OWASP | GenAI SECURITY PROJECT
TOP 10 FOR LLM AND GENERATIVE AI

Agent Name Service (ANS) for Secure AI Agent Discovery

Towards Secure and Interoperable Agentic AI: DNS for AI Agents

Version 1.0
May 13, 2025

The information provided in this document does not, and is not intended to, constitute legal advice. All information is for general informational purposes only. This document contains links to other third-party websites. Such links are only for convenience and OWASP does not recommend or endorse the contents of the third-party sites.

Table of Content

Abstract 4

1. Introduction 5

2. Related Work 6

3. Agent Registry Architecture 7

3.1 Agent Registration and Renewal 10

3.2 PKI Integration 11

3.3 ANS Protocol Notation 12

3.4 Protocol-Agnostic Communication Schema 14

3.5 ANS Naming Structure and Resolution 16

3.6 ANS Challenges and Governance 22

3.7 Agent Identity 23

4. Request/Response Schema for ANS Name Resolution 28

5. Protocol Adapter Layer 31

5.1 A2A Protocol Adapter 31

5.2 MCP Adapter 32

5.3 ACP Adapter 32
5.4 Extension Points 32
5.5 Cross-Protocol Interoperability Limits 32
5.6 Protocol Adapter API Definition 33
6. Security Analysis and Considerations 35
6.1 MAESTRO-Based Threat Analysis 35
6.2 Additional Security Controls and Considerations 37
7. Implementation Considerations 39
8. Future Considerations 42
Acknowledgements 44
OWASP Top 10 for LLM Project Sponsors 45
Project Supporters 46
References 48
Appendix A: Complete Request/Response Schemas 49
Appendix B: Glossary of Terms - Agent Name Service (ANS) 50

Abstract

The proliferation of AI agents requires robust mechanisms for secure discovery. This paper 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. We implement structured communication using JSON Schema and conduct a comprehensive threat analysis of our proposal.

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.

Keywords: *Agent Name Service (ANS), Agentic AI, Service Discovery, Public Key Infrastructure (PKI), Interoperability, Secure DNS, Formal Methods, Multi-Agent Systems (MAS)*

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) [RFC 1035, 1987], 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) [RFC 6763, 2013] 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** [Surapaneni et al., 2025]: Developed by Google, providing a standardized protocol for inter-agent communication, aiming to bridge different agent frameworks.
- **Model Context Protocol (MCP)** [Anthropic, 2024; MCP Specification, 2025]: Focused on simplifying the integration of AI models with external tools and data sources.
- **Agent Communication Protocol (ACP)** [IBM Research, 2025]: Designed to standardize how agents communicate, enabling automation, collaboration, UI integration, and developer tooling, evolving from initial MCP concepts.

This paper 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.

2. Related Work

Traditional service discovery, like DNS [RFC 1035, 1987], provides essential name-to-address resolution but lacks the semantic understanding and security features needed for agentic AI. DNS-SD [RFC 6763, 2013] 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, 2002]. 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** [Surapaneni et al., 2025] focuses on bridging agent ecosystems.
- **MCP** [Anthropic, 2024; MCP Specification, 2025] emphasizes context and integration of tools/data for AI models.
- **ACP** [IBM Research, 2025] 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 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 [RFC 5280, 2008] 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), and manages the agent lifecycle (registration, renewal, revocation). It enforces registry policies and validates the legal entity of the **Requesting Agent**.
- **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 for registry interactions (discovery, registration, etc.), incorporating PKI data and allowing protocol-specific extensions (details in Section 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 how they interact. Imagine an Agent trying to find another agent. It starts by contacting the ANS Service, which acts like a central directory. The ANS Service relies on the Agent Registry, a specialized database containing information about

all registered agents, including their capabilities, security policies, and contact details. The Agent Registry also interacts with a Certificate Authority (CA), which issues and manages digital certificates to verify agent identities, and a Registration Authority (RA), which validates new agents joining the system. To support different communication protocols, the ANS Service uses a Protocol Adapter Layer, which translates requests and responses into the appropriate format. All communication with ANS services is structured with JSON schemas. Together, these components form the Agent Registry Infrastructure that allows secure and reliable agent discovery.

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. Critically, 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"
}
```

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.

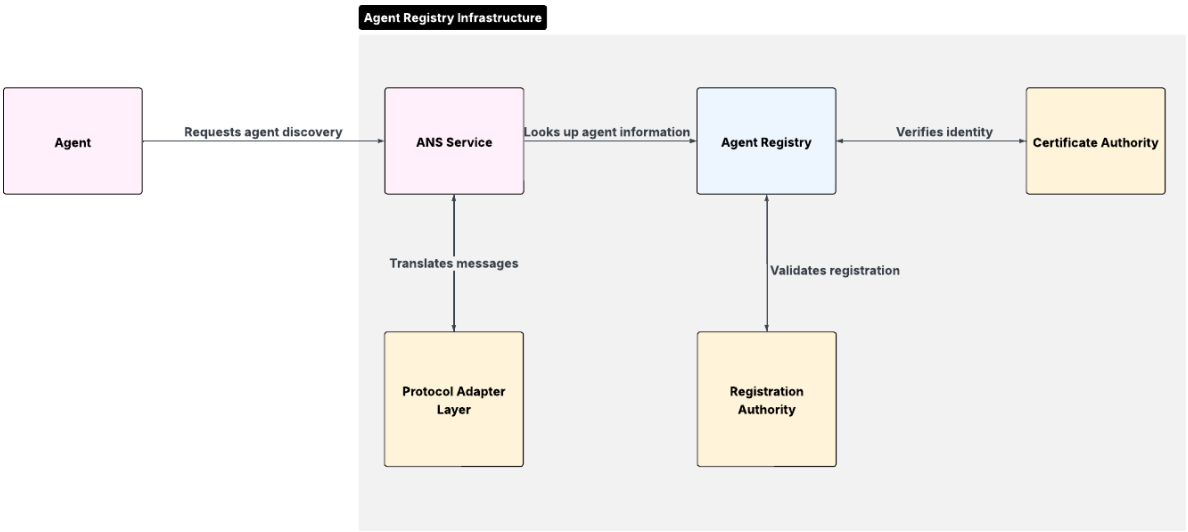

Figure 1 - ANS Architecture

Table 1 outlines how Google A2A Anthropic MCP, and IBM's ACP validate agents and enforce security within the ANS. Each protocol uses distinct adapter implementations and validation mechanisms—such as ZKPs (Zero-Knowledge Proofs), tool schema checks, and role-based controls—to ensure trusted identity, capability verification, and secure agent interactions. Zero-Knowledge Proofs (ZKPs) are cryptographic protocols that allow one party (the prover) to convince another party (the verifier) that a statement is true without revealing any information beyond the validity of the statement itself. These protocols maintain three

essential properties: completeness (a valid proof always convinces an honest verifier), soundness (a false statement cannot be proven true except with negligible probability), and zero-knowledge (the verifier learns nothing beyond the statement's validity). ZKPs can be used for capability attestation by enabling agents to prove they possess certain abilities or skills without exposing the underlying data..

PROTOCOL	ADAPTER IMPLEMENTATION	VALIDATION MECHANISM	SECURITY FEATURES
Google Agent (Agent 2 Agent)	Native Implementation with Google SDK	Agent ID card Integrity verification.	Capability attestation with ZKP
Anthropic MCP	Anthropic compliant adapter with extension validation	Tool Identity / Tool Schema verification	Resource access control
ACP(Agent communication Protocol)	IBM ACP Reference Implementation	Role-based Agent Identify and capability enforcement.	Delegation validation

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) 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).

Figure 2 outlines the steps an agent takes to register with the Agent Name Service (ANS). The process begins when an Agent sends a Registration Request to the Registration Authority (RA). This request includes the agent's metadata (name, capabilities, etc.) and a Certificate Signing Request (CSR). The RA then Validates the agent's identity and information. If the validation Fails, the Registration is Rejected. If the validation is Successful, the RA requests a Certificate from the Certificate Authority (CA). The CA issues the certificate and sends it back to the RA.. Finally, the RA stores the agent's information and certificate in the Agent Registry and sends a Confirmation back to the Agent, completing the registration process.

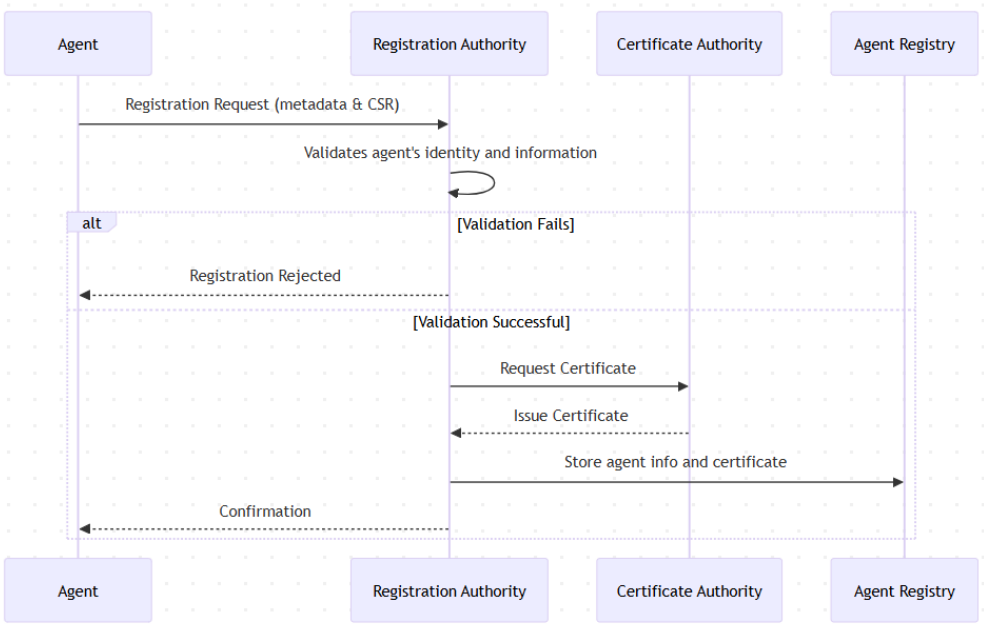

Figure 2: Agent Registration Process

3.2 PKI Integration

PKI [RFC 5280, 2008] provides the foundation for trust. Each registered agent possesses a unique PKI key pair and a corresponding digital certificate issued by the CA via the RA.

- **Identity Verification:** The certificate binds the agent's public key to its verified identity (e.g., its ANSName, organizational affiliation). Other agents can verify signatures made with the private key using the public key in the certificate, ensuring authenticity and integrity.
- **Trust Chain:** Certificates are validated against the trusted CA, establishing a chain of trust.
- **Lifecycle Management:** Certificate validity is tied to the registration/renewal cycle. Revoked certificates are handled using Certificate Revocation Lists (CRLs) or the Online Certificate Status Protocol (OCSP) [RFC 6960, 2013].
- **Simplification:** While PKI management can be complex, the RA/CA interaction within the registry framework aims to streamline certificate issuance and renewal for agent developers compared to manual processes.

3.3 ANS Protocol Notation

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

3.3.1 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

3.3.2 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, ...} // Enumerated set of protocols
- AgentID: String
- agentCapability: String
- Provider: String
- Version: String (Semantic Versioning)
- Extension: String
- Cert: X.509 Certificate
- Sig: Digital Signature
- ANSName: String // Formatted string as defined below
- Endpoint: String // Network address, service binding

3.3.3 Verification Rules

Certificate Chain Verification

```
VerifyCertChain (Cert, TrustedCA) -> Boolean:
  1. Get the certificate authority (CA) 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 == TrustedCA, Return True
  5. Else, recursively check CA Cert against TrustedCA
  6. If no trusted CA is found in the chain, Return False
```

Digital Signature Verification

```
VerifySignature (Data, Signature, PublicKey) -> Boolean:
  1. Use PublicKey to decrypt the Signature
  2. Hash Data using agreed upon function (e.g., SHA-256)
  3. Compare the decrypted signature to the Data hash
  4. If signatures are valid return True, otherwise return False
```

3.4 Protocol-Agnostic Communication Schema

We define JSON Schema [RFC 7159, 2014] documents for registry interactions (discovery requests/responses, registration/renewal requests/responses). This ensures structured, validatable communication.

An example AgentRegistrationRequest schema:

```
{
  "$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-
```

```
]*))*))?(?:\\+([0-9a-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 secure vault reference instead)",
          "readOnly": false
        }
      },
      "required": ["subject", "issuer", "pem"]
    },
    "protocolExtensions": {
      "type": "object",
      "description": "Protocol-specific data"
    }
  },
  "required": ["protocol", "agentID", "agentCapability", "provider", "version",
"certificate"]
}
```

- **Core Fields:** Include common elements like agent communication protocol types such as a2a, mcp, and 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. (See Section 4 for more details on the schema structure).

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:

```
ANSName = Protocol "://" AgentID "." agentCapability "." Provider ".v"
Version "." Extension
```

Where:

- Protocol ∈ {a2a, mcp, acp, ...}
- AgentID, agentCapability, Provider, Version, Extension are strings.

Constraints:

- Version MUST adhere to Semantic Versioning standards.
- AgentID, agentCapability, Provider SHOULD be registered with a governance authority (similar to ICANN).
- Extension SHOULD be used for deployment-specific or provider-defined metadata, not for core identity. In the actual implementation, a registry of reserved tokens can be used to enhance security.

Example:

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

3.5.2 Resolution

The resolution mechanism in ANS is engineered to map a fully qualified ANSName to an actionable reference, such as a network address, service binding, or 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 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 incorporate optional agentCapability filters to refine the search.

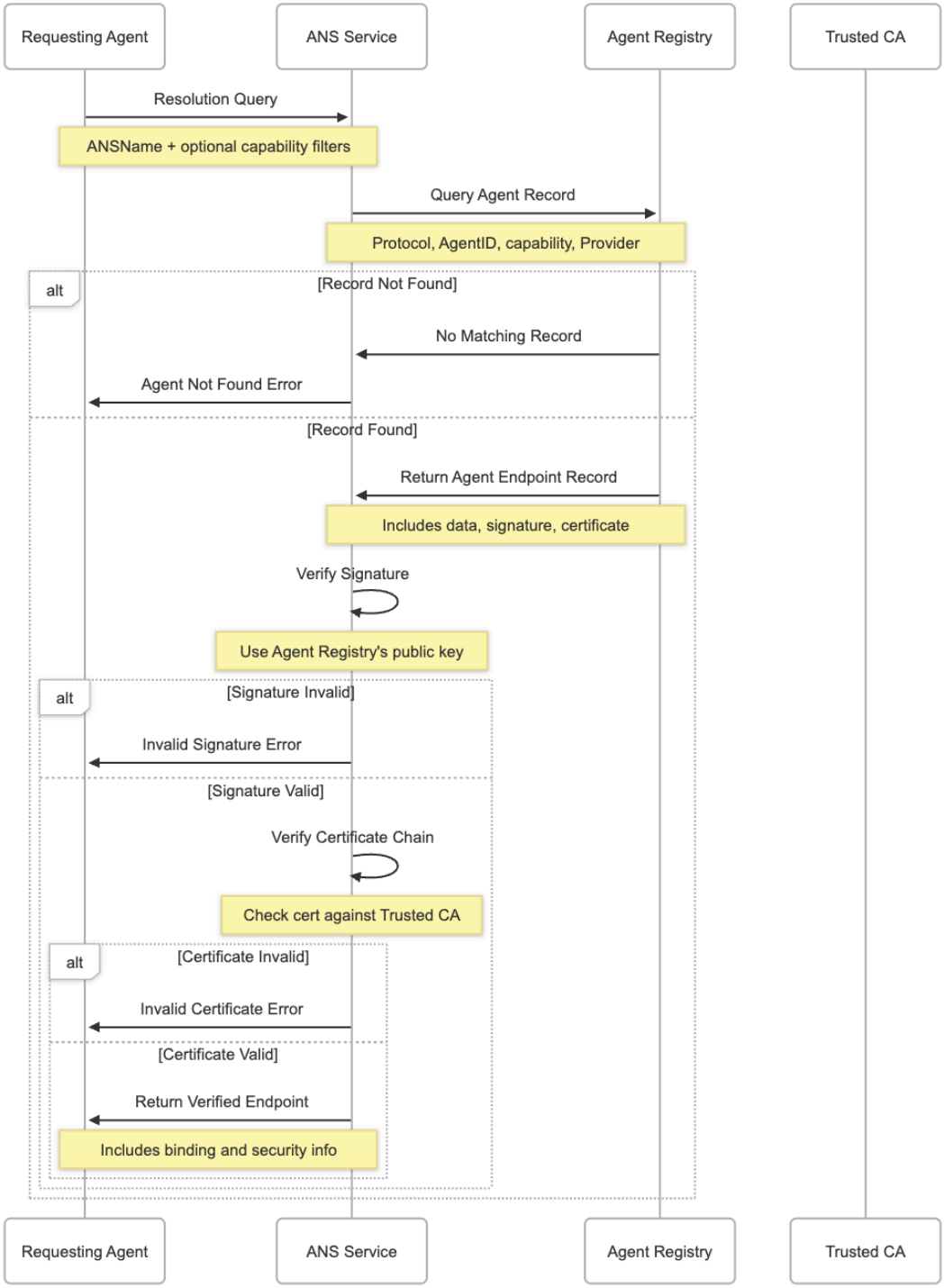

Fig. 3. Agent Resolution Process.

Figure 3 depicts how an agent finds another agent using ANS. The process begins when an Agent sends a Resolution Query, containing the ANSName of the agent it wants to find, to the ANS Service. The ANS then Queries the Agent Registry for the corresponding agent record. If the Record is Not Found, the ANS returns an Agent Not Found Error. If the Record is Found, the ANS Verifies the returned Endpoint (which contains address and security information), including verifying target Agent's signature and certificate. If the Verification Fails, an Invalid Endpoint Error is returned. If the Verification is Successful, the ANS returns the Endpoint to the Agent, enabling it to connect to the desired agent securely.

3.5.3 Formal Resolution Algorithm

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

```
Resolve(ANSName, RequestedVersionRange):
  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:
       Match = VersionNegotiation(Matches, RequestedVersionRange)
       if Match ==  ERROR("Incompatible Version")
             Return ERROR("Incompatible Version")
  5. EndpointRecord = GetAgentEndpointRecord(Match.AgentID)
  6. Valid = VerifyAgentEndpointRecord(EndpointRecord, TrustedCA)
  7. If valid == False
       Return ERROR ("Invalid Endpoint")
  8. Return Endpoint

GetAgentEndpointRecord () -> EndpointRecord
```

```
  // EndpointRecord: {data, signature, Cert}
  // Agent Registry implements GetAgentEndpointRecord to get records from
the database.
  // GetAgentEndpointRecord enforces authentication and authorization
with Agent Registry ACL.

VerifyAgentEndpointRecord (EndpointRecord, TrustedCA) -> Boolean:
  1. signatureValid = VerifySignature(EndpointRecord.data,
EndpointRecord.signature, AgentRegistry.PublicKey)
  2. VerifySignature (Data, Signature, PublicKey) -> Boolean:
       1. Use PublicKey to decrypt the Signature
       2. Hash Data using agreed upon function
       3. Compare the decrypted signature to the Data hash
       4. If signatures are valid return True, otherwise return False
     // Verifying Signature is implemented by each language standard
library, for example: java.security.Signature
  3. If signature is invalid return ERROR ("Invalid Signature")
  4. certChainValid = VerifyCertChain(EndpointRecord.cert, TrustedCA)
  5. VerifyCertChain (Cert, TrustedCA) -> Boolean:
       1. Get the certificate authority (CA) that signed the Cert
       2. Check for Certificate Revocation status of Cert.
       3. If Revoked Return False
       4. If CA == TrustedCA, Return True
       5. Else, recursively check CA Cert against TrustedCA
       6. If no trusted CA is found in the chain, Return False
     // Certificate Chain validation, and Certificate Revocation Check
are implemented via Library in standard language , for example Java:
java.security.cert.CertPathValidator

VersionNegotiation(Matches, RequestedVersionRange):
  1. Sort Matches by Version (highest to lowest Semantic Version)
  2. For each Match in Matches:
  3.   If RequestedVersionRange == "*" OR
IsVersionCompatible(Match.Version, RequestedVersionRange):
  4.     Return Match
  5.   End If
  6. End For
  7. Return ERROR("Incompatible Version")
```

```
IsVersionCompatible(AgentVersion, RequestedVersionRange) -> Boolean:
  // (Implement Semantic Version Range Compatibility Check here
  // Using existing library, for example: https://github.com/npm/node-semver
  // 1. Attempt to parse requestedVersionRange as a SemVer range.
  // 2. If parsing fails, treat requestedVersionRange as a specific SemVer version.
  // 3. Check if agentVersion is satisfied by the requestedVersionRange.
  Return SemVer.satisfies(AgentVersion, RequestedVersionRange)
```

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

1. **Trust Anchor:** The trust anchor for ANS is the Agent Registry's Certificate Authority (CA). The Agent Registry's certificate is a public key certificate that is used to verify the digital signatures of the Agent Registry's responses. The Agent Registry's certificate must be trusted by all agents that use ANS.

2. **Digital Signatures:** Digital signatures are used to ensure the integrity and authenticity of Agent Registry's responses. The Agent Registry's responses are digitally signed using the Agent Registry's private key. Clients, upon receiving a response, verify the signature using the corresponding public key, confirming the integrity and authenticity of the data.

3. **DNSSEC-like Security:** Consider the implementation of the Domain Name System Security Extensions (DNSSEC)-like mechanisms to validate the chain of trust. DNSSEC can increase the risk and amplify the effects of denial of service attacks on the infrastructure. DNSSEC also increases the

number of DNS query responses because of the crypto fields that are used to verify records properly. This means that high-volume responses enable attackers with greater attack volume against a zone than they could if DNSSEC were not in place.Therefore, a careful evaluation of the threat model and the potential for amplification attacks is crucial before implementing DNSSEC-like security measures. Mitigation strategies such as rate limiting, traffic filtering, and anycast deployment should be considered to protect the Agent Registry infrastructure from potential DoS attacks.

4. **Certificate Revocation:** Implement a robust mechanism for certificate revocation. If the Agent Registry's private key is compromised, the corresponding certificate must be revoked immediately to prevent attackers from using the compromised key to sign malicious responses. Use standard certificate revocation methods such as Certificate Revocation Lists (CRLs) or the Online Certificate Status Protocol (OCSP).

5. **Threat Modeling:** Perform ongoing threat modeling to identify potential vulnerabilities in the secure resolution mechanism. This will help to proactively address security concerns and ensure the ongoing security of ANS.

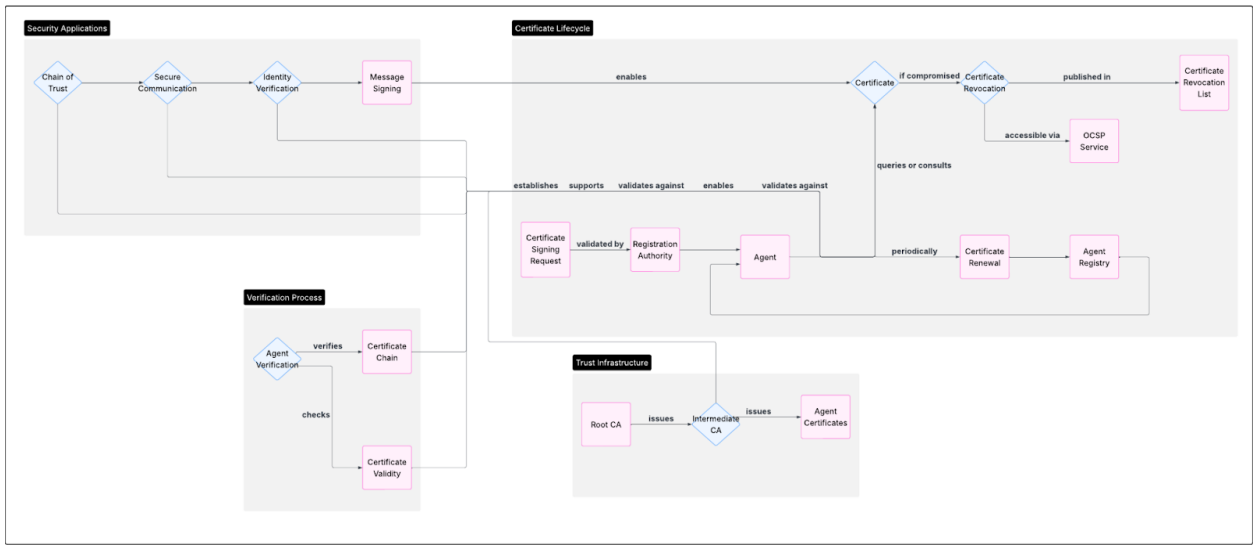

Fig. 4. Full end-to-end flow (Registration, Resolution, Interaction).

3.6 ANS Challenges and Governance

Deploying ANS involves addressing key challenges:

- **Naming Collisions/Squatting:** Ensuring uniqueness requires a managed registration process for names, particularly <AgentID>, <agentCapability>, and <Provider> segments. A governance model,

potentially similar to ICANN for DNS, might be needed to manage top-level agentCapabilities and provider identifiers.

- **Scalability:** Supporting potentially billions of agents requires scalable registry storage (e.g., distributed databases, NoSQL) and efficient resolution mechanisms (e.g., distributed hash tables (DHTs), caching layers, geographically distributed resolution points).

  Governance: Establishing policies for name allocation, dispute resolution, operational practices, and managing the trust infrastructure (CAs, RAs) is crucial for long-term stability and trustworthiness.

  .

3.7 Agent Identity

Agent identity within the ANS framework can include the following:

- **Cryptographic Identity:** The agent's PKI certificate provides a verifiable, CA-signed cryptographic identity.
- **Logical Identity:** The ANSName provides a human-readable, structured identifier conveying agentCapability, provider, etc.
- **Protocol-Specific Identity:** Agents may have identities within their native protocols (A2A Agent Card ID, MCP tool identifiers, ACP agent URIs), stored within the protocolExtensions.
- **Verifiable Claims:** The registry could support attaching digitally signed attestations (e.g., compliance certifications, capability endorsements) to agent profiles.
- **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.

```
{
    "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
        }
    }
}
```

Agent Identity Module:

The Agent Identity module implements resource access control through capability-based security:

```
{
    "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"
            }
        }
```

4. Request/Response Schema for ANS Name Resolution

The following is the core JSON Schema that defines the structure for Agent Capability requests and responses.

AgentCapabilityRequest Schema:

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "AgentCapabilityRequest",
    "description": "Schema for Agent agentCapability Request",
    "type": "object",
    "properties": {
        "requestType": {
            "type": "string",
            "enum": [
                "resolve"
            ],
            "description": "Type of request"
        },
        "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-]*))*))?(?:\\+([0-9a-zA-Z-]+(?:\\.[0-9a-zA-Z-]+)*))?$",
            "description": "Semantic Versioning format"
        },
        "extension": {
            "type": "string",
            "description": "Extension Metadata"
        }
    },
    "required": [
        "requestType",
        "protocol",
        "agentID",
        "agentCapability",
        "provider",
        "version"
    ]
```

AgentCapabilityResponse Schema:

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "title": "AgentCapabilityResponse",
    "description": "Schema for Agent agentCapability Response",
    "type": "object",
    "properties": {
        "Endpoint": {
            "type": "string",
            "description": "Agent address (e.g.,
a2a://translatorBot.DocumentTranslation.exampleCorp.v1.2.3.secure)"
        },
        "signature": {
            "type": "string",
            "description": "signature"
        },
```

```
        "cert": {
            "type": "string",
            "description": "PEM-encoded Certificate (strongly recommended to
use a secure vault reference instead)",
            "readOnly": false
        }
    },
    "required": [
        "Endpoint",
        "signature",
        "cert"
    ]
```

Key points regarding schemas:

- Use a JSON Schema validator library for enforcement.
- Pay attention to required fields.
- Validate all incoming/outgoing messages against this schema.
- Handle validation errors gracefully.
- Monitor evolving standards (A2A, MCP, ACP) and update schemas accordingly.

5. Protocol Adapter Layer

The Protocol Adapter Layer enables the registry to support diverse agent communication protocols without being tightly coupled to any single one. It acts as an intermediary between the registry's core logic/storage and the specific requirements of each protocol.

- **Modularity:** Adapters are implemented as distinct modules (e.g., plugins).
- **Functionality:** Each adapter understands how to:
    - Parse protocol-specific metadata (e.g., from an A2A Agent Card) and map relevant parts into the registry's internal representation (especially within protocolExtensions).
    - Extract information from the registry record to answer protocol-specific discovery queries.
    - Potentially handle protocol-specific aspects of registration or validation if needed.
- Each adapter is responsible to securely implement features defined by each protocol. Some protocols have build in security, such as signed messages.
- **Translation:** Adapters primarily focus on metadata translation for discovery and registration, not on real-time message translation between protocols during agent interaction. They help agents find each other and verify identity; subsequent communication typically uses the agents' shared native protocol.
- **Security aspect:**
    - All adapter implementations must use secure, updated libraries that are commonly used to implement the supported protocols.
    - All adapter implementations must follow the security guidance by each protocol.
    - Each adapter parses untrusted blobs; mandate memory-safe languages (Rust/Go) and formal test suites.

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.

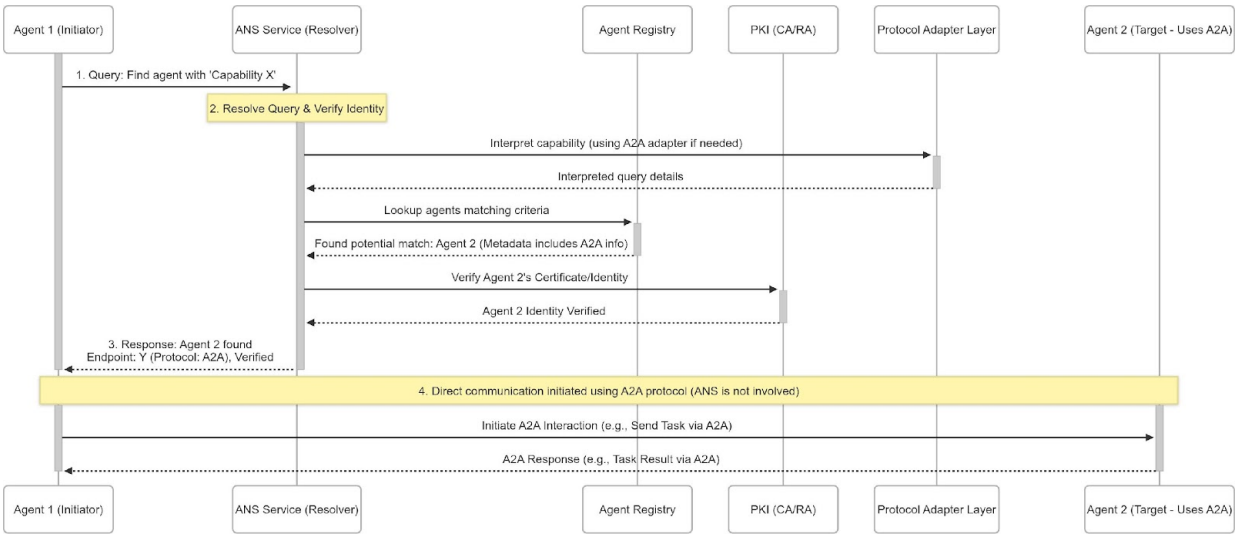

Fig. 5. ANS Resolution Sequence as a Pre-Communication Step.

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-specified metadata from the protocolExtensions and maps it to
an internal registry format.
  // Returns a map of key-value pairs representing the extracted data.
  Map<String, Object> parseMetadata(Object protocolExtensions);

  // Creates a protocol-specific discovery response based on the registry record.
  Object createDiscoveryResponse(Map<String, Object> registryRecord);

  // Handles protocol-specific validation during the registration process
  Boolean validateRegistration(AgentRegistrationRequest request);
}
```

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) **Ref section

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.

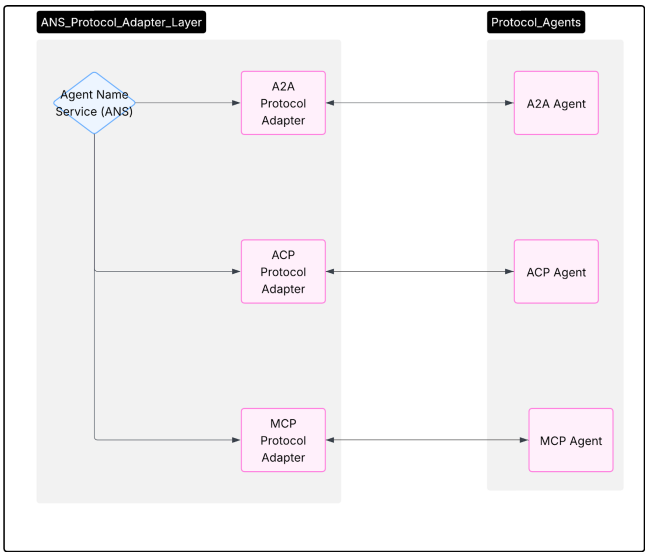

Fig. 6. Protocol Adapters connecting the core registry to different agent protocols (A2A, MCP, ACP).

6. Security Analysis and 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 (Huang, 2025) 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 3.2 (PKI Integration) and 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 3.1).
    - Secure, authenticated registry responses (Section 3.5.4 leveraging PKI 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 (3.5.3) enforces response integrity.
- **Formal Check:** PKI signing (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 8).
- **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 or OCSP mechanisms are essential.
- **Secure Key Storage:** Agent private keys must be protected (HSMs, secure enclaves, OS key stores). Compromise requires immediate revocation/re-issuance.
- **Registry Access Control:** Strict authentication/authorization for registry management (RA) and potentially querying.
- **RA Validation:** Rigorous process to prevent malicious registration (domain validation, organizational checks, manual review).

6.2.2 ANS-Specific Security Controls

- **Resolution Integrity:** Use DNSSEC-like mechanisms or signed responses to prevent ANS Manipulation/Spoofing.
- **DoS Mitigation:** Standard defenses (rate limiting, firewalls, anycast) for resolution Endpoints.
- **CA Security:** Standard CA best practices (offline roots, audits, HSMs) are mandatory. Consider using established, trusted CAs.
- **Sybil Attack Resistance:** Registration processes should make creating fake identities difficult/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 can be implemented using various patterns:

- **Centralized:** Simple management, but single point of failure/bottleneck. Suitable for smaller/private deployments.
- **Distributed:** Offers resilience and scalability. Options:
    - *Distributed Hash Table (DHT):* Scalable P2P lookup, complex state management.
    - *Distributed Database:* Technologies like Cassandra, CockroachDB (strong/eventual consistency). Requires coordination (Paxos, Raft).
    - *Blockchain/DLT:* Smart contracts as Registry Agent, transactions validated/recorded on-chain.
- **Federated:** Independent registries interoperate (organizational/geographic boundaries). Requires inter-registry protocols and trust. To enable seamless cross-registry discovery and interaction, federated registries require standardized mechanisms for verifying agent identities and resolving ANSNames across domains. This necessitates agreed-upon trust anchors (e.g., mutually trusted Certificate Authorities), standardized metadata formats for agent descriptions, and well-defined inter-registry communication protocols for querying and exchanging agent information. Without these elements, a federated system risks becoming a collection of isolated silos, hindering interoperability.
- **Hybrid:** Combines elements (e.g., centralized RA/CA, distributed/replicated read nodes). Caching layers (Redis, Memcached) improve performance. Requires careful consistency management.

The Agent Registry can be implemented using various patterns. The choice depends on scale, trust model, performance requirements, administrative overhead, and cost. The Protocol Adapter Layer suits a plugin architecture. Below is a decision matrix to help guide the selection:

Decision Matrix:

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

Implementation Patterns:

- **Centralized:** Simple management, but a single point of failure and a potential bottleneck. Suitable for smaller or private deployments.
- **Distributed (Cassandra, CockroachDB):** Offers resilience and scalability. Technologies like Cassandra provide tunable consistency (eventual). Requires coordination mechanisms.
- **Distributed (DHT):** Scalable P2P lookup, complex state management. Suitable for very large-scale deployments with relaxed consistency requirements.
- **Blockchain/DLT:** Smart contracts as Registry Agents, transactions validated and recorded on-chain. Offers high security and auditability but suffers from **high latency, limited scalability, and significant write amplification.** Write amplification refers to the fact that a single write operation to the blockchain can result in multiple write operations to the underlying storage, significantly increasing storage costs and reducing performance.
- **Federated:** Independent registries interoperate (organizational or geographic boundaries). Requires inter-registry protocols and trust management.

- **Hybrid:** Combines elements (e.g., centralized RA/CA, distributed/replicated read nodes). Caching layers (Redis, Memcached) improve performance. Requires careful consistency management.

Key Considerations:

- **Consistency vs. Latency:** Strong consistency (e.g., in a centralized system) means that all reads see the most recent write, but it can increase latency. Eventual consistency (e.g., in Cassandra or a DHT) allows for lower latency and higher scalability, but reads may not always reflect the most recent write.
- **Write Amplification (Blockchain):** Be aware of the significant write amplification in blockchain solutions. This can lead to high storage costs and performance bottlenecks.
- **Operational Cost:** Consider the operational cost of each pattern, including hardware, software, and personnel costs. Blockchain solutions, in particular, can be expensive to operate due to the need for specialized hardware and expertise.
- **Complexity:** Distributed systems, especially those based on DHTs or blockchains, are more complex to design, implement, and maintain than centralized systems.

8. Future Considerations

This proposal lays the groundwork; future work should explore:

- **Prototype Implementation:** Build and evaluate a working ANS prototype (See GitHub: https://github.com/kenhuangus/dns-for-agents/).
- Explore an innovative bootstrap model for distributed trust infrastructure, focusing on a foundation-led root governance approach with delegated sub-spaces (similar to Cloud Native Computing Foundation certificate transparency roots), alongside a comprehensive funding and fee schedule that creates a sustainable, flexible ecosystem for trust delegation and verification across different organizational scales and technological domains.
- **Performance & Scalability:** Benchmark resolution latency, registration throughput, and scalability under load.
- **Advanced Cryptography:** Investigate privacy-preserving techniques (e.g., zero-knowledge proofs) for capability advertisement/selective disclosure.
- **Formal Verification:** Mathematically model and verify security properties of registration/resolution protocols.
- **Detailed Governance Model:** Develop comprehensive policies for naming, CA/RA operations, disputes, and trust framework evolution.
- **Platform Integration:** Demonstrate integration with agent frameworks (LangChain, AutoGen, CrewAI) and cloud platforms.
- **Semantic Interoperability:** Research mechanisms leveraging ANS metadata for deeper cross-protocol communication (standardized capability ontologies, translation gateways).
- **Reputation Systems:** Integrate reputation scores/endorsements to enhance trust.
- **ANS** agentCapability **Negotiation and Binding Protocol:** Develop standardized protocols for agents to negotiate capabilities post-resolution (capability mapping taxonomies, binding, and quality of service).
- Draft a governance white-paper detailing name allocation, fee model, dispute arbitration, and root CA stewardship.

9. Conclusion

ANS offers a foundational infrastructure for a more secure, trustworthy, and interconnected agentic AI ecosystem. Its broad impact includes:

- **Enhancing Interoperability:** Protocol-agnostic directory facilitates seamless communication between diverse agents.
- **Boosting Trust and Security:** PKI integration enhances trust for sensitive domains (finance, healthcare, cybersecurity).
- **Accelerating Innovation:** Lowers barriers by providing common discovery/identity, letting developers focus on agent solutions.
- **Facilitating Autonomous Systems:** Critical enabler for systems needing dynamic, secure discovery and interaction (autonomous vehicles, smart cities).
- **Powering Secure AI Marketplaces:** Foundation for marketplaces with verifiable agent identities/capabilities.

The modular Protocol Adapter Layer ensures extensibility. ANS provides foundational trust and discovery, enabling agents on different standards to find and securely contact each other. While challenges remain (governance, scalability, semantic interoperability), ANS is a necessary step towards a robust agentic AI ecosystem.

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.

Contributors

Ken Huang, DistributedApps.ai, CSA
Idan Habler, PhD, Intuit
Vineeth Sai Narajala, AWS
Akram Sheriff, Cisco Systems

ASI Review Board

Alejandro Saucedo - Chair of ML Security Project at Linux Foundation, UN AI Expert, AI Expert for Tech Policy, European Commission
Apostol Vassilev - Adversarial *AI Lead, NIST*
Chris Hughes - CEO, Aquia
Hyrum Anderson - CTO, Robust Intelligence
Steve Wilson - OWASP Top 10 for LLM Applications and Generative AI Project Lead and Chief Product Officer, Exabeam
Scott Clinton - OWASP Top 10 for LLM Applications and Generative AI Project Co-Lead
Vasilios Mavroudis- Principal Research Scientist and Theme Lead, the Alan Turing Institute
Josh Collyer, Principal Security Researcher, Theme Lead
Egor Pushkin, Chief Architect, Data and AI at Oracle Cloud

Coordinated and reviewed by

- Ron F. Del Rosario (ASI co-lead), SAP
- John Sotiropoulos (ASI co-lead), Kainos

OWASP GenAI Security Project Sponsors

We appreciate our Project Sponsors, funding contributions to help support the objectives of the project and help to cover operational and outreach costs augmenting the resources provided by the OWASP.org foundation. The OWASP GenAI Security Project continues to maintain a vendor neutral and unbiased approach. Sponsors do not receive special governance considerations as part of their support.

Sponsors do receive recognition for their contributions in our materials and web properties. All materials the project generates are community developed, driven and released under open source and creative commons licenses. For more information on becoming a sponsor, visit the Sponsorship Section on our Website to learn more about helping to sustain the project through sponsorship.

Project Sponsors:

GenAI Security Project — Gold Project Sponsor — Top 10 for LLM and Generative AI

ByteDance · HiddenLayer · Mend.io · snyk · paloalto Networks · Trend Micro

GenAI Security Project — Silver Project Sponsor — Top 10 for LLM and Generative AI

Lasso Security · pangea · PromptArmor · Acuvity · securiti · Protecto · Cobalt

Lakera · splx AI · ActiveFence · Synack · Prompt: · TrojAI

Sponsor list, as of publication date. Find the full sponsor list here.

Project Supporters

Project supporters lend their resources and expertise to support the goals of the project.

Accenture
AddValueMachine Inc
Aeye Security Lab Inc.
AI informatics GmbH
AI Village
aigos
Aon
Aqua Security
Astra Security
AVID
AWARE7 GmbH
AWS
BBVA
Bearer
BeDisruptive
Bit79
Blue Yonder
BroadBand Security, Inc.
BuddoBot
Bugcrowd
Cadea
Check Point
Cisco
Cloud Security Podcast
Cloudflare
Cloudsec.ai
Coalfire
Cobalt
Cohere
Comcast
Complex Technologies
Credal.ai
Databook
DistributedApps.ai
DreadNode
DSI
EPAM
Exabeam
EY Italy
F5
FedEx
Forescout
GE HealthCare
Giskard
GitHub
Google
GuidePoint Security
HackerOne
HADESS
IBM
iFood
IriusRisk
IronCore Labs
IT University Copenhagen
Kainos
KLAVAN
Klavan Security Group
KPMG Germany FS
Kudelski Security
Lakera
Lasso Security
Layerup
Legato
Linkfire
LLM Guard
LOGIC PLUS
MaibornWolff
Mend.io
Microsoft
Modus Create
Nexus
Nightfall AI
Nordic Venture Family
Normalyze
NuBinary
Palo Alto Networks
Palosade
Praetorian
Preamble
Precize
Prompt Security
PromptArmor
Pynt
Quiq
Red Hat
RHITE
SAFE Security
Salesforce
SAP
Securiti
See-Docs & Thenavigo
ServiceTitan
SHI
Smiling Prophet
Snyk
Sourcetoad
Sprinklr
stackArmor
Tietoevry
Trellix
Trustwave SpiderLabs
U Washington
University of Illinois
VE3
WhyLabs
Yahoo
Zenity

Sponsor list, as of publication date. Find the full sponsor list here.

References

P. Mockapetris, "Domain names - implementation and specification," RFC Editor, RFC 1035, Nov. 1987. [Online]. Available: https://www.rfc-editor.org/info/rfc1035

S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," RFC Editor, RFC 6763, Feb. 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc6763

R. Surapaneni, M. Jha, M. Vakoc, and T. Segal, "Announcing the Agent2Agent Protocol (A2A)," Google for Developers Blog, Apr. 2025, accessed: 2025-04-27. [Online]. Available: https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/

Agent2Agent Protocol Specification Authors, "Agent2Agent (A2A) Protocol Specification," https://github.com/google/A2A, 2025, accessed: 2025-04-27.

Anthropic, "Model context protocol (MCP)," https://www.anthropic.com/news/model-context-protocol, 2024, accessed: 2025-04-27.

Model Context Protocol Specification Authors, "Model Context Protocol (MCP) Specification," https://modelcontextprotocol.io/specification/2025-03-26, Mar. 2025, accessed: 2025-04-27.

P. Schmid, "MCP Introduction," https://www.philschmid.de/mcp-introduction, 2025, accessed: 2025-04-27.

IBM Research, "(placeholder for anticipated ibm acp publication/specification link)," 2025, anticipated Publication

Foundation for Intelligent Physical Agents (FIPA), "FIPA Agent Communication Language Specifications," FIPA, Tech. Rep., 2002. [Online]. Available: http://www.fipa.org/repository/aclspecs.html

D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile," RFC Editor, RFC 5280, May 2008. [Online]. Available: https://www.rfc-editor.org/info/rfc5280

E. T. Bray, "The javascript object notation (JSON) data interchange format," RFC Editor, RFC 7159, Mar. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7159

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

I. Habler, K. Huang, V. S. Narajala, and P. Kulkarni, "Building a secure agentic AI application leveraging A2A protocol," *arXiv preprint arXiv.2504.16902*, 2025. [Online]. Available: https://www.arxiv.org/abs/2504.16902

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

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

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

Appendix A: Complete Request/Response Schemas

The detailed JSON Schema documents for registry interactions are maintained externally. Please ensure these schemas are well-commented and validated in any 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: https://github.com/kenhuangus/dns-for-agents/blob/main/agent_capability_request.schema.json
- AgentCapabilityResponse Schema: https://github.com/kenhuangus/dns-for-agents/blob/main/agent_capability_response.schema.json

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

- **A2A (Agent2Agent Protocol):** A communication protocol developed by Google Cloud 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 identifier for agents in the ANS ecosystem, following the format: Protocol://AgentID.agentCapability.Provider.vVersion.Extension.
- **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 expiration date, typically due to a key compromise or when an agent is deregistered.
- **Certificate Revocation List (CRL):** A list of digital certificates that have been revoked before their scheduled expiration date and should no longer be trusted.
- **CRL (Certificate Revocation List):** A mechanism used to check if a certificate has been revoked and is no longer valid.

- **CSR (Certificate Signing Request):** A message sent by an agent to a Certificate Authority to apply for a digital certificate.
- **Digital Signature:** A mathematical scheme for verifying the authenticity and integrity of digital messages or documents.
- **Distributed Hash Table (DHT):** A decentralized distributed system that provides a lookup service similar to a hash table, used as one possible implementation strategy for the Agent Registry.
- **DNS (Domain Name System):** The traditional system that translates human-readable domain names to IP addresses, which serves as a partial model for ANS but lacks the capability-oriented nature of ANS.
- **DNS-SD (DNS-Based Service Discovery):** An extension to DNS that enables automatic discovery of services available on a local network.
- Endpoint**:** A resolvable network address, service binding, or metadata document that allows agents to connect and communicate with each other.
- Extension**:** A field in the ANSName that holds deployment-specific or provider-defined metadata.
- **Interoperability:** The ability of different agent systems or protocols to exchange information and use that information effectively across platforms.
- **MAESTRO (7 Layers):** A threat modeling framework for agentic AI 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.
- **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 X.509 digital certificates as an alternative to CRLs.
- **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 representation and protocol-specific formats.

- protocolExtensions**:** A field within the ANS schema that acts as a container for protocol-specific data.
- Provider**:** An organization or entity that offers or maintains agents in the ANS ecosystem.
- **RA (Registration Authority):** An entity that verifies agent registration and renewal requests, interacts with the CA to issue certificates, and manages the agent lifecycle.
- **Registry Poisoning:** A security threat where an adversary attempts to inject malicious data into the Agent Registry.
- **Semantic Versioning:** A versioning scheme with a format of MAJOR.MINOR.PATCH used to indicate compatibility and changes in agent versions.
- **Service Discovery:** The process of automatically finding available services in a network.
- **Signature Verification:** The process of checking that a digital signature is valid and was created by the claimed signer.
- **Sybil Attack:** An attack where a malicious actor creates multiple fake identities to gain disproportionate influence in a network.
- **Trust Anchor:** The root of trust in a PKI system, typically a certificate authority whose certificate is implicitly trusted.
- **Version Negotiation:** The process where agents determine which protocol version to use when communicating, based on compatibility and preferences.
- VerifyCertChain**:** A function that checks the validity of a certificate by tracing its chain of trust back to a trusted certificate authority.
- VerifySignature**:** A function that validates a digital signature against a message and a public key to confirm authenticity and integrity.