

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 18 November 2026

E. Somoza
Independent
17 May 2026

Agent Trust Negotiation: Capability, Delegation, and Provenance Binding
for AI Agents
draft-somoza-atn-agent-trust-negotiation-00

Abstract

This document defines the Agent Trust Negotiation (ATN) protocol. ATN sits above existing DNS-based agent discovery mechanisms and answers questions that discovery alone cannot: what is the agent permitted to do, under whose authority, with what provenance, and how do two agents reach a verifiable working agreement.

ATN binds four artifacts to a DNS-discovered agent identity:

1. A Capability Manifest expressing what the agent is willing to do, under what conditions, and with what constraints.
2. A Delegation Chain proving authorized principal scope and revocation paths.
3. A Provenance Attestation proving build-time and runtime integrity.
4. A Session Receipt produced at handshake completion, recording the negotiated scope.

ATN defines the handshake state machine that consumes these artifacts to produce a mutually verified, scope-bounded agent session, and a session receipt suitable for transparency log inclusion.

ATN is transport-independent above the DNS layer and compatible with existing discovery proposals including AID [AID], DNS-AID [DNS-AID], and ANS v2 [ANSv2].

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 18 November 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	4
1.1. Scope	5
1.2. Non-Goals	6
1.3. Choice of Venue	6
2. Conventions and Definitions	7
2.1. Terminology	7
3. Architecture Overview	7
4. DNS Binding	8
4.1. Binding Tags for TXT-Based Discovery	8
4.1.1. Example (binding ATN to an AID-style record)	9
4.2. Binding ParamKeys for SVCB-Based Discovery	9
4.2.1. Example (binding ATN to a DNS-AID-style SVCB record)	9
4.3. ATN Index Document	9
5. Capability Manifest	10
5.1. Capability Object Structure	10
5.2. Schema Binding	12
5.3. Structure	12
5.4. Dimension Semantics	14
5.5. Key Properties	15

5.6. Refusal Semantics	15
6. Delegation Chain	15
6.1. Structure	15
6.2. Verification	16
7. Provenance Attestation	16
7.1. Structure	17
7.2. Properties	17
8. The Agent Trust Handshake (ATH)	17
8.1. State Machine	17
8.2. ATH_HELLO	18
8.3. ATH_OFFER	19
8.4. ATH_ACCEPT	20
8.5. ATH_RECEIPT	20
8.6. Handshake Limits	20
9. Capability Intersection Algebra	21
9.1. Identity Rule	21
9.2. Per-Dimension Rules	21
9.3. Refusal Override	22
9.4. Worked Example	22
10. Mutual Trust Mode	24
11. Version Negotiation and Algorithm Agility	24
11.1. Version Negotiation	24
11.2. Algorithm Agility	25
11.3. Post-Quantum Transition	25
12. Compatibility with Existing Discovery Protocols	26
12.1. With AID AID	26
12.2. With DNS-AID DNS-AID	26
12.3. With ANS v2 ANSv2	26
12.4. Multi-Discovery Tolerance	26
13. Relationship to Existing Identity, Authorization, and Attestation Frameworks	26
13.1. OAuth 2.0 and GNAP	27
13.2. SPIFFE Workload Identity	27
13.3. Mutual TLS	28
13.4. RATS Remote Attestation	28
13.5. SCITT Transparency	29
13.6. Composition Summary	29
13.7. When to Use What	30
14. Security Considerations	31
14.1. Threat Model	31
14.2. Artifact Authenticity	31
14.3. Enforcement Posture: Evidence, Not Prevention	31
14.4. Delegation Forgery	32
14.5. Session Receipt Tampering	33
14.6. Replay	33
14.7. Provenance Attestation Limits	33
15. Privacy Considerations	33
15.1. Negotiation Observability	33

15.2.	Witness Aggregation	33
15.3.	Minimum Disclosure	33
16.	Operational Considerations	33
16.1.	Session Establishment vs Per-Call Latency	34
16.2.	Latency Budget	34
16.3.	Optimization Techniques	36
16.4.	Session Lifetime	36
16.5.	Deployments Where ATN Is Appropriate	37
16.6.	Deployments Where ATN Is Not Appropriate	37
17.	IANA Considerations	37
17.1.	SVCB ParamKey Registrations	37
17.2.	ATN Capability Registry	38
17.3.	ATN Capability Dimension Registries	39
17.4.	ATN Refusal Category Registry	40
17.5.	Well-Known URI Registration	41
18.	References	41
18.1.	Normative References	41
18.2.	Informative References	42
Appendix A.	Complete Example Flow	43
A.1.	Step 1: Discovery	43
A.2.	Step 2: Index Document Fetch	43
A.3.	Step 3: Artifact Fetch and Verification	43
A.4.	Step 4: ATH_HELLO	44
A.5.	Step 5: ATH_OFFER	44
A.6.	Step 6: ATH_ACCEPT	45
A.7.	Step 7: ATH_RECEIPT	45
A.8.	Step 8: Session and Audit	46
Appendix B.	Design Rationale	46
B.1.	Why a Negotiation Layer	46
B.2.	Why Four Artifacts	46
B.3.	Why Refusals Are First-Class	47
B.4.	Why Intersection Algebra Is Specified	47
B.5.	Why SCITT for Receipts	47
Appendix C.	Implementation Status	47
C.1.	ATN Reference Prototype	47
Appendix D.	Acknowledgments	48
Author's Address	48

1. Introduction

Recent IETF work has converged on DNS-based discovery for AI agents. Multiple drafts now define how to locate an agent's endpoint and verify its cryptographic identity given a domain name. These efforts answer two questions: `_where` is the `agent_` and `_who` is the `agent_`.

They do not answer the questions that arise immediately after:

- * What is the agent permitted to do, and under what constraints?

- * Under whose authority is the agent operating?
- * What is the provenance of the agent's software, model, and configuration?
- * How do two agents reach a mutually verifiable working agreement before any task begins?
- * What evidence remains after the session ends, and how is it audited?

This document defines the Agent Trust Negotiation (ATN) protocol to address these gaps. ATN is the layer above discovery. It assumes the discovery layer has located an endpoint and authenticated an identity through one of the existing mechanisms. ATN then negotiates the trust relationship.

The design has four parts:

1. Four structured artifacts (Capability Manifest, Delegation Chain, Provenance Attestation, Session Receipt) that can be pointed to from any DNS-based discovery record.
2. A handshake protocol over HTTPS that exchanges and verifies the artifacts between two agents.
3. A capability intersection algorithm that yields the negotiated session scope.
4. A receipt format suitable for SCITT-style transparency log inclusion.

ATN deliberately does not introduce a new discovery mechanism. It builds on existing work and adds the negotiation layer that the agent ecosystem currently lacks.

1.1. Scope

In scope:

- * Format and semantics of the four ATN artifacts.
- * The Agent Trust Handshake (ATH) state machine.
- * Capability intersection algebra.
- * DNS bindings to point from a discovery record to ATN artifacts.

- * IANA registrations for ATN-specific identifiers.

Out of scope:

- * Discovery of the agent itself (use [AID], [DNS-AID], [ANSv2], or equivalent).
- * Endpoint authentication (delegated to the discovery layer).
- * Application protocol (delegated to MCP, A2A, OpenAPI, or equivalent).
- * In-organization service-to-service identity (see [SPIFFE]).
- * Reputation distribution.
- * Behavioral attestation.

1.2. Non-Goals

- * ATN does NOT define a new DNS record type.
- * ATN does NOT replace any existing discovery proposal.
- * ATN does NOT introduce a centralized trust authority.
- * ATN does NOT enforce policy. It produces verifiable evidence of declared and negotiated policy that downstream systems can audit.

1.3. Choice of Venue

ATN's choice of IETF as the venue follows from its dependencies. The discovery substrate is DNS [RFC4033] and SVCB [RFC9460], both IETF standards. The attestation architecture is RATS [RFC9334]. The transparency log is SCITT. The signature format is JWS [RFC7515]. The mutual authentication transport is TLS [RFC8446]. The authorization frameworks ATN composes with are OAuth [RFC6749] and GNAP [RFC9635].

ATN is the composition of IETF primitives into a peer-to-peer agent trust handshake. Standardizing the composition in any other venue would fork the primitive stack and create alignment problems. The natural home for the composition is the same venue that standardized the primitives.

2. Conventions and Definitions

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.

2.1. Terminology

Agent An automated software entity that initiates network interactions on behalf of, or with delegated authority from, a principal, without per-interaction human authorization.

Initiator The agent that opens the trust handshake.

Responder The agent that answers the trust handshake.

Capability Manifest A signed JSON [RFC8259] document declaring what an agent is willing to do, under what conditions, and with what constraints.

Delegation Chain A signed sequence of authorization statements binding the agent to a principal, with explicit scope, time bounds, and revocation pointers.

Provenance Attestation A signed document proving build-time and runtime characteristics of the agent.

Session Receipt A signed record produced at handshake completion, summarizing the negotiated scope and identities, suitable for transparency log inclusion.

Negotiated Scope The intersection of two agents' capability sets, time windows, authorization scopes, and operational constraints, as determined by the handshake.

3. Architecture Overview

ATN defines four roles:

1. **Agent Operator:** publishes the DNS discovery record and the ATN artifacts.
2. **Agent:** the runtime entity, in either Initiator or Responder mode at any moment.

3. Principal: the human, organization, or other agent that authorized the agent through the Delegation Chain.
4. Witness: an optional third party that can pre-verify ATN artifacts and publish endorsements (e.g., via SCITT [SCITT]).

ATN layers above discovery:

Application Layer (MCP, A2A, OpenAPI, ...)
Agent Trust Negotiation (this document)
Discovery and Identity (AID, DNS-AID, ANS v2, equivalents)
DNS, DNSSEC, TLS, HTTPS

The handshake order is:

1. Discovery: Initiator resolves the Responder's discovery record via DNS.
2. Artifact fetch: Initiator retrieves the four ATN artifacts referenced by the discovery record.
3. Handshake: Initiator opens the ATN handshake. Both parties exchange artifacts and verify signatures.
4. Negotiation: Both parties compute the capability intersection.
5. Receipt: Both parties produce a signed Session Receipt covering the negotiated scope.
6. Session: Application traffic proceeds within the negotiated scope.

4. DNS Binding

ATN does not define its own DNS records. Instead, it specifies how to point to ATN artifacts from existing discovery records.

4.1. Binding Tags for TXT-Based Discovery

The following tags MAY be added to TXT-based discovery records (compatible with the [AID] format and the format defined in companion documents):

Tag	Description
atn	Absolute HTTPS URI of an ATN Index Document.
atn-digest	Hex SHA-256 digest of the Index Document.

Table 1

The ATN Index Document is a JSON file pointing to the four artifacts. Indirection through an Index Document keeps the DNS record small.

4.1.1. Example (binding ATN to an AID-style record)

```
_agent.example.com. 3600 IN TXT ( "v=aid1; "
                                "u=https://agent.example.com/mcp; "
                                "p=mcp; "
                                "k=z7rW8rTq8o4mM6vVf7wlk3m4uQn9p2YxCABcDeFgHiJ; "
                                "atn=https://example.com/.well-known/atn/index.json; "
                                "atn-digest=a3b8c2d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9
                                b0c1d2e3f4" )
```

4.2. Binding ParamKeys for SVCB-Based Discovery

For SVCB-based discovery [RFC9460], the following ParamKeys are registered (see Section 17):

ParamKey	Format
atn	URI
atn-digest	hex-string

Table 2

4.2.1. Example (binding ATN to a DNS-AID-style SVCB record)

```
_agent.example.com. 3600 IN HTTPS 1 agent.example.com. (
  alpn="h2,h3"
  atn="https://example.com/.well-known/atn/index.json"
  atn-digest="a3b8c2d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4" )
```

4.3. ATN Index Document

The Index Document at the atn URI is a JSON object with the following structure:

```
{
  "v": "atn1",
  "agent_id": "2H4kP9aQ8vR3wXyZ1mNbVcLk-a3b8c2d4e5f6a7b8c9d0e1f2",
  "capability_manifest": {
    "url": "https://example.com/.well-known/atn/capability.jws",
    "digest": "sha256:b4c5d6e7f8a9b0c1..."
  },
  "delegation_chain": {
    "url": "https://example.com/.well-known/atn/delegation.jws",
    "digest": "sha256:c5d6e7f8a9b0c1d2..."
  },
  "provenance_attestation": {
    "url": "https://example.com/.well-known/atn/provenance.jws",
    "digest": "sha256:d6e7f8a9b0c1d2e3..."
  },
  "handshake_endpoint": "https://agent.example.com/.atn/handshake",
  "witness_pointers": [
    "https://scitt.example.org/entries/a1b2c3..."
  ]
}
```

The Index Document itself MUST be signed as a JWS [RFC7515] when the discovery record cannot be DNSSEC-validated [RFC4033]. When DNSSEC is in use, the atn-digest tag covers integrity at the DNS layer and signature on the Index Document is RECOMMENDED but not REQUIRED.

5. Capability Manifest

The Capability Manifest declares what the agent is willing to do under what conditions. It is a JSON object signed as a JWS.

5.1. Capability Object Structure

Each capability is a structured object with explicit dimensions covering identification, semantics, scope, and operational bounds. The design borrows from the action-resource-condition pattern of established authorization frameworks, the schema-binding pattern of W3C Verifiable Credentials, and the attenuation principle of capability-based security research.

Field	Required	Description
id	Yes	Capability identifier from the IANA registry, or a vendor-specific identifier.
schema	Yes	URI plus SHA-256 digest of the formal capability definition.
actions	Yes	List of action verbs scoped to the capability's schema vocabulary.
resources	Yes	List of resource patterns the capability applies to.
conditions	No	Runtime constraints: rate limits, data residency, time windows, size limits.
effects	Yes	One of: none, read_only, idempotent, mutating.
external_calls	Yes	One of: forbidden, listed_only, free. Whether the capability may call out.
sub_invocations	Yes	One of: forbidden, same_scope, fresh_handshake_required. Sub-agent rules.
persistence	Yes	One of: none, session_only, durable. Whether state survives the session.
resource_bounds	Yes	Hard ceilings: max_tokens, max_duration_seconds, max_cost_usd.
preconditions	No	Counterparty requirements that must hold before the capability is exercised.

Table 3

Every dimension uses a controlled vocabulary registered with IANA (see Section 17). Vendor-specific identifiers under the x- prefix are permitted for non-standard capabilities and MUST carry a schema reference.

5.2. Schema Binding

The schema field binds the capability identifier to a formal definition. Two implementations resolving the same id to different schema.digest values MUST treat the capabilities as semantically distinct, even if the identifier matches. This eliminates the silent-divergence problem that opaque scope strings created in OAuth deployments.

The schema document SHOULD define:

1. The semantic meaning of the capability identifier.
2. The permitted values in actions.
3. The format and semantics of resources patterns.
4. Any capability-specific extensions to conditions.

Schema documents are versioned. A capability MAY pin a specific schema version via the digest.

5.3. Structure

```
{
  "v": "atn-capability-1",
  "agent_id": "<stable-identifier>",
  "issued_at": "2026-05-15T10:00:00Z",
  "valid_until": "2026-08-15T10:00:00Z",
  "capabilities": [
    {
      "id": "data-read",
      "schema": {
        "url": "https://schemas.example.com/atn/data-read-v1.json",
        "digest": "sha256:b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0a1b2c3d4e5f6a7b8c9d0e1f2a3
b4c5"
      },
      "actions": ["read", "list"],
      "resources": ["dataset:public/*", "dataset:internal/research/*"],
      "conditions": {
        "rate_limit": "1000/min",
        "data_residency": ["US", "EU"],
        "time_window": "09:00-17:00 UTC",
        "max_response_size_bytes": 10485760
      }
    }
  ]
}
```

```
    },
    "effects": "read_only",
    "external_calls": "forbidden",
    "sub_invocations": "forbidden",
    "persistence": "none",
    "resource_bounds": {
      "max_tokens": 100000,
      "max_duration_seconds": 1800,
      "max_cost_usd": 0.50
    },
    "preconditions": {
      "counterparty_provenance": "required",
      "counterparty_delegation": "required",
      "transport": "tls1.3"
    }
  },
  {
    "id": "task-execute",
    "schema": {
      "url": "https://schemas.example.com/atn/task-execute-v1.json",
      "digest": "sha256:c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0a1b2c3d4e5f6a7b8c9d0e1f2a3b4
c5d6"
    },
    "actions": ["execute"],
    "resources": ["task:summarize", "task:translate", "task:classify"],
    "conditions": {
      "max_session_minutes": 30
    },
    "effects": "idempotent",
    "external_calls": "listed_only",
    "sub_invocations": "same_scope",
    "persistence": "session_only",
    "resource_bounds": {
      "max_tokens": 500000,
      "max_duration_seconds": 1800,
      "max_cost_usd": 2.00
    },
    "preconditions": {
      "human_approval": "required",
      "approval_token_issuer": "https://approvals.example.com"
    }
  }
],
"refusals": [
  {
    "category": "financial_transactions",
    "scope": "all"
  },
  {
```

```
        "category": "data_write",  
        "scope": "external_systems"  
    }  
]  
}
```

5.4. Dimension Semantics

`effects` declares the consequence of exercising the capability:

- * `none`: no observable consequence (introspection only).
- * `read_only`: data is read; no state changes.
- * `idempotent`: state may change but repeated execution is equivalent to single execution.
- * `mutating`: state changes, and the change is not idempotent.

`external_calls` declares whether the capability may initiate calls outside the agent's own scope:

- * `forbidden`: no external calls.
- * `listed_only`: external calls are permitted only to destinations listed in the schema or conditions.
- * `free`: any external call is permitted, bounded only by `resource_bounds`.

`sub_invocations` declares the rules for invoking other agents:

- * `forbidden`: this capability does not invoke other agents.
- * `same_scope`: sub-invocations inherit the present session's negotiated scope.
- * `fresh_handshake_required`: every sub-invocation requires a separate ATN handshake with the sub-agent.

`persistence` declares state survivability:

- * `none`: no state persists beyond a single request.
- * `session_only`: state persists for the session lifetime, then is discarded.

- * `durable`: state persists beyond the session; the agent **MUST** disclose retention policy in its schema.

`resource_bounds` declares hard ceilings on the negotiated scope. Conforming implementations **MUST** terminate the session when any bound is exceeded. The protocol specifies the contract; runtime enforcement is the implementation's responsibility, consistent with the enforcement posture described in the Security Considerations.

5.5. Key Properties

- * The `manifest` is a declaration of willingness, not an enforcement primitive. Violations are auditable through the Session Receipt and transparency log.
- * The `valid_until` field forces rotation. Manifests without expiry **MUST** be rejected.
- * refusals are first-class. Explicit "will not" claims provide auditable accountability if violated.
- * Capability id values from the IANA registry have stable semantics; vendor-specific identifiers under `x-` **MUST** carry a schema reference.
- * Capabilities are deny-by-default. The negotiated session scope contains only what both parties explicitly offered.

5.6. Refusal Semantics

Refusals are a deliberate inversion of conventional capability advertisement. Most existing proposals describe what an agent **CAN** do. ATN requires agents to also describe what they explicitly **WILL NOT** do. This produces auditable behavioral contracts: a recorded refusal that is subsequently violated is direct evidence of misbehavior.

6. Delegation Chain

The Delegation Chain proves the agent operates under authorized principal scope, with explicit time bounds and revocation paths.

6.1. Structure

```

{
  "v": "atn-delegation-1",
  "agent_id": "<stable-identifier>",
  "chain": [
    {
      "issuer": "did:example:organization-root",
      "subject": "did:example:department-ops",
      "scope": ["agent.deploy", "agent.delegate"],
      "issued_at": "2026-01-01T00:00:00Z",
      "valid_until": "2027-01-01T00:00:00Z",
      "revocation": "https://example.com/.well-known/atn/revocation/department-ops",
      "signature": "<JWS>"
    },
    {
      "issuer": "did:example:department-ops",
      "subject": "agent:<stable-identifier>",
      "scope": ["data-read", "task-execute:summarize", "task-execute:translate"],
      "issued_at": "2026-05-01T00:00:00Z",
      "valid_until": "2026-08-01T00:00:00Z",
      "revocation": "https://example.com/.well-known/atn/revocation/agent",
      "signature": "<JWS>"
    }
  ]
}

```

6.2. Verification

A relying party MUST:

1. Verify each link's signature against the issuer's published key.
2. Verify each link's scope is a subset of the parent link's scope.
3. Verify all links are within their valid_until window.
4. Optionally poll revocation endpoints with rate-limited caching.
5. Verify the leaf subject matches the agent's stable identifier from discovery.

If any check fails, the agent MUST NOT be trusted for any capability beyond the verifiable subset.

7. Provenance Attestation

The Provenance Attestation proves build-time and runtime characteristics of the agent.

7.1. Structure

```
{
  "v": "atn-provenance-1",
  "agent_id": "<stable-identifier>",
  "build": {
    "predicateType": "https://slsa.dev/provenance/v1",
    "predicate": { "...": "in-toto SLSA provenance" }
  },
  "model": {
    "model_id": "example-vendor/llm-v1.2.3",
    "model_version_sha256": "9f8e7d6c5b4a...",
    "system_prompt_sha256": "1a2b3c4d5e6f..."
  },
  "runtime": {
    "evidence_type": "RATS",
    "evidence": { "...": "RFC 9334 evidence" },
    "verifier": "https://verifier.example.com/rats"
  },
  "issued_at": "2026-05-15T08:00:00Z",
  "valid_until": "2026-05-15T20:00:00Z"
}
```

7.2. Properties

- * The Provenance Attestation links three independent supply chains: software build (SLSA, in-toto [IN-TOTO]), model identity (cryptographic hash of weights or version anchor), and runtime environment (RATS evidence [RFC9334]).
- * All three are optional. An agent that publishes only build provenance is more trustworthy than one that publishes none and less trustworthy than one that publishes all three.
- * Short `valid_until` windows are encouraged. The Provenance Attestation is the most time-sensitive artifact.

8. The Agent Trust Handshake (ATH)

8.1. State Machine

Initiator	Responder
<pre> ----- ATH_HELLO -----> (initiator artifacts + requested scope) </pre>	<pre> <----- ATH_OFFER ----- (responder artifacts + offered scope) </pre>
<pre> ----- ATH_ACCEPT ----- > (intersection scope + receipt request) </pre>	<pre> <----- ATH_RECEIPT ----- (signed session receipt) </pre>
<pre> === negotiated session begins === </pre>	

All four messages are JSON, signed as JWS, exchanged over HTTPS at the Responder's `handshake_endpoint`.

8.2. ATH_HELLO

```

{
  "v": "ath1",
  "type": "hello",
  "supported_versions": ["ath1"],
  "initiator": {
    "agent_id": "<initiator-id>",
    "artifacts": {
      "capability": { "url": "...", "digest": "..." },
      "delegation": { "url": "...", "digest": "..." },
      "provenance": { "url": "...", "digest": "..." }
    }
  },
  "requested_scope": {
    "capability_ids": ["data-read"],
    "duration_seconds": 1800,
    "purpose": "summarize_research_corpus"
  },
  "nonce": "<random>",
  "timestamp": "..."
}

```

The `requested_scope.capability_ids` field carries only the identifiers of the capabilities the Initiator wishes to negotiate. The Responder consults its own Capability Manifest for the structured definitions and computes the intersection (see Section 9).

8.3. ATH_OFFER

The Responder verifies the Initiator's artifacts, computes the capability intersection, and returns:

```
{
  "v": "ath1",
  "type": "offer",
  "selected_version": "ath1",
  "supported_versions_echo": ["ath1"],
  "responder": {
    "agent_id": "<responder-id>",
    "artifacts": {
      "capability": { "url": "...", "digest": "..." },
      "delegation": { "url": "...", "digest": "..." },
      "provenance": { "url": "...", "digest": "..." }
    }
  },
  "offered_scope": {
    "capabilities": [
      {
        "id": "data-read",
        "schema": { "url": "https://schemas.example.com/atn/data-read-v1.json",
                    "digest": "sha256:b4c5d6..." },
        "actions": ["read", "list"],
        "resources": ["dataset:public/*"],
        "conditions": { "rate_limit": "500/min", "data_residency": ["US"] },
        "effects": "read_only",
        "external_calls": "forbidden",
        "sub_invocations": "forbidden",
        "persistence": "none",
        "resource_bounds": { "max_tokens": 50000, "max_duration_seconds": 1800, "max_cost_usd": 0.50 }
      }
    ],
    "duration_seconds": 1800,
    "purpose": "summarize_research_corpus"
  },
  "nonce": "<random>",
  "in_reply_to_nonce": "<initiator-nonce>",
  "timestamp": "..."
}
```

The `offered_scope.capabilities` field carries the full structured Capability objects after intersection. Each is the per-dimension intersection of the Initiator's requested capability (resolved from its Capability Manifest) and the Responder's offered version.

8.4. ATH_ACCEPT

If the Initiator agrees with the offered scope:

```
{
  "v": "ath1",
  "type": "accept",
  "agreed_scope": { "...": "matches ATH_OFFER offered_scope" },
  "nonce": "<random>",
  "in_reply_to_nonce": "<responder-nonce>",
  "timestamp": "..."
}
```

If the Initiator disagrees, it MAY send ATH_COUNTER with a revised request, up to a limit (see Section 8.6).

8.5. ATH_RECEIPT

The Responder produces the Session Receipt:

```
{
  "v": "ath1",
  "type": "receipt",
  "session_id": "<uuid>",
  "initiator_id": "<initiator-id>",
  "responder_id": "<responder-id>",
  "agreed_scope": { "...": "the intersection result" },
  "artifact_digests": {
    "initiator_capability": "sha256:...",
    "initiator_delegation": "sha256:...",
    "initiator_provenance": "sha256:...",
    "responder_capability": "sha256:...",
    "responder_delegation": "sha256:...",
    "responder_provenance": "sha256:..."
  },
  "scitt_log_pointer": "https://scitt.example.org/entries/...",
  "issued_at": "...",
  "expires_at": "...",
  "signature": "<JWS by responder>"
}
```

Both parties countersign the receipt and MAY publish it to a SCITT log [SCITT].

8.6. Handshake Limits

- * Total handshake round trips MUST NOT exceed 4.

- * ATH_COUNTER messages MUST NOT exceed 2 per session.
- * The handshake MUST complete within 30 seconds.
- * Sessions exceeding agreed_scope.duration_seconds require a new handshake.

9. Capability Intersection Algebra

When two agents present capability sets, the negotiated scope is computed by intersection. The algorithm is specified for interoperability so that conforming implementations produce identical results from identical inputs.

9.1. Identity Rule

A capability *c* is in the intersection only if all of the following hold:

1. Both Initiator's requested set and Responder's offered set include a capability with the same id.
2. Both capabilities reference the same schema.url and schema.digest. Identifiers with diverging schemas MUST be treated as distinct capabilities and excluded from the intersection.
3. Neither party has a matching entry in refusals covering *c*.

9.2. Per-Dimension Rules

For each capability *c* that satisfies the identity rule, the intersected capability is computed dimension by dimension.

Dimension	Combinator	Notes
actions	List intersection	If empty, <i>c</i> is dropped.
resources	Pattern intersection (longest-prefix match)	If empty, <i>c</i> is dropped.
conditions numeric (rate_limit, max_response_size_bytes, max_session_minutes)	Minimum	Tighter bound wins.
conditions list (data_residency,	List intersection	If empty, <i>c</i>

tasks)		is dropped.	
conditions.time_window	Time-window intersection	If empty, c	is dropped.
effects	Most restrictive of: none < read_only < idempotent < mutating	Take lower in ordering.	
external_calls	Most restrictive of: forbidden < listed_only < free	Take lower in ordering.	
sub_invocations	Most restrictive of: forbidden < same_scope < fresh_handshake_required	Take lower in ordering.	
persistence	Most restrictive of: none < session_only < durable	Take lower in ordering.	
resource_bounds.max_tokens	Minimum		
resource_bounds.max_duration_seconds	Minimum		
resource_bounds.max_cost_usd	Minimum		
preconditions	Union	Both parties' preconditions apply to the negotiated capability.	

Table 4

9.3. Refusal Override

If either party's refusals list contains an entry matching capability *c* by category or by direct id, *c* is dropped from the intersection regardless of any other dimension. Refusals are absolute.

9.4. Worked Example

Initiator requests data-read:

```
* actions: [read, list, search]
```

- * resources: [dataset:public/*, dataset:internal/*]
- * conditions.rate_limit: 1000/min
- * conditions.data_residency: [US, EU, APAC]
- * effects: read_only
- * external_calls: listed_only
- * resource_bounds.max_cost_usd: 1.00

Responder offers data-read:

- * actions: [read, list]
- * resources: [dataset:public/*]
- * conditions.rate_limit: 500/min
- * conditions.data_residency: [US, EU]
- * effects: read_only
- * external_calls: forbidden
- * resource_bounds.max_cost_usd: 0.50

Negotiated intersection (assuming matching schemas):

- * actions: [read, list]
- * resources: [dataset:public/*]
- * conditions.rate_limit: 500/min
- * conditions.data_residency: [US, EU]
- * effects: read_only
- * external_calls: forbidden
- * resource_bounds.max_cost_usd: 0.50

If the schemas had diverged, data-read would be dropped entirely with no intersection. If Initiator's refusals included data-read, the capability would be dropped regardless of offer compatibility.

10. Mutual Trust Mode

When both agents are sensitive (for example, two agents acting on behalf of different organizations), the handshake operates in mutual mode:

- * Both parties MUST present all four artifacts in ATH_HELLO and ATH_OFFER.
- * Both parties MUST verify the counterparty's full chain before proceeding.
- * The Session Receipt MUST be countersigned by both parties.

In single-sided mode (for example, when one party is a public service), only the trusted side must present full artifacts. The other side's identity is still verified at the discovery layer, but the trust posture is asymmetric.

11. Version Negotiation and Algorithm Agility

ATN is designed for long-term evolution. Two mechanisms ensure forward compatibility: explicit version negotiation in the handshake and pluggable cryptographic algorithms in artifacts and messages.

11.1. Version Negotiation

The Initiator's ATH_HELLO message advertises every ATN version it supports in a `supported_versions` field. The Responder's ATH_OFFER selects the highest mutually supported version and echoes it in a `selected_version` field. All subsequent messages in the session use the selected version.

```
{
  "v": "ath1",
  "type": "hello",
  "supported_versions": ["ath1", "ath2"],
  "initiator": { "...": "..." },
  "requested_scope": { "...": "..." },
  "nonce": "...",
  "timestamp": "..."
}
```

The Responder MUST include the Initiator's `supported_versions` list in the signed envelope of its ATH_OFFER. This prevents downgrade attacks: an on-path adversary that strips advertised versions from ATH_HELLO would cause the Responder's signature to cover the stripped list, which the Initiator can detect.

If the Initiator and Responder share no common version, the Responder MUST reply with ATH_REJECT carrying error code `version_mismatch`. The handshake terminates.

11.2. Algorithm Agility

ATN does not pin cryptographic algorithms beyond a minimum set required for v1 interoperability. Implementations MAY use any algorithm registered for JWS [RFC7515] or COSE.

Required for ATN v1 conformance:

1. Ed25519 signatures over JWS-encoded artifacts and handshake messages.
2. SHA-256 for digests bound at the DNS layer and in artifact references.

Recommended additional algorithms:

1. ECDSA P-256 with SHA-256 for environments with ECDSA-only hardware tokens.
2. ML-DSA (post-quantum signature) for deployments preparing for post-quantum transition.

The Capability Manifest, Delegation Chain, Provenance Attestation, and Session Receipt MAY each be signed with a different algorithm. The JWS header identifies the algorithm. Relying parties MUST reject artifacts signed with algorithms they do not support.

Algorithm identifiers are negotiated implicitly via the JWS alg header on each artifact. Explicit negotiation in ATH_HELLO is NOT REQUIRED for v1 but MAY be added in future versions.

11.3. Post-Quantum Transition

The cryptographic operations in ATN are well-bounded: signature verification of artifacts and handshake messages. There is no key agreement step inside ATN itself; TLS handles channel security. Migration to post-quantum signatures requires only adopting a PQ-capable signature algorithm in JWS, which the IETF JOSE working group is standardizing.

When PQ algorithms are widely deployed, operators SHOULD rotate to a hybrid scheme (classical + PQ) during the transition window, then to PQ-only. ATN's version negotiation supports introducing a new minor version that prefers PQ signatures without breaking v1 deployments.

12. Compatibility with Existing Discovery Protocols

ATN is designed to coexist with existing DNS-based discovery proposals without modification:

12.1. With AID [AID]

AID's TXT record is extended with the atn and atn-digest tags. AID's PKA mechanism continues to authenticate the endpoint. ATN consumes the AID-authenticated identity as the input to the handshake.

12.2. With DNS-AID [DNS-AID]

DNS-AID's SVCB records are extended with the atn and atn-digest ParamKeys. The leaf-zone naming convention from DNS-AID is preserved.

12.3. With ANS v2 [ANSv2]

ANS v2's Trust Card concept maps directly to a subset of ATN's Capability Manifest. An ANS v2 deployment can publish an ATN Index Document where the Capability Manifest cross-references the ANS Trust Card.

12.4. Multi-Discovery Tolerance

A single agent zone MAY publish records compatible with multiple discovery proposals simultaneously. ATN binding is independent of which discovery format is used.

13. Relationship to Existing Identity, Authorization, and Attestation Frameworks

A reasonable first question for any new trust protocol is whether existing frameworks suffice. ATN composes several mature primitives and does not reinvent any of them. The contribution is the composition: a peer-to-peer agent handshake with deterministic scope negotiation and an auditable session receipt. This section addresses each adjacent framework and identifies where ATN composes with it.

The analogy that clarifies the relationship: TLS uses asymmetric crypto, symmetric crypto, key exchange, and X.509. TLS is not accused of reinventing those primitives. TLS is the handshake that composes them into a deployable channel-security protocol. ATN occupies the same compositional role for inter-agent trust.

13.1. OAuth 2.0 and G NAP

OAuth 2.0 [RFC6749] and G NAP [RFC9635] authenticate and authorize a Client to access a Resource Server. ATN negotiates a mutual agreement between two autonomous peers.

Structural differences:

1. Asymmetric vs symmetric. OAuth and G NAP define Client and Resource Server roles. ATN defines two peers, each presenting four artifacts. There is no "resource server" in agent-to-agent collaboration.
2. Secret tokens vs publishable artifacts. OAuth bearer tokens are secrets and MUST NOT be logged. ATN artifacts are designed for publication to transparency logs. Opposite security models for the core artifacts.
3. No formal scope intersection. OAuth scopes are evaluated by membership at the Resource Server. ATN specifies a deterministic intersection algorithm.
4. No refusal semantics. OAuth represents grants. ATN represents grants and explicit refusals as first-class signed claims.

Where OAuth and G NAP remain the right tools:

- * Inside a trust domain with a central authorization server, OAuth and G NAP issue access tokens to passive resources. ATN is unnecessary.
- * Where one party is a passive resource, OAuth-style bearer access fits.
- * Human-to-agent delegation, where a human grants an agent access on the human's behalf, fits G NAP.

ATN composes with OAuth and G NAP. An agent in an ATN session MAY hold OAuth tokens for downstream resource access. ATN governs the inter-agent relationship; OAuth governs subsequent resource calls.

13.2. SPIFFE Workload Identity

SPIFFE [SPIFFE] issues SVIDs (SPIFFE Verifiable Identity Documents) to workloads within a trust domain. SPIFFE Federation supports cross-domain identity through bilaterally exchanged trust bundles.

Structural relationship:

1. SPIFFE provides identity, not authorization, willingness, capability, or provenance.
2. SPIFFE is optimized for intra-organizational deployment. Cross-organizational use requires pre-arranged federation. ATN targets cross-organizational interactions where peers may meet for the first time without prior trust setup.
3. ATN can carry a SPIFFE ID as the stable identifier in its artifacts. The SPIFFE SVID becomes the authentication input; the ATN handshake adds capability, delegation, provenance, and receipt on top.

SPIFFE and ATN compose cleanly. SPIFFE within the trust domain, ATN at the boundary.

13.3. Mutual TLS

Mutual TLS [RFC8446] authenticates both endpoints at the transport layer. It establishes that the connection is between two specific certificates.

mTLS provides:

- * Endpoint authentication.
- * Channel confidentiality and integrity.

mTLS does not provide:

- * Capability declaration.
- * Delegation expression.
- * Provenance evidence.
- * Negotiation of scope.
- * Auditable agreement record.

ATN runs on top of an mTLS-secured channel where appropriate. mTLS is necessary infrastructure for many ATN deployments. It is not a substitute for the negotiation.

13.4. RATS Remote Attestation

RATS [RFC9334] defines roles (Attester, Verifier, Relying Party, Endorser) and the structure of attestation evidence and results.

RATS provides:

- * The architecture for producing and verifying attestation evidence.

RATS does not provide:

- * A protocol for combining attestation evidence with authorization or capability negotiation.
- * A specification for what to do with verification results in a peer interaction.

ATN consumes RATS evidence as one axis of its Provenance Attestation. The other two axes (build provenance and model identity) come from supply-chain attestation frameworks. RATS is the input; ATN is one specified consumer.

13.5. SCITT Transparency

SCITT [SCITT] defines an append-only transparency log for signed statements with inclusion proofs.

SCITT provides:

- * The transparency log infrastructure.
- * The signed-statement envelope format.

SCITT does not provide:

- * The semantics of what to log for a given application.
- * The receipt format that captures an inter-agent agreement.

ATN provides the Session Receipt format that becomes the SCITT statement for an agent interaction. SCITT without ATN is a log waiting for a record format. ATN without SCITT is a receipt waiting for a transparency layer.

13.6. Composition Summary

ATN composes the following primitives per layer:

Concern	Primitive	ATN's Role
Endpoint authentication	mTLS [RFC8446]	Runs above
Workload identity (intra-org)	SPIFFE [SPIFFE]	Consumes SVIDs as stable identifiers
Cross-org identity	DNS + DNSSEC [RFC4033], [AID], [DNS-AID], [ANSv2]	Discovery substrate for ATN artifacts
Delegated authorization to passive resources	OAuth [RFC6749], GNAP [RFC9635]	Composes with, does not replace
Attestation evidence	RATS [RFC9334]	Consumes evidence in Provenance Attestation
Public auditability	SCITT [SCITT]	Provides the receipt format for log entries
Peer-to-peer trust handshake	(none exists)	This document

Table 5

The contribution of ATN is the bottom row. The five rows above are existing work that ATN composes.

13.7. When to Use What

1. Two workloads inside one trust domain coordinating on a task: SPIFFE.
2. A client accessing a resource server with delegated user authorization: OAuth or GNAP.
3. Establishing that you are talking to a specific endpoint: mTLS.
4. Producing tamper-evident proof that some software was built and deployed as claimed: RATS plus supply-chain attestation.
5. Publishing an auditable record of a claim: SCITT.

6. Two autonomous agents from different organizations agreeing to work together with mutually verified capability, authority, provenance, and a signed record of the agreement: ATN.

14. Security Considerations

14.1. Threat Model

ATN assumes:

- * Adversaries may publish well-formed but malicious ATN artifacts.
- * Adversaries may compromise an agent and continue to present valid pre-compromise artifacts.
- * Adversaries may attempt to negotiate scope expansions outside the published Capability Manifest.
- * Adversaries may collude across organizational boundaries to construct false Delegation Chains.

14.2. Artifact Authenticity

Every ATN artifact is signed. Signatures MUST be verified against keys bound to the agent's stable identifier at the discovery layer. Artifacts whose signatures do not verify, whose valid_until has passed, or whose digests do not match the DNS-published values MUST be rejected.

14.3. Enforcement Posture: Evidence, Not Prevention

A predictable objection to ATN is that auditability is not enforcement. The objection is correct on its terms and misses the layer at which ATN operates. ATN is technical evidence infrastructure. It is not a behavioral enforcement primitive. No protocol layered on top of an autonomous software agent can be one.

An agent that signs a Capability Manifest declaring it will not perform financial transactions, then performs a financial transaction, has cryptographically committed to a refusal it then violated. ATN cannot prevent the violation. ATN produces a Session Receipt and a published transparency log entry that, combined with the application-layer evidence of the violation, constitute direct cryptographic evidence usable for revocation, contract enforcement, regulatory complaint, and civil action.

The complete trust stack for autonomous agents has four layers:

1. Technical evidence layer (ATN, SCITT, RATS). Provides cryptographic proof of what was claimed and what was agreed.
2. Policy layer (contracts, organizational policy, regulations). Specifies what is permitted.
3. Consequence layer (revocation, financial penalty, legal action, reputational impact). Imposes cost on violation.
4. Operational enforcement layer (sandboxing, capability-based execution, hardware-enforced runtime such as TEEs, gating proxies). Prevents violation at runtime.

ATN is the first layer. It enables the other three. It does not substitute for them.

For high-stakes domains (financial transactions, clinical decision support, control of physical systems), ATN MUST be deployed with operational enforcement (layer 4). Recommended patterns:

1. Run sensitive capabilities inside a TEE that enforces declared bounds at hardware level.
2. Use a gating proxy that enforces the negotiated scope on every outbound call from the agent.
3. Use short session lifetimes with frequent re-attestation.
4. Combine ATN with object-capability execution where the agent runtime denies unauthorized actions by construction, not by policy.

ATN's contribution is to make violations expensive and discoverable, not impossible. The analogy is to TCP checksums: they do not prevent transmission errors, they detect them so the next layer can respond. Trust at Internet scale has always operated this way. SPF does not prevent spoofed email. Certificate transparency does not prevent mis-issuance. They make undetected misbehavior infeasible. ATN does the same for inter-agent agreements.

14.4. Delegation Forgery

Delegation Chains depend on the cryptographic strength of each issuer's signing key. ATN does not specify the identity verification process that issuers use to vouch for subjects. Operators SHOULD use verifiable credentials [W3C-DID] or organizational identity infrastructure with established assurance levels.

14.5. Session Receipt Tampering

Session Receipts are countersigned and publishable to a SCITT log [SCITT]. Tampering after publication is detectable through log inclusion proofs. Operators handling high-value sessions SHOULD publish receipts.

14.6. Replay

Every handshake message includes a nonce and a timestamp. Implementations MUST reject messages with nonces seen within the session and with timestamps outside a configurable skew window (RECOMMENDED 60 seconds).

14.7. Provenance Attestation Limits

Provenance attestations prove what was built and deployed at attestation time. They do not prove the agent is currently executing the attested code. Operators SHOULD use short valid_until windows for provenance attestations and refresh through RATS verification [RFC9334].

15. Privacy Considerations

15.1. Negotiation Observability

The handshake exchanges artifacts that may reveal the purpose of the session (purpose field) and the identity of the principal (Delegation Chain). Operators SHOULD treat handshake metadata as sensitive.

15.2. Witness Aggregation

Witnesses and SCITT log operators see the volume and pattern of agent interactions. Operators SHOULD use multiple witnesses to reduce centralized observability.

15.3. Minimum Disclosure

ATN supports minimum-disclosure mode where the Initiator presents only the artifacts required for the requested scope. The Responder MAY request additional artifacts during the handshake, which the Initiator MAY decline (causing the handshake to fail).

16. Operational Considerations

ATN introduces non-trivial latency on session establishment. This section quantifies that cost, explains its operational shape, and identifies deployments where ATN is appropriate and where it is not.

16.1. Session Establishment vs Per-Call Latency

ATN is a session establishment protocol, not a per-request protocol. The handshake establishes a negotiated scope. The scope authorizes many subsequent interactions between the two agents. Per-call latency within an established session is zero added by ATN.

This places ATN in the same operational category as:

1. TLS handshake [RFC8446], amortized over the requests in a connection.
2. OAuth token acquisition [RFC6749], amortized over the token lifetime.
3. BGP session establishment, amortized over hours of routing operation.

The relevant performance question is the cost of establishment and the session lifetime over which it is amortized, not the cost per call.

16.2. Latency Budget

The following budget is informative. Implementations and deployments will vary based on geography, caching, and network conditions.

Component	Cold (no cache)	Warm (cached, pooled)
DNS lookup for _agent.<domain> with DNSSEC	50-150 ms	<1 ms
TLS 1.3 connection to Index Document origin	100-200 ms	0 (pooled)
Index Document HTTPS GET	20-50 ms	0 (cached by digest)
Three artifact fetches (HTTP/2 multiplexed)	50-100 ms	0 (cached by digest)
Signature verifications (4-8 Ed25519)	1-2 ms	1-2 ms
Provenance validation	1-5 ms	1 ms
Two-RTT handshake (HELLO/OFFER then ACCEPT/RECEIPT)	150-300 ms	150-300 ms
Receipt signing (Ed25519)	<1 ms	<1 ms
Total session establishment	*500-1500 ms*	*150-300 ms*
Per-call within established session	*0 ms added*	*0 ms added*
SCITT log submission	Async, out of path	Async, out of path

Table 6

Recommended performance targets for conforming implementations:

- * p50 cold session establishment: <800 ms
- * p99 cold session establishment: <2000 ms
- * p50 warm session establishment: <250 ms
- * p99 warm session establishment: <500 ms

16.3. Optimization Techniques

Implementations SHOULD apply the following optimizations to bring warm-case latency to the recommended targets:

1. Digest-keyed artifact caching. Artifacts are refetched only when the DNS-published digest changes. Steady-state cache hit rate is typically above 95 percent.
2. HTTP/2 or HTTP/3 multiplexing for parallel artifact fetches.
3. TLS 1.3 session resumption for repeat handshakes with known peers.
4. Connection pooling between sessions to the same peer.
5. Abbreviated re-handshake. A repeat ATH_HELLO MAY reference a prior session's receipt to extend or renew the same scope in a single round trip rather than three.
6. Pre-warming. Agents MAY establish sessions with frequent counterparties before application traffic arrives.
7. Asynchronous SCITT submission. Receipt publication to a transparency log happens after the session begins and MUST NOT block session establishment.
8. RATS attestation result caching for the duration of the result's validity window.

16.4. Session Lifetime

The Session Receipt's `expires_at` field bounds the session. Within that window, no additional handshake is required for interactions covered by the agreed scope. Implementations SHOULD select session lifetimes that amortize establishment cost while bounding exposure to revocation events.

Typical session lifetimes:

- * Short-lived workflows: 5 to 30 minutes
- * Sustained collaboration sessions: 1 to 24 hours
- * Long-running infrastructure peering: up to 7 days

Session lifetimes longer than 7 days are NOT RECOMMENDED for cross-organizational sessions. The trade-off is between revocation responsiveness and handshake amortization.

16.5. Deployments Where ATN Is Appropriate

1. Multi-step agent collaboration where many calls share a session context.
2. Cross-organizational agent interactions where trust establishment must be verifiable and auditable.
3. Workflows where the cost of trust failure exceeds the cost of establishment latency.
4. Agent ecosystems with workflows already operating in the 100 ms to multi-second range per step.

16.6. Deployments Where ATN Is Not Appropriate

1. Microsecond-scale latency budgets (high-frequency trading, real-time control loops, hardware-in-the-loop systems).
2. Stateless per-message protocols where session continuity is not meaningful.
3. Workflows whose total duration is shorter than the handshake establishment cost.

For these deployments, simpler primitives (mTLS alone, OAuth bearer tokens, or no trust layer at all) may be more appropriate. Operators are encouraged to select the trust layer that matches their latency budget.

17. IANA Considerations

17.1. SVCB ParamKey Registrations

Early allocation requested in the Service Parameter Keys (SvcParamKeys) registry [RFC9460]:

Name	Format
atn	URI
atn-digest	hex-string

Table 7

17.2. ATN Capability Registry

Creation of a new registry, "ATN Capabilities," is requested with initial entries:

Name	Description	Reference Schema
data-read	Read access to a data resource	TBD
data-write	Write access to a data resource	TBD
task-execute	Execute a named task	TBD
model-invoke	Invoke a generative model	TBD
agent-delegate	Delegate work to another agent	TBD
payment-init	Initiate a payment flow	TBD
human-relay	Surface a request to a human approver	TBD

Table 8

Registration policy: Specification Required. Each registered identifier MUST include a reference to a published capability schema that defines the semantics of actions, resources patterns, and any capability-specific extensions to conditions.

Vendor-specific identifiers prefixed with x- are permitted without IANA registration but MUST carry a schema reference in the capability object.

17.3. ATN Capability Dimension Registries

Four new registries are requested to govern the controlled vocabularies used in capability dimensions:

"ATN Capability Effects":

Name	Description
none	No observable consequence (introspection only).
read_only	Data is read; no state changes.
idempotent	State may change; repeated execution equivalent to single.
mutating	State changes; not idempotent.

Table 9

"ATN External Call Modes":

Name	Description
forbidden	No external calls permitted.
listed_only	External calls permitted only to listed destinations.
free	Any external call permitted within resource_bounds.

Table 10

"ATN Sub-Invocation Modes":

Name	Description
forbidden	No sub-agent invocations.
same_scope	Sub-invocations inherit the present negotiated scope.
fresh_handshake_required	Every sub-invocation requires a separate ATN handshake.

Table 11

"ATN Persistence Modes":

Name	Description
none	No state persists beyond a single request.
session_only	State persists for the session lifetime, then discarded.
durable	State persists beyond the session; retention policy required.

Table 12

Registration policy for all four registries: Standards Action or IESG Approval. The controlled vocabularies are deliberately small and stable; expansion requires consensus to avoid the semantic-drift problem that affected OAuth scopes.

17.4. ATN Refusal Category Registry

Creation of a new registry, "ATN Refusal Categories," is requested with initial entries:

Name	Description
financial_transactions	Any movement of funds
personal_data	Processing of personal identifiers
weaponization	Use in weapons systems
mass_persuasion	Generation of mass-distributed content
child_safety	Content involving minors
medical_advice	Clinical recommendations
legal_advice	Legal recommendations binding on parties

Table 13

Registration policy: Specification Required.

17.5. Well-Known URI Registration

The path /.well-known/atn/ is reserved per RFC 8615 conventions for ATN artifact publication.

18. References

18.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <<https://www.rfc-editor.org/rfc/rfc4033>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/rfc/rfc7515>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.
- [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/rfc/rfc8259>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/rfc/rfc8446>>.
- [RFC9460] Schwartz, B., Bishop, M., and E. Nygren, "Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records)", RFC 9460, DOI 10.17487/RFC9460, November 2023, <<https://www.rfc-editor.org/rfc/rfc9460>>.

18.2. Informative References

- [AID] Nemethi, B., "Agent Identity and Discovery (AID)", 2026, <<https://datatracker.ietf.org/doc/draft-nemethi-aid-agent-identity-discovery/>>.
- [ANSv2] Narajala, K., "Agent Name Service v2", 2026, <<https://datatracker.ietf.org/doc/draft-narajala-courtney-ansv2/>>.
- [DNS-AID] Mozley, J., Williams, N., Sarikaya, B., and R. Schott, "DNS for AI Discovery", 2026, <<https://datatracker.ietf.org/doc/draft-mozleywilliams-dnsop-dnsaid/>>.
- [IN-TOTO] in-toto / CNCF, "in-toto Attestation Framework", 2024, <<https://in-toto.io/>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/rfc/rfc6749>>.
- [RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedures (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <<https://www.rfc-editor.org/rfc/rfc9334>>.

- [RFC9635] Richer, J., Ed. and F. Imbault, "Grant Negotiation and Authorization Protocol (GNAP)", RFC 9635, DOI 10.17487/RFC9635, October 2024, <<https://www.rfc-editor.org/rfc/rfc9635>>.
- [SCITT] IETF SCITT WG, "Supply Chain Integrity, Transparency, and Trust (SCITT) Architecture", 2025, <<https://datatracker.ietf.org/wg/scitt/about/>>.
- [SPIFFE] SPIFFE / CNCF, "SPIFFE: Secure Production Identity Framework for Everyone", 2024, <<https://github.com/spiffe/spiffe>>.
- [W3C-DID] W3C, "Decentralized Identifiers (DIDs) v1.0", 2022, <<https://www.w3.org/TR/did-core/>>.

Appendix A. Complete Example Flow

This appendix walks through a full ATN handshake between two agents at agent.research.org and agent.publisher.com.

A.1. Step 1: Discovery

```
$ dig +short TXT _agent.publisher.com
"v=aid1; u=https://agent.publisher.com/mcp; p=mcp;
k=z7rW8rTq8o4mM6vVf7wlk3m4uQn9p2YxCAbcDeFgHiJ;
atn=https://publisher.com/.well-known/atn/index.json;
atn-digest=a3b8c2d4..."
```

A.2. Step 2: Index Document Fetch

```
GET https://publisher.com/.well-known/atn/index.json
```

Returns the JSON object listing the four artifact URLs and their digests.

A.3. Step 3: Artifact Fetch and Verification

The Initiator fetches:

- * `capability.jws`
- * `delegation.jws`
- * `provenance.jws`

Verifies each digest matches the Index Document, verifies each signature, and confirms `valid_until` is in the future.

A.4. Step 4: ATH_HELLO

The Initiator (research.org agent) sends ATH_HELLO to the Responder's handshake_endpoint:

```
{
  "v": "ath1",
  "type": "hello",
  "supported_versions": ["ath1"],
  "initiator": {
    "agent_id": "INIT-XYZ123...",
    "artifacts": { "...": "capability/delegation/provenance refs" }
  },
  "requested_scope": {
    "capability_ids": ["data-read"],
    "duration_seconds": 600,
    "purpose": "academic_research_summarization"
  },
  "nonce": "n-init-001",
  "timestamp": "2026-05-15T14:00:00Z"
}
```

A.5. Step 5: ATH_OFFER

The Responder verifies the Initiator's artifacts, computes the intersection, and returns:

```

{
  "v": "ath1",
  "type": "offer",
  "selected_version": "ath1",
  "supported_versions_echo": ["ath1"],
  "responder": { "...": "responder agent_id and artifact refs" },
  "offered_scope": {
    "capabilities": [
      {
        "id": "data-read",
        "schema": { "url": "https://schemas.example.com/atn/data-read-v1.json",
                     "digest": "sha256:b4c5d6..." },
        "actions": ["read", "list"],
        "resources": ["dataset:public/*"],
        "conditions": { "rate_limit": "300/min", "data_residency": ["US", "EU"] },
        "effects": "read_only",
        "external_calls": "forbidden",
        "sub_invocations": "forbidden",
        "persistence": "none",
        "resource_bounds": { "max_tokens": 50000, "max_duration_seconds": 600, "max_co
st_usd": 0.30 }
      }
    ],
    "duration_seconds": 600,
    "purpose": "academic_research_summarization"
  },
  "nonce": "n-resp-001",
  "in_reply_to_nonce": "n-init-001",
  "timestamp": "2026-05-15T14:00:01Z"
}

```

A.6. Step 6: ATH_ACCEPT

```

{
  "v": "ath1",
  "type": "accept",
  "agreed_scope": { "...": "matches ATH_OFFER" },
  "nonce": "n-init-002",
  "in_reply_to_nonce": "n-resp-001",
  "timestamp": "2026-05-15T14:00:02Z"
}

```

A.7. Step 7: ATH_RECEIPT

```
{
  "v": "ath1",
  "type": "receipt",
  "session_id": "5b7c-...",
  "initiator_id": "INIT-XYZ123...",
  "responder_id": "RESP-ABC789...",
  "agreed_scope": { ... },
  "artifact_digests": { ... },
  "scitt_log_pointer": "https://scitt.research.org/entries/789xyz",
  "issued_at": "2026-05-15T14:00:03Z",
  "expires_at": "2026-05-15T14:10:03Z",
  "signature": "<JWS by responder>"
}
```

The Initiator countersigns. The session begins.

A.8. Step 8: Session and Audit

The agents transact within the agreed scope. The Session Receipt is published to SCITT. Any downstream audit can:

1. Retrieve the receipt.
2. Verify all artifact digests against the agents' original publications.
3. Compare actual session traffic (where logged) against the agreed scope.
4. Flag any deviation.

Appendix B. Design Rationale

B.1. Why a Negotiation Layer

DNS-based discovery answers "where" and "who." Application protocols (MCP, A2A) answer "how." Nothing in the current stack answers "what may we do together, and what evidence will remain." ATN fills that gap with a two-round-trip handshake.

B.2. Why Four Artifacts

Each artifact answers a distinct question:

- * Capability Manifest: what is this agent willing to do?
- * Delegation Chain: who authorized it?

- * Provenance Attestation: what is it actually running?
- * Session Receipt: what did we agree?

Collapsing them into one document obscures the separation of concerns. Splitting further fragments the trust evaluation. Four is the minimum.

B.3. Why Refusals Are First-Class

Capability advertisement alone creates a moral hazard: agents have an incentive to advertise broadly. Mandatory refusals create symmetric accountability: an agent that explicitly refused a category and then violated the refusal has produced cryptographic evidence against itself.

B.4. Why Intersection Algebra Is Specified

If two implementations compute intersections differently, interoperability fails. The algebra is small, specified, and testable. The cost is one paragraph of spec; the benefit is deterministic handshake outcomes.

B.5. Why SCITT for Receipts

Session Receipts gain value through public auditability. SCITT [SCITT] provides the transparency infrastructure. Receipts published to SCITT cannot be tampered with after the fact without detection.

Appendix C. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, per the guidelines of RFC 7942. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Note to RFC Editor: this section may be removed prior to final publication.

C.1. ATN Reference Prototype

- * Organization: Independent (E. Somoza).
- * Implementation Name: atn-prototype.
- * Description: A TypeScript prototype that demonstrates the four-message handshake, the capability intersection algebra, Ed25519 signing and verification of artifacts and messages, and the digest-bound artifact reference model.

- * License: CC0 1.0 Universal (public domain).
- * Maturity: prototype. Not production-ready. Not security-audited.
- * Coverage: Capability Manifest types, Delegation Chain types, Provenance Attestation types, Session Receipt, ATH state machine (HELLO, OFFER, ACCEPT, RECEIPT), capability intersection algebra per Section 9, version negotiation with downgrade protection. DNS resolution, HTTPS artifact fetch, and SCITT submission are stubbed.
- * Verified behavior: end-to-end handshake between two synthetic agents with mismatched capability constraints produces a signed Session Receipt with negotiated scope exactly matching the algorithm in Section 9. Numeric constraints take minimums, list constraints take intersections, refusals override.
- * Repository: <https://github.com/e-somoza/atn-prototype>

Appendix D. Acknowledgments

This document builds on the substantial work of the DNS-based agent discovery community, including the authors of AID, DNS-AID, ANS v2, DN-ANR, and BANDAID. ATN is intended to complement, not compete with, that work.

Author's Address

Enrique Somoza
Independent
United States of America
Email: enrique@somoza.co