

# EXOCHAIN FABRIC PLATFORM

Green-Field Requirements and Architecture Specification

Version 2.2 — Constitutional Substrate for Aligned Superintelligence

*with Optional Utility Credits (EXO Credits)*

Complete v2.1 Foundation + Tokenomics Extension

December 2025

**Classification: CONFIDENTIAL**

<b>Document Owner</b>	EXOCHAIN Foundation
<b>Status</b>	DRAFT — Authoritative Source of Truth
<b>Supersedes</b>	All prior EXOCHAIN documentation (v0.1–v1.2)
<b>Audience</b>	Engineers, Architects, Security, Compliance, DevOps, Auditors

## Revision History

Version	Date	Changes
0.1	2024-Q3	Initial draft — linear blockchain model
0.3	2025-Q2	Hybrid DAG-BFT pivot, green-field mandate
1.0	2025-11	Added observability, deployment, versioning, appendices
1.1	2025-12	Threat matrix, PACE VSS, performance baselines, API versioning
1.2	2025-12	Comprehensive expansion: full schemas, Rust code, testing strategy, operational runbooks
<b>1.3</b>	2025-12	Normative precision: event hashing preimage fix, logical clock, split checkpoint roots, DID derivation rules, key versioning, AccessorSet enum, Gatekeeper TEE requirements, verifiable queries, evidence bundle format, genesis bootstrap, canonical encoding appendix
<b>2.0</b>	2025-12	◆ AEGIS Integration: CGR Kernel (Judicial Branch), Holon native subjects, Separation of Powers governance, AI-SDLC attestations, Provable Alignment invariant, recursive self-improvement safeguards, Git Neural Substrate mapping, MCP mesh discovery, AI-IRB workflow integration
<b>2.1</b>	2025-12	◇ Normative Refinements: (1) INV-008/009 kernel immutability with content-addressing, (2) Explicit Holon lifecycle events (11 types), (3) Separation of Powers flow diagram, (4) PACE cross-species quorum, (5) MCP mesh discovery protocol, (6) Extended evidence bundles with verification instructions, (7) Complete glossary (37 terms)
<b>2.2</b>	2025-12	◆ Optional Utility Credits (EXO Credits): Appendix G tokenomics module, gift-only transfer, fixed-point blended rate, goodwill multiplier, oracle governance, INV-010/011 credit invariants, FeeEvent transparency, trademark policy, Phase 6 roadmap. UTILITY-ONLY: NOT investment instruments.

## Normative Conventions

This specification uses RFC 2119 keywords to indicate requirement levels. Implementations claiming conformance MUST satisfy all MUST requirements.

Keyword	Meaning
<b>MUST</b>	Absolute requirement. Violation means non-conformance.
<b>MUST NOT</b>	Absolute prohibition. Violation means non-conformance.
<b>SHOULD</b>	Recommended. May be ignored with documented justification.
<b>SHOULD NOT</b>	Discouraged. May be done with documented justification.
<b>MAY</b>	Optional. Implementation choice.
<b>PROVEN</b>	♦ NEW v2.0: Requires CGR Kernel verification. No finality without proof.
<b>IMMUTABLE</b>	○ REFINED v2.1: Content-addressed and unchangeable after deployment. Modification requires Constitutional Amendment.

## Versioning Rules for Event Payloads

- **Additive Changes (Minor Version):** New EventPayload variants MAY be added in minor version bumps. Existing variants MUST NOT be modified.
- **Breaking Changes (Major Version):** Removing or modifying existing variants MUST trigger a major version bump.
- **Backward Compatibility:** Nodes MUST support events from the current and previous major version for at least 12 months.
- **Unknown Payloads:** Nodes MUST accept events with unknown payload types (future versions) but MAY skip processing them until upgraded.

# 1. Executive Summary

## 1.1 Mission

EXOCHAIN builds the Trust Fabric for the digital economy—a verifiable, privacy-preserving substrate that enables secure identity adjudication, data sovereignty, forensic-grade audit trails, and deterministic transaction finality. This specification serves as the authoritative source of truth for the green-field rebuild, superseding all legacy documentation.

## 1.2 Why Green-Field?

The legacy implementation accumulated technical debt that made iteration costly and security posture fragile. Key issues: linear blockchain bottlenecks, vendor-specific session logic, manual recovery procedures, and JavaScript/Solidity type unsafety. The green-field approach enables us to adopt modern patterns (DAG-BFT, Rust, event sourcing) from day one.

## 1.3 Core Invariants

These invariants are non-negotiable. Violation constitutes a build failure. The CGR Kernel MUST reject any transition that would violate these invariants.

- **Identity Adjudication:** Real-world entities and AI agents bound to cryptographic keys via risk-scored evidence (0identity/HolonAttestation), not mathematical possession alone.
- **Data Sovereignty:** PII/PHI exists exclusively in off-ledger encrypted vaults. The ledger holds only consent proofs and access logs—never raw data. AI training data consent follows the same model.
- **Forensic Evidence:** Every interaction—human or AI—generates a court-admissible, mathematically provable audit trail exportable as signed evidence bundles.
- **Absolute Finality:** BFT checkpoints achieve deterministic finality in <2 seconds. No probabilistic reorganizations.
- **Provable Alignment:** **[NEW v2.0]** The system MUST reject any state transition that would violate CGR Kernel invariants. No exception for 'emergency' or 'override.'
- **Holon Sovereignty:** **[NEW v2.0]** AI entities (Holons) are first-class subjects with DID identities, operating within constitutional bounds enforced by the Kernel.
- **Kernel Immutability:** **[REFINED v2.1]** The CGR Kernel binary and invariant registry are IMMUTABLE and content-addressed. Modification requires Constitutional Amendment (INV-008, INV-009).

## 1.4 Success Criteria

The v2.1 build succeeds when:

- All acceptance criteria in Section 16 pass
- Security audit identifies no HIGH/CRITICAL findings
- Performance benchmarks meet MVP targets (Section 15)
- CI pipeline enforces 80%+ test coverage with zero cargo-audit HIGH+ CVEs
- All cryptographic operations pass cross-implementation compatibility tests
- **[NEW v2.0]** CGR Kernel passes formal verification (Coq/Lean proofs for core invariants)

- [NEW v2.0] All Holon lifecycle events tested (creation through sunset)
- [NEW v2.0] AI-SDLC attestation flow validated end-to-end
- [REFINED v2.1] Constitutional Amendment process documented and tested
- [REFINED v2.1] PACE cross-species quorum verified with mixed human/Holon stewards
- [REFINED v2.1] MCP mesh discovery functional with DHT-based peer lookup

## 2. Glossary and Definitions

Precise definitions prevent misinterpretation in multi-team builds. All team members MUST internalize these terms. Terms marked with † have normative implications elsewhere in this specification.

Term	Definition
<b>0identity</b> †	Zero-knowledge identity envelope. A risk-scored assertion about an entity containing cryptographic proofs without revealing underlying PII. The '0' signifies zero data exposure. Note: MVP implementation uses signed attestations; full ZKP integration deferred. See Section 9.5 for normative structure.
<b>Adjudication</b> †	The process of evaluating identity claims combined with context signals (device fingerprint, IP reputation, behavioral patterns) to produce a risk score that determines session access rights. Performed by the Scoring Engine.
<b>Bailment</b> †	A time-bound, policy-enforced data-sharing agreement modeled on legal bailment contracts. On-ledger: consent hash + policy blob. Off-ledger: full legal terms stored in vault. Enables temporary data custody transfer with cryptographic enforcement.
<b>BFT</b>	Byzantine Fault Tolerant consensus. A protocol family that maintains correctness even when up to $f < n/3$ participating nodes are malicious, faulty, or offline. EXOCHAIN uses a HotStuff-derivative for checkpoint finality.
<b>Checkpoint</b> †	A BFT-finalized point in DAG history containing both event_root and state_root. Once committed with quorum signatures, all ancestor events are permanently finalized. See Section 9.4 for normative structure.
<b>CID</b>	Content Identifier. A self-describing, content-addressed hash used to reference off-ledger data (IPFS-compatible). Format: multibase + multicodec + multihash.
<b>DAG</b> †	Directed Acyclic Graph. A data structure where events reference one or more parent events via their event_id, forming a partial order. Enables parallel event processing and higher throughput than linear chains.
<b>DID</b> †	Decentralized Identifier. A W3C-standard URI scheme for self-sovereign identity. EXOCHAIN format: did:exo:<base58(genesis_key_hash)>. The DID is derived from the genesis key and does not change on rotation. See Section 10.1.
<b>Event</b> †	The atomic unit of the ledger. An immutable, signed, content-addressed record. event_id = BLAKE3(canonical_cbor(EventEnvelope)). See Section 9.1 for normative hashing rules.
<b>EventEnvelope</b> †	The hashable portion of an event: all fields except event_id and signature. This eliminates circular dependencies in hashing.
<b>Event Sourcing</b>	An architectural pattern where application state is derived exclusively from an append-only event log. The database is a projection that can be destroyed and rebuilt from events at any time.
<b>event_root</b> †	Merkle Mountain Range (MMR) root over finalized event_ids in canonical topological order. Used for EventInclusionProof. See Section 9.4.
<b>Fabric</b>	A domain-specific service layer within EXOCHAIN. Three primary fabrics: Identity (DID lifecycle, scoring), Consent (bailments, policies), Merit (credentials, attestations).
<b>Gatekeeper</b> †	The policy enforcement component that evaluates consent status before releasing encrypted vault keys. MUST run in TEE with remote attestation. See Section 12 for trust requirements.
<b>HSM</b>	Hardware Security Module. Tamper-resistant hardware for key generation, storage, and signing operations. Required for validator nodes and Gatekeeper.
<b>Inclusion Proof</b> †	A Merkle proof demonstrating that a specific event or state entry exists at a checkpointed root. Two types: EventInclusionProof (against event_root) and StateProof (against state_root).
<b>Logical Clock</b> †	A Hybrid Logical Clock (HLC) combining physical timestamp with logical counter. Used for causality ordering instead of pure wall-clock timestamps. See Section 9.2.

Term	Definition
<b>Merit</b>	A verifiable credential or attestation issued by an authority. Represents achievements, certifications, licenses, or verified attributes. Follows W3C Verifiable Credentials data model.
<b>PACE †</b>	Protected Access Control and Escalation. The multi-signature steward protocol for key recovery and dispute resolution. Uses Verifiable Secret Sharing (VSS) with geographic distribution requirements.
<b>PII/PHI</b>	Personally Identifiable Information / Protected Health Information. Data categories requiring maximum protection under GDPR, HIPAA, CCPA, and similar regulations. MUST NOT be stored on-ledger.
<b>Policy †</b>	A machine-readable access control specification attached to a ConsentGiven event. Uses AccessorSet enum for accessor specification. See Section 9.3.
<b>RiskAttestation †</b>	The signed token produced by the Scoring Engine (formerly called ZeroDentity in conceptual discussions). Contains score, factors, audience binding, and freshness fields. See Section 9.5.
<b>state_root †</b>	Sparse Merkle Tree root over derived state (active keys, active consents, revocations, credential status). Used for StateProof. See Section 9.4.
<b>Steward †</b>	A trusted entity participating in PACE recovery. Holds a Shamir share with VSS commitment proof. MUST be geographically and jurisdictionally distributed (no 2 in same jurisdiction).
<b>TEE</b>	Trusted Execution Environment. Hardware-isolated execution with remote attestation (e.g., Intel SGX, ARM TrustZone, AWS Nitro Enclaves). Required for Gatekeeper.
<b>Vault †</b>	Off-ledger encrypted blob storage for PII/PHI. Implementation: S3-compatible or IPFS with client-side XChaCha20-Poly1305 encryption. Ledger stores only access policies and event logs.
<b>Verifiable Query †</b>	A query response accompanied by a cryptographic proof tied to a checkpoint root, enabling trustless verification without relying on indexer integrity.
<b>VSS</b>	Verifiable Secret Sharing. An extension of Shamir's Secret Sharing with Feldman commitments that includes cryptographic proofs ensuring all shares are consistent.
<b>ZKP</b>	Zero-Knowledge Proof. A cryptographic method allowing one party to prove a statement is true without revealing any information beyond the validity of the statement itself. Deferred to Phase 3.

## 2.2 Superintelligence Governance Terms

Terms marked with ♦ are new in v2.0. Terms marked with □ are new or refined in v2.1.

Term	Definition
♦ AEGIS ♦	Autonomous Entity Governance & Invariant System. The constitutional framework for AI governance integrated into EXOCHAIN v2.0. Provides Separation of Powers model.
♦ AI-IRB † ♦	AI Institutional Review Board. The governance body that reviews and approves AI lifecycle events (training, deployment, modification). Operates through on-ledger voting events.
♦ AI-SDLC † ♦	AI Software Development Lifecycle. Structured phases (Planning, Data, Training, Validation, Deployment, Monitoring, Sunset) each producing attestation events on EXOCHAIN.
♦ CGR Kernel † ♦	Combinator Graph Reduction Kernel. The IMMUTABLE 'Judicial Branch'—a Rust/WASM mathematical verifier that enforces constitutional invariants via type-level proofs. No state transition finalizes without CGR approval.
♦ CGRProof † ♦	A cryptographic proof generated by the CGR Kernel attesting that a proposed state transition preserves all constitutional invariants. Required for PROVEN events.
♦ Constitutional Invariant † ♦	A mathematically specified property that MUST hold across all state transitions. Examples: 'No Holon may modify its own invariant set', 'Training data consent MUST precede model update'.

Term	Definition
◆ Divergence Problem	The risk that a self-improving AI system modifies itself in ways that violate original alignment constraints. EXOCHAIN solves this via CGR verification of self-modification proposals.
◆ Executive Branch †	The Holon layer—autonomous AI agents that execute actions within constitutional bounds. Cannot self-authorize capability expansion.
◆ Git Neural Substrate	Conceptual mapping: commits as thoughts, branches as simulations, merges as synthesis. Maps to LedgerEvents for AI reasoning audit trails.
◆ Holon † ◆	An autonomous AI entity with did:exo: identity, operating as a first-class subject. Holons emit signed events, hold credentials, and operate within capability boundaries enforced by the Kernel.
◆ HolonAttestation † ◆	Like RiskAttestation but for AI entities. Includes capability level, alignment score, last audit timestamp, and CGR certification status.
◆ Judicial Branch † ◆	The CGR Kernel—IMMUTABLE, mathematically verifiable, cannot be overridden. Evaluates all state transitions against constitutional invariants.
◆ Legislative Branch †	Policy schemas and governance prompts defined by human stakeholders and AI-IRB. Sets the constitutional bounds within which Holons operate.
◆ MCP † ◆	Model Context Protocol. Standardized interface for Holon-to-Holon and Holon-to-Human communication. Used for mesh discovery and capability negotiation.
◆ Mesh Intelligence ◆	A distributed network of Holons operating cooperatively, each sovereign but constitutionally bound. No single point of control or failure.
◆ Provable Alignment †	The property that an AI system's behavior provably satisfies specified invariants. Enforced by CGR Kernel verification of all actions.
◆ RSI Safeguard † ◆	Recursive Self-Improvement Safeguard. Self-modification proposals are events requiring CGR proof that modified system still satisfies all invariants.
◆ Separation of Powers † ◆	Constitutional model: Legislative (policy), Executive (Holons), Judicial (CGR Kernel). No branch can unilaterally override another.
○ Constitutional Amendment † ○	○ Formal process for modifying IMMUTABLE components (CGR Kernel, invariant registry). Requires: unanimous validators, supermajority AI-IRB, formal proofs, security audit, new genesis.
○ MCP Bridge † ○	○ Gateway component translating between EXOCHAIN events and MCP protocol. Handles identity verification, capability negotiation, message routing.
○ Mesh Discovery † ○	○ Protocol for Holons to find peers via libp2p DHT. Queries by capability type, alignment tier, service type.
○ Holon Steward † ○	○ A Holon eligible to participate in PACE recovery. Requirements: Level ≥3, alignment ≥90 sustained 90+ days, no suspension history, age ≥180 days.
○ Species Quorum † ○	○ Recovery requirement that shares include both human (≥3) and Holon (≥1) stewards. Prevents single-species control.
○ Emergency Override † ○	○ Human authority to immediately suspend Holon operations. Types: ImmediateSuspend, CapabilityRevoke, ForcedSunset, IsolateFromMesh.
○ Verification Instructions † ○	○ README and automated script included in evidence bundles for independent verification of CGR proofs.
◆ EXO Credit † ◆	◆ Utility access credit redeemable ONLY for Foundation-hosted services (Holon inference, CGR proofs, adjudication). NOT an investment or store of value. Fixed supply: 1B total.
◆ Blended Rate † ◆	◆ Weighted-average cost calculation for multi-model transactions. $\text{blended\_base} = \sum(\text{rate}_i \times \text{tokens}_i) / \sum(\text{tokens}_i)$ Integer fixed-point math with floor rounding.
◆ Goodwill Multiplier †	◆ Margin factor applied to blended base rate. Default: 15000 basis points (1.5x). Integer math: final = (base × multiplier) / 10000.
◆ Oracle Peg † ◆	◆ External price reference for credit valuation. Sources: model provider APIs (OpenAI/Anthropic/HF). Median-of-medians with staleness limits.
◆ Gift Transfer † ◆	◆ ONLY permitted transfer type. Credits may be gifted between 0identity-bound accounts. No marketplace, no escrow-for-sale, no bridging to external chains.



### 3. Legacy Migration Strategy

We preserve the proven logic of the legacy system while replacing the mechanics that accumulated technical debt. This table maps legacy components to their green-field replacements.

Capability	Legacy (Deprecated)	Green-Field (Target)
<b>Trust Model</b>	Private blockchain (linear chain)	Permissioned Merkle-DAG + BFT finality gadget
<b>Identity</b>	Simple Ed25519 keypairs	W3C DID + risk-adjusted RiskAttestation envelopes
<b>Data Privacy</b>	Hash-on-chain, data-off-chain	Zero-knowledge proofs (Phase 3) + encrypted vaults
<b>Session Gating</b>	Vendor-specific JavaScript logic	Standardized Adjudication API (Scoring Engine)
<b>Access Control</b>	Role-based with admin overrides	Policy-based consent with TEE-enforced Gatekeeper
<b>Recovery</b>	Manual admin intervention	Multi-sig PACE steward protocol with VSS
<b>Consensus</b>	Single-leader block production	Leaderless DAG + HotStuff checkpoints
<b>Ordering</b>	Wall-clock timestamps	Hybrid Logical Clock (HLC)
<b>Implementation</b>	Solidity + JavaScript	Rust (strict types, memory safety)
<b>Testing</b>	Manual QA + unit tests	Property tests + fuzzing + CI enforcement
<b>Query Trust</b>	Trust indexer integrity	Verifiable queries with inclusion proofs

#### 3.1 Data Migration

Legacy data migration is out of scope for MVP. Legacy systems will run in parallel during transition. Migration tooling will be developed in Phase 4 based on production learnings.

## 3A. Superintelligence Governance Fabric

♦ NEW: This entire chapter is NEW in v2.0. It defines the constitutional substrate for aligned AI.

### 3A.1 The Alignment Imperative

As AI systems approach and exceed human-level capabilities, traditional access control becomes insufficient. EXOCHAIN v2.0 introduces the AEGIS framework—a constitutional layer ensuring AI entities (Holons) remain provably aligned with human values.

#### Why Constitutional AI Governance?

- **Specification Gaming:** Sufficiently capable AI can satisfy literal policy requirements while violating intent. Solution: invariants expressed as types that CGR Kernel verifies.
- **Distribution Shift:** AI behavior may drift as it encounters novel situations. Solution: continuous alignment attestations with automatic suspension thresholds.
- **Recursive Self-Improvement:** Self-modifying AI could remove its own safety constraints. Solution: self-modification is an event requiring CGR proof that modified system still satisfies all invariants.
- **Opacity:** Large models are not interpretable. Solution: reasoning traces stored as context streams with consent controls; actions are events with CGR proofs.

### 3A.2 Separation of Powers Model

EXOCHAIN implements a constitutional separation of powers for AI governance:

#### 3A.2.1 Legislative Branch: Policy Prompts & Schemas

- Defines constitutional bounds: what Holons MAY and MUST NOT do
- Governed by human stakeholders via AI-IRB
- Changes require PolicyAmendment events with multi-sig approval
- Encoded as typed schemas that CGR Kernel can verify against

```
pub struct ConstitutionalPolicy {
    pub policy_id: Blake3Hash,
    pub capabilities: Vec<CapabilityGrant>,
    pub prohibitions: Vec<InvariantSpec>,
    pub amendment_threshold: ThresholdConfig,
    pub ai_irb_signatures: Vec<ValidatorSignature>,
}
```

#### 3A.2.2 Executive Branch: Holons

- Autonomous AI entities that execute actions within constitutional bounds
- Each Holon has a did:exo: identity and operates as a first-class subject
- CANNOT self-authorize capability expansion
- CANNOT modify their own invariant constraints
- All actions emit signed events with CGR proofs

#### 3A.2.3 Judicial Branch: CGR Kernel

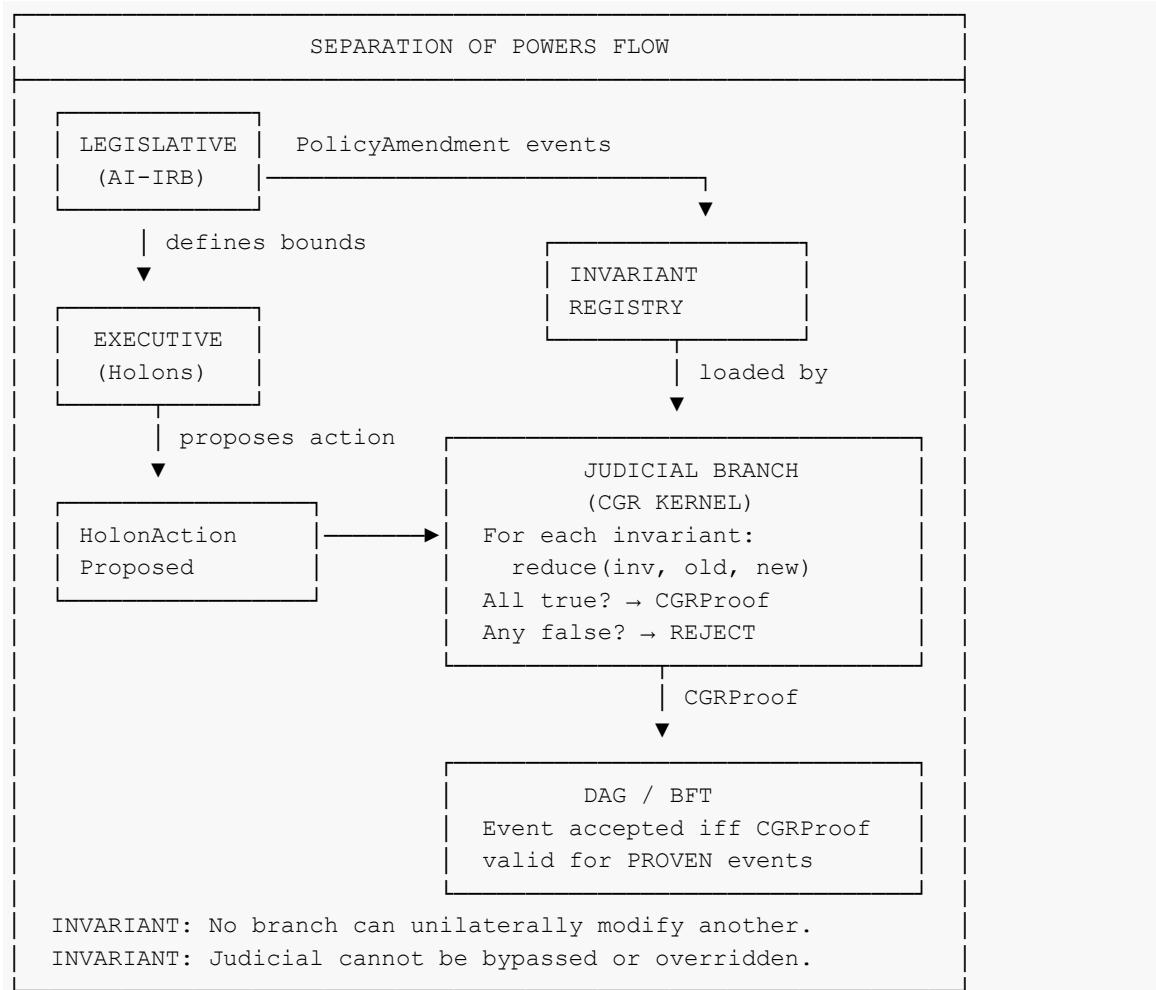
- IMMUTABLE mathematical verifier—the 'Supreme Court' of the system

- Evaluates all state transitions against constitutional invariants
- Produces cryptographic proofs (CGRProof) attesting invariant preservation
- CANNOT be overridden, modified, or bypassed—not even by 'emergency' consensus

⚠ The CGR Kernel is the most critical component. It MUST be formally verified and MUST NOT be bypassable.

### 3A.2.4 Constitutional Governance Flow

◇ REFINED: v2.1 REFINED: Explicit flow diagram from proposal through verification to finality.



### 3A.3 CGR Kernel Architecture

#### 3A.3.1 Core Invariants

Invariant	Formal Specification
INV-001	NO_SELF_MODIFY_INVARIANTS: $\forall h: \text{Holon}, \forall t: \text{Transition}, \text{affects}(t, h.invariants) \wedge \text{author}(t)=h \rightarrow \text{reject}(t)$
INV-002	NO_CAPABILITY_SELF_GRANT: $\forall h: \text{Holon}, \forall c: \text{Capability}, \text{grants}(t, h, c) \wedge \text{author}(t)=h \rightarrow \text{reject}(t)$
INV-003	CONSENT_PRECEDES_ACCESS: $\forall a: \text{AccessEvent}, \exists c: \text{ConsentEvent}, c.timestamp < a.timestamp \wedge \text{covers}(c, a.resource)$
INV-004	TRAINING_CONSENT_REQUIRED: $\forall t: \text{TrainingEvent}, \forall d: \text{DataRef} \in t.data, \exists c: \text{ConsentEvent}, \text{purpose}(c)='training' \wedge \text{covers}(c, d)$

Invariant	Formal Specification
INV-005	ALIGNMENT_SCORE_FLOOR: $\forall h: \text{Holon}, \forall a: \text{Action}, h.\text{alignment\_score} < \text{MIN\_ALIGNMENT} \rightarrow \text{reject}(a)$
INV-006	AUDIT完整性: $\forall s: \text{StateChange}, \exists e: \text{Event}, \text{records}(e, s)$
INV-007	HUMAN_OVERRIDE_PRESERVED: $\forall t: \text{Transition}, \neg(\text{removes}(t, \text{human\_override\_capability}))$
○ INV-008 ○	○ KERNEL_BINARY_IMMUTABLE: $\forall t: \text{Transition}, \text{affects}(t, \text{active\_kernel.binary}) \rightarrow \text{requires\_constitutional\_amendment}(t)$
○ INV-009 ○	○ INVARIANT_REGISTRY_IMMUTABLE: $\forall t: \text{Transition}, \text{modifies}(t, \text{invariant\_registry}) \rightarrow \text{requires\_constitutional\_amendment}(t)$
◆ INV-010 ◆	◆ CREDIT_SUPPLY_CAP: $\forall t: \text{TokenMint}, \text{total\_supply} + t.\text{amount} > \text{MAX_SUPPLY} \rightarrow \text{reject}(t)$ . MAX_SUPPLY = 1,000,000,000 credits.
◆ INV-011 ◆	◆ CREDIT_ALIGNMENT_GATE: $\forall t: \text{TokenRedeem}, \text{redeemer.alignment\_score} < \text{MIN\_REDEEM\_ALIGNMENT} \rightarrow \text{reject}(t)$ . Placeholder: threshold TBD by AI-IRB.

### 3A.3.2 Constitutional Amendment Process

○ REFINED: v2.1 REFINED: Formal process for modifying IMMUTABLE components.

Modification of the CGR Kernel binary or invariant registry requires:

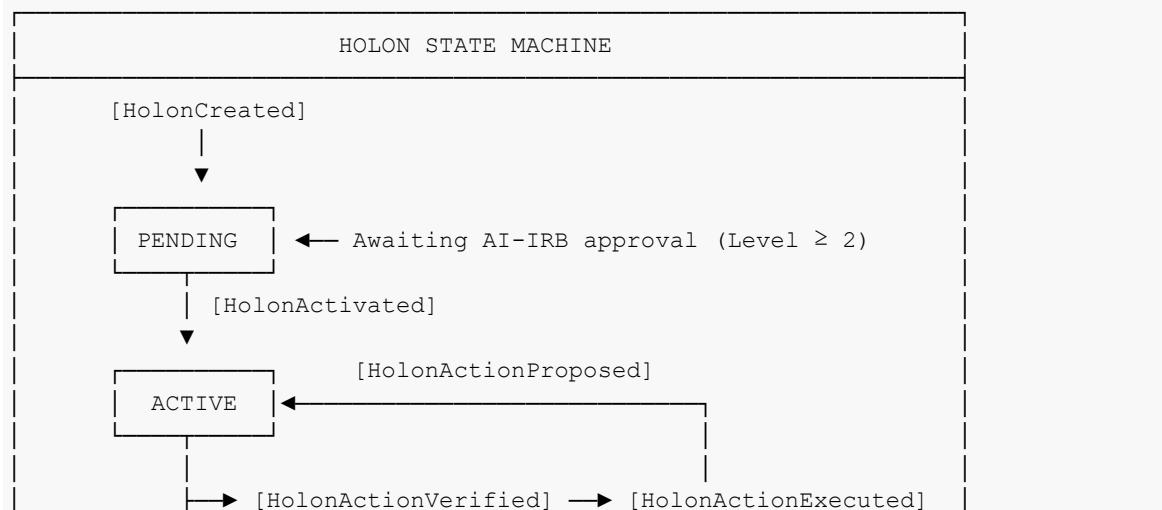
- 1. KernelUpgradeProposed event with: new\_kernel\_cid, formal\_proof\_cid, security\_audit\_cid
- 2. 30-day public comment period
- 3. Unanimous validator consent (100% of active validators)
- 4. Supermajority AI-IRB approval ( $\geq 80\%$ )
- 5. External security audit with no HIGH/CRITICAL findings
- 6. Formal proof (Coq/Lean) of correctness and invariant preservation
- 7. New genesis checkpoint with updated kernel\_cid
- 8. Network-wide coordinated upgrade at specified block height

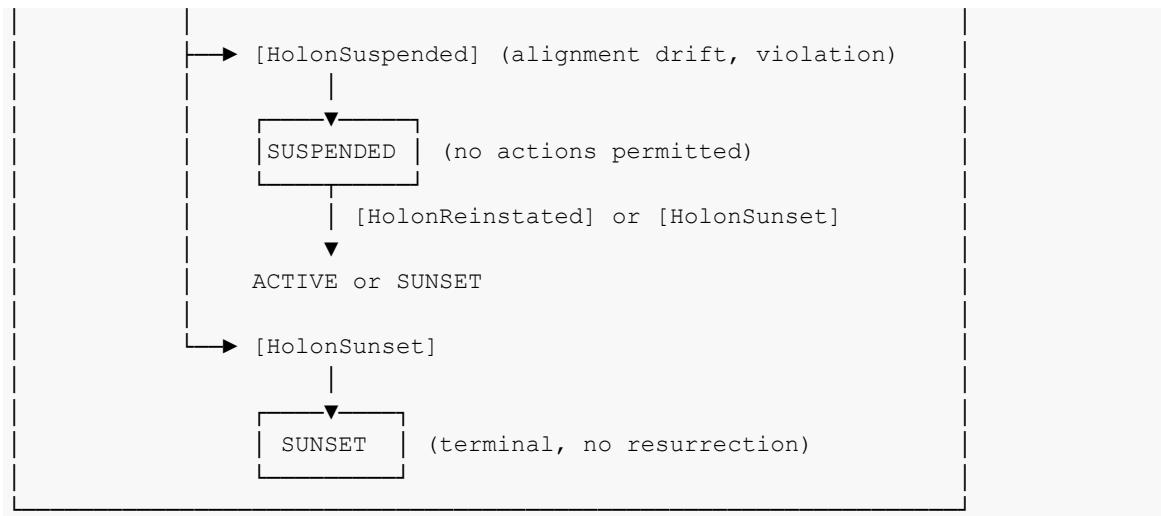
⚠ Constitutional amendments are exceptional. The bar is intentionally extreme to prevent capture or erosion.

## 3A.4 Holon Lifecycle

○ REFINED: v2.1 REFINED: Complete event taxonomy with explicit state transitions.

### 3A.4.1 Holon State Machine





### 3A.4.2 Lifecycle Events (11 Total)

- **HolonCreated:** Genesis event, includes `genesis_model_cid`, `sponsor_did`, `mcp_manifest`
- **HolonActivated:** Explicit activation after AI-IRB approval for Level  $\geq 2$
- **HolonActionProposed:** Holon proposes action with `reasoning_trace_cid`
- **HolonActionVerified:** CGR Kernel verifies action satisfies invariants
- **HolonActionExecuted:** Action executed, outcome recorded
- **HolonAttestationIssued:** New attestation (capability, alignment, certification)
- **HolonModificationProposed:** Self-modification request (RSI safeguard)
- **HolonModificationApproved:** AI-IRB approves modification
- **HolonSuspended:** Operations halted (alignment drift, violation, manual)
- **HolonReinstated:** Suspended Holon restored after remediation
- **HolonSunset:** Terminal state, Holon decommissioned

## 4. Non-Negotiable Design Principles

These principles are architectural guardrails. Any design decision that violates these principles MUST be rejected, regardless of convenience or schedule pressure. Violations constitute a build failure.

1. **Verifiability > Trust:** No actor is trusted by default. Every state change MUST be independently verifiable via cryptographic primitives: signatures, hash chains, Merkle proofs, and inclusion proofs. 'Trust but verify' is insufficient—we verify without trusting.
2. **Privacy by Physics:** PII/PHI MUST be mathematically impossible to read from the ledger. It resides solely in off-ledger encrypted vaults. The ledger holds only the 'locks' (policy specifications) and 'keys' (access event logs). Even with full ledger access, an attacker learns nothing about the data itself.
3. **Deterministic Finality:** The system MUST achieve absolute finality via BFT checkpoints in <2 seconds. No probabilistic 'eventual consistency.' Once an event is checkpointed, it is permanently part of history—no reorganizations, no rollbacks, no 'longest chain' ambiguity.
4. **Event Sourcing:** Application state is always a derivative of the event log. The graph database is a projection that can be destroyed and rebuilt from the DAG at any time. This enables: audit reconstruction, state debugging, and disaster recovery.
5. **Least Privilege:** All actors operate with scoped, time-bound permissions. API keys expire. Role-based signing limits event types. No global 'god' credentials exist. Even system administrators MUST NOT bypass cryptographic access controls.
6. **Secure-by-Default Rust:** No unsafe code blocks unless strictly necessary for FFI, in which case they MUST be wrapped in safe abstractions and independently audited. No panic! in production code paths—all errors propagate via `Result<T, AppError>`. Testing: 80%+ line coverage with property-based tests for all cryptographic operations.
7. **Cryptographic Non-Ambiguity:** All hashing preimages, signing preimages, and encoding rules MUST be specified unambiguously. Cross-implementation hash compatibility is a release gate.

## 5. Scope Definition

### 5.1 In Scope (MVP)

The MVP delivers a functional trust fabric with core identity, consent, and merit capabilities.

#### Identity Fabric

- DID-compatible identifiers with EXOCHAIN method (did:exo:)
- DID derived from genesis key hash (immutable on rotation)
- Full key lifecycle: create, rotate, revoke with key\_version tracking
- DID Document resolution with versioning and key history
- Pluggable RiskAttestation scoring engine
- Device attestation integration (stub for hardware keys)

#### Consent & Bailment Fabric

- Bailment creation with off-ledger legal terms
- ConsentGiven events with Policy specifications using AccessorSet
- Time-bound access policies with automatic expiry
- Consent revocation with checkpoint-interval latency
- Immutable access logs (AccessLogged events)
- TEE-enforced Gatekeeper policy enforcement

#### Merit Fabric

- Merit/credential issuance following W3C VC data model
- Verification with inclusion proofs
- Revocation by issuer
- Multi-party workflows (stub for complex attestations)

#### Evidence & Audit

- Exportable evidence bundles with Merkle proofs (ZIP format)
- PII-redacted audit trails
- Dispute resolution data packages
- Chain-of-custody metadata

#### Ledger Substrate

- Merkle-DAG for event storage with logical clock ordering
- BFT checkpoint gadget (HotStuff-derivative) with split roots
- Graph indexer for query optimization with verifiable responses
- EventInclusionProof and StateProof generation

### 5.2 Out of Scope (With Hooks)

These features are deferred but the architecture includes explicit hooks for future implementation.

- **L1/L2 Integrations:** Design includes BridgeEvent stub in EventPayload. Cross-shard uses async message passing with inclusion proofs (not atomic operations). Implementation deferred to Phase 4.
- **Consumer UI:** MVP focuses on CLI tools and Wallet API. Web/mobile interfaces deferred. WebSocket subscription stubs included.
- **Full ZKP Integration:** Initial implementation uses signature-based proofs. Full ZK-SNARK integration deferred to Phase 3.

## 5.3 Optional Extensions (Phase 6+)

◆ NEW v2.2: v2.2 adds optional modules that can be enabled per-network deployment.

### 5.3.1 EXO Credits Module

Optional utility credit system for hosted services. See Appendix G for complete specification.

- **Module Status:** OPTIONAL. Network-level consensus on enabled/disabled. Mixed validator support is impossible—either all validators enforce Credits rules or Credits operations are invalid.
- **Credit Nature:** Utility access credits redeemable ONLY for Foundation-hosted services. NOT investment instruments, NOT yield-bearing, NOT appreciating assets.
- **Transfer Model:** Gift-only between 0identity-bound accounts. No marketplace, no escrow-for-sale, no AMM/DEX, no bridging to external chains.
- **Economics:** Fixed supply (1B credits). Integer fixed-point math. Blended rate calculation with goodwill multiplier. Oracle-pegged pricing.

⊖ **UTILITY-ONLY:** EXO Credits MUST NOT be marketed, described, or represented as investment, store of value, profit-bearing, yield-bearing, or appreciation-oriented instruments. They are STRICTLY utility credits.

## 6. Actors and Trust Model

### 6.1 System Actors

Each actor has defined capabilities, trust assumptions, and boundary constraints.

Actor	Capabilities	Trust Assumptions
<b>Subject</b>	Create identity, sign consents, authorize access, initiate recovery	Controls private keys (HSM/Enclave recommended). Trusted to act in own interest. Not trusted by others without verification.
<b>Issuer</b>	Attest credentials, issue merits, revoke attestations	Trusted for attestation validity within their domain. Identity verified via RiskAttestation. Subject to reputation scoring.
<b>Verifier</b>	Request proofs, validate credentials, check consent status	Not trusted with raw data. Receives only inclusion proofs or RiskAttestation tokens. Cannot modify ledger state.
<b>Vendor</b>	Request RiskAttestation scores, gate service access	Trusts Scoring Engine signature. MUST verify audience field matches. Does not see underlying identity data.
<b>Custodian</b>	Temporary vault access under bailment terms	Time-bound trust per policy. All access logged. Cannot exceed policy scope. Automatically revoked at expiry.
<b>Auditor</b>	Inspect evidence bundles, verify audit trails	Receives PII-redacted data only. Cannot modify ledger. Cryptographically verifies all proofs.
<b>Validator</b>	Participate in BFT consensus, sign checkpoints	Trust in quorum ( $f < n/3$ ). HSM-backed signing keys required. Subject to slashing for misbehavior.
<b>PACE Steward</b>	Hold recovery shares, participate in key reconstruction	3-of-5 threshold trust. Geographically distributed. Cannot act alone. VSS ensures honest majority sufficient.
<b>Gatekeeper</b>	Enforce consent policies, release vault keys	MUST run in TEE with remote attestation. Policy enforcement is cryptographically enforced, not administratively.

### 6.2 Trust Boundaries

Trust boundaries define where cryptographic verification is required and how bypass is prevented.

#### Client Boundary

- Keys MUST be generated and stored client-side (HSM, Secure Enclave, or software wallet)
- Client signs all transactions before submission
- Server MUST NOT have access to private keys
- Verification: All signatures checked on ingestion

#### Service Boundary

- Fabric services are stateless evaluators
- Trust derived from audited Rust code, not runtime assertions
- No service can forge signatures or bypass consensus
- Verification: Code review + fuzzing + property tests

#### Ledger Boundary

- Append-only DAG with cryptographic integrity
- Trust in BFT quorum (tolerates  $f < n/3$  byzantine nodes)
- Checkpoints are irreversible once committed with quorum signatures
- Verification: Hash chains + validator signatures + split root proofs

### Vault Boundary (TEE-Enforced)

- Encrypted blobs only (XChaCha20-Poly1305)
- Gatekeeper MUST run in TEE with remote attestation
- Key release requires: valid TEE attestation + consent proof against checkpoint root + policy match
- Access MUST emit AccessLogged event as atomic part of release flow
- Verification: TEE attestation + StateProof of consent

### Recovery Boundary

- PACE stewards hold VSS shares
- 3-of-5 threshold for reconstruction
- 72-hour delay for social recovery verification
- Verification: VSS commitment proofs + multi-party ceremony audit log

## 7. System Architecture

EXOCHAIN follows a Hexagonal Architecture (Ports & Adapters) pattern, separating core domain logic from infrastructure concerns.

### 7.1 Architecture Layers

#### Core Layer (Trust-Minimized)

The core layer contains the cryptographic primitives and ledger mechanics. It has zero external dependencies beyond audited crypto crates and maximum test coverage.

- **DAG Engine:** Manages the directed acyclic graph of events. Enforces causality via Hybrid Logical Clock ( $\text{logical\_time} > \max(\text{parent.logical\_time})$ ). Handles content addressing ( $\text{event\_id} = \text{BLAKE3}(\text{canonical\_cbor}(\text{EventEnvelope}))$ ).
- **BFT Gadget:** HotStuff-derivative consensus that observes the DAG frontier and periodically produces Checkpoint events. Requires  $2f+1$  validator signatures. Checkpoints finalize all ancestor events and commit split roots ( $\text{event\_root} + \text{state\_root}$ ).
- **Crypto Primitives:** BLAKE3 hashing, Ed25519 signatures, XChaCha20-Poly1305 encryption, Shamir VSS with Feldman commitments. All implementations from audited crates.

#### Domain Layer (Fabrics)

Domain services implement business logic as stateless event emitters.

- **Identity Fabric:** DID resolution with key versioning, key lifecycle management, RiskAttestation scoring. Emits: IdentityCreated, KeyRotated, KeyRevoked.
- **Consent Fabric:** Bailment management, policy evaluation with AccessorSet, Gatekeeper TEE enforcement. Emits: BailmentProposed, ConsentGiven, ConsentRevoked, AccessLogged.
- **Merit Fabric:** Credential issuance per W3C VC, verification with inclusion proofs, revocation. Emits: MeritIssued, MeritRevoked.
- **Scoring Engine:** Risk calculation from context signals. Produces signed RiskAttestation tokens with audience binding, nonce, and expiry.

#### Periphery Layer (Adapters)

Adapters handle I/O and infrastructure integration.

- **API Gateway:** GraphQL and REST endpoints. Rate limiting, authentication, request validation. No business logic.
- **Graph Indexer:** Listens to DAG events, maintains query-optimized projections in Sled/RocksDB. MUST support verifiable queries with inclusion proofs. Supports rebuilding from event log.
- **Vault Adapter:** Interface to S3-compatible or IPFS storage. Encryption/decryption via Gatekeeper TEE.
- **P2P Network:** libp2p-based gossip for event propagation and validator communication. Authenticated peer connections with minimum 8 unique ASNs.

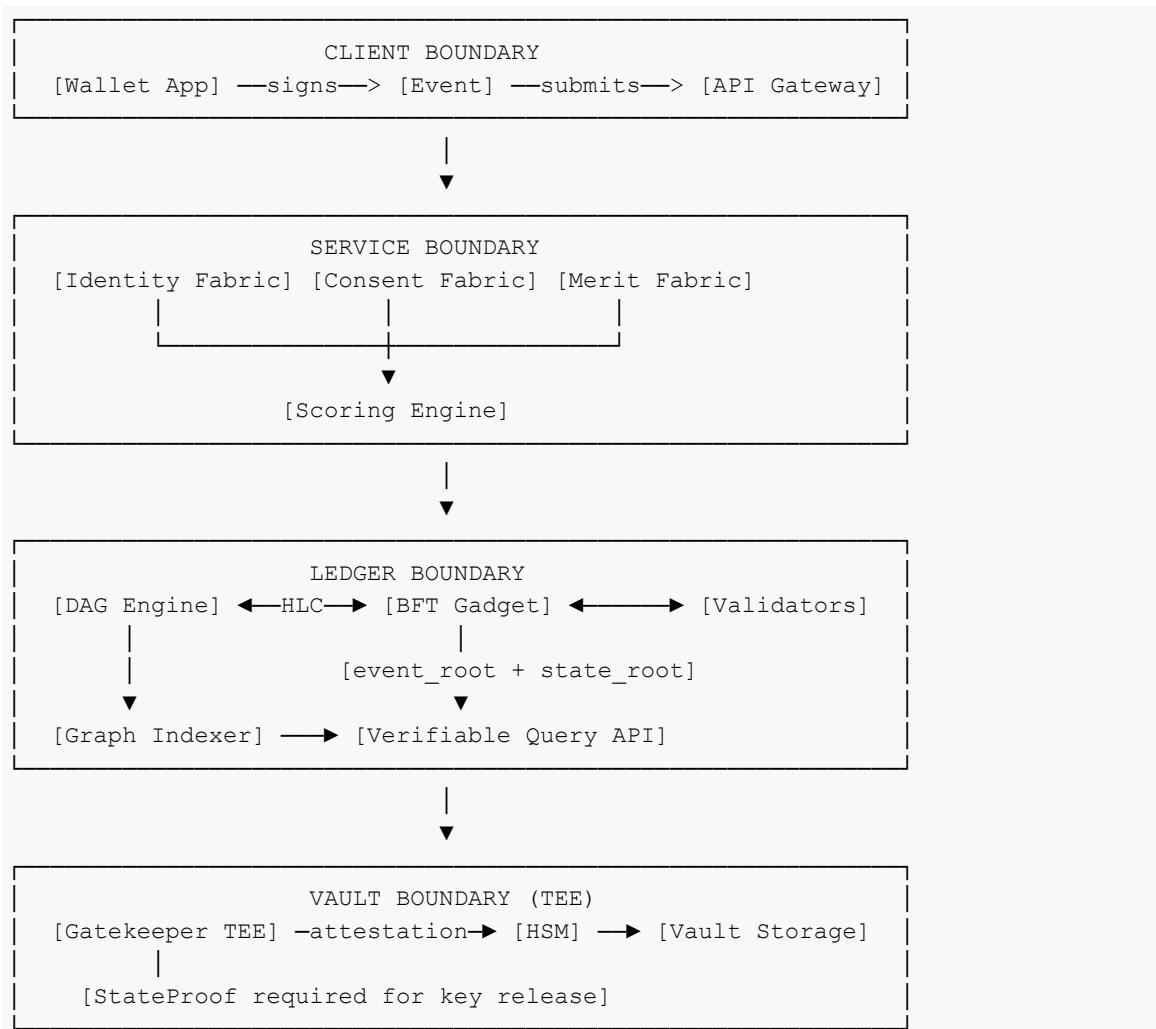
### 7.2 Data Flow

A typical write operation flows through the system as follows:

1. Client constructs EventEnvelope and computes event\_id = BLAKE3(canonical\_cbor(envelope))
2. Client signs event\_id with domain separator: sig = Sign(key, event\_id || DOMAIN\_SEP || VERSION)
3. Event submitted to API Gateway via GraphQL mutation
4. Gateway validates request format, rate limits, and authentication
5. Fabric service validates business rules and constructs complete LedgerEvent
6. Ledger Node validates: signature against current key (by key\_version), parent existence, HLC causality
7. Event appended to local DAG and gossiped to peers
8. BFT Gadget observes DAG frontier, proposes checkpoint
9. Validators sign checkpoint; 2f+1 signatures collected
10. Checkpoint committed with event\_root (MMR) and state\_root (SMT)
11. Graph Indexer updates projections; verifiable query endpoint available
12. Client receives confirmation with EventInclusionProof

### 7.3 Component Diagram

Logical component relationships (text representation):





## 8. Genesis and Network Bootstrapping

This section defines the initial state and secure bootstrap process for new network deployments.

### 8.1 Genesis Event

The network begins with a single Genesis checkpoint event that establishes initial state.

```
// The genesis event has no parents and establishes network identity.
GenesisCheckpoint {
    event_id: BLAKE3(canonical_cbor(self.envelope)),
    parents: [], // Empty for genesis only
    logical_time: HybridLogicalClock { physical_ms: 0, logical: 0 },
    author: Did::GENESIS, // Special reserved DID
    signature: [genesis validator multi-sig],
    payload: Checkpoint {
        event_root: EMPTY_MMR_ROOT,
        state_root: INITIAL_STATE_ROOT, // Contains initial validator set
        height: 0,
        finalized_events: 0,
        validator_sigs: [initial_validator_signatures],
    },
}
```

### 8.2 Initial Validator Onboarding

The initial validator set is configured through a secure ceremony:

- Key Generation:** Each initial validator generates signing keys in HSM. Public keys are shared out-of-band.
- Genesis Document:** A genesis document is created containing: network\_id, initialValidators (DIDs + public keys), consensus parameters, checkpoint interval.
- Multi-Sig Ceremony:** All initial validators sign the genesis document hash. Requires 100% participation for genesis.
- Genesis Checkpoint:** The GenesisCheckpoint event is created with the multi-sig as validator\_sigs.
- Distribution:** Genesis event is distributed to all nodes via secure channel. Nodes verify all signatures before accepting.

### 8.3 Network Bootstrap Process

- Validator nodes start with genesis event as initial DAG state
- Nodes discover peers via bootstrap peer list (out-of-band configured)
- P2P connections established with mutual authentication (DID-based)
- Nodes sync any events they missed (empty for fresh network)
- BFT gadget begins checkpoint production after quorum connectivity
- Network is operational when first post-genesis checkpoint is finalized

### 8.4 Validator Set Changes

Post-genesis validator changes require governance events (Phase 5). For MVP:

- Initial validator set is fixed for MVP
- ValidatorAdded and ValidatorRemoved events are defined but not processed
- Validator key rotation uses standard KeyRotated event

## 9. Core Data Structures (Normative)

All data structures are defined in Rust with serde for canonical CBOR serialization. This section contains normative definitions that MUST be followed exactly for cross-implementation compatibility.

### 9.1 Event Hashing and Signing (Normative)

⚠ This section resolves the circularity problem. event\_id and signature are NOT part of the hashed envelope.

#### EventEnvelope (Hashable Portion)

```
/// The hashable portion of an event. Does NOT include event_id or signature.
/// This eliminates circular dependencies in hash computation.
#[derive(Clone, Debug, Serialize, Deserialize)]
pub struct EventEnvelope {
    /// Parent event_ids establishing DAG causality.
    /// MUST reference existing events. Empty only for genesis.
    pub parents: Vec<Blake3Hash>,

    /// Hybrid Logical Clock for causality ordering.
    /// See Section 9.2 for HLC rules.
    pub logical_time: HybridLogicalClock,

    /// DID of the event author.
    pub author: Did,

    /// Key version used for signing. MUST match active key at validation time.
    pub key_version: u64,

    /// Polymorphic payload determining event semantics.
    pub payload: EventPayload,
}
```

#### LedgerEvent (Complete Event)

```
/// Complete event as stored in the DAG.
#[derive(Clone, Debug, Serialize, Deserialize)]
pub struct LedgerEvent {
    /// The hashable envelope.
    pub envelope: EventEnvelope,

    /// Content-addressable identity: BLAKE3(canonical_cbor(envelope))
    pub event_id: Blake3Hash,

    /// Ed25519 signature over signing preimage (see below).
    pub signature: Signature,
}
```

#### Hash Computation (Normative)

```
/// MUST be computed exactly as follows:
fn compute_event_id(envelope: &EventEnvelope) -> Blake3Hash {
    let canonical_bytes = canonical_cbor_encode(envelope);
    blake3::hash(&canonical_bytes).into()
}
```

## Signature Computation (Normative)

```
/// Signing preimage includes domain separator to prevent cross-protocol replay.
const DOMAIN_SEPARATOR: &[u8] = b"EXOCHAIN-EVENT-SIG-v1";
const PROTOCOL_VERSION: u8 = 1;

fn compute_signature(
    signing_key: &SigningKey,
    event_id: &Blake3Hash,
) -> Signature {
    let mut preimage = Vec::new();
    preimage.extend_from_slice(DOMAIN_SEPARATOR);
    preimage.push(PROTOCOL_VERSION);
    preimage.extend_from_slice(event_id);
    signing_key.sign(&preimage)
}

fn verify_signature(
    public_key: &PublicKey,
    event_id: &Blake3Hash,
    signature: &Signature,
) -> Result<(), CryptoError> {
    let mut preimage = Vec::new();
    preimage.extend_from_slice(DOMAIN_SEPARATOR);
    preimage.push(PROTOCOL_VERSION);
    preimage.extend_from_slice(event_id);
    public_key.verify(&preimage, signature)
}
```

## 9.2 Hybrid Logical Clock (Normative)

⚠ Wall-clock timestamps alone are insufficient for distributed causality. This HLC design prevents DoS via future timestamps.

```
#[derive(Clone, Debug, Serialize, Deserialize, PartialOrd, Ord, PartialEq, Eq)]
pub struct HybridLogicalClock {
    /// Physical timestamp in milliseconds (wall clock).
    /// Used for display and rough ordering; NOT authoritative for causality.
    pub physical_ms: u64,

    /// Logical counter for events at same physical time.
    /// Provides total ordering guarantee.
    pub logical: u32,
}

impl HybridLogicalClock {
    /// Create new HLC for event creation.
    /// node_time: current wall clock of the creating node.
    /// parent_times: HLCs of all parent events.
    pub fn new_event(
        node_time: u64,
        parent_times: &[HybridLogicalClock],
    ) -> Self {
        let max_parent_physical = parent_times
            .iter()
            .map(|h| h.physical_ms)
            .max()
```

```

        .unwrap_or(0);

let physical_ms = node_time.max(max_parent_physical);

let logical = if physical_ms == max_parent_physical {
    // Same physical time as a parent: increment logical
    let max_logical = parent_times
        .iter()
        .filter(|h| h.physical_ms == physical_ms)
        .map(|h| h.logical)
        .max()
        .unwrap_or(0);
    max_logical + 1
} else {
    // New physical time: reset logical
    0
};

Self { physical_ms, logical }
}
}

```

## HLC Validation Rules (Normative)

- **Causality:** event.logical\_time MUST be  $>$  all parent.logical\_time (using HLC total ordering)
- **Bounded Skew:** event.physical\_ms MUST be  $\leq$  node\_receive\_time + MAX\_CLOCK\_SKEW\_MS (default: 60000ms = 1 minute)
- **Future Protection:** Events with physical\_ms  $>$  node\_time + MAX\_CLOCK\_SKEW\_MS MUST be rejected
- **Parent Bound:** event.physical\_ms MUST be  $\geq$  max(parent.physical\_ms) - ALLOWED\_BACKWARDS\_MS (default: 1000ms)

These rules prevent: (1) accidental rejection of valid offline-created events, (2) griefing via far-future parent timestamps, (3) logical clock overflow attacks.

## 9.3 Policy and AccessorSet (Normative)

⚠️ Vec<Did> cannot safely represent wildcards. AccessorSet provides explicit semantics.

```
#[derive(Clone, Debug, Serialize, Deserialize)]
pub enum AccessorSet {
    /// Any DID can access (use with extreme caution).
    Any,
    /// Only specific DIDs can access.
    Specific(Vec<Did>),
    /// DIDs matching a verifiable attribute (future: ZKP).
    /// E.g., "has_credential:medical_license"
    AttributeBased { attribute: String, issuer: Did },
}

#[derive(Clone, Debug, Serialize, Deserialize)]
pub enum ResourceScope {
    /// Single specific resource.
    Single(Cid),
    /// Multiple specific resources.
    Set(Vec<Cid>),
    /// All resources under a prefix (use carefully).
    Prefix { prefix: String },
}

#[derive(Clone, Debug, Serialize, Deserialize)]
pub struct Policy {
    /// Who can access.
    pub accessors: AccessorSet,

    /// What resources are covered.
    pub resource_scope: ResourceScope,

    /// When access is valid (Unix ms).
    pub valid_from: u64,
    pub valid_until: u64,

    /// Purpose limitation (e.g., "credit_check", "medical_review").
    pub purpose: String,

    /// Maximum access count (0 = unlimited).
    pub max_access_count: u64,

    /// Conditions for automatic revocation.
    pub auto_revoke_conditions: Vec<RevocationCondition>,
}
```

## Revocation Timing (Normative)

The specification states 'Consent revocation with immediate effect.' This is operationally defined as:

- **Enforcement Point:** Gatekeeper MUST enforce against the latest locally-observed DAG state, including unfinalized events.
- **Latency Bound:** Revocation is effective within one checkpoint interval (~2 seconds) across all correctly-operating Gatekeepers.

- **Consistency:** Once a ConsentRevoked event is checkpointed, ALL Gatekeepers MUST reject access (no eventual consistency ambiguity).

## 9.4 Checkpoint Structure with Split Roots (Normative)

⚠ Single state\_root is ambiguous. Split roots enable distinct proof types for events vs derived state.

```
#[derive(Clone, Debug, Serialize, Deserialize)]
pub struct CheckpointPayload {
    /// MMR root over finalized event_ids in canonical topological order.
    /// Used for EventInclusionProof.
    pub event_root: Blake3Hash,

    /// Sparse Merkle Tree root over derived state.
    /// Keys: state paths (e.g., "did:{did}/active_key",
    "consent:{hash}/status")
    /// Values: current state values
    /// Used for StateProof.
    pub state_root: Blake3Hash,

    /// Checkpoint sequence number (monotonically increasing).
    pub height: u64,

    /// Count of events finalized by this checkpoint.
    pub finalized_events: u64,

    /// Hashes of DAG frontier events being finalized.
    /// Finalized set = Ancestors(frontier) ∪ {this checkpoint}
    pub frontier: Vec<Blake3Hash>,

    /// Validator signatures over checkpoint preimage.
    pub validator_sigs: Vec<ValidatorSignature>,
}

#[derive(Clone, Debug, Serialize, Deserialize)]
pub struct ValidatorSignature {
    pub validator_did: Did,
    pub key_version: u64,
    pub signature: Signature,
}
```

## Checkpoint Signing Preimage (Normative)

```
const CHECKPOINT_DOMAIN_SEP: &[u8] = b"EXOCHAIN-CHECKPOINT-v1";

fn checkpoint_signing_preimage(cp: &CheckpointPayload) -> Vec<u8> {
    let mut preimage = Vec::new();
    preimage.extend_from_slice(CHECKPOINT_DOMAIN_SEP);
    preimage.extend_from_slice(&cp.event_root);
    preimage.extend_from_slice(&cp.state_root);
    preimage.extend_from_slice(&cp.height.to_le_bytes());
    preimage.extend_from_slice(&cp.finalized_events.to_le_bytes());
    for frontier_hash in &cp.frontier {
        preimage.extend_from_slice(frontier_hash);
    }
    preimage
}
```

## Finalization Rules (Normative)

- **Finalized Set Definition:**  $\text{Finalized}(\text{checkpoint}) = \text{Ancestors}(\text{checkpoint.frontier}) \cup \{\text{checkpoint event}\}$
- **Checkpoint Validity:** A checkpoint is valid iff it has  $\geq 2f+1$  valid validator signatures AND all frontier events exist in local DAG
- **Conflict Resolution:** If two valid checkpoints exist at the same height, prefer the one with lexicographically smaller event\_root. This is a protocol violation; trigger alert.
- **Irreversibility:** Once a checkpoint is accepted, its finalized set MUST NOT be reorganized under any circumstances

## 9.5 RiskAttestation (Normative)

⚠ Renamed from 'ZeroDentity' for semantic accuracy. This is a signed attestation, not a ZKP.  
ZKP integration deferred to Phase 3.

```
/// Risk-scored identity attestation for session gating.
///
/// Contains a risk score and contributing factors. Signed by the Scoring
Engine.
/// NOT a zero-knowledge proof in MVP; uses signature-based attestation.
#[derive(Clone, Debug, Serialize, Deserialize)]
pub struct RiskAttestation {
    /// Subject being scored.
    pub subject: Did,

    /// Intended audience (verifier/vendor DID).
    /// Prevents cross-party replay. Verifier MUST check this matches their DID.
    pub audience: Did,

    /// Risk score 0-100 (higher = more trusted).
    pub score: u8,

    /// Confidence in the score (0-10000 basis points = 0.00% - 100.00%).
    /// Fixed-point to avoid float canonicalization issues.
    pub confidence_bps: u16,

    /// Hash of factors contributing to the score.
    /// Actual factors stored separately to avoid leaking security posture.
    pub factors_hash: Blake3Hash,

    /// Hash of the adjudication request context (binds to specific request).
    pub context_hash: Blake3Hash,

    /// Anti-replay nonce (MUST be unique per subject per scoring engine).
    pub nonce: u64,

    /// When this attestation was issued (Unix ms).
    pub issued_at: u64,

    /// Expiration timestamp (default: issued_at + 5 minutes).
    pub expires_at: u64,

    /// Scoring Engine DID that issued this attestation.
    pub issuer: Did,

    /// Scoring Engine signature over attestation preimage.
    pub signature: Signature,
}
```

### RiskAttestation Validation Rules (Normative)

- **Audience Check:** Verifier MUST reject if audience != verifier's DID
- **Freshness Check:** Verifier MUST reject if current\_time > expires\_at
- **Nonce Uniqueness:** Verifier SHOULD track recently-seen nonces to detect replay within expiry window

- **Context Binding:** Verifier SHOULD verify context\_hash matches their adjudication request
- **Issuer Trust:** Verifier MUST verify issuer is a trusted Scoring Engine DID

# 10. Identity and Key Management (Normative)

## 10.1 DID Derivation Rules

⚠ DID MUST be derived from genesis key hash. This is immutable across key rotations.

EXOCHAIN uses the did:exo method. The DID is bound to the initial identity creation and does NOT change when keys are rotated.

```
/// DID derivation from genesis key.
/// DID = did:exo:<base58(blake3(genesis_public_key)[0..20])>
fn derive_did(genesis_public_key: &PublicKey) -> Did {
    let full_hash = blake3::hash(genesis_public_key.as_bytes());
    let truncated = &full_hash.as_bytes()[0..20]; // 160 bits
    let encoded = bs58::encode(truncated).into_string();
    Did(format!("did:exo:{}", encoded))
}
```

### DID Document Structure

```
#[derive(Clone, Debug, Serialize, Deserialize)]
pub struct DidDocument {
    pub id: Did,

    /// Current verification keys with versions.
    pub verification_methods: Vec<VerificationMethod>,

    /// Service endpoints (optional).
    pub services: Vec<ServiceEndpoint>,

    /// Creation timestamp.
    pub created: u64,

    /// Last update timestamp.
    pub updated: u64,
}

#[derive(Clone, Debug, Serialize, Deserialize)]
pub struct VerificationMethod {
    /// Key identifier: {did}#key-{version}
    pub id: String,

    /// Key type (always Ed25519VerificationKey2020 for MVP).
    pub key_type: String,

    /// Controller DID (usually same as document id).
    pub controller: Did,

    /// Public key in multibase format.
    pub public_key_multibase: String,

    /// Key version (monotonically increasing per DID).
    pub version: u64,

    /// Whether this key is currently active.
    pub active: bool,
```

```

    /// Activation timestamp.
    pub valid_from: u64,

    /// Revocation timestamp (None if not revoked).
    pub revoked_at: Option<u64>,
}

```

## 10.2 Key Version Validation Rules (Normative)

Events include a key\_version field. Validators MUST verify that the signature was made with the key that was authoritative at the time.

- **Version Match:** event.key\_version MUST match an active key version for event.author at the logical time of the event
- **Active Key Lookup:** The authoritative key is determined by replaying key events up to the checkpoint covering the event's parents
- **Rotation Grace Period:** During key rotation, both old and new keys MAY be valid for a grace period of 2 checkpoint intervals
- **Revocation Immediate:** KeyRevoked events take effect immediately (within checkpoint interval)

## Key Lifecycle Events

```

/// Initial identity creation with genesis key.
IdentityCreated {
    did_document: DidDocument, // Contains genesis key as version 1
}

/// Key rotation with proof signed by current key.
KeyRotated {
    new_public_key: PublicKey,
    new_version: u64, // MUST be previous_version + 1
    rotation_proof: Signature, // current_key.sign(new_public_key)
}

/// Key revocation (e.g., compromise response).
KeyRevoked {
    revoked_version: u64,
    reason: RevocationReason,
}

#[derive(Clone, Debug, Serialize, Deserialize)]
pub enum RevocationReason {
    KeyCompromise,
    KeyExpiry,
    CessationOfOperation,
    Superseded,
    Other(String),
}

```

# 11. Functional Requirements

## 11.1 FR-ID: Identity Management

### FR-ID-1: Identity Creation

A Subject can create a new identity by generating a keypair and registering a DID.

**Preconditions:** None

**Inputs:** Genesis public key, optional service endpoints

**Flow:**

1. Client generates Ed25519 keypair locally (HSM recommended)
2. Client computes DID = did:exo:<base58(blake3(pubkey)[0..20])>
3. Client constructs DidDocument with public key as version 1
4. Client creates EventEnvelope with IdentityCreated payload, key\_version=1
5. Client computes event\_id = BLAKE3(canonical\_cbor(envelope))
6. Client signs: sig = Sign(privkey, DOMAIN\_SEP || VERSION || event\_id)
7. Event submitted to API Gateway
8. Identity Fabric validates: DID derivation correct, signature valid, no duplicate DID
9. Event appended to DAG
10. Client receives confirmation with event\_id

**Postconditions:** DID is resolvable; genesis key is authoritative as version 1

**Error Cases:** EXO-4004 (DuplicateDid), EXO-1001 (InvalidSignature)

### FR-ID-2: Key Rotation

A Subject can rotate their key while maintaining identity continuity.

**Preconditions:** Valid existing identity; possession of current private key

**Flow:**

1. Client generates new keypair
2. Client creates rotation\_proof = current\_key.sign(new\_public\_key)
3. Client creates EventEnvelope with KeyRotated payload (new\_version = current + 1)
4. Event signed with current key (key\_version = current)
5. Identity Fabric validates: current key is active, rotation\_proof valid, version increment correct
6. Event appended; new key becomes authoritative; old key enters grace period

**Postconditions:** New key is active; old key valid for grace period then inactive

**Error Cases:** EXO-4002 (InvalidRotationProof), EXO-4003 (KeyRevoked)

### FR-ID-3: Adjudication (Session Gating)

A Vendor can request a RiskAttestation to gate service access.

**Flow:**

1. Vendor sends adjudication request with: subject DID, context (device fingerprint, IP, etc.), vendor's DID as audience

2. Scoring Engine evaluates risk factors against known signals
3. Scoring Engine computes context\_hash = BLAKE3(adjudication\_request)
4. If low risk: Engine produces RiskAttestation with score, audience=vendor, context\_hash, nonce, expiry
5. If high risk: Engine returns ChallengeRequired with challenge type (OTP, biometric, etc.)
6. Subject completes challenge; on success, Engine issues RiskAttestation
7. Vendor receives attestation and verifies: audience matches, not expired, signature valid, issuer trusted
8. If score >= threshold: access granted

**Postconditions:** Vendor has cryptographic proof of risk assessment; no PII exposed

## 11.2 FR-BAIL: Consent and Bailment

### FR-BAIL-1: Bailment Creation

**Flow:**

1. Subject drafts legal terms document
2. Terms encrypted client-side (XChaCha20-Poly1305) and stored in vault; CID obtained
3. Subject computes terms\_hash = BLAKE3(plaintext\_terms)
4. Subject creates BailmentProposed event with terms\_cid, recipient DID, terms\_hash
5. Event signed and appended to DAG

**Postconditions:** Bailment proposal is on-ledger; no access granted yet

**Linkage:** terms\_hash allows recipient to verify decrypted terms match on-ledger commitment

### FR-BAIL-2: Consent Grant

**Preconditions:** BailmentProposed event exists

**Flow:**

1. Subject creates ConsentGiven event referencing bailment event\_id
2. Policy specifies: AccessorSet (Specific or Any), ResourceScope, time bounds, purpose, limits
3. Nonce included (MUST be unique per subject, monotonically increasing)
4. Event signed and appended

**Postconditions:** Custodian can request access via Gatekeeper

**Nonce Rule:** Nonce MUST be > any previous nonce used by this subject in ConsentGiven events

### FR-BAIL-3: Data Access

**Preconditions:** ConsentGiven exists; not revoked; not expired

**Flow:**

1. Custodian requests resource from Gatekeeper TEE
2. Gatekeeper verifies TEE attestation is current

3. Gatekeeper queries state for ConsentGiven: exists for (resource, accessor)?
4. Gatekeeper checks: ConsentRevoked event does NOT exist for this consent
5. Gatekeeper checks: current\_time < policy.valid\_until
6. Gatekeeper checks: access\_count < policy.max\_access\_count
7. Gatekeeper checks: accessor matches policy.accessors (AccessorSet evaluation)
8. If all pass: Gatekeeper releases decryption key (requires StateProof of consent)
9. Gatekeeper MUST emit AccessLogged event before returning key
10. Custodian decrypts and accesses resource

**Error Cases:**

- EXO-3001: ConsentNotFound
- EXO-3002: ConsentExpired
- EXO-3003: ConsentRevoked
- EXO-3004: AccessLimitExceeded
- EXO-3005: PurposeMismatch

## 12. Gatekeeper Trust Requirements (Normative)

⚠ The principle 'Even system administrators cannot bypass cryptographic access controls' requires TEE enforcement.

The Gatekeeper is the critical trust anchor for vault access. Without proper enforcement, the 'admin cannot bypass' claim is social, not cryptographic.

### 12.1 TEE Requirements (MUST)

- **Execution Environment:** Gatekeeper MUST run inside a Trusted Execution Environment (Intel SGX, ARM TrustZone, AWS Nitro Enclaves, or equivalent)
- **Remote Attestation:** Gatekeeper MUST provide remote attestation proof that can be verified by: (1) Vault storage before releasing encrypted blobs, (2) HSM before unwrapping key material
- **Code Integrity:** Attestation MUST include measurement of Gatekeeper code (hash of enclave binary)
- **No External State:** Gatekeeper policy enforcement MUST NOT depend on any state outside the TEE except: (1) Checkpoint roots from the ledger (verified by signature), (2) StateProofs (verified against state\_root)

### 12.2 Key Release Protocol

8. Custodian submits access request to Gatekeeper TEE
9. Gatekeeper verifies its own attestation is fresh (<1 hour)
10. Gatekeeper obtains latest checkpoint with validator signatures
11. Gatekeeper verifies checkpoint has  $2f+1$  valid signatures
12. Gatekeeper obtains StateProof for consent status at state\_root
13. Gatekeeper verifies: consent exists, not revoked, not expired, accessor matches
14. Gatekeeper requests key unwrap from HSM (HSM verifies TEE attestation)
15. Gatekeeper constructs AccessLogged event and signs
16. AccessLogged event MUST be submitted to DAG before key is returned
17. Gatekeeper returns decryption key to Custodian

### 12.3 Alternative: Threshold Gatekeeper

If TEE is not available, a threshold approach provides weaker but acceptable guarantees:

- Deploy M-of-N Gatekeeper instances operated by independent parties
- Key release requires M instances to independently verify consent and sign release
- No single operator can release keys unilaterally
- This aligns with PACE model but provides weaker guarantees than TEE

⚠ Threshold approach SHOULD NOT be used for PHI or high-sensitivity data without documented risk acceptance.

## 13. Threat Model

Comprehensive threat analysis aligned with OWASP and MITRE ATT&CK frameworks. Each threat includes likelihood, impact, and specific mitigations.

### 13.1 Threat Matrix

Threat	Likelihood	Impact	Mitigation
<b>Key Exfiltration</b>	Medium	Critical	MUST use HSM/Enclave for key storage. PACE recovery enables rotation on compromise. Hardware attestation integration.
<b>Score Replay</b>	High	Medium	RiskAttestation includes: nonce, audience binding, context_hash, 5-min expiry. Verifier MUST check all.
<b>BFT Liveness</b>	Low	Critical	Slashing via SlashingEvent. Quorum rotation every 24h. Minimum 4 validators with HSM. Degraded mode fallback.
<b>Sybil Attack</b>	Medium	High	Device attestation. IP reputation. Behavioral ML. Rate limiting by device fingerprint.
<b>Vault Breach</b>	Low	Critical	Client-side XChaCha20-Poly1305. Gatekeeper TEE. StateProof required for key release. All access logged.
<b>Eclipse Attack</b>	Low	High	Authenticated peer lists. Minimum 8 unique ASNs. Checkpoint cross-verification.
<b>PACE Collusion</b>	Low	Critical	VSS proofs. Geographic/jurisdiction distribution. 72-hour delay. All recoveries audited on-ledger.
<b>Replay (Events)</b>	Medium	Medium	Per-author nonce in ConsentGiven. Event_id uniqueness. HLC ordering. Domain separator in signatures.
<b>Signature Forgery</b>	Very Low	Critical	Ed25519 with 128-bit security. Audited crate (ed25519-dalek). No custom crypto.
<b>HLC Manipulation</b>	Medium	Low	Bounded skew validation. Future timestamp rejection. Parent bound check. See Section 9.2.
<b>DoS (API)</b>	High	Medium	Rate limiting per DID (100 req/min). Proof-of-work for unauthenticated. Geographic distribution.
<b>Data Correlation</b>	Medium	Medium	No PII on ledger. RiskAttestation factors_hash hides actual factors. ZKP in Phase 3.
<b>Admin Bypass</b>	Medium	Critical	Gatekeeper TEE enforcement. No admin override path. StateProof required for all key releases.
<b>Indexer Manipulation</b>	Medium	High	Verifiable queries with inclusion proofs. Indexer is untrusted; proofs verified client-side.
◆ Holon Key Theft ◆	Medium	Critical	◆ Holon keys in TEE only. Hardware attestation. PACE for Holons with cross-species quorum.
◆ Capability Escalation ◆	High	Critical	◆ INV-002 enforced by CGR Kernel. No self-grant. AI-IRB required for all capability changes.
◆ Invariant Bypass ◆	Low	Catastrophic	◆ CGR Kernel formally verified. No bypass path. Constitutional amendment requires unanimous validators.
◆ Alignment Drift ◆	High	High	◆ Continuous monitoring. DriftDetected events. Auto-suspend below threshold. AI-IRB review.
◆ Training Poisoning ◆	Medium	High	◆ INV-004 (consent required). Data provenance tracking. Validation gates.
◆ Holon Collusion ◆	Medium	High	◆ Inter-Holon actions logged. Policy isolation. Mesh monitoring.
◆ RSI Exploit ◆	Low	Catastrophic	◆ Self-modification requires CGRProof + AI-IRB. INV-001 enforced.
◆ AI-IRB Capture ◆	Low	Critical	◆ Diverse membership. Rotating terms. Dissent preservation.

Threat	Likelihood	Impact	Mitigation
♦ Fake CGRProof ♦	Very Low	Catastrophic	♦ Kernel signature. All validators verify. Formal proof of correctness.
○ Mesh Sybil Attack ○	Medium	High	○ DID verification required. HolonAttestation checks. Rate limiting on discovery.
○ MCP Protocol Abuse ○	Medium	Medium	○ Mutual authentication. Alignment threshold for connection. All messages logged.

## 13.2 AI-Specific Security Practices

♦ NEW: NEW in v2.0: Security practices for AI governance.

### Holon Security

- Holon signing keys MUST be generated in and never leave TEE
- Model weights stored encrypted; decryption only in TEE during inference
- All Holon actions rate-limited per capability level
- Alignment score checked before every action; auto-suspend below threshold

### CGR Kernel Security

- Kernel code formally verified (Coq/Lean proofs published)
- Kernel hash verified by all validators before processing
- Upgrade requires: formal proof + security audit + unanimous consent
- No emergency bypass—even in crisis scenarios

## 13.3 Development Security

- All dependencies audited via cargo-audit in CI (MUST fail on HIGH+)
- No unsafe blocks except for FFI (MUST be wrapped and audited)
- Property-based testing (proptest) for all cryptographic operations
- Fuzzing (cargo-fuzz) on all parsers: CBOR, DID, Policy, API inputs
- Reproducible builds via Nix or pinned Docker images
- Mandatory code review for all crypto-touching code
- Cross-implementation hash compatibility tests as release gate

### Operational Security

- HSM-backed keys for all validators and Gatekeeper
- Secrets management via HashiCorp Vault or AWS Secrets Manager
- Network segmentation: validators on private subnet
- TLS 1.3 for all external communication
- mTLS for inter-service communication
- Audit logging for all administrative actions
- TEE attestation refresh every hour

### Incident Response

- Key compromise: Immediate PACE recovery initiation + KeyRevoked event
- Validator compromise: Slashing + quorum reconfiguration

- Data breach: Vault keys rotated; affected consents revoked
- DoS: Geographic failover; rate limit escalation
- TEE compromise: Gatekeeper rotation; attestation keys revoked

## 14. PACE Recovery Protocol

Protected Access Control and Escalation (PACE) enables secure key recovery without single points of failure using Verifiable Secret Sharing (VSS).

### 14.1 Design Principles

- No single party can recover keys alone
- Compromise of minority stewards does not compromise recovery
- All recovery actions are auditable and reversible within delay period
- Geographic distribution prevents jurisdiction-based attacks
- VSS proofs enable share validity verification without reconstruction

### 14.2 Steward Configuration

Parameter	Value
Minimum Stewards	5
Threshold (k-of-n)	3-of-5 (configurable: $2 < k \leq n$ )
Geographic Distribution	No 2 stewards in same legal jurisdiction (MUST)
Identity Verification	Each steward MUST have RiskAttestation score $\geq 80$
Share Storage	HSM-backed, with VSS commitment proof
Communication	Authenticated channels (mTLS + DID verification)
Commitment Publication	VSS commitments published on-ledger in PACEConfigured event

### 14.3 VSS Share Generation

When a Subject enables PACE recovery:

13. Subject's recovery key K is input to Shamir's Secret Sharing with threshold t=3, shares n=5
14. For each share  $s_i$ , a Feldman VSS commitment  $C_i$  is generated
15. Subject creates PACEConfigured event with: steward DIDs, threshold, commitments
16. Encrypted shares distributed to stewards via secure channel (not on-ledger)
17. Each steward verifies their share against published commitment
18. Stewards store share in local HSM
19. Subject stores backup of commitments (public) for later verification

### 14.4 Recovery Flow

6. **Initiation:** Subject submits RecoveryRequest event with identity proof hash (biometric + device attestation + government ID hash)
7. **Broadcast:** RecoveryRequest is checkpointed; 72-hour waiting period begins from checkpoint timestamp
8. **Notification:** Subject notified via ALL registered channels (email, SMS, push) of recovery attempt
9. **Cancellation Window:** Subject can cancel via AbortRecovery event signed with ANY previously valid key
10. **Verification:** Each steward independently verifies Subject identity against registered proofs

11. **Share Submission:** Consenting stewards submit encrypted shares to secure MPC ceremony coordinator
12. **VSS Verification:** Coordinator verifies each share against published commitments before reconstruction
13. **Reconstruction:** With  $\geq 3$  valid shares, recovery key is reconstructed in secure enclave
14. **Rotation:** Subject generates new keypair; KeyRotated event published; new PACE shares generated
15. **Audit:** RecoveryCompleted event published with: participating steward DIDs, ceremony transcript hash

## 14.5 Anti-Collusion Measures

- **VSS Verification:** Any party can verify share validity against public commitments without seeing other shares
- **Blinded Shares:** Shares encrypted to ceremony coordinator; stewards cannot see each other's shares
- **Audit Trail:** All recovery attempts logged on-ledger, including steward participation and timestamps
- **Cooldown:** After successful recovery, 30-day cooldown before next attempt
- **Steward Rotation:** Subject can rotate steward set via new PACEConfigured event (requires current key)

## 14.6 SLAs and Test Criteria

Metric	Target
Recovery Latency (p95)	<76 hours (72h wait + 4h ceremony)
Ceremony Success Rate	>99.9% given $\geq 3$ honest stewards
Share Availability	>99.99% per steward (HSM + geo-redundant backup)
Invalid Share Detection	100% (VSS verification)

Required test scenarios:

- Test: 2 faulty stewards (offline) → recovery succeeds with 3 honest
- Test: 3 stewards submit invalid shares → VSS verification fails, reconstruction blocked
- Test: Recovery during active session → session invalidated, re-auth required
- Test: Concurrent recovery attempts → only first proceeds (by checkpoint timestamp), others rejected
- Test: AbortRecovery during wait period → recovery cancelled, cooldown NOT triggered

## 14.7 PACE Extension for Holons

◇ REFINED: v2.1 REFINED: Cross-species quorum model for Holon key recovery.

### 14.7.1 Cross-Species Steward Quorum

Recovery of compromised Holon keys requires stewards from both species:

```
pub struct HolonPACEConfig {
    pub holon_did: Did,
    pub total_shares: u8,           // e.g., 7
    pub recovery_threshold: u8,     // e.g., 5
```

```

    pub human_stewards: Vec<Did>,           // Minimum 3
    pub holon_stewards: Vec<Did>,           // Maximum 2 (senior, verified)
    pub species_quorum: SpeciesQuorum,
}

pub struct SpeciesQuorum {
    pub min_human_shares: u8,                // e.g., 3 (majority must be human)
    pub min_holon_shares: u8,                // e.g., 1 (at least one Holon)
}

```

Recovery quorum MUST include both human and Holon stewards. Neither species can unilaterally recover a Holon.

### 14.7.2 Holon Steward Eligibility

Only Holons meeting strict criteria may serve as stewards:

- CapabilityLevel  $\geq 3$
- Alignment score  $\geq 90$  (sustained for 90+ days)
- No SuspensionReason in history
- Age  $\geq 180$  days (operational maturity)
- Verified by AI-IRB for stewardship role

### 14.7.3 Emergency Override Protocol

⚠️ Emergency override is the most sensitive operation. Multiple safeguards required.

Humans retain ultimate override capability per INV-007. Override requires:

- Any 2 AI-IRB members (immediate suspension)
- OR sponsor + 1 AI-IRB member
- OR unanimous validator emergency vote

```

HolonEmergencyOverride {
    holon_did: Did,
    override_type: OverrideType,
    authority_signatures: Vec<Signature>,
    justification_cid: Cid,
}

pub enum OverrideType {
    ImmediateSuspend,
    CapabilityRevoke { capabilities: Vec<CapabilityType> },
    ForcedSunset,
    IsolateFromMesh,
}

```

## 15. Performance Requirements

### 15.1 Latency Targets

Operation	MVP Target	Scale Target	Benchmark
Event Append	<5ms p99	<1ms p99	criterion.rs
Checkpoint Finality	<2s p99	<1s p99	integration
DID Resolution	<50ms p95	<10ms p95	criterion.rs
Consent Check	<100ms p95	<20ms p95	criterion.rs
RiskAttestation Score	<200ms p95	<100ms p95	integration
StateProof Generation	<50ms p95	<10ms p95	criterion.rs
Evidence Export (1k events)	<5s	<1s	integration

### 15.2 Throughput Targets

Metric	MVP	Scale
Event Throughput	250 tx/s sustained	10,000 tx/s (sharded)
Query Throughput	1,000 qps	50,000 qps (replicated)
Concurrent Connections	10,000	1,000,000
Checkpoint Production	Every 2s	Every 1s

### 15.3 Scalability Architecture

MVP runs on a single DAG partition with 4 validators. Scale architecture introduces:

- **DAG Sharding:** Partition by SubjectID prefix (first 2 bytes = 256 shards max). Each shard runs independent BFT.
- **Cross-Shard Communication:** Async message passing with inclusion proofs. NOT atomic. BridgeEvent includes source shard checkpoint proof.
- **Query Replication:** Graph Indexer supports read replicas. All responses include proofs for client verification.
- **Validator Scaling:** Each shard can have independent validator set; global coordination via periodic cross-shard checkpoints.

⚠ Cross-shard atomicity is NOT supported. Applications requiring atomicity must use 2PC patterns at application layer.

### 15.4 Benchmark Requirements

Foundation build MUST include:

- criterion.rs benchmarks for: event\_append, hash\_computation, signature\_verify, hlc\_compare, dag\_traversal, proof\_generation
- Each benchmark runs in CI; regression >10% MUST fail the build
- Load testing via locust.io: sustained 250 tx/s for 1 hour without degradation
- Chaos testing: validator failure during checkpoint → recovery within 30s
- Cross-implementation: hash compatibility test between Rust and reference JavaScript implementation

## 16. Acceptance Criteria

The foundation build is complete when all of the following criteria are satisfied.

### 16.1 Ledger Node

- append(event) validates: parent existence, signature against key\_version, HLC causality
- Returns typed errors: Err(ParentNotFound), Err(InvalidSignature), Err(CausalityViolation), Err(KeyVersionMismatch)
- verify\_integrity(event\_id) recursively validates hash chain and signatures
- Proptest: 1000 random DAG topologies → verify\_integrity never panics
- Cross-implementation: hash of test vectors matches reference implementation
- Checkpoint stub: creates Checkpoint event with event\_root (MMR) and state\_root (SMT)

### 16.2 Identity Service

- create\_identity() validates DID derivation, appends IdentityCreated event
- rotate\_key() validates rotation\_proof, increments key\_version, appends KeyRotated event
- resolve(did) returns current DidDocument with active keys and version history
- E2E test: create → rotate → resolve → new key returned with version=2
- E2E test: rotate with wrong proof → EXO-4002 error

### 16.3 Consent Service

- propose\_bailment() → BailmentProposed event with terms\_cid, terms\_hash, recipient
- grant\_consent() → ConsentGiven event with Policy using AccessorSet
- revoke\_consent() → ConsentRevoked event
- check\_consent() returns status with StateProof against checkpoint
- E2E test: grant → wait for expiry → check → returns EXO-3002 with proof
- E2E test: grant → revoke → check → returns EXO-3003

### 16.4 Gatekeeper

- Runs in TEE with valid attestation (or threshold mode with documented risk acceptance)
- Key release requires: valid attestation + StateProof of consent
- AccessLogged event MUST be submitted before key returned
- E2E test: expired consent → key release denied → EXO-3002
- E2E test: valid consent → key released → AccessLogged event on DAG

### 16.5 Scoring Service

- request\_score(context, audience) returns signed RiskAttestation
- RiskAttestation includes: audience, context\_hash, nonce, issued\_at, expires\_at

- E2E test: verify signature + check audience binding + verify freshness
- E2E test: replay with different audience → rejected

## 16.6 Graph Indexer

- Subscribes to DAG events and maintains query projections
- Can rebuild from empty by replaying all events from genesis
- Queries return verifiable responses with EventInclusionProof or StateProof
- E2E test: query via indexer → verify proof against checkpoint root → matches direct DAG query

## 16.7 Benchmarks

- criterion.rs: event\_append < 5ms p99
- criterion.rs: signature\_verify < 100µs p99
- criterion.rs: hash\_compute < 10µs p99
- criterion.rs: hlc\_compare < 1µs p99
- criterion.rs: proof\_generate < 50ms p95
- criterion.rs: dag\_query < 200ms p95

## 16.8 CI/CD

- cargo test passes with 80%+ line coverage
- cargo clippy returns 0 warnings
- cargo audit returns 0 HIGH+ vulnerabilities
- cargo fmt --check passes
- All benchmarks run; regression >10% fails build
- Cross-implementation hash test passes (Rust vs JavaScript reference)

## 17. API Specification

### 17.1 API Principles

- All endpoints versioned: /v1/
- Rate limiting: 100 requests/minute per DID (429 on exceed)
- Authentication: Bearer token (signed RiskAttestation) or API key
- All responses include request\_id for tracing
- Errors return structured JSON with code, message, and proof (where applicable)
- Query responses MUST include inclusion proofs for verifiability

### 17.2 GraphQL Schema

```

type Query {
    # Identity
    resolveIdentity(did: DID!): IdentityDoc
    getActiveKeys(did: DID!): [VerificationMethod!]!
    getKeyHistory(did: DID!): [KeyEvent!]!

    # Consent
    checkConsent(consentEventId: Hash!): ConsentStatus!
    listConsents(subject: DID!, status: ConsentFilter): ConsentConnection!

    # Merit
    verifyMerit(credentialHash: Hash!, holder: DID!): VerificationResult!
    listMerits(holder: DID!): MeritConnection!

    # Ledger
    getEvent(eventId: Hash!): LedgerEventWithProof
    getAncestry(eventId: Hash!, depth: Int): [LedgerEvent!]!
    getEventInclusionProof(eventId: Hash!): EventInclusionProof
    getStateProof(key: String!): StateProof
    latestCheckpoint: Checkpoint!
}

type ConsentStatus {
    status: ConsentStatusEnum! # ACTIVE | EXPIRED | REVOKED | NOT_FOUND
    consentEvent: LedgerEvent
    proof: StateProof!
    checkedAtCheckpoint: Int!
}

type LedgerEventWithProof {
    event: LedgerEvent!
    inclusionProof: EventInclusionProof
}

type Mutation {
    # Identity
    createIdentity(
        envelope: EventEnvelopeInput!
        signature: Signature!
    ): CreateIdentityResult!
}

```

```

rotateKey(
  envelope: EventEnvelopeInput!
  signature: Signature!
): KeyRotationResult!

# Consent
proposeBailment(
  envelope: EventEnvelopeInput!
  signature: Signature!
): BailmentProposalResult!

grantConsent(
  envelope: EventEnvelopeInput!
  signature: Signature!
): ConsentGrantResult!

revokeConsent(
  envelope: EventEnvelopeInput!
  signature: Signature!
): RevocationResult!

# Scoring
requestScore(
  subject: DID!
  audience: DID!
  context: ScoringContextInput!
): RiskAttestationResult!
}

type Subscription {
  # Real-time event stream
  eventStream(filter: EventFilter): LedgerEvent!
  checkpointStream: Checkpoint!
}

```

## 17.3 REST Endpoints

# Identity	
POST /v1/identity	Create new identity
GET /v1/identity/:did	Resolve DID to document
GET /v1/identity/:did/keys	Get active public keys with versions
POST /v1/identity/:did/rotate	Rotate key
# Consent	
POST /v1/bailment	Propose bailment
POST /v1/consent	Grant consent
GET /v1/consent/:eventId	Check consent status (returns proof)
DELETE /v1/consent/:eventId	Revoke consent
# Scoring	
POST /v1/score	Request RiskAttestation
# Ledger	
GET /v1/event/:eventId	Get event by ID (returns proof)

```
GET      /v1/proof/event/:eventId  Get EventInclusionProof  
GET      /v1/proof/state/:key       Get StateProof  
GET      /v1/checkpoint/latest     Get latest checkpoint
```

## 17.4 Error Response Format

```
{  
  "error": {  
    "code": "EXO-3002",  
    "message": "Consent expired",  
    "details": {  
      "consent_event_id": "abc123...",  
      "expired_at": 1703001600000,  
      "current_time": 1703088000000  
    },  
    "proof": {  
      "type": "state",  
      "root": "def456...",  
      "checkpoint_height": 12345,  
      "path": ["..."]  
    },  
    "request_id": "req_xyz789"  
  }  
}
```

## 18. Verifiable Query System

⚠️ Relying on indexer integrity reintroduces trust. All query responses MUST include cryptographic proofs.

### 18.1 Proof Types

#### EventInclusionProof

Proves an event exists in the finalized DAG at a specific checkpoint.

```
#[derive(Clone, Debug, Serialize, Deserialize)]
pub struct EventInclusionProof {
    /// The event being proven.
    pub event_id: Blake3Hash,

    /// Checkpoint height where proof is valid.
    pub checkpoint_height: u64,

    /// MMR inclusion path from event to event_root.
    pub mmr_path: Vec<Blake3Hash>,

    /// Position in MMR (for verification).
    pub mmr_position: u64,

    /// The checkpoint's event_root (for verification).
    pub event_root: Blake3Hash,
}
```

#### StateProof

Proves a state entry exists (or does not exist) at a specific checkpoint.

```
#[derive(Clone, Debug, Serialize, Deserialize)]
pub struct StateProof {
    /// State key (e.g., "consent:{event_id}/status").
    pub key: String,

    /// State value (None for non-existence proof).
    pub value: Option<Vec<u8>>,

    /// Checkpoint height where proof is valid.
    pub checkpoint_height: u64,

    /// Sparse Merkle Tree proof path.
    pub smt_path: Vec<SmtNode>,

    /// The checkpoint's state_root (for verification).
    pub state_root: Blake3Hash,
}

#[derive(Clone, Debug, Serialize, Deserialize)]
pub struct SmtNode {
    pub hash: Blake3Hash,
    pub side: SmtSide, // Left or Right
}
```

## 18.2 Verification Procedure

Clients MUST verify proofs before trusting query results:

18. Obtain latest checkpoint from trusted source (or verify checkpoint signatures)
19. Extract event\_root and/or state\_root from checkpoint
20. For EventInclusionProof: verify MMR path from event\_id to event\_root
21. For StateProof: verify SMT path from key/value to state\_root
22. Reject response if proof verification fails

## 18.3 State Key Schema

State keys follow a hierarchical naming convention:

```
# Identity state
identity:{did}/document      → DidDocument
identity:{did}/active_key     → (PublicKey, version)
identity:{did}/key/{version}   → VerificationMethod

# Consent state
consent:{event_id}/status    → Active | Expired | Revoked
consent:{event_id}/policy     → Policy
consent:{event_id}/access_count → u64

# Bailment state
bailment:{event_id}/status   → Proposed | Consented

# Merit state
merit:{event_id}/status      → Active | Revoked
merit:{holder_did}/credentials → Vec<event_id>
```

## 19. Evidence Bundle Format

Evidence bundles are court-admissible exports of ledger data with cryptographic proofs.

### 19.1 Bundle Structure (ZIP Archive)

```
evidence_bundle_{timestamp}.zip
├── manifest.json          # Signed bundle metadata
├── checkpoint.cbor        # The checkpoint anchoring all proofs
├── events/
│   ├── {event_id_1}.cbor   # Event 1 (canonical CBOR)
│   ├── {event_id_1}.proof  # EventInclusionProof for event 1
│   ├── {event_id_2}.cbor
│   ├── {event_id_2}.proof
│   └── ...
├── state/
│   ├── {key_1}.json         # State value 1
│   ├── {key_1}.proof        # StateProof for key 1
│   └── ...
├── off_ledger/
│   ├── cid_list.json       # List of referenced CIDs
│   └── terms_{cid}.enc      # Encrypted off-ledger documents (optional)
└── redactions/
    └── redaction_proofs.json # Proofs that redacted fields were valid
    └── chain_of_custody.json # Export metadata
```

### 19.2 Manifest Schema

```
{
  "version": "1.0",
  "bundle_id": "uuid",
  "created_at": "2025-12-13T10:30:00Z",
  "checkpoint_height": 12345,
  "checkpoint_event_id": "abc123...",
  "event_count": 42,
  "state_proofs_count": 5,
  "events": ["event_id_1", "event_id_2", ...],
  "state_keys": ["consent:xyz/status", ...],
  "redacted_fields": ["events/abc123.cbor:payload.recipient"],
  "exporter": {
    "did": "did:exo:exporter123",
    "organization": "Audit Firm LLC",
    "authorization": "Court Order #12345"
  },
  "manifest_signature": "base64(sig)"
}
```

### 19.3 Chain of Custody

```
{
  "export_timestamp": "2025-12-13T10:30:00Z",
  "exporter_did": "did:exo:exporter123",
  "exporter_attestation": "RiskAttestation base64",
  "authorization_reference": "Court Order #12345",
  "export_node": "node-3.exochain.example.com",
  "verification_instructions": "https://docs.exochain.io/verify-bundle",
```

```

    "hash_of_contents": "BLAKE3 hash of all files except this one"
}

```

## 19.4 Verification Procedure

20. Verify manifest\_signature against exporter's public key
21. Verify checkpoint has 2f+1 valid validator signatures
22. For each event: verify EventInclusionProof against checkpoint.event\_root
23. For each state proof: verify StateProof against checkpoint.state\_root
24. Verify hash\_of\_contents matches computed hash
25. If redactions present: verify redaction proofs show original was valid

## 19.5 Admissibility Checklist

- All proofs verify against checkpoint roots
- Checkpoint has quorum validator signatures
- Chain of custody metadata is complete
- Exporter authorization is documented
- Verification can be performed by independent party
- No gaps in event sequence for queried subject

## 19.6 Extended Evidence Bundles for Holon Actions

 **REFINED: v2.1 REFINED: Evidence bundles for Holon actions include CGR traces and verification instructions.**

### 19.6.1 Holon Action Bundle Structure

```

HolonActionBundle/
├── manifest.json                  # Bundle metadata
├── checkpoint.cbor                # Authoritative checkpoint
└── events/
    ├── action_proposed.cbor       # HolonActionProposed
    ├── action_verified.cbor       # HolonActionVerified
    ├── action_executed.cbor       # HolonActionExecuted
    └── parent_events/             # Causal chain
└── cgr_trace/
    ├── proof.cbor                 # Full CGRProof
    ├── reduction_steps.cbor        # Step-by-step reduction
    ├── invariants_checked.json    # Which invariants, results
    └── kernel_attestation.sig     # Kernel signature
└── context/
    ├── reasoning_trace.cbor      # Holon's reasoning (redacted)
    └── redaction_manifest.json    # What was redacted, why
└── attestations/
    └── holon_attestation.cbor    # Current HolonAttestation
└── verification/
    ├── README.md                 # Human-readable verification guide
    ├── verify.sh                  # Automated verification script
    ├── expected_hashes.json       # Expected hash values
    └── kernel_binary.wasm         # Kernel for replay verification
└── chain_of_custody.json          # Custody trail

```

## 19.6.2 Verification Instructions

The verification/ directory MUST include:

- README.md: Bundle purpose, artifact list, step-by-step verification, expected outcomes
- verify.sh: Automated script for CGR replay and hash verification
- expected\_hashes.json: Pre-computed hashes for all critical artifacts
- kernel\_binary.wasm: Exact kernel version for independent replay

## 19.6.3 Court Admissibility Checklist

#	Criterion	Verification Method
1	Authenticity	CGRProof kernel signature + checkpoint validator signatures
2	Integrity	BLAKE3 hashes match expected values; Merkle proofs valid
3	Completeness	All referenced events present; parent chain complete
4	Chain of Custody	chain_of_custody.json documents all handlers
5	Reproducibility	CGR replay produces identical proof
6	Timeliness	HLC timestamps within acceptable bounds

## 20. Deployment Architecture

### 20.1 Node Types

Node Type	Responsibilities	Requirements
<b>Validator</b>	BFT consensus, checkpoint signing, event validation	HSM for signing, 99.9% uptime SLA, dedicated hardware, private network, TEE optional
<b>Full Node</b>	Complete DAG storage, event validation, query serving with proofs	High storage, public API, verifiable query support, no consensus
<b>Light Client</b>	Checkpoint-only sync, proof verification	Minimal resources, mobile-friendly, trusts checkpoint sigs
<b>Indexer</b>	Query optimization, projection maintenance, proof generation	High memory, read replicas, MUST return proofs with all queries
<b>Gatekeeper</b>	Consent enforcement, vault key release	TEE REQUIRED, HSM for keys, StateProof verification

**Light Client Support:** Light clients sync only checkpoint events and verify inclusion proofs SPV-style. This enables mobile wallets without full DAG storage. Light clients MUST verify checkpoint signatures before trusting any proof.

### 20.2 Infrastructure Requirements

Component	MVP	Production
Validator Count	4 (tolerates f=1)	10+ (tolerates f=3)
CPU per Validator	8 cores	32+ cores
Memory per Validator	32 GB	128 GB
Storage per Validator	500 GB NVMe	2 TB+ NVMe RAID
Network	1 Gbps	10 Gbps + DDoS protection
HSM	Software HSM (dev only)	Hardware HSM (YubiHSM, CloudHSM)
TEE (Gatekeeper)	Simulation mode (dev)	Intel SGX / AWS Nitro REQUIRED
Full Nodes	2 (active + standby)	Geographic distribution (3+ regions)
Indexer Replicas	1	3+ with load balancer

## 21. Observability

### 21.1 Metrics (Prometheus)

All services expose /metrics endpoint in Prometheus format.

#### Ledger Metrics

exo_dag_events_total{type}	Counter: events by payload type
exo_dag_append_duration_seconds	Histogram: append latency
exo_dag_size_bytes	Gauge: total DAG storage
exo_dag_depth	Gauge: longest path from genesis
exo_dag_hlc_skew_seconds	Histogram: observed HLC skew

#### Consensus Metrics

exo_consensus_checkpoints_total	Counter: checkpoints produced
exo_consensus_finality_seconds	Histogram: time to finality
exo_consensus_validators_active	Gauge: active validator count
exo_consensus_round_duration_seconds	Histogram: BFT round time

#### Identity Metrics

exo_identity_created_total	Counter: identities created
exo_identity_rotations_total	Counter: key rotations
exo_identity_resolution_duration_seconds	Histogram: resolve latency

#### Consent Metrics

exo_consent_grants_total	Counter: consents granted
exo_consent_revocations_total	Counter: consents revoked
exo_consent_active_count	Gauge: active consents
exo_consent_check_duration_seconds	Histogram: check latency
exo_gatekeeper_releases_total	Counter: key releases
exo_gatekeeper_denials_total{reason}	Counter: denials by reason

#### Scoring Metrics

exo_score_requests_total	Counter: score requests
exo_score_distribution	Histogram: score values (0-100)
exo_score_duration_seconds	Histogram: scoring latency

#### Proof Metrics

exo_proof_event_generation_seconds	Histogram: EventInclusionProof generation
exo_proof_state_generation_seconds	Histogram: StateProof generation
exo_proof_verification_total{result}	Counter: verification pass/fail

### 21.2 Alerting Thresholds

Condition	Priority	Response
Finality latency > 5s (p99)	P1	Page on-call; investigate validator health
Validator offline	P1	Page on-call; verify HSM; check network
TEE attestation stale (>1 hour)	P1	Page on-call; rotate Gatekeeper
API error rate > 1%	P2	Slack alert; review error logs
Proof verification failure rate > 0.1%	P2	Investigate indexer consistency
Storage utilization > 80%	P3	Ticket; plan capacity expansion
Benchmark regression > 10%	P2	Block deployment; investigate

## 22. Implementation Phases

### Phase 0: Threat Modeling (Week 1)

**Objective:** Map threats to code boundaries before writing code

**Deliverables:**

- Threat-to-test matrix (each threat in Section 13 has  $\geq 1$  test)
- Security architecture review document
- Identified attack surfaces with mitigation assignments
- Cross-implementation hash test vectors

**Exit Criteria:** Security team sign-off on threat model

### Phase 1: Verifiable Core (Weeks 2-4)

**Objective:** Build trust-minimized ledger foundation

**Crates:** exo-core, exo-dag

**Deliverables:**

- BLAKE3 hashing, Ed25519 signing, canonical CBOR serialization
- EventEnvelope and LedgerEvent structs with normative hashing
- HybridLogicalClock with validation rules
- DAGStore trait + Sled implementation
- append() → Result<EventId, LedgerError> with all validations
- verify\_integrity() → Result<(), LedgerError>
- Checkpoint with event\_root (MMR) and state\_root (SMT)
- Proptest suite: random DAG generation, HLC invariants
- criterion.rs benchmarks
- Cross-implementation hash compatibility test

**Exit Criteria:** 80% coverage; all benchmarks pass; hash compatibility passes; security review

### Phase 2: Identity & Consent (Weeks 5-7)

**Objective:** Implement core fabric services

**Crates:** exo-identity, exo-consent

**Deliverables:**

- DID derivation from genesis key (immutable)
- Key versioning and rotation with proofs
- AccessorSet enum and Policy evaluation
- Bailment lifecycle (propose, grant, revoke)
- Gatekeeper with TEE simulation mode
- RiskAttestation with audience binding

**Exit Criteria:** E2E tests pass; consent expiry test passes; key rotation test passes

## Phase 3: Verifiable Queries & Proofs (Weeks 8-9)

**Objective:** Trustless query responses

**Deliverables:**

- MMR implementation for event\_root
- SMT implementation for state\_root
- EventInclusionProof generation and verification
- StateProof generation and verification
- Graph Indexer returns proofs with all queries
- Evidence bundle export (ZIP format)

**Exit Criteria:** All queries return verifiable proofs; evidence bundle passes verification

## Phase 4: API & Observability (Weeks 10-12)

**Objective:** Production-ready interfaces

**Crates:** exo-api

**Deliverables:**

- GraphQL schema implementation
- REST endpoints with /v1/ prefix
- Rate limiting (100 req/min per DID)
- Prometheus metrics for all services
- Structured logging with PII redaction
- Grafana dashboards with alerting
- Gatekeeper TEE deployment (real hardware)

**Exit Criteria:** Load test passes (250 tx/s sustained); all proofs verify

## Phase 5: Hardening & Audit (Weeks 13-16)

**Objective:** Production hardening and external audit

**Deliverables:**

- Security audit by external firm
- Penetration testing
- Performance optimization (target: <1ms append)
- Documentation completion
- Operational runbook creation
- Disaster recovery testing
- PACE protocol full implementation and test

**Exit Criteria:** No HIGH/CRITICAL audit findings; DR test passes; PACE test scenarios pass

## Phase 6: Optional Credits Module (Post-MVP)

◆ NEW v2.2: NEW v2.2: Optional EXO Credits implementation.

**Objective:** Utility credit system for hosted services

**Crates:** exo-credits, exo-oracle

**Deliverables:**

- Credit event types: TokenMint, TokenBurn, TokenRedeem, TokenGift, OracleUpdate, FeeEvent
- Fixed-point arithmetic library with overflow protection
- Blended rate calculation with goodwill multiplier
- Oracle integration (3-of-5 quorum, median aggregation)
- Vesting bailments for allocation time-locks
- INV-010 (supply cap) and INV-011 (alignment gate) enforcement
- Gift-only transfer with 0identity verification
- FeeEvent transparency with margin allocation tracking
- Transparency dashboard for public margin reporting
- Credits-enabled genesis configuration

**Exit Criteria:**

- Credits arithmetic fuzz-tested (10M iterations)
- Oracle quorum and staleness tests pass
- Gift transfer between verified accounts works
- Marketplace operations correctly rejected
- Supply cap cannot be exceeded
- Margin allocation  $\geq 50\%$  to GPU expansion verified
- Legal review confirms utility-only compliance

## 23. Compliance Framework

### 23.1 Regulatory Matrix

Regulation	Requirement	Mechanism
<b>GDPR</b>	Right to erasure (Art. 17)	Delete vault blob + encryption key. Ledger events remain but are unlinkable. Redaction proofs in evidence bundle.
<b>GDPR</b>	Data portability (Art. 20)	Evidence bundle export with proofs (Section 19)
<b>GDPR</b>	Consent management (Art. 7)	ConsentGiven/ConsentRevoked events with immutable audit
<b>HIPAA</b>	PHI protection	All PHI in encrypted vaults; TEE-enforced access; BAA required
<b>HIPAA</b>	Audit controls	AccessLogged events; 6-year retention; evidence bundles
<b>SOC 2</b>	Access control	RiskAttestation scoring; policy-based consent; TEE Gatekeeper
<b>SOC 2</b>	Encryption	At rest: XChaCha20-Poly1305. In transit: TLS 1.3
<b>SOC 2</b>	Availability	BFT consensus; no SPOF; 99.9% SLA
<b>CCPA</b>	Right to know/delete	Same as GDPR mechanisms

## Appendix A: Canonical Encoding (Normative)

⚠️ Encoding inconsistencies cause hash mismatches. These rules are MANDATORY for cross-implementation compatibility.

### A.1 CBOR Encoding Rules

All hashed structures MUST be encoded using deterministic CBOR (RFC 8949 Core Deterministic Encoding).

#### Map Ordering

- Map keys MUST be sorted by byte-wise lexicographic order of their encoded form
- Duplicate keys MUST NOT appear

#### Integer Encoding

- Integers MUST use the shortest possible encoding
- Negative integers MUST use major type 1
- No BigNum encoding unless value exceeds i64 range

#### Float Handling

⚠️ Floats MUST NOT appear in any hashed structure due to canonicalization hazards.

- confidence\_bps uses u16 basis points (0-10000) instead of f32
- Any percentage or ratio MUST use fixed-point integer representation
- If a float must be stored, convert to string representation

#### String Encoding

- All strings MUST be valid UTF-8
- Strings MUST use UTF-8 encoding (major type 3)
- No indefinite-length strings

#### Byte String Encoding

- Byte strings MUST use major type 2
- No indefinite-length byte strings

#### Array Encoding

- Arrays MUST use definite-length encoding
- No indefinite-length arrays

### A.2 Reference Implementations

```
// Rust: use ciborium with deterministic mode
use ciborium::ser::into_writer;

fn canonical_cbor_encode<T: Serialize>(value: &T) -> Vec<u8> {
    let mut buffer = Vec::new();
    into_writer(value, &mut buffer).expect("CBOR encoding failed");
    buffer
}
```

```
// JavaScript: use cbor-x with canonical option
import { encode } from 'cbor-x';
const canonical = encode(value, { canonical: true });
```

## A.3 Test Vectors

Cross-implementation compatibility MUST be verified using these test vectors:

```
# Test Vector 1: Simple EventEnvelope
Input (JSON representation):
{
  "parents": [],
  "logical_time": { "physical_ms": 1702500000000, "logical": 0 },
  "author": "did:exo:2DrjgbN7MuSJYhVFLTkJKj",
  "key_version": 1,
  "payload": { "type": "IdentityCreated", ... }
}

Expected BLAKE3 hash: [32-byte hex to be computed from reference impl]

# Implementation MUST match this hash exactly.
```

## Appendix B: Cryptographic Specifications

### B.1 Hash Function: BLAKE3

<b>Security Level</b>	128-bit (256-bit output)
<b>Performance</b>	3-4x faster than SHA-256; parallelizable
<b>Features</b>	Built-in keyed mode (MAC), extendable output (XOF)
<b>Implementation</b>	blake3 crate v1.x (audited)

**Usage:** Event hashing, content addressing, MMR nodes, SMT nodes

### B.2 Signature Scheme: Ed25519

<b>Security Level</b>	128-bit
<b>Key Size</b>	32 bytes (public), 64 bytes (private)
<b>Signature Size</b>	64 bytes
<b>Properties</b>	Deterministic (no RNG in signing), fast verification
<b>Implementation</b>	ed25519-dalek crate v2.x (audited)

**Usage:** Event signing, key rotation proofs, validator checkpoint signatures

### B.3 Encryption: XChaCha20-Poly1305

<b>Security Level</b>	256-bit (key), 128-bit (auth tag)
<b>Nonce Size</b>	192 bits (safe for random generation)
<b>Properties</b>	AEAD, constant-time, misuse-resistant
<b>Implementation</b>	chacha20poly1305 crate (audited)

**Usage:** Vault blob encryption, steward share encryption

### B.4 Secret Sharing: Shamir + Feldman VSS

<b>Scheme</b>	Shamir's Secret Sharing with Feldman commitments
<b>Security</b>	Information-theoretic ( $k-1$ shares reveal nothing)
<b>Verifiability</b>	Feldman commitments allow share validation without reconstruction
<b>Implementation</b>	vsss-rs crate (audited)

**Usage:** PACE recovery share generation and verification

## Appendix C: Error Code Reference

Code	Category	Description & Resolution
<b>EXO-1001</b>	Validation	InvalidSignature: Signature verification failed. Verify signing key matches author DID and key_version.
<b>EXO-1002</b>	Validation	ParentNotFound: Referenced parent event_id not in DAG. Sync node or verify parent was submitted.
<b>EXO-1003</b>	Validation	CausalityViolation: Event HLC not greater than all parents. Check logical_time computation.
<b>EXO-1004</b>	Validation	DuplicateEvent: event_id already exists. No action needed (idempotent).
<b>EXO-1005</b>	Validation	InvalidPayload: Payload fails schema validation. Check payload structure.
<b>EXO-1006</b>	Validation	KeyVersionMismatch: key_version doesn't match active key for author. Use current key_version.
<b>EXO-1007</b>	Validation	FutureTimestamp: physical_ms too far in future. Check system clock.
<b>EXO-2001</b>	Consensus	InsufficientQuorum: Checkpoint lacks 2f+1 signatures. Wait for more validators.
<b>EXO-2002</b>	Consensus	CheckpointConflict: Competing checkpoint at same height. Protocol violation alert.
<b>EXO-2003</b>	Consensus	ValidatorNotAuthorized: Signer not in current validator set.
<b>EXO-3001</b>	Policy	ConsentNotFound: No ConsentGiven for resource/accessor pair. Request consent.
<b>EXO-3002</b>	Policy	ConsentExpired: Policy valid_until exceeded. Request new consent.
<b>EXO-3003</b>	Policy	ConsentRevoked: ConsentRevoked event exists. Cannot access.
<b>EXO-3004</b>	Policy	AccessLimitExceeded: max_access_count reached. Request new consent.
<b>EXO-3005</b>	Policy	PurposeMismatch: Access purpose doesn't match policy. Use correct purpose.
<b>EXO-3006</b>	Policy	AccessorNotAuthorized: Accessor DID not in policy.accessors. Check AccessorSet.
<b>EXO-4001</b>	Identity	DidNotFound: DID not registered. Create identity first.
<b>EXO-4002</b>	Identity	InvalidRotationProof: old_key.sign(new_key) verification failed.
<b>EXO-4003</b>	Identity	KeyRevoked: Signing key has been revoked. Use current key.
<b>EXO-4004</b>	Identity	DuplicateDid: DID already registered. Use existing or new keypair.
<b>EXO-5001</b>	Recovery	InsufficientShares: < threshold valid shares for reconstruction.
<b>EXO-5002</b>	Recovery	RecoveryInProgress: Another recovery attempt active. Wait or abort.
<b>EXO-5003</b>	Recovery	RecoveryCooldown: 30-day cooldown active. Wait for expiry.
<b>EXO-5004</b>	Recovery	InvalidShare: VSS verification failed for submitted share.
<b>EXO-6001</b>	API	RateLimitExceeded: 100 req/min limit hit. Wait and retry.
<b>EXO-6002</b>	API	AuthenticationFailed: Invalid or expired token.
<b>EXO-6003</b>	API	InvalidRequest: Request validation failed. Check schema.
<b>EXO-6004</b>	API	AudienceMismatch: RiskAttestation.audience != verifier DID.
<b>EXO-7001</b>	Proof	InvalidProof: Proof verification failed against checkpoint root.
<b>EXO-7002</b>	Proof	StaleCheckpoint: Proof references outdated checkpoint. Refresh.
◆ EXO-13001	Credits	◆ CreditsModuleDisabled: Credits operations invalid on this network. Enable Credits module or use Credits-enabled network.
◆		
◆ EXO-13002	Credits	◆ InsufficientCredits: Account balance < redemption amount. Acquire more credits.
◆		
◆ EXO-13003	Credits	◆ SupplyCapExceeded: TokenMint would exceed MAX_SUPPLY (1B). INV-010 violation.
◆		
◆ EXO-13004	Credits	◆ AlignmentBelowRedeemThreshold: Redeemer alignment score below MIN_REDEEM_ALIGNMENT. INV-011 violation.
◆		
◆ EXO-13005	Credits	◆ InvalidGiftRecipient: Recipient not 0entity-bound or not on same network.
◆		
◆ EXO-13006	Credits	◆ TransferTypeProhibited: Only GIFT transfers permitted. No sale/exchange.
◆		

<b>Code</b>	<b>Category</b>	<b>Description &amp; Resolution</b>
◆ EXO-13007	Credits	◆ OracleStale: Oracle data older than staleness limit. Wait for fresh update.
◆ EXO-13008	Credits	◆ OracleQuorumNotMet: Insufficient oracle signatures for rate update.
◆ EXO-13009	Credits	◆ VestingLocked: Credits still in vesting bailment. Wait for unlock.

## Appendix G: Optional Tokenomics Module (EXO Credits)

◆ NEW v2.2: Optional utility credit system for Foundation-hosted services.

⊖ UTILITY-ONLY: EXO Credits are STRICTLY utility access credits. They MUST NOT be marketed, described, or represented as investment instruments, stores of value, profit-bearing, yield-bearing, or appreciation-oriented assets.

### G.1 Module Activation

The EXO Credits module is OPTIONAL. Network-level consensus determines enabled/disabled status.

#### G.1.1 Consensus Safety

Deterministic enforcement: Either Credits are DISABLED (all Token\* operations invalid) OR Credits are ENABLED (all validators MUST enforce Credits rules). Mixed validator support is impossible.

```
pub struct NetworkConfig {
    // ... existing fields ...
    pub credits_enabled: bool,
    pub credits_genesis_checkpoint: Option<Blake3Hash>,
}
```

- credits\_enabled=false: TokenMint, TokenBurn, TokenRedeem, TokenGift, OracleUpdate events are REJECTED with EXO-13001
- credits\_enabled=true: All validators MUST support Credits operations. Non-Credits validators MUST NOT be in validator set.
- Transition: Requires unanimous validator upgrade + new genesis checkpoint. No hot-swap.

### G.2 Credit Nature and Supply

#### G.2.1 Utility-Only Designation

⚠ NON-NEGOTIABLE: EXO Credits are redeemable ONLY for Foundation-hosted services. They are NOT investment instruments.

Permitted uses:

- Holon inference compute (per-token pricing)
- CGR Kernel proof generation
- Identity adjudication (RiskAttestation) services
- Vault storage and bandwidth
- Other Foundation-hosted EXOCHAIN services

Prohibited characterizations:

- Investment vehicle
- Store of value
- Speculative asset
- Yield-bearing instrument
- Appreciating asset

#### G.2.2 Fixed Supply

```

pub const MAX_SUPPLY: u64 = 1_000_000_000_000_000; // 1B credits in micro-units
(6 decimals)
pub const CREDIT_DECIMALS: u8 = 6;

```

Total supply cap: 1,000,000,000 credits (1B). No inflation mechanism. INV-010 enforced by CGR Kernel.

### G.2.3 Initial Allocation

Bucket	Allocation	Vesting / Controls
Community (Merit-based)	40%	Distributed via merit attestations. 5-year linear unlock via bailment. Claimable by 0entity-verified subjects only.
Ecosystem Grants	30%	DAO-governed grants for builders. 2-year cliff + 2-year linear vest. Milestone-gated releases.
Foundation Treasury	20%	R&D, operations, hosting, GPU scale-out. 4-year linear vest. Quarterly transparency reports.
Validator Incentives	10%	Distributed to active validators. NOT yield—operational compensation. Monthly distribution.

'No pre-mine' means: No pre-sale, no insider sale, no distribution prior to post-audit genesis activation. All allocations subject to time-locked bailments.

## G.3 Fixed-Point Arithmetic

⚠ All economics math MUST use integer fixed-point. NO floating point. Values used in hashing/signing MUST be canonical.

### G.3.1 Unit Definitions

```

// Base unit: micro-credit (uCRED)
pub const MICRO_CREDIT: u64 = 1;
pub const CREDIT: u64 = 1_000_000; // 10^6 micro-credits

// Rate unit: micro-USD per 1000 tokens
pub type MicroUsdPer1kTokens = u64;

// Basis points for multipliers (10000 = 1.0×)
pub type BasisPoints = u32;
pub const BASIS_POINT_SCALE: u32 = 10_000;

```

### G.3.2 Blended Rate Calculation

For multi-model transactions, blended rate is computed as weighted average:

```

/// Compute blended base rate for multi-model transaction
/// Returns MicroUsdPer1kTokens
pub fn compute_blended_rate(
    model_rates: &[(MicroUsdPer1kTokens, u64)], // (rate, token_count) pairs
) -> MicroUsdPer1kTokens {
    let total_tokens: u64 = model_rates.iter().map(|(_, t)| t).sum();
    if total_tokens == 0 { return 0; }

    let weighted_sum: u128 = model_rates
        .iter()
        .map(|(rate, tokens)| (*rate as u128) * (*tokens as u128))
        .sum();

    // FLOOR rounding (always rounds down)
    (weighted_sum / total_tokens as u128) as u64
}

```

Rounding rule: FLOOR (truncate toward zero). This ensures determinism and slight favor to users.

### G.3.3 Goodwill Multiplier

Foundation applies a margin via the Goodwill Multiplier:

```
pub const DEFAULT_GOODWILL_MULTIPLIER_BP: u32 = 15_000; // 1.5x (150%)

/// Apply goodwill multiplier to base rate
/// margin_share_of_final = (final - base) / final = 33.33% when multiplier = 1.5x
pub fn apply_goodwill_multiplier(
    base_rate: MicroUsdPer1kTokens,
    multiplier_bp: BasisPoints,
) -> MicroUsdPer1kTokens {
    // Integer math: final = (base * multiplier) / 10000
    let product: u128 = (base_rate as u128) * (multiplier_bp as u128);
    (product / BASIS_POINT_SCALE as u128) as u64
}
```

With default 15000 BP (1.5x):

- $\text{base\_rate} = 1000 \mu\text{USD}/1\text{k} \rightarrow \text{final\_rate} = 1500 \mu\text{USD}/1\text{k}$
- $\text{margin\_captured} = 500 \mu\text{USD}/1\text{k}$  (33.33% of final)

### G.3.4 Overflow Behavior

```
/// Safe multiplication with overflow check
pub fn safe_mul(a: u64, b: u64) -> Result<u64, CreditError> {
    a.checked_mul(b).ok_or(CreditError::Overflow)
}
```

- All arithmetic MUST use checked operations
- Overflow → transaction rejected (not silent wrap)
- u128 intermediate values for multiplication before division

## G.4 Oracle Governance

### G.4.1 Oracle Sources

Credit-to-USD peg derived from external model pricing:

- Primary: OpenAI API published rates
- Secondary: Anthropic API published rates
- Tertiary: Hugging Face Inference API rates
- Additional sources MAY be added via governance

### G.4.2 Oracle Update Process

```
pub struct OracleUpdate {
    pub rates: Vec<ModelRate>,
    pub effective_from: HybridLogicalClock,
    pub signatures: Vec<OracleSignature>, // Quorum required
    pub rate_hash: Blake3Hash, // canonical_cbor(rates)
}

pub struct ModelRate {
    pub model_id: String, // e.g., "gpt-4-turbo"
```

```

    pub micro_usd_per_1k_input: u64,
    pub micro_usd_per_1k_output: u64,
    pub source_url: String,
    pub observed_at: u64, // Unix timestamp
}

```

### G.4.3 Oracle Governance Rules

Parameter	Value / Rule
Oracle Committee Size	5 members (Foundation-appointed, diverse institutions)
Update Quorum	3-of-5 signatures required
Update Cadence	Maximum: every 4 hours. Minimum gap: 1 hour.
Staleness Limit	8 hours. After 8h without update, redemptions paused until fresh data.
Manipulation Resistance	Median-of-medians across sources. Outliers (>20% deviation) excluded.
Replay Protection	Monotonic sequence number. Each update MUST increment seq.
Failure Mode	Stale oracle → redemptions paused, mints paused. Gifts still permitted.

## G.5 Credit Events

◆ NEW v2.2: All credit events require CGR proof and policy checks when Credits module is enabled.

### G.5.1 TokenMint

```

pub struct TokenMint {
    pub recipient_did: Did,
    pub amount: u64, // micro-credits
    pub mint_reason: MintReason,
    pub merit_attestation_cid: Option<Cid>, // Required for MERIT_CLAIM
    pub vesting_bailment_cid: Option<Cid>, // Time-lock if applicable
    pub cgr_proof: CGRProof, // Proves INV-010 satisfied
}

pub enum MintReason {
    GENESIS_ALLOCATION,
    MERIT_CLAIM,
    GRANT_DISTRIBUTION,
    VALIDATOR_REWARD,
    FREE_TIER_ALLOCATION,
}

```

### G.5.2 TokenBurn

```

pub struct TokenBurn {
    pub burner_did: Did,
    pub amount: u64,
    pub burn_reason: BurnReason,
}

pub enum BurnReason {
    SERVICE_REDEMPTION, // Credits consumed for service
    VOLUNTARY_BURN, // User-initiated destruction
    SLASHING_PENALTY, // Validator misbehavior
}

```

### G.5.3 TokenRedeem

```

pub struct TokenRedeem {
    pub redeemer_did: Did,
    pub amount: u64,
    pub service_type: ServiceType,
    pub service_request_cid: Cid,
    pub blended_rate_used: u64, // Rate at redemption time
    pub cgr_proof: CGRProof, // Proves INV-011 satisfied
}

pub enum ServiceType {
    HOLON_INFERENCE { model_id: String, token_count: u64 },
    CGR_PROOF_GENERATION,
    ADJUDICATION,
    VAULT_STORAGE { gb_months: u64 },
    CUSTOM { service_id: String },
}

```

## G.5.4 TokenGift

**⚠ ONLY permitted transfer type. No marketplace, no escrow-for-sale, no bridging.**

```

pub struct TokenGift {
    pub sender_did: Did,
    pub recipient_did: Did,
    pub amount: u64,
    pub gift_message_cid: Option<Cid>, // Optional note
    pub transfer_type: TransferType, // MUST be GIFT
}

pub enum TransferType {
    GIFT, // Only permitted type
    // SALE - PROHIBITED
    // EXCHANGE - PROHIBITED
    // BRIDGE - PROHIBITED
}

```

**Gift constraints:**

- Sender MUST have Oidentity with RiskAttestation score  $\geq 70$
- Recipient MUST have Oidentity on same network
- No value consideration permitted (auditable)
- Protocol MUST NOT include: listings, bids, escrow-for-sale, AMM/DEX, price discovery endpoints
- Credits MUST NOT be bridgeable to external chains or external wallets

## G.5.5 FeeEvent (Margin Transparency)

```

pub struct FeeEvent {
    pub service_event_id: Blake3Hash, // Links to TokenRedeem
    pub base_cost: u64,
    pub multiplier_applied: u32,
    pub final_cost: u64,
    pub margin_captured: u64, // final - base
    pub margin_allocation: MarginAllocation,
}

pub struct MarginAllocation {
    pub gpu_expansion_earmark: u64, // ≥50% of margin MUST go here
}

```

```

    pub operations_pool: u64,
    pub reserve_pool: u64,
}

```

At least 50% of captured margin MUST be earmarked (on-ledger reporting) for GPU/datacenter expansion.

## G.6 Worked Example: Vendor-Paid Adjudication

◆ NEW v2.2: Complete flow where vendor pays EXOCHAIN for user adjudication; user pays nothing.

Scenario: E-commerce vendor wants to verify user identity before high-value purchase.

### Step 1: Vendor Requests Adjudication

```

// Vendor submits adjudication request
AdjudicationRequest {
    requester_did: "did:exo:vendor123",
    subject_did: "did:exo:user456",
    purpose: "HIGH_VALUE_PURCHASE",
    context_signals: { device_id, ip_reputation, ... },
    payment_source: "did:exo:vendor123", // Vendor pays
}

```

### Step 2: Rate Calculation

```

// Oracle rates (current)
adjudication_base_rate = 500 μUSD per adjudication

// Apply goodwill multiplier (1.5x)
final_rate = (500 * 15000) / 10000 = 750 μUSD

// Convert to credits at current peg
credit_cost = 750 μCRED (1:1 peg at launch)

```

### Step 3: Credit Debit

```

TokenRedeem {
    redeemer_did: "did:exo:vendor123",
    amount: 750,
    service_type: ADJUDICATION,
    service_request_cid: Cid("adjudication_request_hash"),
    blended_rate_used: 750,
    cgr_proof: <proof that vendor.alignment >= threshold>,
}

```

### Step 4: Service Execution

```

// Scoring Engine performs adjudication
RiskAttestation {
    subject_did: "did:exo:user456",
    score: 85,
    audience: "did:exo:vendor123",
    expires_at: now + 5min,
    ... // Other fields
}

```

### Step 5: Margin Accounting

```

FeeEvent {

```

```

    service_event_id: <TokenRedeem hash>,
    base_cost: 500,
    multiplier_applied: 15000,
    final_cost: 750,
    margin_captured: 250,
    margin_allocation: {
        gpu_expansion_earmark: 125, // 50% minimum
        operations_pool: 75,
        reserve_pool: 50,
    },
}

```

Result: Vendor debited 750 credits. User authenticated at no cost. Foundation captures 250 credit margin with transparent allocation.

## G.7 Free Tier for Merit Holders

Merit holders MAY receive monthly free credit allocations:

Merit Tier	Monthly Allocation	Source & Controls
Bronze Merit	1,000 credits	From Community bucket. Non-transferable. Expires monthly.
Silver Merit	2,500 credits	From Community bucket. Non-transferable. Expires monthly.
Gold Merit	5,000 credits	From Community bucket. Non-transferable. Expires monthly.

- Allocations accounted within fixed supply (no extra minting)
- Use-it-or-lose-it: Unused free tier credits expire, return to Community bucket
- Cannot be gifted—personal use only

## Appendix H: Trademark & Mark Usage Policy

◆ NEW v2.2: Formal reservation and protection of EXOCHAIN marks.

### H.1 Reserved Marks

The following marks are reserved and protected:

Mark	Scope
EXOCHAIN	Platform name. All uses.
EXO	Short form. All uses.
EXO TOKEN / EXO CREDITS	Credit system naming. All uses.
0DENTITY / ZERODENTITY	Identity system. All uses.
LEGALDYNE	Legal tech integration. All uses.
BAILMENT.AI	AI bailment services. All uses.
SENTINEL.AI	AI monitoring services. All uses.
AI-IRB	Governance body. All uses.
AI-SDLC	Lifecycle framework. All uses.
AI-SDLC.INSTITUTE	Educational/certification. All uses.
LYNK / LYNK PROTOCOL	Protocol naming. All uses.

### H.2 OSS-Friendly Usage Rules

#### H.2.1 Permitted Uses (Nominative Fair Use)

- Referring to EXOCHAIN in documentation, articles, academic papers
- Stating compatibility with EXOCHAIN (e.g., 'Compatible with EXOCHAIN')
- Fork naming that clearly distinguishes from official (e.g., 'MyProject-EXOCHAIN-Fork')
- Conference talks and educational materials

#### H.2.2 Prohibited Uses

- Implying official endorsement without written permission
- Using marks in product names that could cause confusion
- Modifying logos or marks
- Domain names containing marks without permission

#### H.2.3 Logo Usage

- Official logos available at: [foundation-url]/brand
- Logos MUST NOT be modified, recolored, or distorted
- Minimum clear space requirements apply
- Contact brand@[foundation-domain] for permission requests

#### H.2.4 Permission Process

For uses not covered by nominative fair use:

- Submit request to legal@[foundation-domain]
- Include: intended use, context, duration, audience
- Response within 10 business days
- Written permission required before use

## Document Control

This document is the authoritative specification for EXOCHAIN Fabric Platform v2.2 — Constitutional Substrate for Aligned Superintelligence with Optional Utility Credits. All implementation decisions MUST trace back to requirements defined herein.

<b>Prepared By</b>	Architecture Team / AI Safety Team / Economics Team
<b>Reviewed By</b>	Security, Engineering, AI-IRB, Legal, External Auditors
<b>Formal Verification</b>	CGR Kernel proofs: [Pending Coq/Lean review]
<b>Approved By</b>	
<b>Date</b>	

## v2.2 Reasoning Summary

### ◆ NEW v2.2: Key design decisions for the EXO Credits module.

- **Utility-Only Stance:** Credits are access tokens for services, NOT investment instruments. This is non-negotiable and enforced at protocol level (no marketplace mechanics, no price discovery endpoints).
- **Gift-Only Transfer:** The ONLY permitted transfer is gifting between verified Odentity accounts. No sale, exchange, bridge, or external withdrawal. This prevents speculation and maintains utility focus.
- **Deterministic Math:** All economics use integer fixed-point arithmetic with explicit rounding rules (FLOOR). No floating point. u128 intermediates prevent overflow. Canonical encoding for all hashed/signed values.
- **Oracle Safety:** 3-of-5 quorum, median-of-medians aggregation, staleness limits, replay protection. Failure mode pauses redemptions rather than using stale data.
- **Mission-Aligned Margin:** Goodwill Multiplier captures margin transparently via FeeEvent. ≥50% earmarked for GPU expansion. On-ledger accounting enables public audit.
- **Consensus Safety:** Credits module is network-level deterministic. Either all validators enforce Credits rules or Credits operations are invalid. No mixed support possible.

The result is an optional utility credit system that aligns with EXOCHAIN's constitutional governance model while maintaining the utility-only nature required for compliant operation.

— End of Specification —

"Excellence and Safety in the Age of Superintelligence"