ORION: A Cryptographically Trusted Overlay

for Distributed Discovery and Presence

Technical Whitepaper v0.1 — Shareable Edition

Luis Mendoza
Scraplay LLC
luis.mendoza@scraplay.com
https://orion.scraplay.com

June 2026

Trust the Identity. Not the Location.

Abstract

Modern distributed systems couple discovery and trust to network location: DNS names, IP addresses, and specific providers. When infrastructure migrates, fails over, or is censored, clients break because they trusted *where* data came from instead of *who* signed it. ORION (Open Resilient Identity Overlay Network) is a protocol and architecture for cryptographically trusted discovery and presence. Identity is hierarchical and offline-rooted; records are versioned, expiring, and Ed25519-signed; lookup is deterministic by namespace and subject rather than by endpoint. ORION replicates signed records through a Kademlia DHT and gossip notifications while maintaining a validating local cache for offline resilience. This whitepaper presents the design principles, eleven-layer architecture, core algorithms, threat model, and compliance framework. It is informative: normative requirements remain in the ORION specification series. ORION explicitly does not execute remote code, bypass host security, or replace TLS or DNS—it provides an identity-first overlay for discovery and presence above existing transport.

1 Introduction

Distributed applications discover services through names and addresses that change. Load balancers move, clouds migrate regions, domains expire, and providers fail. Traditional clients often encode trust in location: allowlists of IP addresses, hardcoded hostnames, or implicit trust in whichever DNS answer arrives first.

ORION inverts this model. Cryptographic identity is the sole basis of trust. A record received from any peer is equally valid if its signature verifies, its trust chain is sound, its version is current, and it has not expired. Network location—IP, domain, or geographic region—MUST NOT be used as a primary trust signal.

ORION is simultaneously a protocol, a language-independent specification, a layered architecture, and an interoperability standard. Any compliant implementation can interoperate with any other without reading another implementation's source code.

1.1 Contributions

This whitepaper contributes:

1. A hierarchical offline-root trust model for distributed discovery records.
2. Deterministic namespace/subject discovery keys with Ed25519-signed canonical JSON records, version precedence, and signed revocation.
3. An eleven-layer architecture separating cryptography, identity, transport, routing, presence, discovery, replication, trust, messaging, persistence, and consumer integration.
4. A compliance framework (SPEC-0008) and explicit security boundaries that forbid remote code execution and stealth behavior.

1.2 Non-Goals

ORION explicitly does **not** aim to:

- Execute remote commands or arbitrary code from network messages.
- Install, update, or persist software on hosts.
- Bypass operating system security or firewalls.
- Replace TLS, DNS, or blockchain naming systems.
- Provide general-purpose messaging (limited pub/sub for record updates only).
- Enforce licensing or access control at the protocol layer.

1.3 Document Structure

Section 2 surveys related systems. Section 3 states design principles. Section 4 describes the layered architecture. Sections 5–6 cover the core protocol. Section 7 presents the threat model. Section 8 outlines evaluation methodology. Section 9 summarizes compliance requirements. Section 11 concludes.

2 Related Work

Table 1 positions ORION relative to adjacent systems.

ORION is an *overlay*: it operates above QUIC [6] and libp2p-style routing, adding signed records, trust chains, deterministic discovery keys, and validation rules purpose-built for infrastructure-independent presence.

3 Design Principles

ORION design follows eight principles derived from the protocol specification.

Trust the Identity, Not the Location. Records are verified by signature and trust chain, not by source address. The anti-pattern "only accept records from 10.0.0.5" is non-compliant; the correct rule is "only accept records signed by an authorized operational key."

Signed Truth. Every ORION record MUST be cryptographically signed. Unsigned or tampered records MUST be rejected. Canonical serialization ensures deterministic signatures across implementations [5].

Table 1: ORION compared to related systems

System	Strength	Gap ORION addresses
DNS / DNSSEC [4]	Hierarchical naming; signed RRsets	Trust still tied to resolution path; limited presence model
X.509 / PKI	Widely deployed identity certificates	CA-centric chains; no namespace/subject discovery overlay
Certificate Transparency [7]	Auditable certificate issuance	Focus on TLS certs, not application discovery records
IPFS [2]	Content-addressed DHT storage	Content hash ≠ operational identity delegation
libp2p [9]	QUIC transport, Kademlia [8], GossipSub [10]	Transport/routing stack; ORION defines signed record layer above
Secure Scuttlebutt [1]	Signed append-only feeds	Social graph model; not namespace/subject service discovery

Replaceable Infrastructure. Servers, domains, IPs, and providers are ephemeral. Bootstrap nodes can be replaced; records are republished; cache fallback sustains operation during outages.

Protocol First. Normative requirements live in the specification, not in any single implementation. The reference implementation demonstrates compliance; it does not define the protocol.

Implementation Independence. All algorithms include deterministic examples and test vectors. Wire format is language-neutral (Canonical JSON; CBOR optional).

Auditability. Records include `issued_at`, `valid_until`, `version`, and `signing_key_id`. Revocation lists are themselves signed records.

Security First. ORION operates within strict boundaries: no remote code execution, no stealth channels, no firewall bypass, no automatic software installation from network messages.

No Single Point of Failure. Distributed DHT storage, gossip propagation, local cache, and multiple bootstrap peers eliminate dependence on a single server or IP.

4 Architecture

ORION is organized into eleven layers (0–10). Each layer has defined responsibilities; no layer bypasses adjacent layers.

Publish and resolve traverse layers 10→5→3→2 on the network path, with layer 7 validation on every accepted record and layer 9 cache on every successful resolution. Diagram sources for bootstrap mesh, record flows, and key hierarchy are maintained in the repository at `docs/diagrams/`.

5 Identity and Records

5.1 Key Hierarchy

ORION identity is a hierarchical cryptographic trust model:

- **Root key** (offline): ultimate trust anchor; signs operational keys only.

Table 2: ORION architectural layers

Layer	Name	Responsibility
0	Cryptography	Ed25519 [3], SHA-256, Base64URL, canonical serialization
1	Identity	Key hierarchy, PeerID, delegation, revocation tracking
2	Transport	QUIC / libp2p multiaddr dial and listen
3	Routing	Kademlia DHT record routing
4	Presence	`service.presence`, `node.presence`, `cluster.presence`
5	Discovery	Resolver, lookup by discovery key, namespace validation
6	Replication	DHT put, GossipSub publish, republishing
7	Trust	Signature verification, expiration, version precedence, revocation
8	Messaging	GossipSub notifications on `orion/records/{namespace}`
9	Persistence	Local validating cache, offline fallback
10	Consumer	Application integration (CLI, SDK, service mesh)

- **Operational key**: signs records for a namespace.
- **Delegated key**: limited-scope authorization from operational key.
- **Node key**: libp2p network participant identity.
- **Service key**: logical service entity identity.

Keeping root keys offline eliminates remote attack surface on the trust anchor and requires deliberate human action for hierarchy changes.

5.2 PeerID

A node's PeerID is deterministically derived:

$$\text{peer\_id} = \text{BASE64URL\_NO\_PAD}(\text{SHA-256}(\text{“orion:peer:v1:”} \parallel 0\text{x}00 \parallel \text{ed25519\_pubkey}))$$

5.3 Discovery Keys

A discovery key uniquely identifies a `(namespace, subject)` pair without embedding location data:

$$\text{discovery\_key} = \text{BASE64URL\_NO\_PAD}(\text{SHA-256}(\text{“orion:discovery:v1:”} \| 0\text{x}00 \| \text{namespace} \| 0\text{x}00 \| \text{subject}))$$

Example. `namespace = com.example`, `subject = api-gateway` yields discovery key `_9_i_HhEBET_80mqYP`

Namespaces use reverse-domain notation (`com.example`). Subjects identify entities within a namespace (`api-gateway`). Both are validated for length and character set before lookup.

5.4 Record Structure

Every ORION record MUST contain: `type`, `namespace`, `subject`, `version`, `issued_at`, `valid_until`, `payload`, `signing_key_id`, and `signature`.

Signing process:

1. Construct record without `signature`.
2. Serialize to Canonical JSON (keys sorted alphabetically, no insignificant whitespace, UTF-8).

3. Sign bytes with Ed25519; encode signature as Base64URL without padding.

Example canonical form:

```
{"issued_at":"2026-06-01T12:00:00Z","namespace":"com.example",
 "payload":{"status":"online"},"signing_key_id":"abc123",
 "subject":"api-gateway","type":"service.presence",
 "valid_until":"2026-06-02T12:00:00Z","version":1}
```

5.5 Versioning and Expiration

`version` MUST monotonically increase per `(namespace, subject)`. Version 0 is invalid. Resolvers MUST reject `incoming.version < cached.version` (downgrade). Equal version triggers deterministic conflict resolution (later `issued_at`, then lexicographically smaller signature). Records MUST include RFC 3339 UTC `valid_until`; expired records MUST NOT be served.

6 Discovery, Replication, and Presence

6.1 Lookup Flow

The resolver:

1. Validates namespace and subject format.
2. Computes discovery key.
3. Checks local cache (layer 9); serves if valid and unexpired.
4. Queries DHT at `/orion/record/{discovery_key}`.
5. Validates signature, trust chain, expiration, version, and revocation (layer 7).
6. Updates cache on success.

6.2 Replication

Publishers replicate via:

- Kademlia DHT `put_record` at `/orion/record/{discovery_key}`.
- GossipSub publish to topic `orion/records/{namespace}`.

Publishers SHOULD republish before `valid_until` to maintain availability. Gossip notifications trigger cache refresh attempts on resolvers.

6.3 Bootstrap and Presence

Bootstrap nodes form a cluster publishing a signed `network.bootstrap` ring record listing active bootstrap peers. Fixed seed addresses are startup hints only; the signed ring is authoritative. Presence record types (`service.presence`, `node.presence`, `cluster.presence`) announce availability and endpoints in the signed payload—clients resolve `namespace + subject`, not hardcoded IPs.

7 Threat Model and Security

7.1 In Scope

- Record tampering and signature forgery (mitigated by Ed25519 verification).
- Replay attacks (mitigated by monotonic `version` and `valid_until`).
- Downgrade attacks (mitigated by version precedence).
- Compromised signing keys (mitigated by signed `revocation.list` and re-delegation from offline root).
- Compromised network nodes (invalid records rejected; cannot forge without key).
- DHT poisoning (mitigated by mandatory validation before cache or serve).

7.2 Out of Scope

- OS-level compromise and physical access to offline root key storage.
- Quantum attacks on Ed25519 (future algorithm versioning path).
- Sybil resistance without additional application policy.

7.3 Limitations

ORION does not encrypt record payloads (application-layer encryption is required for confidentiality). ORION does not provide anonymity. DHT availability depends on network participation. Cache may serve slightly stale but still valid records within `valid_until`.

8 Evaluation

This section describes how ORION implementations are evaluated. Version 0.1 focuses on methodology and conformance criteria; quantitative benchmark tables are planned for version 0.2 and are not required for client or website distribution of this document.

8.1 Conformance

Reference implementations are validated against SPEC-0008 and interoperability test vectors in `tests/interoperability/`. Discovery key generation is deterministic: identical `(namespace, subject)` inputs MUST yield identical keys across runs and implementations (verified in CI for the Node.js and Rust SDKs).

8.2 Test Environment

- Reference implementations: `reference/orion-rs` (PoC) and `reference/orion` (production).
- Local bootstrap mesh (1–3 nodes) on QUIC/libp2p.
- Record type: `service.presence` for `com.example/api-gateway`.

Table 3: Performance metrics scheduled for whitepaper v0.2

Experiment	Metric
Cold resolve (empty cache)	p50 / p95 latency (ms)
Warm resolve (populated cache)	p50 / p95 latency (ms)
Publish → resolve round-trip	Time to valid record (ms)
Bootstrap unavailable + warm cache	Successful resolve rate (%)
Cross-implementation conformance	SPEC-0008 test vector pass rate

8.3 Planned Performance Metrics (v0.2)

8.4 Status

Version 0.1 establishes the evaluation framework. Measured results will be published in version 0.2 for arXiv and formal citation; they are optional for sharing this document with clients or hosting it on https://orion.scraplay.com.

9 Compliance and Interoperability

A compliant ORION implementation MUST satisfy all normative MUST requirements in SPEC-0008, including:

- Ed25519 signatures, SHA-256 hashes, Base64URL encoding, Canonical JSON.
- Full key hierarchy support with PeerID generation and delegation verification.
- Record signing and verification with expiration and downgrade rejection.
- Discovery key generation per Section 5.
- Local validating cache with fallback; MUST NOT serve expired entries.
- Security boundaries: no remote command execution or arbitrary code from messages.

Interoperability test vectors are maintained at `tests/interoperability/`. Implementations SHOULD support QUIC, Kademlia, and GossipSub but MAY substitute equivalent transports if normative record and validation semantics are preserved.

10 Discussion and Future Work

Publication path. The normative specification may remain proprietary during client deployments or be published under Creative Commons CC BY 4.0 with Apache-2.0/MIT reference implementations. This whitepaper is informative in either path.

Future directions include Canonical CBOR wire encoding, formal Internet-Draft submission, expanded observability and bootstrap HA patterns, and SDK coverage beyond the Rust and Node.js references.

11 Conclusion

ORION provides a cryptographically trusted overlay for distributed discovery and presence. By anchoring trust in signed identity rather than network location, it enables applications to survive infrastructure churn without hardcoded endpoints. The eleven-layer architecture, deterministic discovery keys, and explicit security boundaries make ORION auditable and implementation-independent. Normative requirements remain in the ORION specification series; adoption and deployment are arranged with Scraplay LLC.

A Record Types

Type	Purpose
`service.presence`	Service availability and endpoints
`node.presence`	Node status and multiaddrs
`cluster.presence`	Cluster health and membership
`network.bootstrap`	Signed bootstrap peer ring
`revocation.list`	Revoked signing key IDs
`update.channel`	Signed update metadata (no auto-install)

B Example Record (Illustrative)

```
{
  "type": "service.presence",
  "namespace": "com.example",
  "subject": "api-gateway",
  "version": 1,
  "issued_at": "2026-06-01T12:00:00Z",
  "valid_until": "2026-06-02T12:00:00Z",
  "payload": {
    "status": "online",
    "endpoint": "https://api.example.com",
    "health": "healthy"
  },
  "signing_key_id": "<operational_key_id>",
  "signature": "<ed25519_signature_base64url>"
}
```

C Normative Specification

The authoritative protocol definition is the ORION specification series (SPEC-0001 through SPEC-0010) in the project repository. At the time of v0.1, public implementation rights are not granted by this whitepaper. Contact Scraplay LLC for licensing and conformance questions.

References

[1] Dominic Anderson et al. Secure scuttlebutt, 2016.

[2] Juan Benet. IPFS — content addressed, versioned, P2P file system, 2014.

[3] Daniel J. Bernstein et al. High-speed high-security signatures, 2012.

[4] IETF. Domain name system security extensions (DNSSEC). RFC 4033–4035, 2005.

[5] IETF. The JavaScript object notation (JSON) data interchange format. RFC 8259, 2017.

[6] IETF. QUIC: A UDP-based multiplexed and secure transport. RFC 9000, 2021.

[7] IETF. Certificate transparency. RFC 9162, 2022.

[8] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric, 2002.

[9] Protocol Labs. libp2p: A modular network stack, 2024.

[10] Dimitris Vyzovitis et al. GossipSub: Scalable broadcasts for blockchains and beyond, 2020.