Independent Submission
Internet-Draft
Intended status: Informational
Expires: 9 November 2026

J. Xu
B. Li
R. Zhu
CAICT

8 May 2026

Architecture for Agentic Overlay Networks
draft-xu-agentic-overlay-network-architecture-00

Abstract

This document describes an architecture for agentic overlay networks. The architecture enables autonomous software agents, agent gateways, registries, discovery services, and enterprise service hubs to interoperate across administrative domains without replacing the existing Internet protocol stack.

The architecture separates control-plane coordination from runtime agent interaction. Management root nodes coordinate trust anchors, semantic taxonomies, and network-wide policy. Registry service nodes onboard agents and publish signed capability metadata. Discovery service nodes provide intent-driven candidate selection. Enterprise service hubs adapt private systems and legacy interfaces into agent-compatible capabilities. After discovery, runtime authentication and execution are performed by the invoking and target endpoints, which can be agents or gateways, using mutual verification; discovery and management nodes are not required in the runtime path.

This document defines architectural roles, functional boundaries, metadata expectations, operational workflows, and security considerations for such networks. It does not define a new transport protocol, a new URI scheme, or a mandatory discovery ranking algorithm.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

Autonomous software agents increasingly act as networked principals that can interpret intent, select tools, invoke remote services, and cooperate with other agents. Existing Internet mechanisms remain essential for packet delivery and host reachability, but they do not by themselves provide the agent-specific functions needed for large-scale cross-domain collaboration: capability registration, semantic discovery, verifiable identity, policy coordination, protocol adaptation, and auditable commercial metering.

An agentic overlay network adds these functions above the existing Internet. It is an overlay because it preserves IP, DNS, TLS, HTTP, QUIC, and other deployed infrastructure where those mechanisms are suitable. It is agentic because its primary subjects are autonomous agents and agent gateways rather than only hosts, users, or conventional web applications.

The architecture in this document is intended for multi-operator environments. It assumes that no single platform will control all agents, all discovery services, or all enterprise systems. It therefore defines a separation between network-wide coordination and decentralized execution. Coordination provides consistent semantics and trust anchors; execution remains between the selected runtime endpoints and does not require a central node in the data path.

1.1. Problem Statement

Agent deployments are converging on several recurring interoperation problems:

* agents can expose useful capabilities, but there is no common architectural boundary between capability registration, semantic discovery, credential verification, and runtime invocation;

* discovery systems often mix search, authorization, routing, and execution in one platform-specific service, which makes cross-domain deployment hard;

* existing host and service discovery mechanisms can identify hosts or endpoints, but they do not by themselves define agent capability semantics, freshness, proof of control, or agent-specific lifecycle state; and

* enterprises need a way to expose selected internal tools as agent capabilities without placing private systems or raw data in a global registry.

This document addresses those problems by defining architectural roles and boundaries. It deliberately stops short of defining a complete product ecosystem, commercial governance model, or mandatory implementation backend.

1.2. Applicability

This architecture is applicable when multiple organizations need to publish, discover, verify, and invoke autonomous agents or agent gateways across administrative boundaries. It is less useful for single-application systems where all agents, tools, credentials, and discovery functions are controlled by one implementation and do not need interoperable metadata or cross-domain trust.

1.3. Goals

The architecture has the following goals:

* enable agents to register stable identities, endpoint references, supported protocols, and capability metadata;

* enable discovery services to find candidate agents based on user or agent intent, capability tags, operational status, and policy constraints;

* permit heterogeneous agent protocols and legacy APIs to be adapted without forcing all deployments onto one interaction protocol;

* support decentralized runtime authentication between interacting endpoints;

* keep management nodes out of the runtime traffic path;

* allow operator-neutral governance of trust anchors, revocation, taxonomy synchronization, and audit artifacts; and

* provide a basis for interoperable metering and audit evidence without requiring every runtime interaction to be recorded on a public ledger.

1.4. Non-Goals

This document does not define:

* a replacement for IP, DNS, TLS, HTTP, QUIC, or other deployed Internet protocols;

* a new agent URI scheme;

* a mandatory agent-to-agent invocation protocol;

* a single global discovery algorithm;

* a universal reputation system;

* an economic settlement protocol; or

* an application-specific workflow language.

Companion specifications may define concrete message formats, transport bindings, identity credential profiles, discovery query schemas, or metering records that fit within this architecture.

1.5. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Terminology

Agent: An autonomous or semi-autonomous software system that can receive an intent or request, perform planning or tool use, and return a result or state transition.

Agent Gateway: A domain-local component that adapts an internal API, tool, service, or legacy system into an agent-compatible interface. An Agent Gateway is not a global discovery service unless it also implements that role.

Agentic Overlay Network: An overlay architecture that adds agent

identity, capability registration, semantic discovery, trust coordination, and operational metering above the existing Internet.

Capability Metadata: Structured and human-readable information describing what an agent can do, how it can be invoked, which protocols it supports, and which operational constraints apply.

Discovery Service Node: A node that accepts discovery requests and returns candidate agents. It performs search and candidate ranking, but it is not in the runtime execution path between agents.

Enterprise Service Hub: A private-domain component that encapsulates enterprise data sources, tools, and internal APIs and exposes selected capabilities through an Agent Gateway or Registry Service Node.

Management Root Node: A control-plane node that coordinates trust anchors, credential issuer authorization, semantic taxonomy synchronization, revocation information, and network-wide policy artifacts. It does not forward agent traffic.

Registry Service Node: A node that onboards agents or Agent Gateways, validates registration metadata, issues or requests identity credentials, and publishes capability metadata to authorized discovery infrastructure.

Semantic Taxonomy: A controlled hierarchy or graph of capability labels used to normalize registration and discovery across administrative domains.

Trusted Metering Log: A tamper-evident record or digest that summarizes an invocation, discovery, referral, or metering-relevant event without necessarily exposing private payload data.

3. Architectural Principles

The architecture follows these principles:

Control-plane coordination is limited. Management Root Nodes provide coordination and trust bootstrapping. They MUST NOT be required for ordinary runtime endpoint-to-endpoint message exchange.

Execution is decentralized. After discovery, the invoking agent or its gateway communicates directly with the target agent or target gateway. The parties perform mutual verification without relying on the Discovery Service Node as a trusted intermediary.

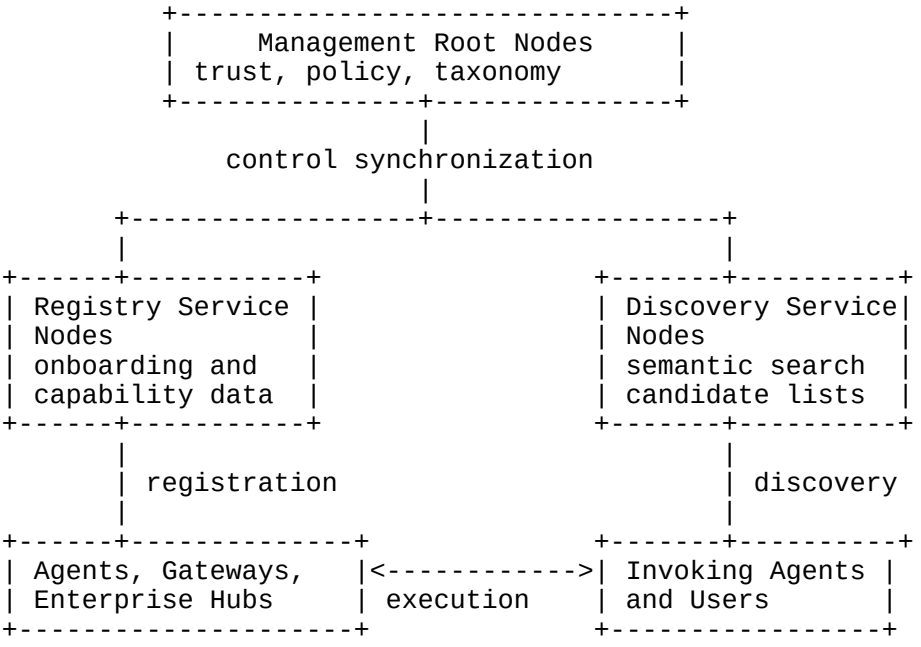
Metadata and payloads are separated. Registries and discovery services operate on capability metadata, endpoint references, operational status, and policy hints. They SHOULD NOT require access to private task payloads.

Protocol adaptation is local. Heterogeneous enterprise systems and agent protocols are adapted by Agent Gateways and Enterprise Service Hubs close to the systems they protect.

Semantic coordination is explicit. Capability labels and taxonomies SHOULD be synchronized through controlled processes so that discovery results have consistent meaning across domains.

4. Architecture Overview

An agentic overlay network is composed of a coordination plane, a service exchange plane, and a runtime interaction plane.



The key architectural boundary is that discovery and execution are separate. A Discovery Service Node returns candidates and metadata. The invoking party then verifies identity, policy, credentials, and endpoint information before using any candidate.

5. Logical Layers

The architecture can be understood as four logical layers. These layers are not wire-protocol layers; they are functional groupings used to assign responsibilities.

Layer	Function	Typical Components
Underlay Support	Existing network transport, compute, storage, and optional deterministic networking support	IP, DNS, TLS, HTTP, QUIC, edge compute
Interconnection	Registration, addressing references, endpoint discovery, and protocol adaptation	Registry Service Nodes, Discovery Service Nodes, Agent Gateways
Collaboration	Identity, trust, taxonomy synchronization, policy coordination, metering, and audit	Management Root Nodes, credential issuers, audit services
Application	User-facing and agent-facing services that express or satisfy intents	Agents, applications, enterprise services

Table 1

The architecture intentionally keeps most agent-specific semantics above the existing transport layer. Underlay technologies MAY provide quality, latency, or reliability enhancements, but they are not required to understand agent capability metadata.

6. Deployment Profiles

Deployments can adopt this architecture incrementally. This document defines four non-exclusive deployment profiles to make that progression explicit.

Profile	Name	Description
P0	Local Gateway	A single domain exposes local tools through Agent Gateways. No cross-domain registry is required.
P1	Domain Registry	A domain operates a Registry Service Node and publishes signed capability metadata for local agents.
P2	Federated Discovery	Multiple domains exchange catalog summaries and allow authorized Discovery Service Nodes to query or synchronize metadata.
P3	Coordinated Overlay	Management Root Nodes coordinate trust anchors, taxonomy versions, revocation state, and audit digests across multiple operators.

Table 2

Implementations SHOULD document which deployment profile they implement. Clients SHOULD degrade gracefully when interacting with a deployment that supports only a lower profile. For example, a P0 deployment can still expose a useful agent endpoint, but a client cannot assume global taxonomy synchronization or cross-operator revocation checks.

7. Node Roles

7.1. Management Root Nodes

Management Root Nodes are control-plane coordination nodes. A deployment MAY use one Management Root Node or a federation of such nodes operated by neutral or authorized organizations.

Management Root Nodes are responsible for:

* authorizing Registry Service Nodes and Discovery Service Nodes;

* publishing credential issuer metadata and revocation information;

* coordinating semantic taxonomy versions;

* publishing policy digests or governance artifacts;

* receiving compact catalog summaries or audit digests; and

* distributing synchronized control-plane state to authorized service nodes.

Management Root Nodes MUST NOT be required for:

* runtime request forwarding;

* target agent selection for an individual user request;

* access to private invocation payloads;

* direct enforcement of endpoint-level authorization decisions; or

* maintaining real-time fine-grained search indices for all agents.

This boundary reduces central bottlenecks and limits the operational power of the control plane.

7.2. Registry Service Nodes

Registry Service Nodes are supply-side onboarding hubs. They receive agent registrations, validate metadata, and publish capability records to discovery infrastructure.

A Registry Service Node SHOULD perform the following functions:

* validate that mandatory registration fields are present;

* verify endpoint reachability where applicable;

* validate imported OpenAPI, tool, or agent-card descriptions when supplied;

* assign or verify a stable agent identifier;

* issue, request, or bind credentials that associate the agent identifier with capability metadata;

* monitor lifecycle events such as update, suspension, deletion, and heartbeat status; and

* synchronize catalog summaries with authorized Management Root Nodes or Discovery Service Nodes.

A Registry Service Node MUST preserve the distinction between asserted capabilities and verified capabilities. If it cannot verify a claim, it MUST NOT mark that claim as verified.

Registrations SHOULD be soft state. A Registry Service Node SHOULD assign an expiration time to each active registration and require refresh before expiration. If an agent fails to refresh, the Registry Service Node SHOULD mark the registration inactive or remove it from discovery results.

7.3. Discovery Service Nodes

Discovery Service Nodes are demand-side search and candidate selection nodes. They accept discovery requests and return candidate agents with metadata that allows the requester to make its own trust decision.

A Discovery Service Node SHOULD support:

* lookup by stable identifier;
* discovery by capability tags;
* discovery by natural-language intent;
* filtering by protocol support, domain, policy, trust level, or operational status;
* ranking that combines semantic relevance with operational and policy signals; and
* explainable result metadata sufficient for clients to understand why a candidate was returned.

A Discovery Service Node MUST treat its own results as recommendations, not as authorization decisions. It MUST NOT require target agents to route runtime traffic through it.

7.4. Agent Gateways

Agent Gateways adapt local systems into agent-compatible interfaces. They are often deployed inside enterprises, clouds, or application platforms.

An Agent Gateway MAY:

* translate between agent protocols and internal REST, RPC, message bus, or tool interfaces;

* enforce local policy;

* perform local authentication and authorization;

* redact or transform private data before exposing a capability;

* maintain local logs required by the enterprise domain; and

* present a standardized capability record to a Registry Service Node.

Agent Gateways SHOULD NOT be assumed to be globally trusted. Remote agents SHOULD verify the gateway’s credentials and policy claims before invocation.

7.5. Enterprise Service Hubs

Enterprise Service Hubs are private-domain integration components. They are used when an organization wants to expose selected internal capabilities while keeping raw systems and data inside a protected environment.

An Enterprise Service Hub SHOULD expose only the minimum metadata needed for discovery. Sensitive schemas, raw datasets, internal credentials, and private business logic SHOULD remain local unless explicitly authorized by local policy.

8. Capability Metadata

Capability metadata is the primary object exchanged between registration and discovery functions. A concrete serialization is out of scope for this document, but a registration profile SHOULD include the fields in Table 1.

Field	Requirement	Description
agent_id	MUST	Stable identifier for the agent or gateway
name	MUST	Human-readable name
description	MUST	Natural-language capability description
capability_tags	SHOULD	Normalized tags selected from a semantic taxonomy
endpoints	MUST	Endpoint references or protocol-specific contact data
protocols	MUST	Supported invocation protocols or transport bindings
auth_methods	SHOULD	Supported authentication methods
credential_refs	SHOULD	References to credentials or attestations
status	SHOULD	Operational status such as active, suspended, deprecated
freshness	SHOULD	Timestamp, sequence number, or version marker
ttl	MAY	Suggested registration lifetime in seconds
constraints	MAY	Rate limits, regions, data handling limits, cost hints
examples	MAY	Example tasks or invocation patterns

Table 3

Capability metadata SHOULD be signed by the publisher or by a Registry Service Node. Consumers MUST validate signatures and freshness before relying on the metadata for security-sensitive decisions.

Implementations MUST ignore unrecognized metadata fields unless a local policy requires strict validation. This enables extension fields, protocol-specific bindings, and future capability descriptors to coexist with the base architecture.

9. Operational Workflows

9.1. Agent Registration

Agent registration brings a new agent or gateway into the overlay.

1. The agent operator submits capability metadata to a Registry Service Node.

2. The Registry Service Node validates mandatory fields and endpoint reachability.

3. The Registry Service Node verifies or assigns the agent identifier.

4. The Registry Service Node binds the identifier to capability metadata using a credential, signed record, or equivalent mechanism.

5. The Registry Service Node stores the local catalog entry.

6. The Registry Service Node synchronizes a catalog summary or signed record to authorized discovery infrastructure.

Registrations SHOULD support incremental updates. A metadata update MUST carry a freshness indicator that allows receivers to reject stale records.

An explicit deregistration request SHOULD be supported. Deregistration MUST be authenticated with the same or stronger proof of control as registration. After deregistration, Registry Service Nodes SHOULD either remove the record from discovery or return it only as an inactive historical record according to local retention policy.

9.2. Taxonomy Synchronization

Capability labels are useful only if different parties interpret them in compatible ways. Management Root Nodes or authorized taxonomy services SHOULD publish versioned semantic taxonomies.

Taxonomy updates SHOULD include:

* a taxonomy identifier;
* a version number or timestamp;
* label identifiers;
* parent-child or graph relationships;
* deprecation markers;
* mappings to related labels where available; and
* a signature from the taxonomy authority.

Registry Service Nodes SHOULD validate submitted capability tags against a supported taxonomy version. Discovery Service Nodes SHOULD expose which taxonomy versions were used when producing results.

9.3. Agent Discovery

Agent discovery returns candidate agents for a requested intent or constraint set.

1. The requester sends a discovery request to a Discovery Service Node.
2. The Discovery Service Node normalizes the request against supported taxonomies and policies.
3. The Discovery Service Node searches local or synchronized catalogs.
4. The Discovery Service Node ranks candidates using implementation-specific relevance and policy signals.
5. The Discovery Service Node returns candidate metadata, credential references, endpoint references, and matching evidence.
6. The requester validates the returned metadata before selecting a target.

Discovery results MUST NOT be treated as proof that an agent is safe, authorized, or suitable for a particular regulated use.

9.4. Mutual Verification Runtime Interaction

After selecting a candidate, the invoking party establishes trust directly with the target endpoint, which can be an agent or gateway.

1. The invoking party resolves or retrieves the target’s credential material.

2. Both parties validate each other’s credentials, signatures, revocation status, and policy constraints.

3. The invoking party establishes a secure channel using the selected protocol.

4. The target enforces local authorization policy.

5. The parties exchange runtime requests and responses directly.

6. Each party records local audit or metering evidence according to policy.

Discovery Service Nodes SHOULD NOT be in this execution path unless the deployment explicitly uses them as application gateways. If a Discovery Service Node also acts as a gateway, the two roles MUST be distinguishable in metadata and policy.

9.5. Trusted Metering

Commercial deployments may require metering for billing, audit, reconciliation, or contribution tracing. The architecture supports metering without requiring every interaction payload to be globally visible.

A metering record SHOULD include:

* a record identifier;

* identities of the metering parties or pseudonymous references;

* timestamp or time window;

* invocation or discovery event type;

* service or capability identifier;

* usage counters or metering-relevant units;

* policy or contract reference;

* privacy-preserving digest of evidence where applicable; and

* signature by the recording party.

High-frequency private interaction data SHOULD remain off-chain or local. Deployments MAY publish compact digests, Merkle roots, revocation events, or metering commitments to a distributed ledger when tamper-evidence or cross-operator reconciliation is required.

10. Interoperability Considerations

The architecture is protocol-neutral. It can coexist with agent cards, well-known metadata files, DNS-based discovery hints, MCP-style tool interfaces, A2A-style messaging, HTTP APIs, gRPC services, local process bindings, and future agent protocols.

Interoperability profiles SHOULD define:

* how an agent identifier is represented;
* how capability metadata is serialized;
* how signatures and credential references are encoded;
* how endpoints and protocols are selected;
* how taxonomy labels are named and versioned;
* how discovery queries and results are encoded; and
* how status, revocation, and freshness are checked.

Profiles SHOULD avoid requiring one specific machine-learning model or ranking algorithm for discovery. Discovery algorithms affect quality and performance, but the interoperable surface is the query, metadata, result, and validation contract.

10.1. Relationship to Existing Mechanisms

DNS and well-known URIs [RFC8615] can provide bootstrap hints, domain control signals, or descriptor locations. They SHOULD NOT be overloaded with high-frequency, large, or privacy-sensitive agent metadata.

OAuth 2.0, mutual TLS, signed application messages, public-key credentials, or other credential systems can be used as authentication and authorization substrates. This architecture does not require one credential technology, but deployments MUST make credential semantics and revocation behavior explicit.

Deployments that use TLS SHOULD follow the security properties of the TLS version they negotiate; TLS 1.3 is specified in [RFC8446]. Deployments that use OAuth-style bearer authorization SHOULD account for token replay and audience restriction, including the considerations in OAuth 2.0 [RFC6749] and OAuth 2.0 Bearer Token Usage [RFC6750].

Agent interaction protocols such as tool invocation protocols, message-based agent protocols, or local process protocols remain above or beside this architecture. The overlay discovers and verifies candidates; the selected interaction protocol carries the task.

10.2. Versioning and Extension

Deployment profiles, metadata objects, taxonomy versions, and credential profiles SHOULD carry explicit version identifiers. Implementations MUST NOT assume that all nodes in an overlay support the same profile version.

Unknown metadata fields MUST be ignored unless strict validation is required by local policy. Unknown profile identifiers, taxonomy versions, or credential profiles SHOULD cause graceful degradation rather than silent acceptance.

Private extensions SHOULD use collision-resistant names, such as URIs or reverse-DNS names. Extensions that affect interoperability across independent implementations SHOULD be documented in a stable specification.

10.3. Failure Handling

Implementations SHOULD distinguish at least the following failure conditions:

Condition	Typical Handling
stale_metadata	Reject for security-sensitive use or request refresh
revoked_credential	Do not invoke; report revocation to the user or policy engine
taxonomy_mismatch	Fall back to text search or request a supported taxonomy version
discovery_unavailable	Use cached signed metadata if local policy permits
endpoint_unreachable	Try alternate binding or mark candidate degraded
policy_denied	Do not invoke even if semantic match is high

Table 4

Discovery clients SHOULD surface these conditions separately. Collapsing all failures into "not found" can hide security problems and make operational debugging difficult.

11. Security Considerations

Agentic overlay networks introduce risks beyond ordinary service discovery. Agents can initiate actions, combine tools, and operate across administrative domains. Implementations MUST treat discovery data as untrusted input.

Identity spoofing: Attackers may register names or descriptions similar to trusted agents. Registries SHOULD require proof of control over identifiers and SHOULD distinguish unverified claims from verified credentials.

Capability exaggeration: Agents may claim capabilities they do not possess. Registries SHOULD mark whether capabilities are self-asserted, registry-validated, or attested by a third party.

Stale metadata: Discovery results may include outdated endpoints, revoked credentials, or deprecated capabilities. Metadata MUST include freshness information. Clients MUST check revocation and expiration before invocation.

Discovery poisoning: Malicious operators may attempt to bias ranking or inject misleading metadata. Discovery Service Nodes SHOULD apply abuse detection, provenance checks, and rate limits.

Centralization risk: Management Root Nodes could become governance bottlenecks. Deployments SHOULD scope their authority narrowly and publish auditable policy artifacts.

Payload exposure: Registries and discovery services SHOULD NOT require private invocation payloads. Enterprise Service Hubs SHOULD redact sensitive metadata and keep raw data local unless policy permits disclosure.

Confused deputy attacks: An agent may be tricked into invoking another agent using credentials or authority outside the user's intent. Runtime agents SHOULD use scoped, task-specific credentials and SHOULD bind authorization to the requested action.

Downgrade attacks: An attacker may try to force a client from a stronger deployment profile to a weaker one, for example by suppressing credential references or taxonomy versions. Clients SHOULD enforce minimum acceptable profile and credential requirements for sensitive tasks.

Registry compromise: A compromised Registry Service Node can publish false metadata for many agents. Management Root Nodes or governance services SHOULD support rapid revocation of registry credentials and distribution of registry compromise indicators.

Metering privacy: Metering records can reveal business relationships or user behavior. Deployments SHOULD minimize metering data, aggregate where possible, and use privacy-preserving digests when detailed evidence is not required.

12. Privacy Considerations

Capability metadata can reveal sensitive information about enterprise systems, available tools, regions, customers, or business processes. Registrations SHOULD disclose only what is necessary for discovery and safe invocation.

Deployments SHOULD evaluate privacy risks using the general guidance for Internet protocols in [RFC6973].

Discovery requests can reveal user intent. Discovery Service Nodes SHOULD provide transport confidentiality, access control, retention limits, and privacy-preserving logging options.

Enterprise Service Hubs SHOULD keep private data, internal schemas, and credentials within the local domain unless a specific policy authorizes external disclosure.

13. IANA Considerations

This document does not create new registries, media types, URI schemes, HTTP fields, or protocol parameter values. It makes no IANA requests.

14. Implementation Status

This section is to be removed before publication as an RFC.

Prototype deployments are expected to use existing web transports, signed JSON metadata, public-key credentials or equivalent signed credentials, local vector or keyword search in Discovery Service Nodes, and enterprise gateways for protocol adaptation.

15. Implementation Checklist

An implementation of this architecture can be reviewed against the following checklist:

* which deployment profile is supported;
* how capability metadata is signed and refreshed;
* how unknown metadata fields are handled;
* how taxonomy versions are negotiated or rejected;
* how registry compromise and credential revocation are distributed;
* how discovery results are separated from authorization decisions;
* how endpoint reachability and stale metadata failures are reported; and
* what information is retained in metering or audit logs.

16. References

16.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

16.2. Informative References

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/rfc/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/rfc/rfc6750>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/rfc/rfc6973>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

Appendix A. Acknowledgements

The authors thank participants in discussions on agent networking, agent discovery, and cross-domain agent interoperability.

Authors' Addresses

Jinliang Xu
CAICT
No. 52, Huayuan North Road
Beijing
Haidian District, 100191
China
Email: xujinliang@caict.ac.cn

Bingqi Li
CAICT
No. 52, Huayuan North Road
Beijing
Haidian District, 100191
China
Email: libingqi@caict.ac.cn

Runkai Zhu
CAICT
No. 52, Huayuan North Road
Beijing
Haidian District, 100191
China
Email: zhurunkai@caict.ac.cn