolution Algorithm

```
RESOLVE(did:cardano:stake1...)

1. Query Blockfrost for metadata label 199674
   GET /metadata/txs/labels/199674?order=asc

2. Filter events where metadata.id == did

3. Build event chain:
   - Start with create event (v=1, prev=null)
   - Link update events via prev → tx_hash
   - Check for revoke event

4. Verify each event:
   - Validate Ed25519 signature
   - Confirm stake address matches DID
   - Verify prev pointer chain

5. Return:
   - If revoked: { deactivated: true }
   - If valid: Latest DID Document from IPFS
```

Indexer Architecture

The indexer layer is split into global and forkable components:

DID Indexer (Global) – Single ecosystem-wide indexer for all `did:cardano` identities:

Endpoint	Method	Description
`/did/:did`	GET	Resolve DID to current document
`/did/:did/history`	GET	Full event chain

VC Indexer (Forkable) – Per-organization indexer for credential status:

Endpoint	Method	Description
`/vc/:vcHash/status`	GET	VC status (active/revoked)
`/issuer/:did/credentials`	GET	VCs issued by DID
`/holder/:did/credentials`	GET	VCs held by DID
`/schemas`	GET	Supported credential types

Verifier Discovery Flow:

Verifiers resolve the issuer's DID Document to find the appropriate VC Indexer:

```
1. Extract issuerDid from credential
2. GET /did/{issuerDid} → DID Document
3. Find service with type: "VCIndexer"
4. GET {serviceEndpoint}/vc/{vcHash}/status → revocation status
```

Fallback: If no `VCIndexer` service is present in the issuer's DID Document, verifiers SHOULD default to the Prisma VC Indexer (`L_VC`) or treat revocation status as unknown.

Custom Label Governance:

Organizations MAY use their own CIP-10 label (e.g., `L_ORG = 888888`) or default to `L_VC` (199675). The issuer's DID Document `service` field determines which indexer verifiers should query.

> ***Warning: Organizations reusing `L_VC` without their own label MUST ensure their credential schemas are compatible with the base `VCEventPayload` structure to avoid collisions. Registering a dedicated label is RECOMMENDED for production deployments.***

Implementation Structure:

The VC Indexer and VC Interface share a common schemas package (monorepo structure):

```
packages/
├── schemas/        # Shared credential type definitions
├── indexer/        # Forkable VC Indexer
└── vc-interface/   # Forkable VC Frontend
```

When forking, organizations modify `packages/schemas/` with their credential types, and both indexer and interface stay synchronized.

> ***Implementation Status: The forkable VC Indexer architecture is specified but not yet implemented. MVP delivers the Prisma-operated indexer for `L_DID` and `L_VC`; the forkable pattern enables future multi-tenant deployments.***

6. Security Considerations

Threat Model

Threat	Mitigation
DID Hijacking	Only stake key holder can sign events
Event Forgery	Ed25519 signatures verified on resolution
Chain Tampering	Blockchain immutability + prev pointer chain
Replay Attack	Version numbers prevent replay
Metadata Spam	CIP-10 label filtering; resolver ignores invalid events

Signature Verification

All DID events are verified using:

1. **Ed25519 signature** over canonicalized payload (UTF-8 bytes)
2. **Stake address derivation** from signing address
3. **Controller match** between derived stake and DID identifier

```
async function verifyDIDEvent(event: DIDEvent): Promise<boolean> {
  // 1. Verify Ed25519 signature
  const validSig = await ed25519.verifyAsync(sig, message, pubKey);
  if (!validSig) return false;

  // 2. Derive stake address from signing address
  const stakeFromSigner = deriveStakeAddress(event.payloadSig.address);

  // 3. Compare with DID
  const stakeFromDid = event.id.replace('did:cardano:', '');
  return stakeFromSigner === stakeFromDid;
}
```

Dependencies

Dependency	Provider	Status
Cardano Node	IOHK	Available
Blockfrost API	Blockfrost.io	Available
IPFS Pinning	Pinata	Available
CIP-30 Wallets	Eternl, Lace, Nami	Available

References

Cardano Improvement Proposals

- CIP-10: Transaction Metadata Label Registry (https://cips.cardano.org/cip/CIP-10)
- CIP-20: Transaction Message/Comment Metadata (https://cips.cardano.org/cip/CIP-20)
- CIP-30: Cardano dApp-Wallet Web Bridge (https://cips.cardano.org/cip/CIP-30)

W3C Standards

- DID Core 1.0 (https://www.w3.org/TR/did-core/)
- Verifiable Credentials Data Model 2.0 (https://www.w3.org/TR/vc-data-model-2.0/)

IETF Standards

- SD-JWT (draft-ietf-oauth-selective-disclosure-jwt) (https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/)

Related Documentation

- TECHNICAL_DESIGN_1.5.md (./TECHNICAL_DESIGN_1.5.md) - Full technical specification
- ADR-001 (./adr/001-metadata-anchoring.md) - Architecture decision record

---