Prisma DIDs Technical Specification

Cardano-Native Decentralized Identifiers and Verifiable Credentials

Authors: Prisma Team

Status: Production-Ready Specification

Last Updated: December 2025

Abstract

This document specifies **Prisma DIDs**, a Cardano-native system for decentralized identifiers (DIDs) and verifiable credentials (VCs). The specification defines:

- A W3C-compliant DID method (`did:cardano`) using Cardano stake addresses
- On-chain DID lifecycle management via CIP-20-style transaction metadata
- Verifiable Credentials with SD-JWT selective disclosure
- VC anchoring and revocation mechanisms
- A resolver architecture for DID resolution and VC status queries

The system enables privacy-preserving identity and credential management on Cardano with transaction costs of approximately 0.17-0.25 ADA per operation.

Table of Contents

---

1. Terminology

1.1 Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 (https://datatracker.ietf.org/doc/html/rfc2119).

1.2 Definitions

Term	Definition
DID	Decentralized Identifier per W3C DID Core specification
DID Document	JSON-LD document containing verification methods for authentication
VC	Verifiable Credential per W3C VC Data Model
SD-JWT	Selective Disclosure JWT per IETF draft specification
Stake Address	Cardano reward address (bech32 format: `stake1...`)
L_DID	CIP-10 metadata label for DID events (199674)
L_VC	CIP-10 metadata label for VC events (199675)
CIP-30	Cardano dApp-Wallet Web Bridge specification
CIP-20	Cardano Transaction Metadata specification
Issuer	Entity that creates and signs a Verifiable Credential
Holder	Entity that receives and presents a Verifiable Credential
Verifier	Entity that validates a presented Verifiable Credential
Controller	Entity authorized to make changes to a DID Document

1.3 Conformance

This specification contains both normative and non-normative content:

- **Normative sections:** Sections 4-10 define the required behavior for conforming implementations. These sections use RFC 2119 keywords (MUST, SHOULD, MAY) to indicate requirement levels.
- **Non-normative sections:** Sections 1-3 (Introduction, Terminology, Architecture), Sections 11-13 (Privacy, SDK Reference, Future Roadmap), and Appendix A (Design Rationale) provide context and guidance but do not define conformance requirements.

A conforming implementation:

1. MUST implement the DID method as specified in Section 4
2. MUST use the on-chain event schemas defined in Sections 5 and 8
3. MUST perform signature verification as specified in Section 10.2
4. SHOULD implement the resolver API endpoints defined in Section 9
5. MAY support any subset of credential formats (SD-JWT, Ed25519, BBS+)

---

2. Introduction

2.1 Purpose

Prisma DIDs provides a lightweight, cost-effective identity layer for Cardano that enables:

- Self-sovereign identity creation using existing Cardano wallets
- Verifiable credentials for contribution tracking and attestations
- Privacy-preserving credential sharing via selective disclosure
- On-chain anchoring for credential verification and revocation

2.2 Design Goals

Cardano-Native: Use native Cardano primitives (stake keys, metadata) without external dependencies
W3C Compliant: Full compliance with DID Core and VC Data Model specifications
Cost-Effective: DID operations at ~0.17-0.25 ADA (approximately $0.10-0.15 USD)
Privacy-Preserving: Cryptographic selective disclosure via SD-JWT, with BBS+ upgrade path
Developer-Friendly: Simple SDK with minimal infrastructure requirements
Wallet-Compatible: Works with any CIP-30 compliant wallet

2.3 Standards Compliance

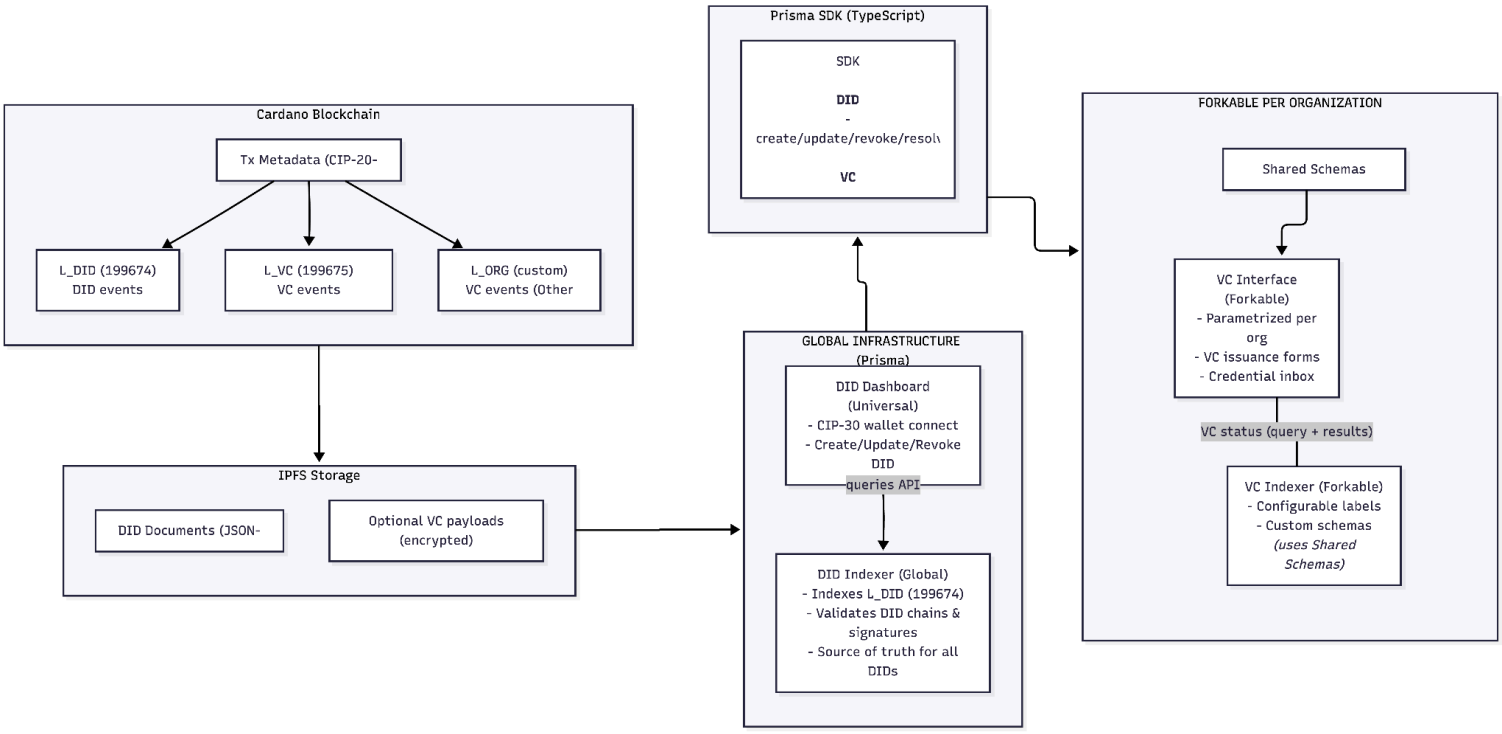
This specification implements or references the following standards:

Standard	Usage
W3C DID Core 1.0 (https://www.w3.org/TR/did-core/)	DID method specification
W3C VC Data Model 2.0 (https://www.w3.org/TR/vc-data-model-2.0/)	Credential format
IETF SD-JWT (https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/)	Selective disclosure
CIP-10 (https://cips.cardano.org/cip/CIP-10)	Metadata label registry
CIP-20 (https://cips.cardano.org/cip/CIP-20)	Transaction metadata
CIP-30 (https://cips.cardano.org/cip/CIP-30)	Wallet dApp connector
RFC 8785 (https://datatracker.ietf.org/doc/html/rfc8785)	JSON Canonicalization Scheme

---

3. Architecture Overview

3.1 System Components



Design Philosophy: DIDs are ecosystem-wide identity infrastructure—one global indexer serves all Cardano users. VCs are organization-specific—each organization defines their credential types, schemas, and runs their own indexer. This separation enables decentralized credential ecosystems while maintaining a unified identity layer.

3.2 Technology Stack

Component	Technology
Blockchain	Cardano (mainnet/preprod)
Metadata Format	CIP-20-style JSON under CIP-10 labels
Wallet Integration	CIP-30 dApp connectors
Document Storage	IPFS (Pinata or equivalent)
Backend	Node.js + TypeScript
SDK	TypeScript client library
Frontend	React / Next.js
Canonicalization	JSON Canonicalization Scheme (RFC 8785)

3.3 Frontend Architecture

The frontend layer is split into two distinct applications:

DID Dashboard (Universal)

A general-purpose DID management interface for any Cardano user:

- **Purpose:** Create, view, update, and revoke `did:cardano` identities
- **Target Users:** Any Cardano wallet holder
- **Scope:** DID-only operations, no VC-specific features
- **Deployment:** Single canonical instance serving the Cardano ecosystem
- **Branding:** Neutral Cardano identity branding

The DID Dashboard is infrastructure serving the entire ecosystem.

VC Interface (Forkable/Parametrized)

A customizable credential interface designed to be forked or parametrized per organization:

- **Purpose:** Issue, receive, present, and verify Verifiable Credentials
- **Target Users:** Organization members (contributors, team members)
- **Scope:** Full VC lifecycle with organization-specific credential types
- **Deployment:** Multiple instances (one per organization or use case)
- **Branding:** Customizable per deployment

Parametrization Options:

Parameter	Description	Example Values
`ORG_NAME`	Organization display name	"Prisma"
`ORG_LOGO`	Logo asset path	`/assets/logo.svg`
`CREDENTIAL_TYPES`	Allowed credential types	`["ContributionCredential"]`
`ISSUER_DIDS`	Authorized issuer DIDs	`["did:cardano:stake1u9..."]`
`THEME`	UI color scheme	`{ primary: "#4F46E5" }`
`INDEXER_ENDPOINT`	VC Indexer service URL	"https://indexer.example.com"

3.4 Indexer Architecture

The indexer layer is split to match the frontend separation:

DID Indexer (Global)

A **single ecosystem-wide indexer** for all `did:cardano` identities:

- **Purpose:** Index and validate all DID lifecycle events
- **Scope:** L_DID (199674) metadata label only
- **Operator:** Prisma (ecosystem infrastructure)
- **Deployment:** Single canonical instance

VC Indexer (Forkable)

A **customizable indexer** designed to be forked or configured per organization:

- **Purpose:** Index organization-specific VC events
- **Scope:** Configurable metadata labels and schemas
- **Operator:** Each organization runs their own
- **Deployment:** Multiple instances (one per organization)

VC Indexer Configuration:

```
export const indexerConfig = {
  labels: {
    199674: { name: 'L_DID', schema: DIDEventPayloadSchema },
    199675: { name: 'L_VC', schema: VCEventPayloadSchema },
  },
  cardanoProvider: 'blockfrost',
  didIndexerEndpoint: 'https://did.prisma-dids.io',
};
```

Organizations can register custom CIP-10 labels for their credential types and configure their indexer accordingly.

---

4. DID Method: did:cardano

4.1 Method Name

The method name for this DID method is: `cardano`

4.2 Method-Specific Identifier

The method-specific identifier is derived from the Cardano stake address:

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

Where `<stake-address>` is the bech32-encoded reward address obtained from a CIP-30 wallet.

Example:

```
did:cardano:stake1u9rqg7a3skqzx8jxzvm9k2w3c5k6h7j8l9m0n1p2q3r4s5
```

4.3 Identifier Properties

The stake address binding provides:

Property	**Benefit**
Persistence	Stable across payment address changes and UTxO consumption
Independence	Not tied to any single spending address
Verifiability	Cryptographically bound to stake key
Wallet Compatibility	Derivable from any CIP-30 wallet

4.4 Network Considerations

Unless otherwise specified, this specification assumes Cardano mainnet (networkId = 1). On testnet/preprod:

- The DID derivation algorithm is identical
- Only the bech32 prefix differs (`stake_test1...` vs `stake1...`)
- The same DID method works across all Cardano networks

4.5 DID Derivation Algorithm

Implementations MUST derive the DID as follows:

Inputs:

- CIP-30 wallet instance
- Result of `wallet.getRewardAddresses()`

Algorithm:

```
async function deriveDID(wallet: CIP30Wallet): Promise<string> {
  // 1. Obtain reward addresses
  const rewardAddresses = await wallet.getRewardAddresses();

  // 2. Use first reward address (primary stake key)
  const stakeAddress = rewardAddresses[0]; // e.g., "stake1u9..."

  // 3. Construct DID
  return `did:cardano:${stakeAddress}`;
}
```

The DID identifier MUST be the complete bech32 stake address, not a hash or derivative.

4.6 DID Document

DID Documents MUST conform to W3C DID Core and SHOULD be stored on IPFS.

Required Structure:

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ],
  "id": "did:cardano:stake1u9rqg7a3skqzx8jxzvm9k2w3c5k6h7j8l9m0n1p2q3r4s5",
  "controller":
"did:cardano:stake1u9rqg7a3skqzx8jxzvm9k2w3c5k6h7j8l9m0n1p2q3r4s5",
  "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"],
  "service": [
    {
```

```
      "id": "did:cardano:stake1u9...#vc-indexer",
      "type": "VCIndexer",
      "serviceEndpoint": "https://indexer.alj.example.com"
    }
  ]
}
```

The `VCIndexer` service enables **verifier discovery**: when verifying a credential, the verifier resolves the issuer's DID Document to find which indexer to query for revocation status.

Field Requirements:

Field	Requirement	Description
`@context`	REQUIRED	MUST include W3C DID context
`id`	REQUIRED	MUST match the derived DID
`controller`	REQUIRED	MUST be the DID itself (self-controlled)
`verificationMethod`	REQUIRED	At least one Ed25519 key
`authentication`	REQUIRED	Reference to verification method
`assertionMethod`	RECOMMENDED	For VC signing
`service`	OPTIONAL	Linked service URLs per W3C DID spec (e.g., credential endpoints, profiles)

---

5. DID Registry (On-Chain)

5.1 Metadata Labels

DID events are stored under reserved CIP-10 labels (formal registration pending):

Constant	Value	Purpose
`L_DID`	199674	DID lifecycle events
`L_VC`	199675	VC anchoring events

5.2 DID Event Schema

All DID events MUST include the following payload structure:

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

Field Validation Rules:

Field	Validation
`id`	MUST be valid `did:cardano:` format
`ipfs`	MUST be valid IPFS CID (v0 or v1)
`action`	MUST be one of: `create`, `update`, `revoke`
`v`	MUST be positive integer; create MUST be 1
`prev`	MUST be null for create; MUST be valid tx hash for update/revoke
`payloadSig`	MUST be valid JSON string containing PrismaPayloadSig

Optional Fields (Backward Compatibility):

Resolvers SHOULD ignore unknown fields in DID events for forward compatibility. Early implementations MAY include additional fields (e.g., `ts` for client-side timestamp convenience). These fields are not validated and do not affect event processing.

5.3 Signature Wrapper

Each DID event MUST include a signature for verification:

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

The signature is over the canonicalized JSON payload using RFC 8785 (JCS).

5.4 On-Chain Format

The complete metadata structure:

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

5.5 Metadata Constraints

Cardano metadata has the following limitations:

Constraint	Limit	Handling
String length	64 bytes max	Chunk with `_0`, `_1` suffixes
Total size	~16KB per tx	Split across transactions if needed
Encoding	UTF-8 only	Ensure valid UTF-8
Null values	Not supported	Convert to empty string `""`

Chunking Example:

```
{
  "payloadSig_0": "first 64 bytes...",
  "payloadSig_1": "next 64 bytes...",
  "payloadSig_2": "remaining bytes..."
}
```

6. DID Operations

6.1 Create DID

Prerequisites:

- CIP-30 wallet connected
- Sufficient ADA for transaction (~0.2 ADA)

Process:

1. Derive DID from wallet stake address
2. Generate DID Document with wallet's Ed25519 public key
3. Pin DID Document to IPFS
4. Build payload with `action: "create"`, `v: 1`, `prev: null`
5. Sign payload using `wallet.signData()`
6. Submit metadata transaction

SDK Example:

```
const result = await prismaSDK.createDID(wallet);
// Returns: { did, txHash, ipfsCid }
```

6.2 Update DID

Prerequisites:

- Existing DID created by this wallet
- Previous transaction hash

Process:

1. Fetch current DID state from resolver
2. Modify DID Document as needed
3. Pin updated document to IPFS

4. Build payload with `action: "update"`, `v: prev.v + 1`, `prev: lastTxHash`
5. Sign and submit

Validation:

- v MUST be exactly `prev.v + 1`
- prev MUST reference the most recent valid transaction
- Signer MUST be the DID controller

6.3 Revoke DID

Process:

1. Build payload with `action: "revoke"`, `v: prev.v + 1`
2. IPFS CID SHOULD point to a document with `deactivated: true`
3. Sign and submit

Effect:

- DID is permanently deactivated
- Resolver returns `{ deactivated: true }`
- All associated VCs SHOULD be considered invalid

6.4 Resolve DID

Algorithm:

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

1. Query blockchain for all L_DID metadata
2. Filter events where payload.id == did
3. Build event chain:
   - Start with create event (v=1, prev=null)
   - Link updates via prev → tx_hash
   - Check for revoke event
4. Verify each event:
   - Validate Ed25519 signature
   - Confirm stake address matches DID
   - Verify prev pointer integrity
5. Return:
   - If revoked: { deactivated: true }
   - If valid: Latest DID Document from IPFS
```