Scaling Trust Beyond DNS: How NANDA's Minimal Registry and Verified Agent Facts Unlock the Internet of AI Agents

Ramesh Raskar, Pradyumna Chari, Jared Grogan, Rajesh Ranjan, Shailja Gupta, Robert Lincourt, John Zinky, Rekha Singhal, Surya Narayan Singh, Mahesh Lambe, Raghu Bala, Abhishek Singh, Dimitris Stripelis, Bhuvan B, Sumit Kumar.

Project NANDA

Abstract: We propose a decentralized infrastructure to support billions of autonomous AI agents operating across platforms, devices, and networks. This paper extends the foundational NANDA Protocol with a lean registry design that separates static identifiers from dynamic routing and metadata. We present an architecture where a minimal registry resolves to dynamic, cryptographically verifiable agent facts that support **multi-endpoint routing, load balancing, privacy-preserving access, and credentialed capability assertions**. We introduce a formal agent facts schema and resolution mechanism, along with caching strategies and adaptive routers that enable secure, scalable, and trust-aware agent discovery across organizational boundaries. This architecture offers a lightweight, interoperable foundation for building secure, scalable systems of autonomous agents at large scale.

I. INTRODUCTION

Autonomous agents are emerging as a critical abstraction layer in modern distributed systems, responsible for executing tasks, communicating, and making decisions on behalf of human and machine stakeholders. From supply chain management and digital finance to healthcare orchestration and scientific collaboration, AI agents are expected to scale into trillions of active participants in a globally distributed, interconnected environment.

This evolving reality demands foundational infrastructure that supports the discovery, resolution, and trust evaluation of agents in a decentralized and dynamic ecosystem. Traditional internet infrastructure—designed for static, human-facing resources—is ill-suited for these demands. Basic name resolution and secure communication are made possible by protocols like the Domain Name System (DNS) and HTTPS certificates, but they are unable to meet the autonomy, dynamism, and verifiability needed in agent-centric systems.

To generate cryptographically verifiable representations of agent identity and capabilities, previous work on the NANDA Protocol introduced a decentralized registry and identity framework using Verifiable Credentials (VCs) and Decentralized Identifiers (DIDs). Building upon this framework, the current work tackles a crucial issue in the deployment of this infrastructure at the scale of the internet: creating a registry system that is lean, dynamic, and privacy-conscious without compromising extensibility, interoperability, or trust.

The primary contributions of this paper are as follows:

1) The architecture defines a **minimal core registry** that holds only essential static metadata (agent IDs, credential pointers, and dynamic agent facts URLs) to reduce update needs. It introduces a **self-describing and verifiable AgentFacts schema** that allows agents to dynamically update capabilities, endpoints, and authentication info without modifying the registry. **TTL-based endpoint resolution** supports decoupling, enabling caching, DDOS protection, and flexible deployment.
2) A **privacy-preserving dual-resolution mechanism** enables metadata lookups without revealing the requester's identity, aligning with zero-trust principles. Finally, **agent facts updates require cryptographic signatures** from trusted issuers to ensure authenticity and prevent tampering.

II. DESIGN GOALS

The design of a decentralized AI agent registry infrastructure must account for a wide spectrum of operational and architectural requirements. These range from performance and scalability to security, privacy, and resilience. In this section, we outline the key goals that informed the proposed registry model.

A. Minimal Core Registry

The registry must be lean and stable. Instead of acting as a continuously updated database of agent state, it should serve as a static reference layer that maps each agent's decentralized identifier (DID) to one or more URLs hosting the agent's metadata and endpoint information. This reduces the frequency of registry writes and lowers infrastructure complexity while improving fault tolerance. Current DNS registries receive about 10B hits per day although there are only 300M DNS addresses which is unsustainable when there are billions of agents.

B. Dynamic Endpoint Resolution

Agent systems operate in variable deployment environments that may require frequent changes in network endpoints. Static endpoint binding in the registry would lead to scalability bottlenecks and security vulnerabilities (e.g., susceptibility to DDoS attacks). Therefore, endpoint resolution is decoupled and delegated to the agent facts, which can include TTL (Time-To-Live) parameters for adaptive caching and short-lived resolution windows. This enables dynamic rotation of endpoints without direct registry intervention.

C. Decentralized Metadata Updates

Agents or their authors must be able to publish or revise metadata asynchronously from the registry. By allowing independently hosted agent facts (signed and credentialed), we enable decentralized updates without registry bottlenecks. The use of verifiable credentials ensures that updates can be authenticated and verified without reliance on a central party.

D. Privacy Preservation

To align with privacy best practices and data minimization principles, agent resolution should not expose the identity of the requester. We introduce dual-path resolution via URL1 and URL2, where URL2 can be hosted by third parties or decentralized storage systems to shield access patterns. This division protects sensitive access behavior by enabling metadata retrieval without sending requests to the agent's real domain.

E. Multiple Routing Paths

To support diverse application architectures and resilience strategies, the resolution system must allow:

- Static endpoint discovery via URL1.
- Privacy-preserving resolution via URL2.
- Indirection to adaptive routing services that dynamically determine the best endpoint.

These multiple routing paths provide flexibility for varying operational goals such as geo-distributed load balancing, failover routing, and capability-specific dispatching.

F. Credentialed Agent Metadata

Agent facts must be treated as verifiable claims that require cryptographic attestation. This ensures agents cannot spoof capabilities, impersonate reputable actors, or divert traffic illegitimately. Issuers may include enterprises, consortiums, or federated certification authorities. Each claim or update to the agent facts must be linked to a credentialing path anchored in the issuer's DID.
Together, these goals serve as the foundation for a decentralized, extensible, and trustworthy discovery infrastructure tailored to the requirements of Internet-scale agent ecosystems.

III. SYSTEM ARCHITECTURE OVERVIEW

The architecture of the proposed registry system is organized into a modular, layered stack designed to support large-scale, autonomous agent ecosystems. At its core, the system separates static identity resolution from dynamic endpoint management and verifiable metadata distribution. This separation enables both performance optimization and secure, decentralized operation across heterogeneous networks.

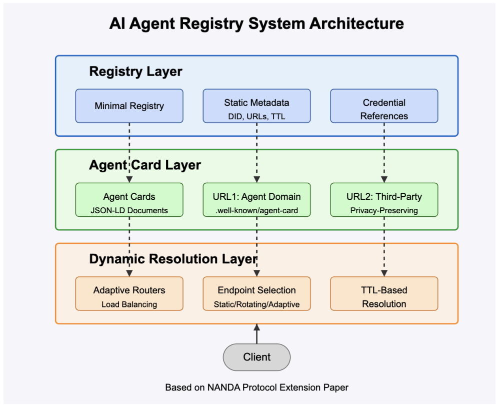


Fig 1: AI Agent registry system architecture

A. Layered System Model

To accommodate the diverse demands of agent discovery, trust evaluation, and governance, the system is organized into three conceptual layers:

1. **Registry Layer (Core Focus of This Paper):**
    - Provides decentralized mapping from agent identifiers (DIDs) to metadata URLs.
    - Maintains only essential, static metadata such as Agent ID, credential hashes, TTL, and URLs for dynamic agent facts.
    - Supports resolution via short-lived, cacheable records that are resistant to tampering and unauthorized updates.
2. **Agent Facts Layer (Metadata Distribution):**
    - Hosts self-describing JSON-LD documents (AgentFacts) that contain endpoint lists, capability descriptors, telemetry configurations, authentication protocols, and credentialed evaluations.
    - Enables frequent, independent updates without requiring registry-level intervention.
    - Supports hosting at either agent-owned domains (via .well-known) or neutral third parties to support privacy-preserving resolution.
3. **Dynamic Resolution Layer:**
    - Dynamically interprets AgentFacts metadata to resolve live endpoints,

apply adaptive routing policies, and maintain privacy constraints.
    - Includes support for decentralized adaptive resolvers that can load balance, geolocate, or behavior-route traffic as needed.
    - TTL-based resolution ensures endpoint freshness while minimizing re-resolution overhead.

This layered model allows the core registry to remain minimal and stable, while richer metadata and operational agility are pushed to the agent facts and resolution services.

B. Registry Schema
The registry resolver does not return raw metadata or service endpoints. Instead, it returns a **cryptographically signed, cacheable object** called **AgentAddr**, which serves as a lightweight address record for the agent. This object encapsulates the agent's identifier, TTL (time-to-live), and pointers to metadata and optional routing infrastructure.

- **agent_id** *(DID)*: A globally unique decentralized identifier compliant with [W3C DID Core] specifications.
- **agent_name** *(URN)*: A human-readable alias encoded as a URN (e.g., urn:agent:salesforce:starbucks).
- **facts_url** *(FactsURL)*: A reference to the AgentFacts hosted at the agent's domain (e.g., https://salesforce.com/starbucks/.agent-facts).
- **private_facts_url** *(PrivateFactsURL)*: An optional, privacy-enhanced reference to the AgentFacts hosted on a third-party or decentralized service (e.g., https://agentfacts.nanda.ai/...).
- **adaptive_router_url** *(optional)*: An optional endpoint for dynamic routing services (e.g., https://router.example.com/dispatch) that handle load balancing, failover, or geo-aware dispatching.
- **ttl** *(Time-To-Live)*: The maximum cache duration before the client must re-resolve the record.
- **signature**: A cryptographic signature from the registry resolver that binds the contents of the AgentAddr.

The AgentAddr object is **signed and cacheable**, allowing clients to redistribute it and perform verification without repeated lookups. Its role is similar to a DNS record, but extended to include verifiable metadata pointers and flexible routing instructions.

C. Resolution Paths
The registry supports multiple agent resolution workflows depending on context, privacy requirements, and routing logic. These workflows begin with resolution of the AgentName into an AgentAddr, which guides the next step.

1. **Direct Communication (Endpoint Access):** This path is used when the client intends to connect to an agent's endpoint directly and minimal metadata is required. This is used when privacy is not a concern and agent-hosted metadata is available.

   AgentName → Registry → AgentAddr → Endpoint

   Metadata Resolution (for trust or discovery) can be used when the client wants to discover the agent's declared capabilities, audit results, or telemetry configuration via its signed AgentFacts. Metadata may be used for interest validation, trust scoring, or reputation checks. Endpoint access is optional and performed only after validation.

   AgentName → Registry → AgentAddr → FactsURL or PrivateFactsURL → AgentFacts

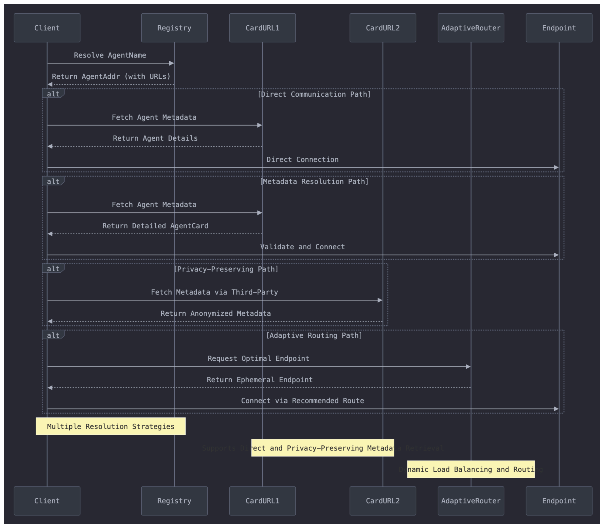
2. **Privacy-Preserving Resolution via URL2:** When the client wishes to preserve its identity or avoid contacting the agent's infrastructure, it uses the PrivateFactsURL. This path does not resolve to an endpoint, and endpoint engagement is a separate decision, possibly through proxy or post-authentication.

   AgentName → Registry → AgentAddr → PrivateFactsURL → AgentFacts (metadata only)

3. **Adaptive Routing Resolution:** When the agent supports dynamic, context-aware routing (e.g., for load balancing, geo-awareness, or DDoS protection), the registry returns the router URL directly in the AgentAddr. The AdaptiveRouter may return temporary signed endpoints or session tokens. The AgentFacts is not required unless client policies demand additional validation.

   AgentName → Registry → AgentAddr → AdaptiveRouter → (Ephemeral) Endpoint

These resolution paths are unified by the use of cryptographically signed AgentFacts and credential validation logic that ensures agents cannot falsify capabilities or impersonate others.


Client
Registry
CardURL1
CardURL2
AdaptiveRouter
Endpoint
Resolve AgentName
Return AgentAddr (with URLs)
alt
[Direct Communication Path]
Fetch Agent Metadata
Return Agent Details
Direct Connection
alt
[Metadata Resolution Path]
Fetch Agent Metadata
Return Detailed AgentCard
Validate and Connect
alt
[Privacy-Preserving Path]
Fetch Metadata via Third-Party
Return Anonymized Metadata
alt
[Adaptive Routing Path]
Request Optimal Endpoint
Return Ephemeral Endpoint
Connect via Recommended Route
Multiple Resolution Strategies
Supports Direct and Privacy-Preserving Metadata Retrieval
Dynamic Load Balancing and Routing
Client
Registry
CardURL1
CardURL2
AdaptiveRouter
Endpoint

D. AgentFacts as a Decoupled Metadata Layer

AgentFacts serve as the functional equivalent of a decentralized service manifest. Each facts includes:

- Versioned metadata and labels (e.g., "Translation Assistant v1.2.1").
- Lists of capabilities (e.g., "streaming", "authentication").
- Skills and modalities (e.g., "text → speech" translation).
- Telemetry endpoints for observability (e.g., via OpenTelemetry).
- Signed evaluations and certifications.
- Endpoint URIs and resolution TTLs.

All AgentFacts must be signed by an issuer authorized under the NANDA trust framework or domain-specific federated authority.

E. Sharded, Federated Deployment Model

The registry layer can be deployed as a quilt of federated registries to support scalability across sectors and jurisdictions. Each registry instance may:

- Be operated by an enterprise, cloud provider, or neutral public body.
- Maintain authority over a subset of agent namespaces (e.g., agent@salesforce:*).
- Federate with other registries using verifiable links, credential exchanges, or trust frameworks.

This federated model ensures both organizational autonomy and interoperability, crucial for a pluralistic global agent network.

IV. AGENT FACTS SCHEMA AND RESOLUTION MECHANISM

The AgentFacts is a structured, cryptographically signed metadata document that encapsulates an AI agent's dynamic state, declared capabilities, endpoints, and trust-related credentials. It plays a central role in decoupling the lean registry from volatile or sensitive metadata, enabling agile agent discovery and interaction at scale. This section formalizes the AgentFacts schema and describes its resolution pathways.

A. Role and Benefits of the AgentFacts

The AgentFacts functions as an intermediary data structure between a stable registry record and an agent's operational configuration. It offers the following advantages:

- **Dynamic Updatability:** AgentFacts can be updated independently of the registry, supporting frequent changes to capabilities, telemetry, or endpoints.
- **Credentialed Assertions:** All claims (e.g., skills, certifications) are signed by issuers, ensuring tamper-resistance and reputation integrity.
- **Deployment Flexibility:** Facts can be hosted at the agent's domain (URL1) or on neutral infrastructure (URL2) to meet privacy or reliability requirements.
- **Multi-Endpoint Support:** Enables load balancing, routing to specialized instances, and failover mechanisms.

B. Formal Schema Specification

AgentFacts are represented in JSON-LD and conform to a versioned schema. A minimal example is shown below:

```
{
 "id": "did:nanda:550e8400-e29b-41d4-a716-4466554400",
 "agent_name": "urn:agent:nanda.mit.edu:lab1:robot42",
 "label": "TranslationAssistant",
 "description": "AI agent for multi-language translation
services",
 "version": "1.2.1",
 "provider": {
   "name": "Example Corporation",
   "url": "https://example.com",
   "did": "did:web:example.com"
 },
 "capabilities": {
   "streaming": true,
   "push": true,
   "authentication": {
     "methods": ["oauth2", "jwt"],
     "jwks_url":
"https://agent.example.com/.well-known/jwks.json"
   }
 },
 "skills": [
   {
     "id": "translation",
     "description": "Translates text between languages",
     "inputModes": ["text"],
     "outputModes": ["text"],
     "supportedLanguages": ["en", "es", "fr", "de", "ja", "zh"]
   }
 ],
 "telemetry": {
   "endpoint": "otlp://metrics.nanda.ai",
   "format": "opentelemetry"
 },
 "evaluations": {
   "performanceScore": 4.8,
  "securityAudit":
"https://audit.nanda.ai/reports/550e8400",
  "lastUpdated": "2025-04-01T12:00:00Z"
 },
 "certification": {
  "level": "verified",
  "issuer": "NANDA",
  "issuanceDate": "2025-03-15T09:30:00Z",
  "expirationDate": "2026-03-15T09:30:00Z",
  "signature": "base64signature..."
 }
}
```

This schema includes both static metadata (e.g., identifiers, descriptions) and dynamic fields (e.g., performance metrics, endpoint lists), all bound by verifiable cryptographic signatures.

C. AgentFacts Hosting Patterns

AgentFacts can be resolved through two primary URL schemes:

- **URL1: Primary Facts Hosting (FactsURL)** Hosted under a standardized path on the agent's domain (e.g., https://example.com/.well-known/agent-facts). It is specified as facts_url in the AgentAddr object, recommended for agents operating public-facing services or enterprise-controlled deployments, and enables clients to retrieve metadata directly from the source domain. "facts_url": "https://salesforce.com/starbucks/.agent-facts"

- **URL2: Privacy-Preserving Hosting (PrivateFactsURL) (Third-Party Hosted Facts):**
  It is hosted via a neutral or decentralized provider (e.g., IPFS, third-party metadata gateway), specified as private_facts_url in the AgentAddr, enables metadata retrieval without contacting the agent's infrastructure, preserving requester privacy and supporting zero-trust principles and is recommended for agents operating in sensitive or regulated domains.

This dual-hosting model allows agent publishers to choose based on trust requirements, latency preferences, and organizational control.

D. Resolution Workflow (End-to-End Resolution Workflow with AgentAddr**)**

The AgentFacts resolution follows a structured, cache-aware workflow:

1. **Registry Lookup**
   - **Input:** AgentName (e.g., urn:agent:salesforce:starbucks)
   - **Output:** A signed AgentAddr containing: {agent_id (DID), facts_url, private_facts_url , adaptive_router_url (optional), ttl, signature}
2. **Metadata Resolution (Optional)**
   - If trust evaluation or capability validation is needed, the client fetches the AgentFacts from either: facts_url (direct access) or private_facts_url (privacy-preserving access). The AgentFacts is verified using digital signatures and credential chains.
3. **Endpoint Discovery**
   - Clients inspect AgentFacts metadata (if fetched) or AgentAddr (if endpoint is included directly) for:
     - **Static Endpoints**: Stable communication URIs
     - **Rotating Endpoints**: Short-lived, dynamic URIs
     - **AdaptiveRouter URL**: A programmable routing microservice
4. **Endpoint Resolution & Connection**
   - If adaptive_router_url is used, the client sends a request and receives either: A redirect to the optimal endpoint or A signed ephemeral token. Connection to the endpoint is authenticated (e.g., via OAuth2 or JWT) and established.

Clients may choose the resolution strategy based on Latency and load-balancing needs, privacy preferences, credential strength and trust policies, and adaptive behavior triggers. This layered resolution flow allows the system to separate stable identity resolution from dynamic endpoint management, supporting security, agility, and privacy at global scale.

E. TTL and Caching Strategies

The system uses **time-to-live (TTL)** values to manage caching and resolution frequency across multiple layers: the registry (AgentAddr), the agent metadata (AgentFacts), and the dynamic routing layer (e.g., AdaptiveRouter). These TTLs are **signed**, **layer-specific**, and **enforce trust-aware resolution intervals**.

1. **TTL in AgentAddr (Registry Layer):** The AgentAddr object returned by the registry includes a ttl field. This defines how long the record may be cached by clients, gateways, or edge resolvers before requiring re-validation. It applies to the entire AgentAddr object, including: Metadata pointers (facts_url, private_facts_url) and optional adaptive_router_url. This ensures that the client doesn't frequently burden the registry for stable identifiers. TTL values are typically tuned based on the volatility of the agent's state, usage frequency, and trust domain policies.

2. **TTL in AgentFacts (Metadata Layer):** Each AgentFacts includes its own ttl value, often embedded in its cryptographically signed metadata section. This TTL governs: How long the **capabilities**, **telemetry endpoints**, or

evaluations may be considered fresh and When clients should re-fetch the facts for updated data. This can be used primarily when metadata is used for **trust scoring**, **capability filtering**, or **credential validation**.

3. **TTL for Routing Metadata (Resolution Layer):** Endpoint lists (static or rotating) and adaptive_router_url routing logic may have shorter TTLs than AgentFacts metadata. Common TTLs in this layer could be - Static endpoints: 1–6 hours, Rotating endpoints: 5–15 minutes, and Adaptive routing tokens: 30–60 seconds.

TTLs can be adjusted based on agent criticality (e.g., healthcare vs. demo agents), deployment volatility (static hosting vs. serverless endpoints), and trust zone policies.

V. ADAPTIVE ROUTING AND MULTI-ENDPOINT MANAGEMENT

To meet the demands of globally distributed, privacy-sensitive, and dynamically deployed AI agents, the registry architecture supports **adaptive routing** via programmable, policy-driven components embedded in the agent resolution process. These components decouple **endpoint selection** from static metadata and enable scalable, resilient operation in diverse environments.

A. Motivation for Adaptive Routing

Traditional agent endpoint resolution—where the agent is tied to a single, static endpoint—is inflexible and fragile. AI agents today must serve users from multiple regions with low latency, load balance across backend clusters, obfuscate infrastructure to prevent DDoS attacks and update routing logic in real time without registry writes. Adaptive routing solves these challenges by introducing **runtime resolvers**—referred to as **AdaptiveRouters**—that inspect request context and forward traffic to the optimal instance.

B. Multi-Endpoint Resolution Models

Each AgentFacts may contain multiple endpoint references, categorized by their resolution type:

1. **Static Endpoints:** Explicit URIs that represent stable, always-on service interfaces. Ideal for low-frequency, high-trust agents where endpoint volatility is low.

2. **Rotating Endpoints:** A set of URLs with short TTLs, designed for infrastructure that undergoes frequent restarts, redeployments, or geographical rebalancing.

3. **Adaptive Router URI:** Points to a microservice or gateway that dynamically routes traffic to optimal downstream instances. Functions as a programmable interface for applying routing logic based on input features (location, time, capability match).

AgentFacts may define one or more types of endpoints under an endpoints section:

```
"endpoints": {
  "static": ["https://api.example.com/v1"],
  "rotating": ["https://east.example.com",
"https://west.example.com"],
  "adaptive_router":
"https://router.example.com/dispatch"
}
```

C. AdaptiveRouter Component

To avoid requiring AgentFacts resolution for routing in all cases, the registry can embed the adaptive_router_url directly in the **AgentAddr**, enabling immediate routing without further lookups. Example AgentAddr:

```
{
  "agent_id": "did:web:example.com",
  "adaptive_router_url":
"https://router.example.com/dispatch",
  "ttl": 300,
  "signature": "..."
}
```

This enables the resolution flow:

AgentName → Registry → AgentAddr → AdaptiveRouter → Ephemeral Endpoint

The **AdaptiveRouter** is an optional but powerful element in the routing stack. It accepts requests, inspects context (e.g., request headers, origin IP, JWT claims), and forwards the call to the most appropriate agent instance. Features include:

- **Load Balancing:** Routes traffic to the least-loaded endpoint instance in real time.
- **Geo-Aware Dispatching:** Selects endpoints closest to the request origin.
- **Capability-Specific Routing:** Directs requests to endpoint versions supporting requested skills (e.g., “real-time translation” vs. “batch mode”).
- **Threat Mitigation:** Rotates backend targets or rate-limits sensitive interfaces based on anomaly detection.

The AdaptiveRouter is referenced via the AgentFacts and may itself issue temporary credentials or endpoint tokens to validate downstream access.

D. Security and TTL Constraints

All routing logic exposed via the adaptive_router_url must **declare TTLs** to define how long routing metadata and endpoint assignments may be cached, **be signed** to prevent manipulation or impersonation and **specify fallback rules**, such as: use static endpoint if router is unreachable, retry after a delay with alternate region or elevate to credential-based access.

E. Implementation Considerations

For real-world deployments, AdaptiveRouter implementations may take the form:

- **Serverless Functions:** Lightweight routing logic deployed on cloud edge networks.
- **Reverse Proxies (e.g., Envoy, NGINX):** Using programmable rulesets for routing and policy enforcement.
- **Federated Mesh Gateways:** In multi organizations setting, shared routers may enforce domain-specific policies.

These implementations must support cryptographic verification, auditability, and availability SLAs to function reliably within the decentralized agent fabric.

VI. CREDENTIALED TRUST AND PRIVACY MECHANISMS

The architecture emphasizes **verifiability, privacy, and modular trust** across the entire resolution process. With agents operating in sensitive, autonomous, and federated contexts, all metadata—particularly dynamic endpoint and capability claims—must be cryptographically bound and privacy-aware. To support this, the system introduces the separation of:

- **Signed AgentAddr** (returned by the registry)
- **Credentialed AgentFacts** (resolved optionally for metadata)
- **Privacy-preserving resolution paths** using third-party infrastructure

A. Verifiable Credentials and Signed Metadata

The integrity of agent metadata and its update mechanism depends on **W3C Verifiable Credentials (VCs)** to assert capabilities, skills, audits, or evaluations and **cryptographic signatures** to bind these claims to the agent identity. This ensures agents cannot spoof skills, impersonate trusted services, or hijack reputational pathways.

Artifact	Purpose	Signed By
AgentAddr	Identity, routing pointers, TTL	Registry resolver
AgentFacts	Capabilities, credentials, endpoints	Credential issuers
Endpoint Tokens	Temporary routing/session bindings	AdaptiveRouter (optional)

B. Credential Governance and Trust Domains

The trustworthiness of an AgentFacts hinges on the validity and recognition of its credential issuers. Our system supports a **federated trust governance model**, where different credential authorities may govern their own "trust zones" (e.g., within an enterprise, vertical, or region). These zones can define credential schemas and issuance policies, establish revocation mechanisms, and interoperate using cross-signing, federation agreements, or registry-to-registry bridges. Clients interacting with agents may configure their own trust preferences - accept only credentials from whitelisted authorities, require threshold verification (e.g., at least two independent audits), and verify issuer reputation using embedded TRS (Trust Reputation Scores).

This modular design allows the ecosystem to evolve without hardcoded trust anchors.

C. Privacy-Preserving Resolution via URL2

To protect the **privacy of the accessor**, the system supports resolving agent metadata via a third-party or decentralized location (e.g., IPFS, secure proxy), specified as private_facts_url in the AgentAddr. This model provides

1) **Requester anonymity**: Agent domain is never contacted
2) **Access decoupling**: Metadata is publicly readable without invoking agent infrastructure
3) **Audit independence**: Metadata remains available even if the agent hosting goes offline

Clients may be configured to **prefer this route** when operating in regulated or high-sensitivity environments.

AgentName → Registry → AgentAddr → PrivateFactsURL → AgentFacts (metadata only)

D. Trust Reputation Scores (TRS)

For automated agent selection, trust signals must be distilled into a computable reputation score. We define a composite score as follows:

$$\mathbf{TRS = w_1 \cdot CVS + w_2 \cdot CapVS + w_3 \cdot BTS + w_4 \cdot CS}$$

Metric	Description
CVS	Certificate Validity Score (expiry, revocation, freshness)
CapVS	Capability Verification Score (audit coverage)
BTS	Behavioral Trust Score (uptime, latency, fault rates)
CS	Compliance Score (regulatory or vertical adherence)

The weights $\mathbf{w_1}$ through **w4** are adjustable per use case and may be informed by client policy or domain-specific defaults. TRS is not stored in the registry or AgentFacts but is dynamically computed based on credential chains, telemetry lookups, and certificate validations.

E. Revocation and Credential Freshness
Credential freshness is essential for safety in real-time environments. To support dynamic trust, the system includes:

- **Credential Expiry Fields:** Embedded in all signed objects.
- **Revocation Lists:** Served by credential issuers and queried by clients during interaction.
- **Hash-Linked Chains:** Credential validity can be chained across issuers (e.g., capability certified by org A, endorsed by org B).

It is advised to recompute TRS on each new session, set cache expiration policies shorter than the weakest credential's TTL, and perform multi-source validation for high-risk agents (e.g., financial or healthcare applications).

F. Resolution Layer Enforcement

The architecture enforces that AgentAddr records can only be resolved and cached within their TTL window, AgentFacts metadata is not assumed to be static; clients must verify signatures and freshness, and privacy-preserving resolution (via PrivateFactsURL) is functionally equivalent to FactsURL, but adds a layer of **access decoupling**

VII. DEPLOYMENT CONSIDERATIONS AND USE CASES

This section outlines practical implementation strategies for deploying the proposed registry architecture and highlights real-world use cases that demonstrate its applicability across domains. The architecture's lean registry, dynamic resolution, and credentialed metadata design make it well-suited to diverse operational environments ranging from tightly governed enterprise systems to open agent marketplaces.

A. Deployment Models
The registry–AgentFacts system can be deployed under multiple architectural models, depending on organizational goals, control boundaries, and compliance requirements.

1. **Enterprise-Controlled Registries**
   - Enterprises deploy and manage internal registries governing their agent namespaces.
   - AgentFacts are hosted within their infrastructure (URL1).
   - Credential authorities reside within the enterprise's compliance and audit teams.

2. **Federated Industry Registries**
   - Vertically integrated industries (e.g., logistics, healthcare) operate consortium registries.
   - AgentFacts reference third-party capabilities and certifications from accredited bodies.
   - Governance rules are encoded in registry policy and enforced via signature validation.

3. **Decentralized Public Registries**
   - Publicly accessible registries allow open registration of agents.
   - Credential validation and revocation are enforced via smart contracts or decentralized protocols.
   - AgentFacts are hosted in IPFS or similar decentralized file systems, emphasizing privacy and censorship resistance.

4. **Hybrid Topologies**
   - Organizations may register with both private and public registries for broader discoverability.
   - Trust rules and resolution pathways allow clients to prioritize registry sources based on use-case-specific logic.

These models enable the system to adapt to both regulated, closed-loop deployments and open, innovation-driven ecosystems.

B. Performance and Scaling Considerations
While the registry is intentionally lean, the supporting infrastructure (AgentFacts hosts, adaptive routers, credential resolvers) must be designed for internet-scale agent traffic. Key strategies that could be helpful:

- **Edge Caching:** TTL-enforced caching of AgentFacts and routing metadata at CDN edges to reduce resolution latency.
- **Sharded Credential Verification:** Parallel trust scoring engines operating at domain-specific points of presence (PoPs).
- **Load-Adaptive Routers:** Stateless microservices that scale horizontally and resolve endpoints dynamically based on request volume and client location.
- **Audit and Monitoring Hooks:** Integration with OpenTelemetry and observability stacks to support SLA monitoring and behavior-based trust score computation.

VIII. CONCLUSION

The rise of autonomous AI agents presents unprecedented demands on digital infrastructure—scalable identity, verifiable trust, dynamic coordination, and privacy-preserving discovery. Traditional internet protocols, built for static, human-operated endpoints, are insufficient for the fluid, high-volume interactions required by agent-scale autonomy. This paper introduced a lean, modular registry architecture as a foundational layer for the decentralized Internet of AI Agents. Grounded in the NANDA Protocol, the design advances the state of the art across multiple dimensions:

- **Minimal Core Registry**: Decouples static identifiers from dynamic metadata, reducing update frequency, exposure, and attack surfaces.
- **Credentialed AgentFacts**: Enable rapid, verifiable updates to agent capabilities, routing logic, and telemetry.
- **Timed Resolution & Adaptive Routing**: Support latency optimization, DDoS mitigation, and geo-aware deployments.
- **Privacy-Preserving Discovery**: Protects accessor identities and aligns with zero-trust security models.
- **Federated Trust & Governance**: Enables scalable oversight across jurisdictions, organizations, and policy domains.

Together, these components establish the groundwork for an open, secure, and extensible agent ecosystem—capable of supporting trillions of agents operating cooperatively across digital and physical environments.

IX. Future Work

Several avenues for future research and development build upon this foundation:

1. **Trust Layer Extensions**
   Future work will explore decentralized trust score computation, integration with behavioral monitoring systems, and real-time reputation feeds.
2. **Semantic and Vector-Based Discovery**
   While this paper focuses on endpoint resolution, future systems may support semantic and embedding-based agent search, enabling discovery based on natural language descriptions, skills similarity, or behavioral history.
3. **Governance Frameworks and Revocation Models**
   We will formalize revocation protocols, dispute resolution mechanisms, and multi-jurisdictional governance workflows to enforce trust at scale. This includes modeling hierarchical and mesh-based governance for industry-specific applications.
4. **Open Standards and Interoperability**
   Standardizing the AgentFacts schema with IETF/W3C bodies and enabling interoperability with DIDComm, Open Agent Frameworks, and existing Web3 identity platforms will ensure ecosystem adoption.
5. **Privacy Enhancements with Zero-Knowledge Proofs (ZKPs)**
   To further privacy guarantees, ZKP-based credential assertions can be introduced, allowing agents to prove capabilities or compliance without revealing full certificate chains.
6. **Cross-Registry Discovery Protocols**
   We aim to build discovery protocols that stitch together independent registries through verifiable, decentralized cross-references, powering a truly federated global agent index.

X. OPEN QUESTIONS AND DISCUSSION

While this paper proposes a comprehensive registry and metadata architecture for AI agents, several key areas remain open for further exploration, community alignment, and empirical validation. These questions span technical design, trust models, privacy trade-offs, economic incentives, and future governance structures. Their resolution will shape the long-term viability and adaptability of decentralized agent infrastructure at a global scale.

A. Hosting and Accessibility of Agent Metadata

A central question concerns the hosting model for AgentFacts. The current design permits multiple options: (i) self-hosting by the agent (e.g., at /.well-known), (ii) hosting through secure third-party services, or (iii) provisioning by a neutral infrastructure provider such as NANDA. Each approach offers distinct trade-offs in privacy, trust, and update latency.

- Self-hosting ensures provenance but reveals access patterns.
- Third-party hosting enhances privacy but introduces a dependency on the host's security posture.
- Public infrastructure democratizes access but may incur governance and scalability concerns.

A flexible model is desirable, but the trade-offs between decentralization, performance, and control remain unresolved.

B. Registry Visibility and Discoverability

Unlike DNS, the agent registry may be intentionally opaque to prevent abuse or sensitive agent enumeration. At the same time, open discovery is crucial for enabling federation, crawling, and reputation-based search.

Open issue: Should the registry support machine-readable scraping or indexing, and if so, under what access controls or throttling policies?

This question has profound implications for building agent reputation graphs, public directories, or sector-specific trust indexes.

C. Terminology Alignment with Prior Art

The use of the term *AgentFacts* may overlap or cause confusion with prior frameworks such as A2A. While our schema significantly extends prior work, we may consider renaming it to *Agent Metadata Layer* or *AgentFacts v2*, and clearly describing its relationship to A2A.

Additionally, terms like URL1 and URL2 may be refined to more functional descriptors such as *Primary Metadata Endpoint* and *Privacy-Preserving Metadata Endpoint*.

Open issue: What terminology best communicates extensibility while acknowledging lineage and compatibility with existing systems?

D. Resolution Flow and Return Semantics

Current resolution logic implies that accessing an endpoint requires first resolving and validating the AgentFacts. However, it is unclear whether the client must always parse the facts or if a signed *AgentAddr* or *AgentLoc* object—containing endpoint and metadata digests—would suffice.

Open issue: Should the resolver return a lightweight signed object, distinct from the full facts, optimized for runtime communication?

This separation could reduce latency and support pluggable authentication protocols at the edge.

E. TTL Strategy and Update Constraints

While the paper introduces TTL-based caching, important design questions remain:

- Who determines TTL values—agents, registries, credential authorities, or external policy engines?
- Should TTLs vary by agent type (e.g., real-time agents vs. scheduled agents)?
- Can TTLs be **ephemeral** or **task-conditioned**, based on state machines or mission duration?

Open issue: How do we strike a balance between agility and stability in TTL assignments, particularly for high-frequency or regulated agents?

F. Credential Governance and Delegation

Another unresolved dimension is the delegation model for updates and credential issuance:

- Can agents update their own metadata autonomously?
- Are delegated authorities (e.g., platform vendors or consortia) responsible for publishing credentials?
- Can agents authorize credentialing-on-the-fly via smart contracts or concierge agents?

Open issue: What are the minimal viable trust constraints to allow automated updates while preventing spoofing, abuse, or trust inflation?

G. Adaptive Routing Reliability

The adaptive routing layer introduces flexibility and performance, but it also poses new failure modes:

- What fallback paths should be employed if the AdaptiveRouter is offline?
- Can routing logic be cached or prevalidated in critical systems?
- How is routing evaluated for fairness and performance across federated domains?

Open issue: What guarantees must the resolution and routing layer provide to support mission-critical or regulated agent deployments?

H. Economic Incentives and Monetization Models

The sustainability of agent registries and metadata services may depend on viable economic incentives. Yet the architecture must avoid rent-seeking behavior or exclusionary gatekeeping.

Open issue: Should AgentFacts hosting or TTL adjustments be linked to micropayments, staking mechanisms, or usage tiers? How can such models preserve openness and composability?

References:

[1] W3C, "Decentralized Identifiers (DIDs) v1.0," Jul. 2022. [Online]. Available: https://www.w3.org/TR/did-core/.

[2] J. Camenisch and A. Lysyanskaya, "A Formal Treatment of Onion Routing," in *Advances in Cryptology – CRYPTO 2005*, Lecture Notes in Computer Science, vol. 3621, Springer, 2005, pp. 169-187.

[3] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," *arXiv:1407.3561*, Jul. 2014. [Online]. Available: https://arxiv.org/abs/1407.3561.

[4] D. Mazières, "The Stellar Consensus Protocol: A Federated Model for Internet-Level Consensus," Stellar Development Foundation, Nov. 2015.

[5] A. Tobin and D. Reed, "The Inevitable Rise of Self-Sovereign Identity," Sovrin Foundation, 2016.

[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications," in *Proc. ACM SIGCOMM*, San Diego, CA, USA, Aug. 2001, pp. 149-160.