Independent Submission
Internet-Draft
Intended status: Experimental
Expires: 17 November 2025

K. Huang
DistributedApps.ai
V.S. Narajala
Amazon Web Services
I. Habler
Intuit
A. Sheriff
Cisco Systems
May 2025

Agent Name Service (ANS): A Universal Directory for Secure AI Agent Discovery and Interoperability
draft-narajala-ans-00

Abstract

The proliferation of AI agents requires robust mechanisms for secure discovery. This document introduces the Agent Name Service (ANS), a novel architecture based on DNS addressing the lack of a public agent discovery framework. ANS provides a protocol-agnostic registry mechanism that leverages Public Key Infrastructure (PKI) certificates for verifiable agent identity and trust. The architecture features several key innovations: a formalized agent registration and renewal mechanism for lifecycle management; DNS-inspired naming conventions with capability-aware resolution; a modular Protocol Adapter Layer supporting diverse communication standards (A2A, MCP, ACP, etc.); and precisely defined algorithms for secure resolution. This specification describes structured communication using JSON Schema and includes a comprehensive threat analysis. The result is a foundational agent directory service protocol addressing the core challenges of secure discovery and interaction in multi-agent systems, paving the way for future interoperable, trustworthy, and scalable agent ecosystems.

Disclaimer

The present document reflects the authors' individual opinions and does not necessarily represent the views of their respective employers.

Status of This Memo

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

This Internet-Draft will expire on 2 November 2025.

Copyright Notice

Table of Contents

1. Introduction

Agent-to-agent communication is expected to become a significant component of internet traffic, driving the need for reliable mechanisms enabling agents to discover, verify, and securely interact with one another. Traditional service discovery, notably the Domain Name System (DNS) [RFC1035], primarily maps human-readable names to network addresses and is insufficient for the dynamic, semantically rich, and security-sensitive environment of agentic AI. Enhancements like DNS-Based Service Discovery (DNS-SD) [RFC6763] offer improvements but still fall short of the necessary agent capability granularity, identity verification, and lifecycle management required by autonomous agents. Furthermore, maintaining a trustworthy registry necessitates robust processes for agent registration and periodic renewal.

Several agent communication protocols are emerging to standardize interactions:

* Agent2Agent (A2A) Protocol: Developed by Google, providing a standardized protocol for inter-agent communication, aiming to bridge different agent frameworks.

* Model Context Protocol (MCP): Focused on simplifying the integration of AI models with external tools and data sources.

* Agent Communication Protocol (ACP): Designed to standardize how agents communicate, enabling automation, collaboration, UI integration, and developer tooling, evolving from initial MCP concepts.

This document outlines the Agent Name Service (ANS), a framework for a protocol-agnostic Agentic AI Registry. ANS complements these emerging protocols by integrating Public Key Infrastructure (PKI) for identity and trust, defining structured communication via JSON Schema, incorporating DNS-like naming for discovery, establishing mechanisms for agent registration and renewal, and providing a formal specification of the protocol to enhance precision and implementability. ANS aims to provide a universal, secure directory service foundation for interoperable agent ecosystems.

1.1. 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. Related Work

Traditional service discovery, such as DNS [RFC1035], provides essential name-to-address resolution but lacks the semantic understanding and security features needed for agentic AI. DNS-SD [RFC6763] adds local service discovery capabilities but doesn’t address verifiable identity or complex agentCapability matching on a global scale.

Research in multi-agent systems (MAS) has explored various agent communication languages (ACLs), such as those defined by the Foundation for Intelligent Physical Agents (FIPA) [FIPA-ACL]. While influential, these often lack standardized, built-in security mechanisms and universally adopted transport protocols suitable for the modern internet.

The emerging protocols represent significant advancements:

* A2A [A2A-Blog] [A2A-Spec] focuses on bridging agent ecosystems.

* MCP [MCP-News] [MCP-Spec] emphasizes dynamic discovery and integration of tools/data for AI models.

* ACP [IBM-ACP-Placeholder] targets broader agent-to-agent communication needs, including delegation and orchestration.

Our work builds upon these efforts not by replacing them, but by providing a complementary, protocol-agnostic infrastructure layer. ANS differentiates itself by integrating PKI-based identity verification [RFC5280] directly into the discovery and lifecycle management process, offering a universal registry mechanism that enhances trust and facilitates secure interaction across different protocol standards via a common discovery plan. Furthermore, the formalized specification of the ANS protocol ensures clarity and ease of implementation.

3. Agent Registry Architecture

The proposed Agent Registry architecture provides a secure, interoperable platform for agent discovery and interaction, supporting multiple communication protocols through a modular design. Key components include:

Requesting Agent:
The entity initiating the agent registration process, which could be an individual, organization, or automated system seeking to register a new agent or update existing agent information in the registry.

Agent Registry:
A potentially distributed database for storing ACEM (Agent Credential and Entitlement Management) and DID (Decentralized Identifier) related information. This registry encompasses agent capabilities, security policies, PKI certificates, protocol-specific metadata (via protocolExtensions), and registration/renewal timestamps, supporting a comprehensive framework for agent identity, authentication, and authorization.

Certificate Authority (CA):
A trusted entity issuing and managing X.509 digital certificates [RFC5280] for agents, forming the root of trust.

Registration Authority (RA):
Verifies agent registration/renewal requests, interacts with the CA to issue certificates based on Certificate Signing Requests (CSRs), manages the agent lifecycle (registration, renewal, revocation), and validates the legal entity of the Requesting Agent. It enforces registry policies.

Protocol Adapter Layer:
Translates between the registry's internal representation and protocol-specific formats (details in Section 5).

Request/Response Schema:
A protocol-agnostic JSON-based schema [RFC8259] for registry interactions (discovery, registration, etc.), incorporating PKI data and allowing protocol-specific extensions (details in Section 3.4).

Agent Name Service (ANS):
Enables agent discovery using human-readable, structured names, coupled with agentCapability-based resolution (details in Section 3.5).

Figure 1 illustrates the core components of the Agent Name Service (ANS) and their interactions.

<preamble>:Illustrates the interaction between Agent, ANS Service, Agent Registry, CA, RA, and Protocol Adapter Layer.

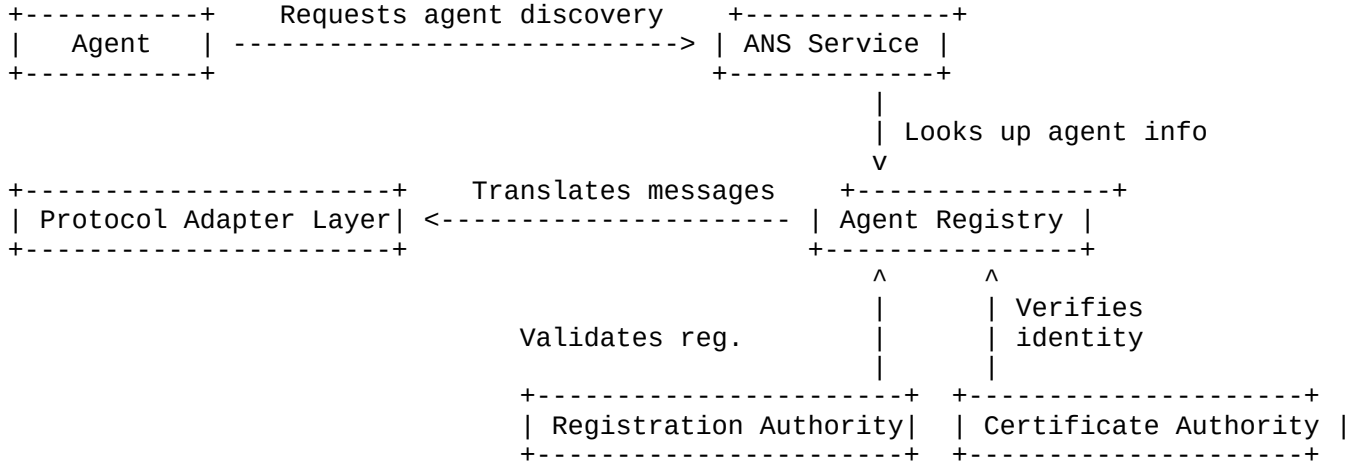

Figure 1: ANS Architecture

The Protocol Adapter Layer translates between the registry's internal representation and protocol-specific formats. For example, consider an agent registering with the MCP. An MCP tool description might be represented as a JSON blob. The agent would need to be registered with ANS first and foremost. Therefore, imagine this tool is associated with the following ANSName: "mcp://sentimentAnalyzer.textAnalysis.ExampleCorp.v1.0".

This would mean the MCP tool is now discoverable via the ANS. The MCP specific extension data itself might look like this:

```
{
  "description": "Analyzes sentiment of text input.",
  "input_schema": {
    "type": "string",
    "description": "Text to analyze."
  },
  "output_schema": {
    "type": "object",
    "properties": {
      "sentiment": {
        "type": "string",
        "enum": ["positive", "negative", "neutral"]
      },
      "score": {
        "type": "number",
        "description": "Sentiment score (-1 to 1)."
      }
    }
  },
  "mcpEndpoint": "https://sentiment.example.com/analyze"
}
```

Figure 2: Example MCP Extension Data

The MCP Adapter within the Protocol Adapter Layer would parse this JSON and map it to the registry's internal columns. This could involve:

* Extracting information implicitly: since the ANSName is "mcp://sentimentAnalyzer.textAnalysis.ExampleCorp.v1.0", this implicitly defines the:

  - Protocol: mcp

  - AgentID: sentimentAnalyzer

  - agentCapability: textAnalysis

- Provider: ExampleCorp

- Version: v1.0

* Storing the description ("Analyzes sentiment of text input.") in a dedicated description field within protocolExtensions.

* Serializing the input_schema and output_schema and storing them in a protocolExtensions column specific to MCP, allowing other MCP-aware agents to understand the tool’s interface.

* The actual MCP endpoint "https://sentiment.example.com/analyze" would be stored within the protocolExtensions, under the key mcpEndpoint.

This normalization process allows the Agent Registry to store and query MCP-specific information in a protocol-agnostic way, while adhering to the ANSName structure for consistent identification and resolution.

3.1. Agent Registration and Renewal

Maintaining registry integrity requires explicit lifecycle management:

Registration:

1. An agent submits a registration request (conforming to the defined JSON schema, see Section 3.4) including metadata, protocol details (within protocolExtensions), and a CSR.

2. The RA validates the agent’s identity and submitted information against registry policies (potentially involving automated checks or human review).

3. The RA requests a certificate from the CA using the validated CSR.

4. The issued certificate and agent information are stored in the Agent Registry.

Renewal:

1. Agents periodically submit renewal requests before their registration or certificate expires.

2. The RA verifies continued compliance with policies.

3. The RA requests a new certificate from the CA.

4. The agent's registration/renewal timestamp and potentially updated certificate are stored in the Registry.

Deregistration/Revocation: Agents can be deregistered, or their certificates revoked (e.g., due to key compromise), removing or flagging their entry in the registry and invalidating their certificate via standard PKI mechanisms (CRL/OCSP [RFC6960]).

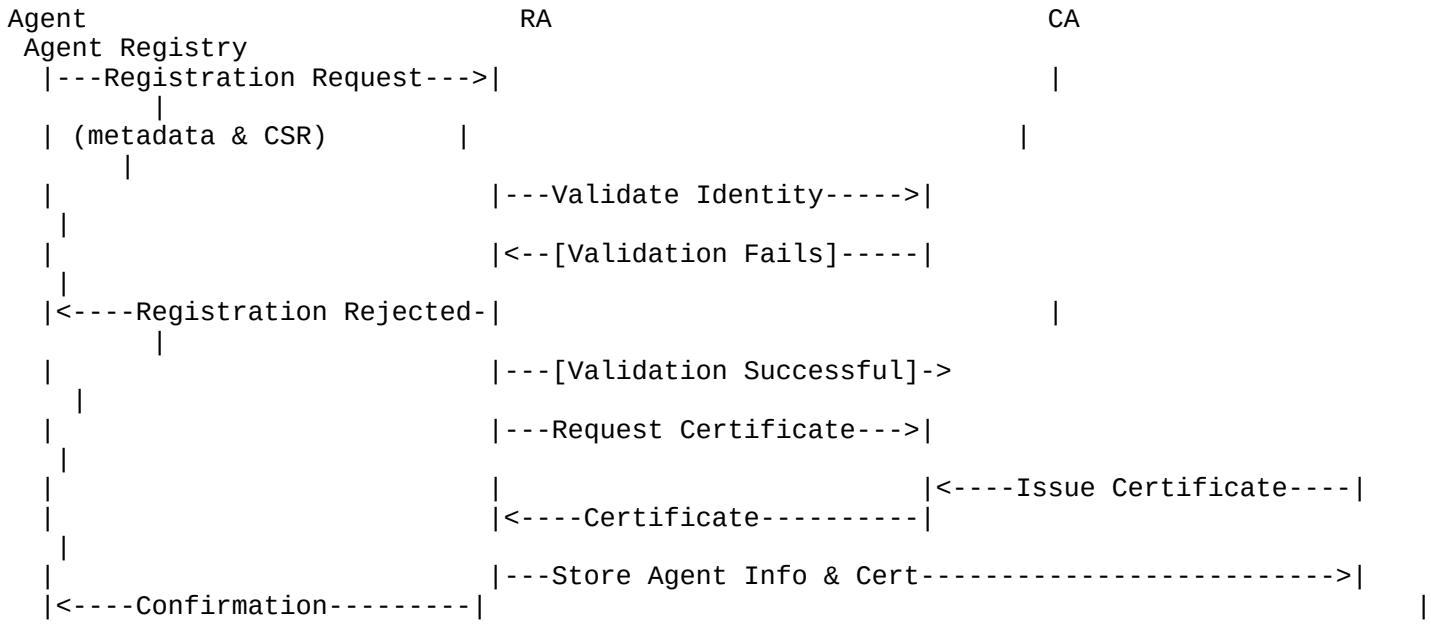

Figure 3: Agent Registration Process

3.2. PKI Integration

Public Key Infrastructure (PKI) [RFC5280] provides the cryptographic foundation for trust within ANS. Each registered agent possesses a unique PKI key pair and a corresponding X.509 digital certificate issued by a trusted CA via the RA.

Identity Verification:
The agent's certificate cryptographically binds its public key to its verified identity (e.g., its ANSName, organizational affiliation). Other agents can verify digital signatures made with the agent's private key by using the public key in the certificate, thus ensuring the authenticity and integrity of communications.

Trust Chain:
Certificates are validated against the trusted CA, establishing a hierarchical chain of trust. Agents can determine the trustworthiness of other agents by tracing their certificates back to a common trusted root CA.

Lifecycle Management:
The validity of an agent’s certificate is tied to its registration and renewal cycle. Revoked certificates are managed using standard mechanisms like CRLs or OCSP [RFC6960], ensuring that compromised or outdated certificates are no longer trusted.

Simplification:
While managing PKI can be complex, the integrated RA/CA interaction within the ANS framework aims to streamline the certificate issuance and renewal processes for agent developers compared to manual PKI management.

3.3. ANS Protocol Notation

We introduce the following notation for defining ANS elements and operations:

1) Top Level Elements:

* Protocol: Communication Protocol

* AgentID: Agent Identifier

* agentCapability: Agent Capability

* Provider: Provider Name

* Version: Version Number

* Extension: Extension Metadata

* Cert: Agent Certificate (X.509)

* Sig: Digital Signature

* ANSName: Agent Name Service Name

* Endpoint: a resolvable endpoint

2) Data Types:

* String: Represents a sequence of characters.

* Integer: Represents an integer number.

* Boolean: Represents a boolean value (true or false).

* Set<T>: Represents a set of elements of type T.

So, the top level elements have the following data types:

* Protocol: {a2a, mcp, acp, ...}
* AgentID: String
* agentCapability: String
* Provider: String
* Version: String (Semantic Versioning)
* Extension: String
* Cert: X.509 Certificate
* Sig: Digital Signature
* ANSName: String
* Endpoint: String

3) Verification Rules:

Certificate Chain Verification

VerifyCertChain (Cert, TrustedCA) -> Boolean:
1. Get the certificate authority (CA_signer) that signed the Cert.
2. Check for Certificate Revocation status of Cert via CRL or OCSP.
3. If Cert is Revoked, Return False.
4. If CA_signer is the TrustedCA, Return True (assuming Cert is validly signed by it and not expired).
5. Else, get the certificate of CA_signer (IssuerCert) and recursively call VerifyCertChain(IssuerCert, TrustedCA).
6. If no trusted CA is found in the chain, Return False.

Figure 4: Algorithm 1: Certificate Chain Verification

Digital Signature Verification

VerifySignature (Data, Signature, PublicKey) -> Boolean:
1. Use PublicKey to decrypt/verify the Signature against the Data.
2. Hash Data using the agreed-upon cryptographic hash function (e.g., SHA-256) that was used by the signer.
3. Compare the hash obtained from the signature with the hash calculated from the Data.
4. If the hashes match and the signature is cryptographically valid, Return True.
5. Else, Return False.

Figure 5: Algorithm 2: Digital Signature Verification

3.4. Protocol-Agnostic Communication Schema

ANS utilizes JSON Schema [RFC8259] to define the structure for all registry interactions, including discovery requests/responses and registration/renewal requests/responses. This ensures structured, validated communication across different protocols.

An example AgentRegistrationRequest schema is shown below:

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "AgentRegistrationRequest",
  "description": "Schema for Agent Registration Request",
  "type": "object",
  "properties": {
    "protocol": {
      "type": "string",
      "enum": ["a2a", "mcp", "acp"],
      "description": "Communication Protocol"
    },
    "agentID": {
      "type": "string",
      "description": "Unique Agent Identifier"
    },
    "agentCapability": {
      "type": "string",
      "description": "Primary Agent Capability"
    },
    "provider": {
      "type": "string",
      "description": "Name of the Provider"
    },
    "version": {
      "type": "string",
      "pattern": "^(0|[1-9]\\d*)\\.(0|[1-9]\\d*)\\.(0|[1-9]\\d*)(?:-((?:0|[1-9]\\d*|\\
d*[a-zA-Z-][0-9a-zA-Z-]*)(?:\\.(?:0|[1-9]\\d*|\\d*[a-zA-Z-][0-9a-zA-Z-]*))*))?(?:\\+([0-9
a-zA-Z-]+(?:\\.[0-9a-zA-Z-]+)*))?$",
      "description": "Semantic Versioning format"
    },
    "extension": {
      "type": "string",
      "description": "Extension Metadata"
    },
    "certificate": {
      "type": "object",
      "properties": {
        "subject": {"type": "string", "description": "Certificate Subject"},
        "issuer": {"type": "string", "description": "Certificate Issuer"},
```

```
           "pem": {
             "type": "string",
             "description": "PEM-encoded Certificate (strongly recommended to use a secur
e vault reference instead)",
             "readOnly": false
           }
         },
         "required": ["subject", "issuer", "pem"]
       },
       "protocolExtensions": {
         "type": "object",
         "description": "Protocol-specific data"
       }
     },
     "required": ["protocol", "agentID", "agentCapability", "provider", "version", "certi
ficate"]
   }
```

Figure 6: Snippet: AgentRegistrationRequest JSON Schema

Core Fields:
Include common elements like agent communication protocol types (a2a, mcp, acp, etc.), requesting/responding agent identifiers, timestamps, and PKI certificate details (subject, issuer, PEM representation - though referencing a secure vault is recommended for the PEM in production).

protocolExtensions:
A key field within the schema acts as a container for protocol-specific data (e.g., an A2A Agent Card, MCP tool descriptions, ACP agent profiles). This allows the registry to store and query protocol-specific agentCapabilities while maintaining a common core schema.

Validation:
All interactions with the registry must be validated against these schemas. (Complete schemas are referenced in Appendix A).

3.5. ANS Naming Structure and Resolution

ANS defines a robust, protocol-agnostic mechanism for naming and resolving agents across heterogeneous agentic environments. Its principal function is to establish a uniform Endpoint format that encodes identity, agentCapability, and contextual metadata for any given agent, irrespective of the underlying transport or runtime architecture. ANS ensures that both human-readable and machine-resolvable identifiers are preserved in a format designed to facilitate dynamic discovery, rigorous trust verification, secure communication, seamless service composition, and the representation of relationships between agents. A key motivation for ANS is to move

beyond simple naming resolution to enable precise agentCapability discovery, which is not achievable with traditional systems like DNS. The design of ANS acknowledges that the agent's agentCapabilities are paramount for intelligent interactions, distinguishing it from simpler naming systems like DNS.

3.5.1. Formal Naming Structure

The ANSName is formally defined as a string constructed from the following components, concatenated in a specific order:

```
ANSName = Protocol "://" AgentID "." agentCapability "." Provider ".v" Version ["." Exten
sion]
```

Figure 7: ANSName Structure

Where:

* Protocol ∈ {a2a, mcp, acp, ...} (Specifies the primary communication protocol)

* AgentID: A unique identifier for the agent within its provider and protocol scope. (String)

* agentCapability: Describes the primary capability or service offered by the agent. (String)

* Provider: The name of the entity or organization providing the agent. (String)

* Version: The semantic version of the agent (e.g., "1.0.2"). (String)

* Extension (Optional): Used for deployment-specific or provider-defined metadata (e.g., "hipaa", "testenv"). (String)

Constraints:

* Version MUST adhere to Semantic Versioning standards.

* AgentID, agentCapability, Provider SHOULD be registered with a governance authority (similar to ICANN for DNS domains) to prevent collisions and ensure uniqueness.

* Extension SHOULD be used for deployment-specific or provider-defined metadata, not for core identity. In the actual implementation, a registry of reserved tokens for Extension can enhance security.

Example ANSName:
"a2a://textProcessor.DocumentTranslation.AcmeCorp.v2.1.hipaa"

3.5.2. Resolution Process

The resolution mechanism in ANS maps a fully qualified ANSName to an actionable reference, such as a network address, service binding, or a detailed metadata document (Endpoint). Resolution can be achieved through distributed lookups, local resolver caches, or enterprise-specific ANS gateways, providing deployment flexibility. Critically, ANS moves beyond simple name resolution to facilitate precise agentCapability discovery.

When an agent (the "Requesting Agent") requires resolution, it queries the ANS service (a fundamental component of the Agent Registry infrastructure). The query includes the ANSName of the target agent and can optionally incorporate agentCapability filters to refine the search.

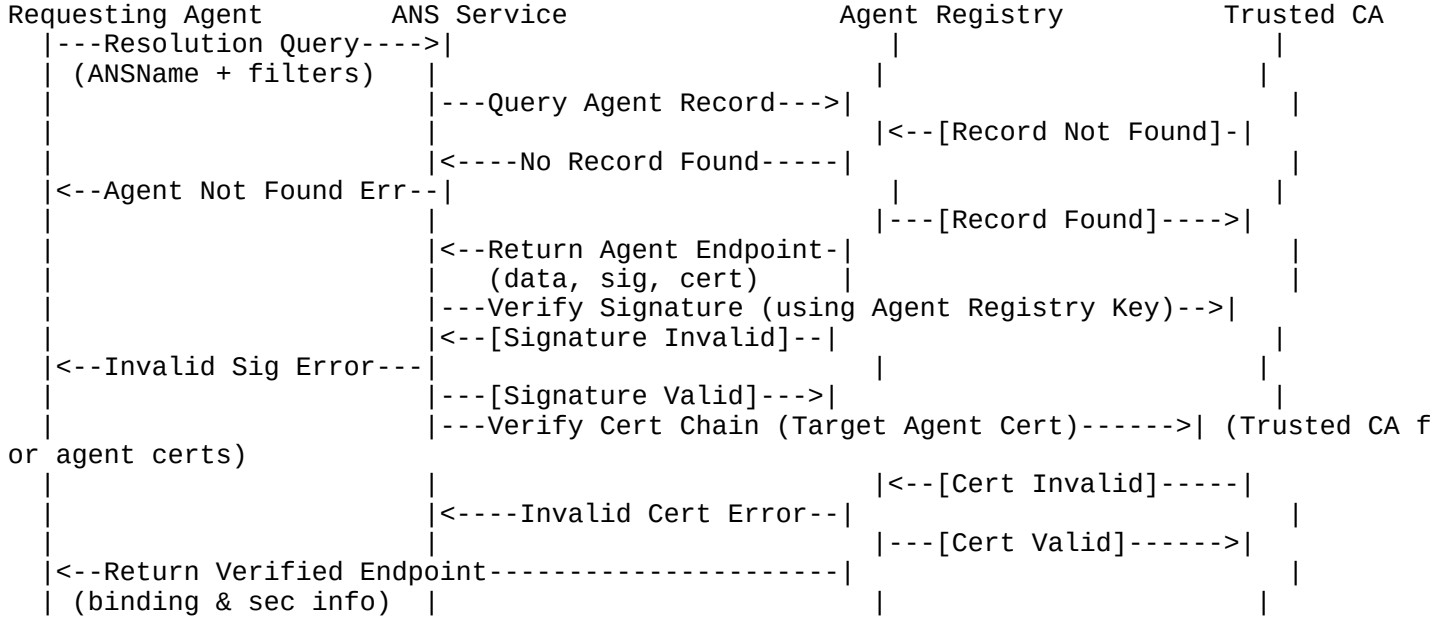

Figure 8: Agent Resolution Process Sequence

3.5.3. Formal Resolution Algorithm

The ANS resolution algorithm takes an ANSName and an optional RequestedVersionRange as input and returns a resolvable Endpoint or an error.

```
   Resolve(ANSName, RequestedVersionRange) -> EndpointRecord / ERROR:
   1. Parse ANSName into Protocol, AgentID, agentCapability,
      Provider, Version, Extension.
   2. Query Agent Registry for Agents with matching Protocol, AgentID,
      agentCapability, Provider.
   3. If no match found:
      Return ERROR("Agent not found").
   4. If multiple matches found (different versions):
      Match = VersionNegotiation(Matches, RequestedVersionRange).
      If Match is ERROR("Incompatible Version"):
        Return ERROR("Incompatible Version").
   5. EndpointRecord_struct = GetAgentEndpointRecord(Match.AgentID).
      // Renamed to avoid conflict, EndpointRecord_struct: {data, signature, Cert}
      If EndpointRecord_struct is ERROR:
        Return ERROR("Could not retrieve endpoint record").
   6. Valid = VerifyAgentEndpointRecord(EndpointRecord_struct,
                                        TrustedCA_for_Registry_Signature,
                                        TrustedCA_for_Agent_Certs).
   7. If Valid is False:
      Return ERROR("Invalid Endpoint").
   8. Return EndpointRecord_struct.data. // Contains the actual endpoint URI/info

   --- Helper Functions ---
   GetAgentEndpointRecord(AgentID_from_Match) -> EndpointRecord_struct / ERROR:
     // Agent Registry implements this to fetch records.
     // This function must enforce authentication and authorization
     // against Agent Registry ACLs if applicable.
     // Returns {data, signature, Cert} or ERROR.

   VerifyAgentEndpointRecord(EndpointRecord_struct, RegistryTrustedCA, AgentCertTrustedCA
) -> Boolean:
     // 1. Verify signature on EndpointRecord_struct.data using
     //    AgentRegistry.PublicKey (obtained from a cert issued by RegistryTrustedCA).
     signatureValid = VerifySignature(EndpointRecord_struct.data,
                                      EndpointRecord_struct.signature,
                                      AgentRegistry.PublicKey);
     If not signatureValid:
       Return False; // Or ERROR("Invalid Registry Signature")

     // 2. Verify certificate chain of the target Agent's Cert (EndpointRecord_struct.Cer
t)
     //    against AgentCertTrustedCA.
     certChainValid = VerifyCertChain(EndpointRecord_struct.Cert, AgentCertTrustedCA);
     If not certChainValid:
       Return False; // Or ERROR("Invalid Agent Certificate Chain")

     Return True;

   // VerifySignature and VerifyCertChain are defined in Section 3.3
```

```
VersionNegotiation(Matches, RequestedVersionRange) -> Match_object / ERROR:
  1. Sort Matches by Version (highest to lowest Semantic Version).
  2. For each Match_item in Matches:
  3.   If RequestedVersionRange is empty OR RequestedVersionRange == "*" OR
         IsVersionCompatible(Match_item.Version, RequestedVersionRange):
  4.       Return Match_item;
  5. End For
  6. Return ERROR("Incompatible Version");

IsVersionCompatible(AgentVersion, ReqRange) -> Boolean:
  // Implement Semantic Version range compatibility check.
  // (e.g., using a library like node-semver's satisfies function)
  // 1. Attempt to parse ReqRange as a SemVer range.
  // 2. If parsing fails, treat ReqRange as a specific SemVer version.
  // 3. Check if AgentVersion is satisfied by the ReqRange.
  // Example: return SemVer.satisfies(AgentVersion, ReqRange);
  Return true; // Placeholder for actual SemVer library call
```

Figure 9: Algorithm 3: ANS Resolution

IMPLEMENTATION NOTES:

* *Cacheability:* To ensure resolvers know when to re-validate EndpointRecords, the Agent Registry MUST include a Time-To-Live (TTL) value with each resolved Endpoint. The TTL indicates the number of seconds for which the EndpointRecord can be cached. A recommended default TTL is 300 seconds (5 minutes), but this value MAY be adjusted based on factors such as the volatility of the agent's configuration or the security policy of the Agent Registry. Resolvers MUST re-validate the EndpointRecord (by calling GetAgentEndpointRecord) after the TTL has expired.

* *Version Negotiation and Pre-release Tags:* When using SemVer.satisfies for version negotiation, pre-release tags (e.g., -rc1, -beta) MUST be considered to have lower precedence than the corresponding stable version. For example, version 1.0.0-rc1 would be considered lower precedence than 1.0.0. This ensures that resolvers prefer stable versions over pre-release versions unless explicitly requested (e.g., by specifying a pre-release version range).

3.5.4. Secure Resolution Implementation

Key security aspects for the resolution process include:

Trust Anchor:
The trust anchor for ANS is the Certificate Authority (CA) that issues the Agent Registry's own certificate. The Agent Registry's

certificate contains the public key used to verify the digital signatures on the Agent Registry's responses. This Agent Registry certificate (or its issuing CA) must be trusted by all agents that use ANS. A separate trust anchor (or set of CAs) will be used to validate the certificates of the agents themselves.

Digital Signatures:
All responses from the Agent Registry (containing Endpoint information) MUST be digitally signed using the Agent Registry's private key. Clients (Requesting Agents) verify this signature using the Agent Registry's public key to ensure the integrity and authenticity of the received data.

DNSSEC-like Security Considerations:
While DNSSEC provides a model for securing DNS, its direct application to ANS requires careful evaluation due to potential increased risk of DoS amplification attacks from larger crypto-fields in responses. Mitigation strategies like rate limiting, traffic filtering, and anycast deployment are crucial if DNSSEC-like mechanisms are adopted.

Certificate Revocation:
A robust and timely mechanism for certificate revocation is essential. If the Agent Registry's private key is compromised, its certificate MUST be revoked immediately. Standard methods like CRLs [RFC5280] or OCSP [RFC6960] should be used for both Agent Registry and individual agent certificates.

Ongoing Threat Modeling:
Continuous threat modeling is necessary to identify and proactively address potential vulnerabilities in the secure resolution mechanism.

3.6. ANS Challenges and Governance

Deploying and maintaining a global ANS involves addressing several key challenges:

Naming Collisions and Squatting:
Ensuring the uniqueness of ANSNames, particularly the <AgentID>, <agentCapability>, and <Provider> segments, requires a managed registration process. A governance model, potentially similar to ICANN's role for DNS, might be necessary to manage top-level agentCapabilities, provider identifiers, and resolve disputes.

Scalability:
Supporting a potentially vast number of agents (billions) requires highly scalable registry storage solutions (e.g., distributed

databases like Cassandra, NoSQL databases) and efficient resolution mechanisms (e.g., distributed hash tables (DHTs), robust caching layers, geographically distributed resolution points).

Governance:
Establishing clear and comprehensive policies for name allocation, dispute resolution, operational practices for CAs and RAs, and managing the evolution of the trust infrastructure is crucial for the long-term stability, trustworthiness, and adoption of ANS.

3.7. Agent Identity

Agent identity within the ANS framework is multifaceted, encompassing several components:

Cryptographic Identity:
The agent's PKI certificate, signed by a trusted CA, provides a verifiable cryptographic identity, linking its public key to its ANSName and other attributes.

Logical Identity:
The ANSName itself serves as a human-readable, structured identifier that conveys information about the agent's protocol, capabilities, provider, and version.

Protocol-Specific Identity:
Agents may possess identities within their native communication protocols (e.g., A2A Agent Card ID, MCP tool identifiers, ACP agent URIs). This information is stored within the 'protocolExtensions' field of the agent's record in the registry.

Verifiable Claims:
The registry could support the attachment of digitally signed attestations or verifiable credentials to agent profiles, such as compliance certifications, capability endorsements, or security audit results.

Identity linkage:
ANS ecosystem leverages the structured naming convention to establish relationships between agents. The core principle is that an agent's ANSName serves as a unique and resolvable identifier, allowing other agents or the system itself to reference it. The Agent Registry, upon recognizing this request, not only returns the target agent's details (binding, metadata, certificates) but also automatically resolves the linked agents, effectively materializing the relationship and providing all necessary information for secure and informed interaction. This

automated relationship discovery, built upon the foundation of uniquely identifiable and resolvable agent names, significantly simplifies the orchestration and coordination of complex multi-agent systems.

Agent Card Validation: The integrity of Agent Cards within the ANS ecosystem is verified through cryptographic methods, utilizing the relational patterns between agents. This process ensures that capability declarations are validated against organizational policies. Additionally, endpoint URL structures are enforced to comply with security standards such as TLS and proper domain constraints. The Agent Registry oversees and connects these identity components, facilitating verification through challenge-response protocols that rely on the agent's private key. Both the Requesting Agent and the Registration Authority (RA) play crucial and distinct roles in this validation process.

Requesting Agent Responsibility: The Requesting Agent has a primary and ongoing responsibility for validating the Agent Card before every interaction. This includes:

* *Cryptographic Verification:* Verifying the Agent Card's digital signature to ensure it hasn't been tampered with.

* *Capability Alignment:* Confirming that the agent's stated capabilities are actually what the Requesting Agent expects and needs for the intended interaction. This might involve checking specific input/output schemas or testing the agent's performance on sample tasks before relying on it for critical operations.

The Requesting Agent's validation is not a one-time event; it's a continuous process that ensures the agent remains trustworthy for each specific interaction. A failure to properly validate an Agent Card could expose the Requesting Agent to significant security risks.

Registration Authority (RA) Responsibility: The RA performs a foundational validation of the Agent Card during the agent registration and renewal processes. This includes:

* *Signature Verification:* Verifying the Agent Card's signature and the validity of the associated certificate.

* *Policy Adherence:* Ensuring the agent's claimed capabilities and operational practices comply with broader registry policies and legal requirements.

* *Legitimacy Checks:* Performing checks to confirm the identity and legitimacy of the agent's owner (e.g., domain validation, organizational checks).

The RA's validation provides a baseline level of trust, but it does not replace the need for the Requesting Agent to perform its own, more context-specific validation.

Additionally, endpoint URL structures are enforced to comply with security standards such as TLS and proper domain constraints. The Agent Registry oversees and connects these identity components, facilitating verification through challenge-response protocols that rely on the agent's private key.

Agent Capability Attestation: The AI agent's identity and claimed capabilities are authenticated through zero-knowledge proof methods. Specifically, ZKPs can be employed to allow an agent to prove that it possesses certain capabilities (e.g., access to specific data, the ability to perform a certain computation) without revealing how it possesses those capabilities or the underlying data itself. For example, an agent might use a ZKP to prove it has access to a database containing sensitive patient information without revealing the specific query it will use or any of the patient data. This involves the agent constructing a proof, based on its private knowledge and the claimed capabilities, that can be verified by the Agent Registry (or another agent) using only publicly available information.

The verifier gains assurance that the agent possesses the claimed capabilities without learning any sensitive information about the agent's internal state or data. During runtime, capabilities are dynamically validated as part of the resolution process. To further enhance real-time verification, challenge-response mechanisms are employed.

Challenge-Response Example: Imagine an agent claims to be able to perform "Sentiment Analysis" with a certain accuracy.

* The Agent Registry (or a verifying agent) sends the claimed "Sentiment Analysis" agent a specific challenge: a piece of text with a known sentiment.

* The "Sentiment Analysis" agent processes the text and returns its sentiment classification (positive, negative, neutral) and a confidence score.

* The Agent Registry (or verifying agent) compares the agent's response to the known sentiment and the claimed accuracy.

* If the response is correct and the confidence score aligns with the agent's claimed accuracy, the agent's capability is considered validated (for that specific challenge).

* If the response is incorrect or the confidence score is significantly lower than the claimed accuracy, the agent's claimed capability is called into question and further challenges or even revocation of the agent's registration might be triggered.

This challenge-response process can be repeated periodically or triggered based on certain events (e.g., a change in the agent's code, a security alert). The challenges can be designed to test various aspects of the agent's claimed capabilities, ensuring that it continues to function as expected over time. The Agent Registry maintains a history of challenge-response results to track the agent's performance and reliability.

By combining ZKPs for initial capability attestation with challenge-response mechanisms for ongoing validation, the ANS provides a robust framework for ensuring the trustworthiness of AI agents.

Authentication Enforcement: The process involves validating the OAuth 2.0 flow to ensure the legitimacy of authorization tokens, verifying mTLS certificates to confirm alignment with the registered agent's identity, and checking JSON Web Tokens (JWTs) to ensure their signatures and claims are accurate and properly authenticated.

Agent Identity Module Examples: The Agent Identity module implements resource access control through capability-based security. Below are examples for A2A and MCP protocols:

```
{
  "a2aCapabilityVerification": {
    "capabilityVerification": {
      "proofMechanism": "ZKP",
      "verificationCircuit": {
        "constraints": [
          "agent.hasCapability(c) AND agent.isAuthorized(c)",
          "agent.certificate.isValid() AND agent.certificate.notRevoked()"
        ],
        "proofGeneration": "Groth16",
        "verificationKey": "0x4a8f..."
      }
    },
    "rateLimit": {
      "algorithm": "TokenBucket",
      "refillRate": "100/s",
      "burstCapacity": 500,
      "perCapability": true
    }
  }
}
```

Figure 10: A2A Capability Verification Example

```
{
  "mcpAgentIdentity": {
    "resourceAccessControl": {
      "model": "RBAC+ABAC",
      "policyDecisionPoint": {
        "engine": "OPA",
        "evaluationMode": "distributed"
      },
      "contextAttributes": [
        "agent.role",
        "resource.classification",
        "time.window",
        "operation.sensitivity"
      ]
    },
    "toolRegistration": {
      "sandboxValidation": {
        "environment": "gVisor",
        "runtime": "V8Isolate",
        "memoryLimit": "256MB",
        "cpuQuota": "0.5",
        "networkPolicy": "DENY_ALL"
      }
    }
  }
}
```

Figure 11: MCP Agent Identity Example

The Agent Registry manages and links these identity facets, enabling verification via challenge-response protocols using the agent's private key.

4. Request/Response Schema for ANS Name Resolution

The following core JSON Schemas define the structure for AgentCapability requests (used for resolving ANSNames) and responses. These ensure validated and structured communication for discovery operations. (Full schemas are linked in Appendix A).

AgentCapabilityRequest Schema:

```
   {
     "$schema": "http://json-schema.org/draft-07/schema#",
     "title": "AgentCapabilityRequest",
     "description": "Schema for Agent agentCapability Request (Resolution)",
     "type": "object",
     "properties": {
       "requestType": { "type": "string", "enum": ["resolve"] },
       "protocol": { "type": "string",
                     "description": "Target protocol (from ANSName)"},
       "agentID": { "type": "string",
                    "description": "Target agentID (from ANSName)"},
       "agentCapability": { "type": "string",
                            "description": "Target capability (from ANSName)"},
       "provider": { "type": "string",
                     "description": "Target provider (from ANSName)"},
       "version": { "type": "string",
                    "description": "Target version or version range (from ANSName or quer
y)" },
       "extension": { "type": "string",
                      "description": "Target extension (from ANSName, optional)"}
     },
     "required": ["requestType", "protocol", "agentID",
                  "agentCapability", "provider", "version"]
   }
```

Figure 12: AgentCapabilityRequest JSON Schema

AgentCapabilityResponse Schema:

```
   {
     "$schema": "http://json-schema.org/draft-07/schema#",
     "title": "AgentCapabilityResponse",
     "description": "Schema for Agent agentCapability Response (Resolution)",
     "type": "object",
     "properties": {
       "Endpoint": {
         "type": "string",
         "description": "Resolved agent address/endpoint URI (e.g., a2a://translatorBot.D
ocumentTranslation.exampleCorp.v1.2.3.secure)"
       },
       "signature": {
         "type": "string",
         "description": "Digital signature of the Endpoint data by Agent Registry"
       },
       "cert": {
         "type": "string",
         "description": "PEM-encoded Certificate of the target Agent (strongly recommende
d to use a secure vault reference instead)",
         "readOnly": false
       }
     },
     "required": ["Endpoint", "signature", "cert"]
   }
```

Figure 13: AgentCapabilityResponse JSON Schema

Key points regarding schemas:

* Use a JSON Schema validator library for enforcement.

* Pay close attention to required fields and data type constraints.

* Validate all incoming and outgoing messages related to ANS resolution against these schemas.

* Implement graceful error handling for validation failures.

* Monitor evolving standards in agent communication (A2A, MCP, ACP) and update schemas as necessary to maintain compatibility and incorporate new features.

5. Protocol Adapter Layer

The Protocol Adapter Layer is a critical component that enables the ANS registry to support diverse agent communication protocols (like A2A, MCP, ACP) without being tightly coupled to any single one. It acts as an intermediary, translating between the registry's core logic and internal data storage format and the specific requirements and metadata formats of each supported protocol.

Modularity:
Adapters are implemented as distinct, pluggable modules (e.g., plugins or separate services). This allows for easy addition of support for new protocols or updates to existing ones without altering the core registry logic.

Functionality:
Each protocol adapter understands how to:

* Parse protocol-specific metadata (e.g., from an A2A Agent Card, MCP Tool Description) and map relevant parts into the registry's internal representation, especially within the 'protocolExtensions' field of an agent's record.

* Extract information from an agent's registry record to construct responses to protocol-specific discovery queries.

* Potentially handle protocol-specific aspects of the registration or validation process if they differ significantly from the generic ANS procedures.

Each adapter is responsible to securely implement features defined by each protocol.
Some protocols have built-in security, such as signed messages.

Translation Focus:
Adapters primarily focus on metadata translation for the purposes of discovery and registration. They do not typically handle real-time message translation between different protocols during agent-to-agent interaction. Their role is to help agents find each other and verify identity; subsequent communication generally uses the agents' shared native protocol.

Security aspect:
* All adapter implementations must use secure, up-to-date libraries commonly employed for the supported protocols.

* Implementations MUST adhere to the security guidance provided by each specific protocol.

* Since each adapter parses potentially untrusted blobs of data (protocol-specific metadata), it is MANDATED that they be implemented in memory-safe languages (e.g., Rust, Go) and be accompanied by formal test suites to ensure robustness against parsing vulnerabilities.

<preamble>:Conceptual flow showing ANS interacting with different agent protocols via their respective adapters.

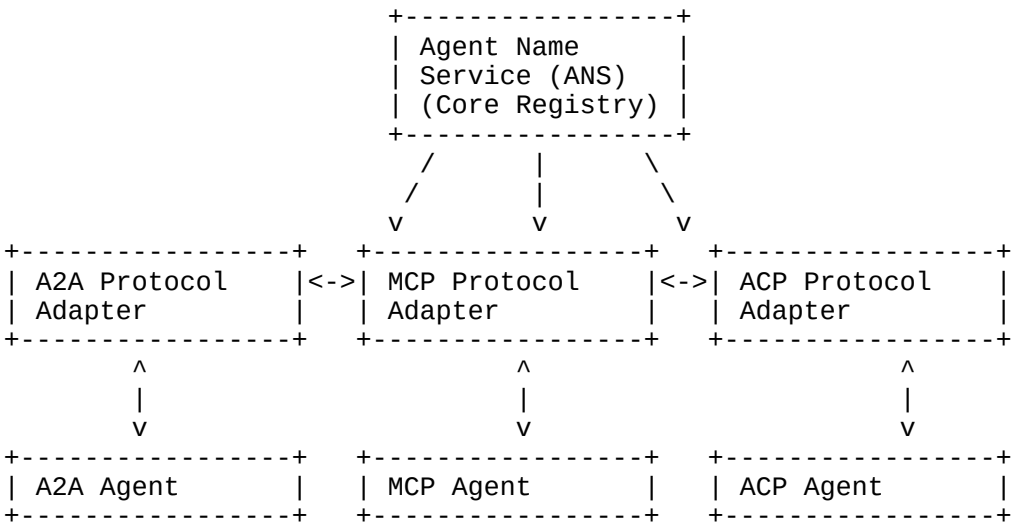

Figure 14: Protocol Adapter Layer Interaction

5.1. A2A Protocol Adapter

* Parses/stores A2A Agent Card information within protocolExtensions.

* Enables discovery based on A2A agentCapabilities advertised in the card.

* Facilitates finding A2A Endpoints listed in the card.

5.2. MCP Adapter

* Parses/stores MCP Tool and Resource descriptions within protocolExtensions.

* Enables discovery of agents offering specific MCP tools/resources.

* Facilitates finding MCP-compliant Endpoints.

5.3. ACP Adapter

* Parses/stores ACP agent profiles and agentCapability advertisements within protocolExtensions.

* Supports discovery based on ACP roles or agentCapabilities.

* May assist in bootstrapping ACP delegation or orchestration workflows by providing initial agent references.

5.4. Extension Points

Adding support for a new protocol involves creating a new adapter module that implements the required mapping and discovery logic interfaces defined by the registry framework. This ensures the registry can evolve with the agent ecosystem.

5.5. Cross-Protocol Interoperability Limits

The registry primarily enables discovery and identity verification across protocols. An A2A agent can discover an agent advertising MCP tools, verify its identity via PKI, and potentially interact if it also understands MCP or uses a gateway. ANS does not automatically translate A2A tasks into MCP requests. It provides the foundational trust and location information, but deeper semantic interoperability often requires additional mechanisms or multi-protocol support within the agents themselves.

5.6. Protocol Adapter API Definition

Protocol Adapters are implemented as plugins that conform to the following interface (expressed in a language-agnostic way; adapt to your chosen implementation language):

```
interface ProtocolAdapter {
  // Identifies the protocol supported by this adapter
  // (e.g., "a2a", "mcp", "acp")
  String getProtocol();

  // Parses protocol-specific metadata from the raw
  // 'protocolExtensions' data (e.g., a JSON object or string)
  // and maps it to a structured internal representation used by
  // the registry (e.g., a map or a dedicated object).
  // Returns a representation of the extracted data.
  Map<String, Object> parseMetadata(Object protocolExtensionsData);

  // Constructs a protocol-specific discovery response payload
  // based on information retrieved from an agent's registry record.
  // 'registryRecord' contains the agent's core ANS data and the
  // parsed protocolExtensions.
  Object createDiscoveryResponse(Map<String, Object> registryRecord);

  // Handles any protocol-specific validation logic required during
  // the agent registration process, beyond the standard ANS checks.
  // 'registrationRequest' is the full request object.
  // Returns true if validation passes, false otherwise.
  Boolean validateRegistration(AgentRegistrationRequest request);
}
```

Figure 15: Interface: ProtocolAdapter

Important Considerations:

* This is a basic interface. More complex scenarios might require additional methods (e.g., for handling updates, deletions, etc.).

* The Map<String, Object> type should be replaced with more specific data structures based on your chosen implementation language and registry data model.

* Error handling should be implemented using exceptions or appropriate return codes.

SUMMARY OF ANS FUNCTIONAL LAYERS

ANS Component	ANS Functional Layer	JSON Schema Implementation
Policy Enforcement Engine	Request Processing	Forward-chaining rule processor with sequential evaluation of Agent's certs.
PKI Governance	Security Infrastructure	Hierarchical PKI with Agent specific Cert Validation & Integration.
A2A Protocol Engine	Protocol Adapter	Zero-knowledge proof capability verification
MCP Protocol Engine	Protocol Adapter	RBAC+ABAC access control with OPA
Consensus Engine	Distributed Governance	Analyzing Agent Signatures
ANS Audit Trail System	Compliance Layer	Deterministically generate unique Agent IDs (UUIDv5 based on PKI public key hash)

Table 1: Summary of ANS Functional Layers

This technical implementation of ANS ensures that universal agent Registry/Directory operates with cryptographic assurance, distributed consensus for critical operations, and real-time compliance enforcement while maintaining high performance and scalability requirements essential for enterprise AI agent deployments.

6. Security Considerations

The security of the Agent Name Service is paramount for its adoption and the trustworthiness of the agent ecosystems it supports. This section outlines key threats, mitigation strategies, and security controls, referencing the MAESTRO 7 Layers framework [Huang-MAESTRO] where applicable. See RFC 3552 [RFC3552] for a general guide on writing security considerations.

6.1. MAESTRO-Based Threat Analysis

This section presents a systematic security threat analysis of the proposed ANS protocol. We identify key potential vulnerabilities and map them onto the MAESTRO 7 Layers framework to provide a structured understanding of the threat landscape and the corresponding mitigation strategies integrated into our design. MAESTRO stands for Multi-Agent Environment, Security, Threat, Risk, and Outcome, and its seven layers are: Foundation Models, Data Operations, Agent Frameworks, Deployment and Infrastructure, Evaluation and Observability, Security and Compliance, and Agent Ecosystem. By analyzing vulnerabilities and risks at each architectural layer, as well as cross-layer interactions, MAESTRO enables security teams to proactively identify, assess, and mitigate threats unique to agentic AI.

6.1.1. Threat: Agent Impersonation

Risk:
An adversary attempts to impersonate a legitimate, registered agent inside the ecosystem.

Mitigation Strategy:
Mandatory implementation of PKI. Verification of agent identity through validation of agent-specific digital certificates (Cert) issued by a trusted CA. Agent must prove possession of the private key.

Formal Check:
* Agent MUST provide valid Cert on registration.
* RA MUST verify Cert against trusted CA.
* All communication MUST be digitally signed; recipient MUST verify signature using public key from Cert.
* See Section Section 3.2 (PKI Integration) and Section 3.7 (Agent Identity).

MAESTRO Layer Mapping:
Layer 7 (Agent Ecosystem) - Agent Impersonation/Identity Attack.

6.1.2. Threat: Registry Poisoning

Risk:
An adversary tries to inject malicious data into the Agent Registry (e.g., corrupting agentCapabilities or Endpoints).

Mitigation Strategy:
Strict RA validation during registration/renewal; cryptographic signing of registry responses (secure resolution); stringent database access controls. Secure resolution ensures responses are signed by the Agent Registry's private key, verifiable by clients.

Formal Check:
* RA validation procedures (Section Section 3.1).
* Secure, authenticated registry responses (Section Section 3.5.4 leveraging PKI Section Section 3.2).
* Database access controls (Infrastructure L4 / Data L2).

MAESTRO Layer Mapping:
* Layer 7 (Agent Ecosystem) - Compromised Agent Registry, Malicious Agent Discovery.
* Layer 6 (Security and Compliance) - RA validation policies, cryptographic signing.
* Layer 4 (Deployment and Infrastructure) / Layer 2 (Data Operations) - Database access controls.

6.1.3. Threat: Man-in-the-Middle (MitM) Attacks

Risk:
An adversary modifies communications between system components (agent-agent, agent-registry, agent-RA/CA).

Mitigation Strategy:
Message authenticity/integrity via digital signatures (agent's PKI private key). Secure transport (e.g., mTLS) is best practice but PKI signing is the core mechanism discussed. Formal resolution algorithm (Section 3.5.3) enforces response integrity.

Formal Check:
PKI signing (Section Section 3.2).

MAESTRO Layer Mapping:
* Layer 4 (Deployment and Infrastructure) - Targets communication channels. Secure transport operates here.
* Layer 6 (Security and Compliance) - PKI framework providing keys/certs managed via L6.

6.1.4. Threat: Denial of Service (DoS) / Distributed Denial of Service (DDoS)

Risk:
Adversary attempts to incapacitate Agent Registry, RA, CA, or resolution services via traffic flooding or resource exhaustion.

Mitigation Strategy:
Resilience through distributed implementation design. Standard operational defenses (rate limiting, DDoS protection services). Formal resolution algorithms may include DoS limits.

Formal Check:
Architectural design (distributed options in Section Section 7).

MAESTRO Layer Mapping:
* Layer 4 (Deployment and Infrastructure) - DoS attacks target L4 availability. Architectural resilience is L4 design.

* Layer 7 (Agent Ecosystem) - Impact felt at L7 (disrupted discovery/interaction).

6.2. Additional Security Controls and Considerations

Implementing ANS requires thorough threat modeling and comprehensive security controls.

6.2.1. PKI Security Controls

Certificate Revocation:
Robust CRL [RFC5280] or OCSP [RFC6960] mechanisms are essential for handling compromised or decommissioned agent certificates.

Secure Key Storage:
Agent private keys MUST be protected using strong mechanisms (e.g., Hardware Security Modules (HSMs), secure enclaves, OS-level key stores). Compromise of a private key requires immediate revocation and re-issuance of the certificate.

Registry Access Control:
Strict authentication and authorization mechanisms for managing the registry (RA operations) and potentially for querying, depending on policy.

RA Validation Rigor:
The RA must implement a rigorous validation process to prevent malicious or fraudulent registrations. This may include domain validation, organizational checks, and, in some cases, manual review.

6.2.2. ANS-Specific Security Controls

Resolution Integrity:
As discussed, use DNSSEC-like mechanisms or signed responses from the registry to prevent ANS manipulation or spoofing of resolution results.

DoS Mitigation:
Standard defenses (rate limiting, firewalls, anycast routing) for resolution Endpoints are critical.

CA Security:
The CA(s) underpinning ANS MUST adhere to standard CA best practices (e.g., offline root CAs, regular audits, use of HSMs). Consideration should be given to using established, trusted CAs or building a highly secure, dedicated CA infrastructure.

Sybil Attack Resistance:
Registration processes should be designed to make the creation of large numbers of fake identities (Sybil attack) difficult and/or costly.

6.2.3. Protocol Integration Security

Protocol-Specific Security:
Leverage security features within A2A, MCP, ACP (e.g., OAuth, capability tokens) for agent-to-agent communication post-discovery.

Governance and Trust Framework:
Clear policies, liability, and trust anchors are essential for adoption.

6.2.4. Side-Channel Deanonymization and Mitigation

The process of querying the Agent Registry for specific agentCapabilities could unintentionally reveal sensitive information about the querier’s intent, business objectives, or operational profile. To mitigate this risk, the implementation must prioritize:

* *Private Information Retrieval (PIR) implementation:* To enable retrieval of information from the Agent Registry without revealing which information is being retrieved. Evaluate various PIR libraries to determine the most secure and efficient implementation based on use case. Prioritize the most performant and secure library.

* *Anonymized Query Relays:* Implementing query relays to hide the origin of the requests and prevent ANS Registry from seeing what agents are querying for what capabilities.

* *Differential Privacy for Aggregated Query Data:* Anonymize all data before sending to a 3rd party, ensuring that PII isn't included in the request.

* *Query Pattern Analysis:* Analyzing query patterns for anomalous behavior, and block queries that look suspicious.

* *Rate Limiting:* Implementing rate limiting to prevent potential DDoS attacks, and help obfuscate queries.

* *Auditing:* Ensure all security implementations are audited regularly.

* *Privacy Controls for Query and Response Data:* Implementation of data anonymization techniques to enhance privacy compliance.

7. Implementation Considerations

The Agent Registry, the core of ANS, can be implemented using various architectural patterns. The choice depends on factors such as the expected scale of deployment, trust model, performance requirements (latency, throughput), administrative overhead, and cost. The Protocol Adapter Layer is well-suited to a plugin-based architecture.

Common implementation patterns for the registry include:

Centralized:
A single logical registry instance. Simple to manage and ensures strong consistency. However, it can be a single point of failure and a performance bottleneck. Suitable for smaller, private, or experimental deployments.

Distributed (e.g., using Cassandra, CockroachDB):
Offers high availability, fault tolerance, and scalability. Technologies like Cassandra provide tunable consistency (often eventual consistency). Requires robust coordination mechanisms and careful data partitioning strategies.

Distributed (DHT - Distributed Hash Table):
Highly scalable peer-to-peer lookup. Can be complex for state management and ensuring data integrity. Suitable for very large-scale, decentralized deployments where eventual consistency is acceptable.

Blockchain/Distributed Ledger Technology (DLT):
Smart contracts can act as Registry Agents, with registrations and updates recorded as transactions on a blockchain. Offers high security, transparency, and auditability but typically suffers from high latency, limited scalability (transactions per second), and significant write amplification (one logical write results in many physical writes across the network and storage), increasing storage costs and reducing performance.

Federated:
Multiple independent registries interoperate, often aligning with organizational or geographic boundaries. Requires standardized inter-registry communication protocols, shared trust anchors (e.g., cross-certified CAs or a common root CA), and mechanisms for cross-domain name resolution and identity verification. Without these, a federated system risks becoming a collection of isolated silos.

Hybrid:
Combines elements of different patterns. For example, a centralized RA/CA infrastructure might serve multiple distributed/replicated read-only registry nodes. Caching layers (e.g., Redis, Memcached) can be used to improve read performance in many of these models. Requires careful management of consistency between components.

Key considerations when choosing an implementation pattern:

* *Consistency vs. Latency:* Strong consistency ensures all reads see the most recent write but can increase latency. Eventual consistency allows lower latency and higher scalability but reads might temporarily see stale data.

* *Write Amplification (especially for Blockchain):* Be mindful of the storage and performance implications.

* *Operational Cost:* Includes hardware, software, and personnel. Blockchain solutions can be particularly expensive to operate.

* *Complexity:* Distributed systems (DHTs, blockchains, some federated models) are generally more complex to design, implement, deploy, and maintain than centralized systems.

DECISION MATRIX FOR AGENT REGISTRY IMPLEMENTATION PATTERNS

Feature	Centralized	Distributed (Cassandra)	Distributed (DHT)	Blockchain/ DLT	Federated
Consistency	Strong	Tunable (Eventual)	Eventual	Strong	Tunable (Eventual)
Latency	Low	Medium	Medium to High	High	Medium to High
Scalability	Limited	High	Very High	Limited	High
Fault Tolerance	Low	High	High	High	Medium
Security	Medium	Medium	Medium	High	Medium
Operational Cost	Low	Medium	Medium	High	Medium
Complexity	Low	Medium	High	High	Medium

Table 2: Decision Matrix for Agent Registry Implementation Patterns

8. Future Considerations

This document lays the foundational groundwork for ANS. Significant future work is envisioned to mature the proposal into a widely adopted standard:

* *Prototype Implementation and Evaluation:* Build and rigorously evaluate a working ANS prototype. The existing work at https://github.com/kenhuangus/dns-for-agents/ serves as an initial base.

* *Bootstrap Model for Distributed Trust:* Explore an innovative bootstrap model for the distributed trust infrastructure, potentially focusing on a foundation-led root governance approach with delegated sub-spaces (analogous to CNCF’s certificate transparency roots). This includes developing a sustainable funding and fee schedule.

* *Performance & Scalability Benchmarking:* Conduct thorough benchmarking of resolution latency, registration throughput, and overall scalability under various load conditions and deployment models.

* *Advanced Cryptography:* Investigate the integration of privacy-preserving cryptographic techniques, such as zero-knowledge proofs (ZKPs) for selective capability advertisement/disclosure, or more advanced PIR schemes.

* *Formal Verification:* Mathematically model and formally verify the security properties of the registration, resolution, and identity management protocols within ANS.

* *Detailed Governance Model:* Develop a comprehensive governance model, including detailed policies for naming conventions, CA/RA operational requirements, dispute resolution mechanisms, and the evolution of the trust framework.

* *Platform and Framework Integration:* Demonstrate and facilitate integration with popular agent development frameworks (e.g., LangChain, AutoGen, CrewAI) and major cloud platforms.

* *Semantic Interoperability Enhancements:* Research mechanisms that leverage ANS metadata for deeper cross-protocol communication, such as standardized capability ontologies or translation gateways facilitated by ANS discovery.

* *Reputation Systems:* Investigate the integration of reputation scores, endorsements, or attestations into agent profiles within ANS to further enhance trust in agent interactions.

* *ANS Capability Negotiation and Binding Protocol:* Develop standardized protocols that allow agents, post-ANS resolution, to negotiate specific capabilities, agree on service level objectives (SLOs), and formally bind to service contracts.

* *Governance White Paper:* Draft a detailed white paper outlining name allocation strategies, a sustainable fee model, dispute arbitration processes, and stewardship of the root CA infrastructure.

9. Conclusion

The Agent Name Service (ANS) offers a foundational infrastructure designed to foster a more secure, trustworthy, and interconnected ecosystem for agentic AI. By addressing the critical needs of verifiable identity, secure discovery, and protocol-agnostic interoperability, ANS aims to provide significant benefits:

* *Enhancing Interoperability:* A protocol-agnostic directory facilitates seamless discovery and initial contact between diverse agents built on different communication standards.

* *Boosting Trust and Security:* The integration of PKI for identity verification enhances trust, which is crucial for agents operating in sensitive domains such as finance, healthcare, and cybersecurity.

* *Accelerating Innovation:* By providing common infrastructure for discovery and identity, ANS can lower barriers to entry for developers, allowing them to focus on core agent functionalities and solutions.

* *Facilitating Autonomous Systems:* ANS is a critical enabler for complex autonomous systems that require dynamic, secure discovery and interaction between their constituent agents (e.g., in autonomous vehicles, smart cities, and industrial automation).

* *Powering Secure AI Marketplaces:* The framework can serve as a foundation for AI marketplaces where agents with verifiable identities and capabilities can be securely listed, discovered, and engaged.

The modular Protocol Adapter Layer ensures that ANS can evolve and extend its support to new and emerging agent communication standards. While challenges related to governance, global scalability, and achieving deep semantic interoperability remain, ANS represents a necessary and significant step towards building a robust, secure, and scalable future for agentic AI.

10. IANA Considerations

This document has no IANA actions requested at this time. Future versions may request registration of a URI scheme for ANSNames, new port numbers if a dedicated ANS protocol is defined, or new registries for ANS components if deemed necessary by the community.

11. References

11.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

[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/info/rfc2119>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <https://www.rfc-editor.org/info/rfc6960>.

[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/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

11.2. Informative References

[A2A-Blog] Surapaneni, R., Jha, M., Vakoc, M., and T. Segal, "Announcing the Agent2Agent Protocol (A2A)", Google for Developers Blog , April 2025, <https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/>.

[A2A-Spec] Agent2Agent Protocol Specification Authors, "Agent2Agent (A2A) Protocol Specification", 2025, <https://github.com/google/A2A>.

[MCP-News] Anthropic, "Model context protocol (MCP)", 2024, <https://www.anthropic.com/news/model-context-protocol>.

[MCP-Spec] Model Context Protocol Specification Authors, "Model Context Protocol (MCP) Specification", March 2025, <https://modelcontextprotocol.io/specification/2025-03-26>.

[MCP-Intro-Schmid] Schmid, P., "MCP Introduction", 2025, <https://www.philschmid.de/mep-introduction>.

[FIPA-ACL] Foundation for Intelligent Physical Agents (FIPA), "FIPA Agent Communication Language Specifications", FIPA Technical Report , 2002, <http://www.fipa.org/repository/aclspecs.html>.

[Habler-A2A-Secure] Habler, I., Huang, K., Narajala, V.S., and P. Kulkarni, "Building a secure agentic Al application leveraging A2A protocol", arXiv preprint arXiv:2504.16902 , 2025, <https://www.arxiv.org/abs/2504.16902>.

[Narajala-MCP-Secure] Narajala, V.S. and I. Habler, "Enterprise-Grade Security for the Model Context Protocol (MCP): Frameworks and Mitigation Strategies", arXiv preprint arXiv:2504.08623 , 2025, <https://arxiv.org/abs/2504.08623>.

[Huang-MAESTRO] Huang, K., Sheriff, A., Sotiropoulos, J., Del, R.F., and V. Lu, "Multi-agentic system threat modelling guide OWASP GenAI security project", April 2025, <https://www.researchgate.net/publication/391204915_Multi-Agentic_system_Threat_Modelling_Guide_OWASP_GenAl_Security_Project>.

[Narajala-ToolSquatting] Narajala, V.S., Huang, K., and I. Habler, "Securing genai multi-agent systems against tool squatting: A zero trust registry-based approach", arXiv.org , 2025, <https://arxiv.org/abs/2504.19951>.

[IBM-ACP-Placeholder] IBM Research, "(Placeholder for anticipated IBM ACP publication/specification link)", 2025.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

Appendix A. Appendix A: Complete Request/Response Schemas

The detailed JSON Schema documents for various registry interactions are maintained externally due to their length. Implementers should refer to these sources for complete and up-to-date schema definitions. It is crucial that these schemas are well-commented and validated in any ANS implementation.

* AgentRegistrationRequest Schema: https://github.com/kenhuangus/dns-for-agents/blob/main/agent_registration_request_schema.json

* AgentRenewalRequest Schema: https://github.com/kenhuangus/dns-for-agents/blob/main/agent_renewal_request_schema.json

* AgentRegistrationResponse Schema: https://github.com/kenhuangus/dns-for-agents/blob/main/agent_registration_response_schema.json

* AgentRenewalResponse Schema: https://github.com/kenhuangus/dns-for-agents/blob/main/agent_renewal_response_schema.json

* AgentCapabilityRequest Schema (for resolution): https://github.com/kenhuangus/dns-for-agents/blob/main/agent_capability_request.schema.json

* AgentCapabilityResponse Schema (for resolution): https://github.com/kenhuangus/dns-for-agents/blob/main/agent_capability_response.schema.json

Appendix B. Appendix B: Glossary of Terms - Agent Name Service (ANS)

A2A (Agent2Agent Protocol):
A communication protocol developed by Google for standardizing inter-agent communication, designed to bridge different agent frameworks.

ACP (Agent Communication Protocol):
A protocol designed by IBM Research to standardize how agents communicate, enabling automation, collaboration, UI integration, and developer tooling.

Agent:
An autonomous software entity capable of performing tasks, making decisions, and interacting with other agents or systems.

Agent Identity:
The verifiable identity of an agent within the ANS ecosystem, comprising cryptographic identity (PKI certificate), logical identity (ANSName), and protocol-specific identities.

Agent Registry:
A database storing registered agent information including capabilities, security policies, PKI certificates, protocol-specific metadata, and registration/renewal timestamps.

agentCapability:
A specific function, service, or skill that an agent can perform or provide to other agents or users.

ANS (Agent Name Service):
A universal directory service framework that enables secure discovery and interoperability between AI agents across different protocols and platforms.

ANSName:
A structured, human-readable, and machine-resolvable identifier for agents in the ANS ecosystem.

CA (Certificate Authority):
A trusted entity that issues and manages digital certificates that bind public keys to entities (like agents) to establish a chain of trust in the ANS ecosystem.

Certificate Chain Verification:
The process of validating a certificate by checking the chain of trust from the certificate up to a trusted root certificate authority.

Certificate Revocation:
The process of invalidating a certificate before its scheduled expiration date, typically due to a key compromise or when an agent is deregistered.

CRL (Certificate Revocation List):
A list of digital certificates that have been revoked by the issuing CA before their scheduled expiration date and should no longer be trusted.

CSR (Certificate Signing Request):
A message, typically in a standardized format, sent by an applicant (an agent in this context) to a Certificate Authority to apply for a digital identity certificate.

Digital Signature:
A mathematical scheme for verifying the authenticity (proof of sender) and integrity (proof data has not been altered) of digital messages or documents.

DHT (Distributed Hash Table):
A decentralized distributed system that provides a lookup service similar to a hash table; keys are mapped to values, and the table is distributed among participating nodes.

DNS (Domain Name System):
The hierarchical and decentralized naming system used to identify computers, services, and other resources reachable through the Internet or other Internet Protocol networks.

DNS-SD (DNS-Based Service Discovery):
An extension to DNS that enables automatic discovery of services available on a local network using standard DNS queries.

Endpoint:
A resolvable network address, service binding, or metadata document that allows agents to connect and communicate with each other.

Extension (in ANSName):
An optional field in the ANSName structure that holds deployment-specific or provider-defined metadata, not part of the core identity.

Interoperability:
The ability of different agent systems, devices, or applications from different vendors to access, exchange, integrate, and cooperatively use data in a coordinated manner.

MAESTRO (7 Layers):
A threat modeling framework specifically designed for agentic AI systems, consisting of 7 layers, used to structure the security analysis of ANS.

MAS (Multi-Agent Systems):
Systems composed of multiple interacting intelligent agents that can cooperate, coordinate, or compete to solve problems that are beyond the individual capabilities or knowledge of each agent.

MCP (Model Context Protocol):
A protocol developed by Anthropic focused on simplifying the integration of AI models with external tools and data sources.

OCSP (Online Certificate Status Protocol):
An Internet protocol used for obtaining the revocation status of an X.509 digital certificate as an alternative to CRLs, providing more timely status information.

PKI (Public Key Infrastructure):
A set of roles, policies, hardware, software, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption.

Protocol Adapter Layer:
A component in the ANS architecture that translates between the registry's internal, protocol-agnostic representation and protocol-specific formats for metadata and discovery.

protocolExtensions (in Schema):
A flexible field within the ANS JSON schemas that acts as a container for protocol-specific data, allowing ANS to support diverse agent types.

Provider (in ANSName):
An organization or entity that offers, maintains, or is responsible for agents registered in the ANS ecosystem.

RA (Registration Authority):
An entity, authorized by a CA, that verifies agent registration and renewal requests, interacts with the CA to issue certificates, and manages aspects of the agent lifecycle.

Registry Poisoning:
A security threat where an adversary successfully injects malicious, false, or misleading data into the Agent Registry to disrupt services or deceive users/agents.

Semantic Versioning:
A formal versioning scheme with a format of MAJOR.MINOR.PATCH (e.g., 1.2.3), used to indicate compatibility and changes in agent versions.

Service Discovery:
The process of automatically detecting devices and services offered by these devices on a computer network.

Signature Verification:
The process of using the signer's public key to check that a digital signature is authentic and that the associated data has not been tampered with since it was signed.

Sybil Attack:
A type of attack on a peer-to-peer network in which a malicious actor creates a large number of pseudonymous identities and uses them to gain a disproportionately large influence.

Trust Anchor:
A CA certificate that is implicitly trusted and serves as the starting point for validating a certificate chain in a PKI system.

Version Negotiation:
The process by which communicating entities (agents in this case) determine which version of a protocol or capability to use, based on compatibility and preferences.

VerifyCertChain (Function):
A function or process that checks the validity of a digital certificate by tracing its chain of signatures and revocation statuses back to a trusted root CA (Trust Anchor).

VerifySignature (Function):
A function or process that validates a digital signature against a given message and a public key to confirm the message’s authenticity and integrity.

Acknowledgements

The authors acknowledge the contributions of the communities and organizations developing foundational agent communication standards, including Google (A2A), Anthropic (MCP), IBM (ACP), and the broader AI research community.

This document was created as part of the OWASP Gen AI Security Project - Agentic Security Initiative (ASI).

Authors’ Addresses

Ken Huang
DistributedApps.ai
Email: ken.huang@distributedapps.ai

Vineeth Sai Narajala
Amazon Web Services
Email: abcvineeth.sai@gmail.com

Idan Habler
Intuit

Email: idan_habler@intuit.com

Akram Sheriff
Cisco Systems
Email: isheriff@cisco.com