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.

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:

16. MUST implement the DID method as specified in Section 4
17. MUST use the on-chain event schemas defined in Sections 5 and 8
18. MUST perform signature verification as specified in Section 10.2
19. SHOULD implement the resolver API endpoints defined in Section 9
20. 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

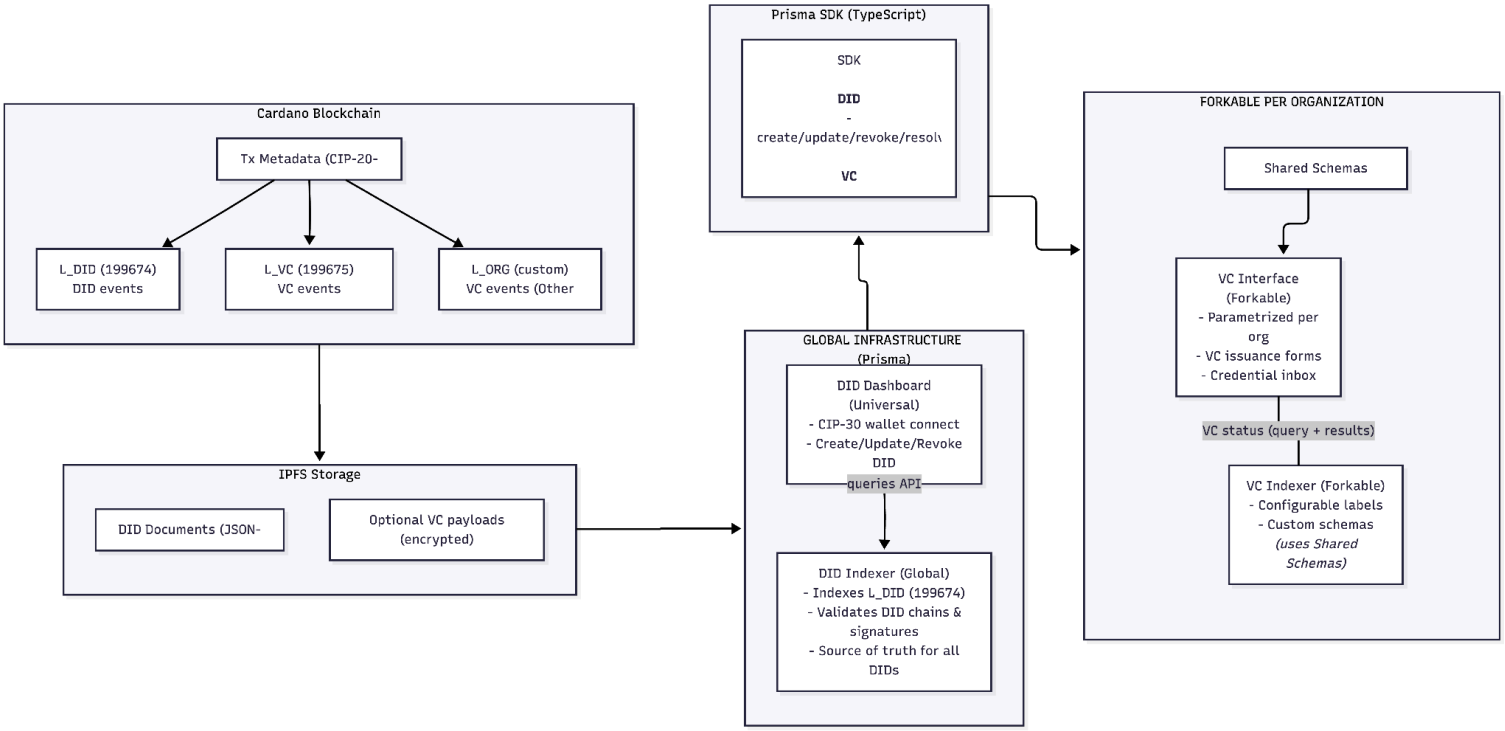
2.3 Standards Compliance

This specification implements or references the following standards:

Standard	Usage
W3C DID Core 1.0	DID method specification
W3C VC Data Model 2.0	Credential format
IETF SD-JWT	Selective disclosure
CIP-10	Metadata label registry
CIP-20	Transaction metadata
CIP-30	Wallet dApp connector
RFC 8785	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

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:

21. Derive DID from wallet stake address
22. Generate DID Document with wallet's Ed25519 public key
23. Pin DID Document to IPFS
24. Build payload with action: "create", v: 1, prev: null
25. Sign payload using wallet.signData()
26. 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:

27. Fetch current DID state from resolver
28. Modify DID Document as needed
29. Pin updated document to IPFS
30. Build payload with action: "update", v: prev.v + 1, prev: lastTxHash
31. 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:

32. Build payload with action: "revoke", v: prev.v + 1
33. IPFS CID SHOULD point to a document with deactivated: true
34. 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
```

7. Verifiable Credentials

7.1 Credential Format

Prisma DIDs supports two credential formats:

Format	Use Case	Privacy Level
Ed25519	Simple credentials, full disclosure	None
SD-JWT	Selective disclosure needed	Claim-level

7.2 ContributionCredential Schema

The primary credential type for contribution tracking:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://prisma.events/contexts/contribution/v1"
  ],
  "type": ["VerifiableCredential", "ContributionCredential"],
  "issuer": "did:cardano:stake1u9...",
  "issuanceDate": "2025-01-15T10:30:00Z",
  "credentialSubject": {
    "id": "did:cardano:stake1ux...",
    "projectId": "catalyst-fund-14-prisma",
    "contributionType": "code",
    "hours": 42,
    "organization": "Prisma",
    "evidenceUrl": "https://github.com/..."
  }
}
```

7.3 SD-JWT Selective Disclosure

SD-JWT enables holders to reveal only specific claims during presentation.

Issuance Process:

35. Issuer identifies disclosable claims
36. Each disclosable claim is replaced with a hash
37. Original values are stored as "disclosures"
38. Holder receives: <issuer-jwt>~<disclosure1>~<disclosure2>~...

Presentation Process:

39. Holder selects claims to reveal
40. Holder includes only selected disclosures
41. Verifier can verify revealed claims
42. Hidden claims remain cryptographically committed but unrevealed

Example:

```
// Issuance - all claims disclosable
const sdJwtVc = await issueSDJwtVC(
  issuer,
  holder,
  {
    projectId: "catalyst-fund-14",    // Always disclosed
    contributionType: "code",          // Disclosable
    hours: 42,                         // Disclosable
    organization: "Prisma",            // Disclosable
  },
  {
    disclosable: ["contributionType", "hours", "organization"],
  }
);

// Presentation - reveal only contributionType
const presentation = await createPresentation(sdJwtVc, ["contributionType"]);
// Verifier sees: projectId, contributionType
// Verifier does NOT see: hours, organization
```

7.4 JWT ID (jti) Requirement

All SD-JWT credentials MUST include a jti claim in URN UUID format:

```
const credential = {
  iss: issuerDid,
  sub: holderDid,
  jti: `urn:uuid:${crypto.randomUUID()}`,  // REQUIRED: URN UUID format
  iat: Math.floor(Date.now() / 1000),
  vct: "ContributionCredential",
  // ... other claims
};
```

jti Format Requirements:

- MUST be a URN UUID: urn:uuid:<uuid-v4>
- MUST use lowercase hexadecimal characters
- Example: urn:uuid:550e8400-e29b-41d4-a716-446655440000

The jti is used for:

- On-chain anchoring reference (vcHash field)
- Revocation lookup
- Credential identification across presentations

extractJti() Implementation:

```
function extractJti(sdJwtVc: string): string {
  // 1. Split SD-JWT into parts (jwt~disclosure1~disclosure2~...)
  const jwtPart = sdJwtVc.split("~")[0];

  // 2. Decode JWT payload (base64url)
  const [, payloadB64] = jwtPart.split(".");
  const payload = JSON.parse(
    Buffer.from(payloadB64, "base64url").toString("utf8")
  );

  // 3. Validate jti format
  if (!payload.jti) {
    throw new Error("SD-JWT missing required jti claim");
  }

  const jtiPattern =

/^urn:uuid:[0-9a-f]{8}-[0-9a-f]{4}-4[0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$
/i;
  if (!jtiPattern.test(payload.jti)) {
    throw new Error("jti must be URN UUID format: urn:uuid:<uuid-v4>");
  }

  return payload.jti;
}
```

8. VC Anchoring and Revocation

8.1 VC Event Schema

VC events are stored under label L_VC (199675):

```
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 for verification
  validatorDid?: string;  // For validate events
  reason?: string;        // For revoke events
}
```

Field Requirements:

Field	Requirement	Description
event	REQUIRED	One of: issue, validate, revoke
issuerDid	REQUIRED	DID of credential issuer
holderDid	REQUIRED	DID of credential holder
vcHash	REQUIRED	Credential identifier (see §8.2)
vcType	REQUIRED	Credential type name
vcFormat	REQUIRED	Format identifier for verification rules
validatorDid	OPTIONAL	Required only for validate events
reason	OPTIONAL	Recommended for revoke events

8.2 vcHash Strategy

The vcHash field serves as the credential identifier for on-chain operations:

Format	vcHash Value	Rationale
SD-JWT	jti claim (see §7.4)	Preserves selective disclosure
Ed25519	SHA-256 of credential	Simple content hash
BBS+ (future)	Commitment hash	Preserves unlinkability

Ed25519 vcHash Computation:

For Ed25519-signed credentials, the vcHash MUST be computed over the complete signed credential, including the proof object with the signature. This ensures:

- The hash uniquely identifies the specific signed instance
- Tampering with any field (including the signature) changes the hash
- Issuance anchoring commits to the exact credential issued

Algorithm:

43. Canonicalize the complete credential JSON (including proof) using RFC 8785 (JSON Canonicalization Scheme)
44. Hash the canonicalized UTF-8 bytes using SHA-256
45. Encode the hash as lowercase hexadecimal

```
import { canonicalize } from "json-canonicalize";
import { createHash } from "crypto";

function computeEd25519VcHash(credential: object): string {
  // 1. Canonicalize (deterministic JSON ordering)
  const canonicalJson = canonicalize(credential);

  // 2. Hash with SHA-256
  const hash = createHash("sha256").update(canonicalJson,
"utf8").digest("hex");

  // 3. Return lowercase hex (64 characters)
  return hash.toLowerCase();
}
```

8.3 On-Chain Event Examples

Issuance:

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

Revocation:

```
{
  "199675": {
    "event": "revoke",
    "issuerDid": "did:cardano:stake1u9...",
    "holderDid": "did:cardano:stake1ux...",
    "vcHash": "urn:uuid:550e8400-e29b-41d4-a716-446655440000",
    "vcType": "ContributionCredential",
    "vcFormat": "sd-jwt",
    "reason": "Duplicate credential issued in error"
  }
}
```

8.4 Revocation Verification

Verifiers MUST check revocation status before accepting a credential:

```
async function verifyCredential(presentation: string): Promise<boolean> {
  // 1. Extract jti from SD-JWT
  const jti = extractJti(presentation);

  // 2. Check on-chain revocation status
  const status = await resolver.getVCStatus(jti);
  if (status.revoked) {
    return false; // Credential has been revoked
  }

  // 3. Verify cryptographic signature
  return await verifySignature(presentation);
}
```

9. Indexer Services

9.1 Architecture

The indexer layer is split to match the DID/VC separation:

DID Indexer (Global)

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

- Indexes L_DID (199674) only from Cardano
- Validates DID event chains and signatures
- Caches resolved DID Documents
- Provides REST API for DID resolution
- Operator: Prisma (ecosystem infrastructure)
- Deployment: Single canonical instance at https://did-indexer.prisma-dids.io

VC Indexer (Forkable)

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

- Indexes organization-specific VC events (L_VC or custom labels)
- Configurable metadata labels and credential schemas
- Queries global DID Indexer for issuer verification
- Provides REST API for VC status queries
- Operator: Each organization runs their own
- Deployment: Multiple instances (one per organization)

9.2 Database Schema

did_events table:

Column	Type	Description
id	UUID	Primary key
did	TEXT	DID identifier
tx_hash	TEXT	Transaction hash (unique)
action	TEXT	create, update, revoke
version	INTEGER	Event version number
prev_tx_hash	TEXT	Previous transaction reference
ipfs_cid	TEXT	IPFS CID of DID Document
valid	BOOLEAN	Validation result
block_height	BIGINT	Block number
timestamp	TIMESTAMP	Block time

vc_events table (VC Indexer):

Column	Type	Description
id	UUID	Primary key
tx_hash	TEXT	Transaction hash (unique)
event	TEXT	issue, validate, revoke
issuer_did	TEXT	Issuer DID
holder_did	TEXT	Holder DID
validator_did	TEXT	Validator DID (for validate only)
vc_hash	TEXT	Credential identifier (jti)
vc_type	TEXT	Credential type
vc_format	TEXT	sd-jwt, ed25519, bbs
reason	TEXT	Revocation reason (optional)
block_height	BIGINT	Block number
timestamp	TIMESTAMP	Block time

9.3 REST API

DID Indexer API (Global)

Endpoint	Method	Description
/did/:did	GET	Resolve DID to current document
/did/:did/history	GET	Full event chain for DID
/health	GET	Indexer sync status

VC Indexer API (Per-Organization)

Endpoint	Method	Description
/vc/:vcHash	GET	VC anchor events (issue/validate/revoke)
/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 schemas
/health	GET	Indexer health and sync status

Pagination Parameters:

Endpoints returning lists SHOULD support pagination:

Parameter	Type	Default	Description
limit	integer	50	Maximum items per page (1-100)
offset	integer	0	Number of items to skip
order	string	desc	Sort order: asc or desc

Example: GET /issuer/:did/credentials?limit=20&offset=40&order=asc

Response Codes:

Code	Status	Description
200	OK	Request successful
400	Bad Request	Invalid parameters or malformed DID
404	Not Found	DID or credential not found
410	Gone	DID has been revoked (deactivated)
500	Internal Server Error	Server error

VC Status Enum:

The status field in VC responses MUST be one of:

Value	Description
active	Credential is valid and not revoked
revoked	Credential has been revoked by issuer
unknown	Credential not found in registry

Example Response - DID Resolution:

```
{
  "did": "did:cardano:stake1u9...",
  "document": {
    "@context": ["https://www.w3.org/ns/did/v1"],
    "id": "did:cardano:stake1u9...",
    "verificationMethod": [...]
  },
  "metadata": {
    "created": "2025-01-15T10:30:00Z",
    "updated": "2025-01-20T14:00:00Z",
    "version": 2,
    "deactivated": false
  }
}
```

Example Response - VC Status:

```
{
  "vcHash": "urn:uuid:550e8400-e29b-41d4-a716-446655440000",
  "status": "active",
  "issuer": "did:cardano:stake1u9...",
  "holder": "did:cardano:stake1ux...",
  "issuedAt": "2025-01-15T10:30:00Z",
  "revokedAt": null
}
```

10. Security Considerations

10.1 Threat Model

Threat	Mitigation
DID Hijacking	Only stake key holder can sign valid events
Event Forgery	Ed25519 signatures verified on resolution
Chain Tampering	Blockchain immutability + prev pointer chain
Replay Attack	Monotonic version numbers prevent replay
Metadata Spam	CIP-10 label filtering; resolver ignores invalid events
Credential Theft	Credentials bound to holder DID via cryptographic proofs

10.2 Signature Verification

All DID events MUST be verified using:

46. Ed25519 signature validation over canonicalized payload (UTF-8 bytes)
47. Stake address derivation from the signing address
48. Controller match verification between derived stake and DID identifier

```
async function verifyDIDEvent(event: DIDEvent): Promise<boolean> {
  // 1. Parse signature wrapper
  const payloadSig = JSON.parse(event.payloadSig);

  // 2. Reconstruct and canonicalize payload
  const payload = canonicalize({
    id: event.id,
    ipfs: event.ipfs,
    action: event.action,
    v: event.v,
    prev: event.prev,
  });

  // 3. Verify Ed25519 signature
  const message = new TextEncoder().encode(payload);
  const valid = await ed25519.verify(
    hexToBytes(payloadSig.sig),
    message,
    hexToBytes(payloadSig.key)
  );
  if (!valid) return false;

  // 4. Derive stake address from signing address
  const stakeFromSigner = deriveStakeAddress(payloadSig.address);

  // 5. Compare with DID stake address
  const stakeFromDid = event.id.replace("did:cardano:", "");
  return stakeFromSigner === stakeFromDid;
}
```

10.3 Private Key Security

- Private keys NEVER leave the user's wallet
- All signing operations use CIP-30 signData() API
- The SDK has no access to private key material

10.4 IPFS Content Integrity

- DID Documents are content-addressed via IPFS CID
- Any modification changes the CID
- On-chain CID serves as commitment to document content

10.5 Credential Verification Requirements

Verifiers MUST perform the following checks before accepting a credential:

1. Revocation Check (REQUIRED):

```
async function verifyCredential(
  presentation: string,
  vcFormat: string
): Promise<boolean> {
  // Extract credential identifier
  const vcHash = vcFormat === "sd-jwt"
    ? extractJti(presentation)
    : computeEd25519VcHash(presentation);

  // Check revocation status - MUST fail if revoked
  const status = await resolver.getVCStatus(vcHash);
  if (status.status === "revoked") {
    return false;
  }

  // Continue with cryptographic verification...
}
```

2. Controller Verification (REQUIRED):

For DID update and revoke operations, implementations MUST verify that:

- The signing address derives to the same stake address as the DID identifier
- The stake address in the signature matches the DID controller
- No delegation of control is permitted (controller MUST equal id for self-sovereign DIDs)

3. Unknown vcFormat Handling (REQUIRED):

Implementations MUST reject credentials with unknown vcFormat values:

```
function getVerifier(vcFormat: string): CredentialVerifier {
  switch (vcFormat) {
    case "sd-jwt":
      return new SDJwtVerifier();
    case "ed25519":
      return new Ed25519Verifier();
    case "bbs":
      return new BBSVerifier();
    default:
      throw new Error(`Unknown vcFormat: ${vcFormat}. Credential rejected.`);
  }
}
```

Rationale: Unknown formats may have different verification rules. Silently accepting them could lead to security vulnerabilities.

10.6 Authorization Rules

Operation	Authorization Requirement
DID Create	Any valid stake key holder
DID Update	MUST be signed by current DID controller
DID Revoke	MUST be signed by current DID controller
VC Issue	Issuer MUST have valid, non-revoked DID
VC Revoke	MUST be signed by original issuer DID
VC Validate	Any valid DID holder can add validation

11. Privacy and Selective Disclosure

11.1 Privacy Levels

Level	Technology	Privacy Guarantee
None	Ed25519 VCs	Full credential disclosed
Claim-level	SD-JWT	Holder chooses which claims to reveal
Unlinkable	BBS+ (future)	Multiple presentations cannot be correlated

11.2 SD-JWT Privacy Properties

- Issuer binding: Only issuer can create valid credentials
- Holder binding: Presentations can be bound to holder DID
- Claim hiding: Undisclosed claims remain cryptographically hidden
- Minimal disclosure: Holder reveals only necessary claims

11.3 Revocation Privacy

The vcHash strategy (§8.2) ensures revocation works from any presentation without revealing undisclosed claims.

11.4 On-Chain Privacy Considerations

On-chain anchoring reveals:

- Issuer DID
- Holder DID
- Credential type
- Issuance/revocation events

On-chain anchoring does NOT reveal:

- Credential content
- Specific claims
- Disclosed vs hidden claims

For maximum privacy, consider:

- Batched anchoring to reduce on-chain footprint
- Delayed anchoring for time-sensitive information
- BBS+ upgrade for unlinkable presentations (future)

12. SDK Reference

12.1 Installation

```
npm install @prisma-dids/sdk
```

12.2 DID Operations

```
import { PrismaSDK } from "@prisma-dids/sdk";

const sdk = new PrismaSDK({
  network: "preprod",  // or "mainnet"
  blockfrostKey: "your-api-key",
  pinataJwt: "your-pinata-jwt",
});

// Create DID
const { did, txHash, ipfsCid } = await sdk.createDID(wallet);

// Update DID
await sdk.updateDID(wallet, did, {
  newPublicKey: "z6Mk...",
});

// Revoke DID
await sdk.revokeDID(wallet, did);

// Resolve DID
const document = await sdk.resolveDID(did);
```

12.3 VC Operations

```
// Issue SD-JWT VC
const { sdJwtVc, jti } = await sdk.issueSDJwtVC(
  issuerWallet,
  holderDid,
  {
    projectId: "catalyst-fund-14",
    contributionType: "code",
    hours: 42,
  },
  { disclosable: ["contributionType", "hours"] }
);

// Create presentation (selective disclosure)
const presentation = await sdk.createPresentation(
  sdJwtVc,
  ["contributionType"]  // Only reveal contributionType
);

// Verify presentation
const { valid, claims } = await sdk.verifyPresentation(presentation);

// Check revocation status
const status = await sdk.checkRevocationStatus(jti);
```

12.4 Anchoring Operations

```
// Anchor VC issuance
await sdk.anchorVCIssuance(wallet, {
  issuerDid,
  holderDid,
  vcHash: jti,
  vcType: "ContributionCredential",
  vcFormat: "sd-jwt",
});

// Revoke VC
await sdk.revokeVC(wallet, {
  issuerDid,
  holderDid,
  vcHash: jti,
  reason: "Issued in error",
});
```

13. Future Roadmap

Note: *This section is non-normative. Future features described here are planned enhancements and do not define conformance requirements. Implementation details may change as these features are developed.*

13.1 BBS+ Signatures

BBS+ signatures will enable:

- Unlinkable presentations: Same credential, multiple presentations cannot be correlated
- Predicate proofs: Prove age > 18 without revealing exact age
- Holder binding: Cryptographic binding to holder key

Implementation Plan:

49. Evaluate BLS12-381 library options
50. Implement issueBBSVC() function
51. Add vcFormat: "bbs" support
52. Maintain SD-JWT compatibility

13.2 NFC Validation

Hardware-based validation for in-person credential presentation:

- NFC-enabled credential display
- Quick verification without internet
- Suitable for events and physical access control

13.3 Plutus Integration

Optional on-chain enforcement via Plutus smart contracts:

- Contract-enforced revocation checks
- Programmable credential policies
- Atomic credential-based transactions

13.4 DIDComm Integration

Peer-to-peer messaging between DID holders:

- Credential offer/request protocols
- Encrypted communication
- Standard DIDComm v2 compatibility

14. References

Standards

53. W3C. "Decentralized Identifiers (DIDs) v1.0." https://www.w3.org/TR/did-core/
54. W3C. "Verifiable Credentials Data Model v2.0." https://www.w3.org/TR/vc-data-model-2.0/
55. IETF. "SD-JWT: Selective Disclosure for JWTs." https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/
56. RFC 8785. "JSON Canonicalization Scheme (JCS)." https://datatracker.ietf.org/doc/html/rfc8785
57. RFC 2119. "Key words for use in RFCs." https://datatracker.ietf.org/doc/html/rfc2119

Cardano Improvement Proposals

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

Related Projects

61. Hyperledger Identus (formerly Atala PRISM). https://hyperledger.github.io/identus-docs/
62. KERI (Key Event Receipt Infrastructure). https://keri.one/

15. Appendix A: Design Rationale

A.1 Why Metadata Over Smart Contracts?

Approach	Pros	Cons
Metadata (chosen)	Simple, cheap, no Plutus complexity	No on-chain enforcement
Smart Contracts	On-chain validation	Higher cost, complexity
Hybrid	Best of both	Increased complexity

Decision:

Metadata was chosen because:

63. DID operations don't require on-chain logic (verification happens off-chain)
64. Cost is 10-100x lower than smart contract execution
65. Simpler developer experience
66. Optional Plutus layer can be added later

A.2 Why Stake Address for DID?

Option	Pros	Cons
Stake address (chosen)	Persistent, independent of UTxOs	Requires stake key
Payment address	Simple	Changes with each UTxO
Custom identifier	Flexible	No cryptographic binding

Decision:

Stake address provides:

- Persistence across wallet operations
- Native cryptographic binding (stake key can sign)
- Alignment with Cardano's identity model

A.3 Why IPFS for Documents?

Option	Pros	Cons
IPFS (chosen)	Content-addressed, decentralized	Requires pinning
On-chain	Fully decentralized	Size limits, high cost
Centralized storage	Simple	Single point of failure

Decision:

IPFS provides:

- Content integrity via CID hashing
- Decentralized availability
- Cost-effective storage for larger documents

A.4 Why SD-JWT Over Alternatives?

Option	Pros	Cons
SD-JWT (chosen)	Industry standard, minimal dependencies	No unlinkability
BBS+	Unlinkable, predicate proofs	Complex, immature libraries
ZKP custom	Maximum flexibility	High complexity, audit burden

Decision:

SD-JWT provides:

- Industry-standard selective disclosure
- Straightforward implementation path
- Clear upgrade path to BBS+ when needed

A.5 Why Not Hyperledger Identus (Prism)?

Identus was evaluated but not adopted because:

- Requires significant infrastructure (Prism Node, IOHK services)
- Higher complexity for our use case
- Our approach is simpler and more cost-effective

However, we align with Identus on:

- W3C standards compliance
- Cardano integration patterns
- SD-JWT adoption

A.6 Why Not KERI?

KERI was evaluated but not adopted because:

- Different paradigm (key event logs, not blockchain-anchored)
- Overkill for current requirements
- Would require significant infrastructure changes

— END OF SPECIFICATION —