

Domain Name System
Internet-Draft
Intended status: Informational
Expires: 3 September 2026

Y. Cui
Tsinghua University
2 March 2026

DNS-Native AI Agent Naming and Resolution
draft-cui-dns-native-agent-naming-resolution-01

Abstract

This document specifies DNS-Native Agent Naming and Resolution (DN-ANR) for AI agents. DN-ANR has three goals: (1) use domain names (FQDNs) as stable Agent Identifiers, (2) resolve Agent Identifiers to verifiable endpoints and supported protocol/version information with a cryptographic integrity chain (DNSSEC preferred), and (3) provide only minimal and stable pointer/index capabilities that can be referenced by upper-layer discovery systems. DN-ANR intentionally does not carry heavy semantic metadata in DNS, and does not define semantic discovery, ranking, or routing decisions.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at <https://nobrowning.github.io/dns-native-agent-naming-resolution/draft-cui-dns-native-agent-naming-resolution.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-cui-dns-native-agent-naming-resolution/>.

Discussion of this document takes place on the Domain Name System Working Group mailing list (<mailto:namedroppers@nic.ddn.mil>), which is archived at nicfs.nic.ddn.mil:~/namedroppers/*.Z.

Source for this draft and an issue tracker can be found at <https://github.com/nobrowning/dns-native-agent-naming-resolution>.

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 3 September 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. Goals	4
1.2. Non-Goals	4
2. Conventions and Definitions	5
3. Design Principles	5
4. Architecture Overview	6
5. Naming and Resource Location	6
5.1. Domain Name as Identity	6
5.1.1. Naming Rules	6
5.1.2. Naming Examples	7
5.2. Resource Location via DNS	7
6. DNS Record Design	7
6.1. Mandatory DNS Data (MUST)	7
6.2. Recommended DNS Data (SHOULD)	8
6.3. Optional DNS Data (MAY)	8
6.4. TXT Record: Identity Anchor (Conditional Metadata)	8
6.4.1. TXT Record Format	9
6.4.2. TXT Field Descriptions	9
6.5. SVCB Record: Version Distribution and Protocol Negotiation	10
6.5.1. SVCB Record Example	10
6.6. Version and Protocol Resolution	10
6.6.1. SVCB Private Parameters	11
6.6.2. ALPN Usage	11
6.6.3. Relationship Between Version and Protocol	11

6.6.4.	External Descriptor Locator and Digest in TXT (Optional)	12
6.6.5.	Interoperability Gating for Descriptor-Dependent Clients	13
7.	Performance and Determinism	13
7.1.	Why Address Hints	13
7.2.	Recommended SVCB Publication Strategy	13
7.3.	TTL Guidance	13
8.	HTTPS Fallback Mechanism	13
8.1.	agent-dns.json	14
8.1.1.	Media Type	14
8.1.2.	JSON Signature	14
8.1.3.	JSON Schema Definition	14
8.1.4.	File Structure Example	16
8.1.5.	JSON Signature Computation	17
8.1.6.	JSON Signature Verification	18
8.2.	Design Principles	18
8.3.	Applicable Scenarios	18
9.	Security	19
9.1.	Security Model Overview	19
9.1.1.	DNSSEC-based Security (RECOMMENDED)	19
9.1.2.	DNSSEC Deployment Recommendations	19
9.1.3.	Signature-based Security (OPTIONAL but RECOMMENDED)	20
9.1.4.	Choosing a Security Mechanism	21
9.2.	SVCB Integrity Digest (Optional)	22
9.2.1.	SVCB Canonicalization	22
9.2.2.	Digest Computation	23
9.3.	Signature Specification	23
9.3.1.	Public Key Requirements	23
9.3.2.	Signature Input Construction	24
9.3.3.	Signature Generation	25
9.3.4.	Signature Verification Procedure	25
9.3.5.	TLS Certificate Binding Verification (Option 1 Only)	26
10.	Implementation Checklist	26
10.1.	For Agent Publishers	26
10.2.	For Client Developers	26
10.3.	DNS Record Configuration Example	27
11.	Security Considerations	27
11.1.	Threat Model	28
11.2.	Mandatory Security Requirements	28
11.3.	Deployment Recommendations	28
11.4.	Specification Scope	29
12.	IANA Considerations	29
13.	References	29
13.1.	Normative References	30
13.2.	Informative References	30

Acknowledgments	31
Author's Address	31

1. Introduction

The emergence of AI agents as autonomous software entities creates concrete requirements for naming, trusted resolution, and endpoint verification. Existing deployments often mix discovery, semantic matching, and resolution into one control plane, which increases coupling and weakens interoperability.

This document defines DN-ANR as a DNS-native resolution layer built on [RFC1035] and Service Binding (SVCB/HTTPS RRs, [RFC9460], [RFC9461]). The design objective is strict scope control: discovery systems produce candidate Agent Identifiers, while DN-ANR securely resolves a chosen Agent Identifier into connection material.

1.1. Goals

DN-ANR goals are:

1. **Identity naming**: use domain names/FQDNs as administratively managed Agent Identifiers.
2. **Trusted resolution and connection guidance**: resolve an Agent Identifier to endpoint(s), protocol/version declarations, and verifiable integrity material.
3. **Foundational support for discovery systems**: expose only minimal stable pointers/indexes that upper-layer discovery systems MAY reference.

1.2. Non-Goals

DN-ANR non-goals are:

- * DN-ANR does not provide cross-domain agent discovery by semantics or capability.
- * DN-ANR does not provide semantic matching, capability ranking, or task-routing decisions.
- * DN-ANR does not standardize heavy capability metadata schemas inside DNS.
- * DN-ANR only specifies how to securely and deterministically connect after an Agent Identifier has been selected.

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.

The following terms are used throughout this document:

Agent: An autonomous software entity capable of communicating with other agents or humans using defined protocols.

Agent Identifier: A Fully Qualified Domain Name (FQDN) that uniquely identifies an agent.

Agent Protocol: The application-layer protocol used for agent-to-agent communication (e.g., [A2A], [ANP]).

3. Design Principles

This specification follows five core principles:

Principle	Description
DNS-First	DNS is the authoritative source for Agent Identifier resolution; HTTP serves only as a fallback mirror
Layered Scope	Discovery and semantic selection are out of scope; DN-ANR resolves selected identifiers
Path-Independent	Version and endpoint selection are controlled by DNS, not URL paths
Protocol Autonomy	Agent interaction protocols are decoupled from transport
Default Availability	A/AAAA records guarantee minimum connectivity; enhanced features are optional. Default version is provided when not specified

Table 1

4. Architecture Overview

DN-ANR is a resolution-layer specification inside a three-layer architecture:

Layer	Examples	Scope
Discovery Layer	Web registry, agent gateway, search engine, semantic router, DNS-SD/mDNS	OUT OF SCOPE
Resolution Layer	Agent Identifier (FQDN) -> endpoint, protocol/version, integrity material	DN-ANR scope
Connection Layer	A2A, MCP, HTTPS, gRPC, other application protocols	OUT OF SCOPE

Table 2

Interface boundary:

- * Discovery Layer outputs one or more candidate Agent Identifiers (FQDNs).
- * DN-ANR takes one selected Agent Identifier and resolves it into verifiable connection guidance.
- * Connection Layer consumes DN-ANR outputs and executes protocol-specific session logic.

5. Naming and Resource Location

5.1. Domain Name as Identity

Each agent is uniquely identified by a stable Fully Qualified Domain Name (FQDN). Domain ownership combined with TLS certificates forms the foundation of agent identity.

5.1.1. Naming Rules

- * Use domain names or subdomains owned by the organization
- * Agent version changes do not introduce new identities
- * No registration with any central authority is required

5.1.2. Naming Examples

```
# Recommended: dedicated subdomains
translator.agents.example.com
assistant.ai.company.com
agent123.agents.example.com
```

5.2. Resource Location via DNS

This specification does not use URL paths for version expression.
All version and endpoint selection is controlled by DNS records:

1. Obtain (from Discovery Layer or local policy) a candidate Agent Identifier (FQDN)
2. Query DNS SVCB records -> obtain version, endpoint, and protocol information
3. If SVCB is unavailable, use A/AAAA resolution of the Agent Identifier as the default Agent Endpoint
4. Query DNS TXT records (if present) -> obtain optional application-layer security and descriptor metadata
5. Apply local security policy (e.g., DNSSEC validation and/or TXT signature validation)
6. Connect to the selected endpoint and interact according to the selected protocol specification

DN-ANR provides only deterministic resolution and verification for an already-selected identifier; it does not perform semantic discovery or ranking.

6. DNS Record Design

This specification keeps DNS payloads minimal and operationally stable. DNS data is classified as MUST/SHOULD/MAY to separate core resolution from optional optimization.

6.1. Mandatory DNS Data (MUST)

- * *A/AAAA* [RFC1035]: provide baseline reachability and interoperability for resolvers and clients.

Rationale: A/AAAA guarantees minimum connectability and provides a default Agent Endpoint when no SVCB policy is available.

6.2. Recommended DNS Data (SHOULD)

- * ***SVCB*** [RFC9460] [RFC9461] with endpoint parameters and address hints (ipv4hint, ipv6hint): provide deterministic endpoint selection, protocol/version signaling, and reduced lookup latency.
- * ***DNSSEC*** [RFC4033]: provide origin authentication and integrity for DNS RRsets.
- * ***TXT identity anchor*** [RFC1035]: publish optional application-layer security metadata and optional descriptor pointers.

Rationale: SVCB and DNSSEC substantially improve determinism, performance, and security. TXT metadata supports alternative or additional security models and descriptor linkage when needed.

6.3. Optional DNS Data (MAY)

- * ***TXT signature fields*** (alg, pk, sig): used when signature-based verification is enabled.
- * ***TXT SVCB integrity digest*** (svcb-digest): optional integrity cross-check material, especially for HTTPS fallback workflows.
- * ***TXT descriptor pointer fields*** (agent-desc, agent-desc-sha256): pointer + digest for heavy external metadata.

Rationale: Heavy metadata evolves quickly and can grow large; keeping it out of DNS preserves DNS efficiency while retaining verifiable linkage.

6.4. TXT Record: Identity Anchor (Conditional Metadata)

TXT records [RFC1035] provide optional application-layer metadata. Their responsibilities are strictly limited to:

1. Declare identity metadata (e.g., v, kid)
2. Optionally publish key/signature material (alg, pk, sig) for signature-based security
3. Optionally publish SVCB digest (svcb-digest) for integrity cross-check, especially with HTTPS fallback
4. Optionally publish external descriptor pointer metadata (agent-desc, agent-desc-sha256)

6.4.1. TXT Record Format

```

_agent.translator.example.com. IN TXT (
  "v=1;"
  "kid=key-2025-01;"
  "alg=Ed25519;" ; OPTIONAL
  "pk=base64-encoded-public-key;" ; OPTIONAL
  "sig=base64-encoded-signature;" ; OPTIONAL
  "svcb-digest=base64-encoded-sha256-digest;" ; OPTIONAL
  "agent-desc=https://translator.example.com/
    .well-known/agent-descriptor.json;" ; OPTIONAL
  "agent-desc-sha256=x48E9qOokqqr=" ; OPTIONAL
)

```

6.4.2. TXT Field Descriptions

Field	Description
v	Version identifier, fixed as 1
kid	Key identifier, used for key rotation
alg	Signature algorithm: Ed25519 (RECOMMENDED) or ES256 (OPTIONAL; REQUIRED when sig is present)
pk	Base64-encoded public key (OPTIONAL; REQUIRED when sig is present)
sig	Signature over selected TXT content (OPTIONAL; used in signature-based security mode)
svcb-digest	Base64-encoded SHA-256 digest of canonicalized SVCB records (OPTIONAL; useful for HTTPS fallback integrity cross-check)
agent-desc	Descriptor URI for external heavy metadata (OPTIONAL)
agent-desc-sha256	Base64-encoded SHA-256 digest of descriptor content (OPTIONAL; RECOMMENDED when agent-desc is present)

Table 3

6.5. SVCB Record: Version Distribution and Protocol Negotiation

SVCB (Service Binding) records [RFC9460] are the core resolution mechanism, serving the following responsibilities:

Level	SVCB Role
Service Location	TargetName + port specify the service endpoint
Version Distribution	Private SvcParam declares agent version
Protocol Negotiation	Private parameters declare supported agent protocols
Performance Optimization	ipv4hint / ipv6hint reduce additional address lookups

Table 4

6.5.1. SVCB Record Example

```
# Complete SVCB record example
_agent.translator.example.com. IN SVCB 1 agent-v3.example.com. (
  alpn=h2
  port=443
  ipv4hint=203.0.113.50
  ipv6hint=2001:db8::50
  key65480="v3"           ; Agent version
  key65481="a2a,anp"      ; Supported agent protocols
)

# v2 version (lower priority)
_agent.translator.example.com. IN SVCB 2 agent-v2.example.com. (
  alpn=h2
  port=443
  ipv4hint=203.0.113.51
  key65480="v2"
  key65481="a2a"
)
```

6.6. Version and Protocol Resolution

6.6.1. SVCB Private Parameters

This specification introduces private SVCB parameters (SvcParam) as defined in [RFC9460]:

Parameter	Semantics	Example
key65480	Agent version	"v3", "v2.1.0"
key65481	Agent protocols	"a2a", "a2a,anp"

Table 5

6.6.1.1. Version Selection Behavior

Clients can:

- * ***Default selection***: When version is not specified, the highest priority version based on SVCB priority is used
- * ***Specific selection***: Specify key65480 value to select a particular version
- * ***Protocol filtering***: Select only versions supporting specific protocols (key65481)

6.6.2. ALPN Usage

ALPN is used for TLS-layer protocol negotiation (e.g., h2, h3). Agent interaction protocols ([A2A], [ANP]) are declared via SVCB private parameters (key65481), not ALPN values. This ensures compatibility with existing TLS ecosystems and reserves space for future IANA registration.

6.6.3. Relationship Between Version and Protocol

This specification clearly distinguishes two layers:

Layer	Declaration Location	Example
Agent Version	key65480	v3, v2.1.0
Agent Protocol	key65481	a2a, anp

Table 6

6.6.4. External Descriptor Locator and Digest in TXT (Optional)

DN-ANR supports optional linkage to heavy external metadata while keeping DNS payloads minimal:

- * agent-desc in TXT contains an absolute URI that identifies a descriptor resource.
- * agent-desc-sha256 in TXT contains the SHA-256 digest of the descriptor in Base64 encoding.

DN-ANR standardizes only:

- * URI syntax and transport locator semantics.
- * Digest algorithm (SHA-256) and digest encoding.
- * Client verification flow (fetch descriptor -> compute digest -> compare -> consume).

DN-ANR does not standardize descriptor content schema (capability model, OpenAPI, model card, I/O schema, etc.).

Descriptor digest computation rules:

1. Fetch descriptor bytes from the URI in agent-desc.
2. If the descriptor media type is JSON, canonicalize using JCS [RFC8785] before hashing.
3. For non-JSON media types, hash the raw octet stream as retrieved.
4. Compute SHA-256 and Base64-encode the result.
5. Compare with agent-desc-sha256; mismatch MUST be treated as verification failure.

6.6.5. Interoperability Gating for Descriptor-Dependent Clients

- * A client that depends on descriptor data MUST require agent-desc; otherwise it MUST treat descriptor-based logic as unavailable.
- * A client that requires descriptor-integrity verification MUST require both agent-desc and agent-desc-sha256.
- * A publisher that wants interoperable descriptor verification SHOULD publish both TXT fields together.

7. Performance and Determinism

7.1. Why Address Hints

SVCB ipv4hint and ipv6hint improve resolution behavior by:

- * reducing extra A/AAAA lookup round-trips;
- * improving first-connection determinism;
- * reducing resolver-path jitter under recursive caching variance.

7.2. Recommended SVCB Publication Strategy

- * Publishers SHOULD keep each SVCB RRSSet compact and avoid excessive per-version record expansion.
- * When many versions exist, publishers SHOULD keep only stable externally supported versions in DNS and move detailed capability/version matrices to external descriptors (agent-desc + agent-desc-sha256 in TXT).
- * Publishers SHOULD keep endpoint migration agility by using shorter TTLs for SVCB than TXT.

7.3. TTL Guidance

- * Identity anchors (TXT) SHOULD use relatively longer TTL values.
- * Endpoint/control-plane records (SVCB) SHOULD use relatively shorter TTL values to support endpoint migration and rapid rollback.

8. HTTPS Fallback Mechanism

To ensure "works by default" behavior, this specification introduces an optional but strongly recommended fallback mechanism.

8.1. agent-dns.json

For clients that do not support SVCB queries, agents can publish a JSON mirror of DNS records at an HTTPS endpoint:

```
https://{agent-id}/.well-known/agent-dns.json
```

8.1.1. Media Type

The agent-dns.json file MUST be served with the following HTTP headers:

```
Content-Type: application/json; charset=utf-8
Cache-Control: max-age=300
```

Servers SHOULD set an appropriate Cache-Control header. A value between 300 seconds (5 minutes) and 3600 seconds (1 hour) is RECOMMENDED.

8.1.2. JSON Signature

The JSON file MUST include a signature for integrity protection. Unlike the TXT record signature which covers only TXT fields, the JSON signature covers the complete service binding information.

The signature is computed over the canonical JSON representation of the document (excluding the sig field) as defined in [RFC8785] (JSON Canonicalization Scheme).

8.1.3. JSON Schema Definition

The agent-dns.json file MUST conform to the following JSON Schema:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/agent-dns.schema.json",
  "title": "Agent DNS JSON",
  "description": "Mirror of DNS records for agent resolution",
  "type": "object",
  "required": ["agentId", "txt", "sig"],
  "properties": {
    "agentId": {
      "type": "string",
      "description": "The FQDN identifying the agent",
      "pattern": "^[a-zA-Z0-9]([a-zA-Z0-9-]*[a-zA-Z0-9])?(\.\\.[a-zA-Z0-9]([a-zA-Z0-9-]*[a-zA-Z0-9])?)*$"
    },
    "txt": {
```

```
"type": "object",
"description": "Core identity fields from DNS TXT record",
"required": ["v", "kid"],
"properties": {
  "v": {
    "type": "string",
    "const": "1"
  },
  "kid": {
    "type": "string",
    "description": "Key identifier"
  },
  "alg": {
    "type": "string",
    "enum": ["ES256", "Ed25519"],
    "description": "Signature algorithm"
  },
  "pk": {
    "type": "string",
    "description": "Base64-encoded TLS certificate public key"
  }
},
"svcb": {
  "type": "array",
  "description": "Mirror of DNS SVCB records",
  "items": {
    "type": "object",
    "required": ["priority", "target", "port"],
    "properties": {
      "priority": {
        "type": "integer",
        "minimum": 1,
        "maximum": 65535
      },
      "target": {
        "type": "string",
        "description": "Target hostname"
      },
      "port": {
        "type": "integer",
        "minimum": 1,
        "maximum": 65535
      },
      "alpn": {
        "type": "array",
        "items": {
          "type": "string"
        }
      }
    }
  }
}
```

```
    }
  },
  "agentVersion": {
    "type": "string",
    "description": "Agent version (mirrors key65480)"
  },
  "agentProtocols": {
    "type": "array",
    "items": {
      "type": "string"
    },
    "description": "Supported protocols (mirrors key65481)"
  }
}
},
"sig": {
  "type": "string",
  "description": "Base64-encoded signature over canonical JSON
                  (excluding sig field)"
}
}
```

8.1.4. File Structure Example

```

{
  "agentId": "translator.example.com",
  "txt": {
    "v": "1",
    "kid": "key-2025-01",
    "alg": "ES256",
    "pk": "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE..."
  },
  "svcb": [
    {
      "priority": 1,
      "target": "agent-v3.example.com",
      "port": 443,
      "alpn": ["h2"],
      "agentVersion": "v3",
      "agentProtocols": ["a2a", "anp"]
    },
    {
      "priority": 2,
      "target": "agent-v2.example.com",
      "port": 443,
      "alpn": ["h2"],
      "agentVersion": "v2",
      "agentProtocols": ["a2a"]
    }
  ],
  "sig": "MEUCIQC7..."
}

```

8.1.5. JSON Signature Computation

The signature over the JSON file is computed as follows:

1. Construct the JSON object without the sig field.
2. Serialize using JSON Canonicalization Scheme (JCS) as defined in [RFC8785].
3. Compute the signature using the TLS private key.
4. Encode the signature using Base64.

```

json_without_sig = { agentId, txt, svcb }
canonical_json = JCS(json_without_sig)
signature = Sign(TLS_private_key, UTF-8(canonical_json))
sig = Base64Encode(signature)

```

8.1.6. JSON Signature Verification

Clients MUST verify the JSON signature:

1. Fetch the JSON file over HTTPS.
2. Extract the sig field and remove it from the object.
3. Serialize the remaining object using JCS.
4. Obtain the public key from the txt.pk field in the JSON.
5. Verify the signature.
6. (RECOMMENDED for TLS-based signing) Verify that txt.pk matches the TLS certificate's public key.

If verification fails, the client MUST reject the JSON file.

Note: When agent providers use separate key pairs (not TLS-based), the verification in step 6 is not applicable. In such cases, the integrity of the JSON file depends on the authenticity of the public key in txt.pk, which has the same trust anchor limitations as described in the Security Model Overview.

8.2. Design Principles

- * ***Mirror, not addition***: JSON only mirrors information already in DNS; it does not introduce content absent from DNS
- * ***DNS remains authoritative***: HTTPS JSON is only a "readable mirror", not a new authoritative source
- * ***Signature required***: JSON files MUST be signed for integrity protection
- * ***Schema validated***: Clients SHOULD validate JSON against the defined schema

8.3. Applicable Scenarios

- * Clients that do not support SVCB queries
- * Browsers / debugging tools
- * Early ecosystem transition period
- * Environments where DNS resolution is limited

9. Security

This section defines the security mechanisms for ensuring the integrity and authenticity of agent resolution data.

9.1. Security Model Overview

This specification provides two complementary mechanisms for ensuring integrity and authenticity of agent resolution data:

1. *DNSSEC (RECOMMENDED for Internet-facing deployments)*: Protocol-level cryptographic authentication of DNS data
2. *Signature-based Security (OPTIONAL)*: TXT key/signature validation (pk, sig) with optional digest cross-checks (svcb-digest); if HTTPS fallback JSON is used, fallback signature validation is REQUIRED

Mechanism	Protection Scope	Trust Anchor
DNSSEC	All DNS records (TXT, SVCB, A/AAAA)	DNS root zone
Signature-based Security	TXT signed fields, optional SVCB digest consistency, JSON fallback signature (when fallback is used)	Web PKI (when using TLS keys) or self-declared (when using separate keys)

Table 7

9.1.1. DNSSEC-based Security (RECOMMENDED)

DNSSEC [RFC4033] provides cryptographic authentication of DNS data at the protocol level.

9.1.2. DNSSEC Deployment Recommendations

- * For publicly reachable agents, the authoritative zone SHOULD deploy DNSSEC.
- * When DNSSEC validation is available and the SVCB RRSet (or TXT RRSet, when used) validates as *bogus*, clients MUST treat resolution as failure (fail-closed) and MUST NOT use that endpoint.

- * Clients SHOULD apply stricter fail-closed behavior at least to SVCB and TXT (when TXT is part of the selected trust path).
- * In enterprise/private networks where DNSSEC is not deployed, operators MAY rely on TXT signatures and TLS certificate binding as a minimum trust baseline.

When DNSSEC is enabled:

- * All DNS records are signed by the zone's DNSSEC keys.
- * Clients with DNSSEC validation can verify record authenticity.
- * Application-layer signatures remain useful for defense in depth and for JSON fallback integrity.

9.1.3. Signature-based Security (OPTIONAL but RECOMMENDED)

This specification defines an optional but recommended signing mechanism for integrity protection. Agent providers have two options for key management:

9.1.3.1. Option 1: TLS Certificate Keys (RECOMMENDED)

Using the domain's TLS certificate keys provides a complete trust chain:

- * Uses the domain's TLS certificate private key for signing
- * Public key is published in the TXT record (pk field)
- * Enables verification through the established Web PKI trust chain
- * Clients can verify that pk matches the TLS certificate presented during HTTPS connection

When TLS-based signing is used:

1. The TXT record contains the TLS certificate's public key
2. A signature covers the selected TXT fields
3. svcb-digest MAY be included as optional integrity cross-check material (especially for HTTPS fallback consistency checks)
4. Clients can verify the signature using the public key from the TLS certificate chain

9.1.3.2. Option 2: Separate Key Pair

Agent providers MAY use a separate key pair (not derived from TLS certificates) for signing:

- * Agent provider generates and manages their own key pair
- * Public key is published in the TXT record (pk field)
- * Signature is computed using the corresponding private key

***Trust Anchor Limitation*:** When using separate keys, the trust anchor is limited to the TXT record itself. If the TXT record is tampered with (e.g., via DNS spoofing or cache poisoning), an attacker could replace both the public key and signature, rendering the integrity protection ineffective. This is because:

- * The public key in the TXT record is self-declared without external verification
- * Clients have no independent trust anchor to verify the authenticity of the public key
- * SVCB records and agent-dns.json cannot be reliably verified if the TXT record is compromised

For this reason, when using separate keys:

- * DNSSEC deployment becomes more important to protect the TXT record itself
- * Clients SHOULD treat records from non-DNSSEC zones with appropriate caution
- * Out-of-band key distribution mechanisms MAY be used to establish trust

9.1.4. Choosing a Security Mechanism

Scenario	Recommended Approach
DNSSEC fully deployed	DNSSEC alone is sufficient
DNSSEC not available	Use TLS-based signing (Option 1)
High security requirements	Use both DNSSEC and signing (defense-in-depth)

HTTPS fallback required	JSON signing and/or svcb-digest consistency checks are recommended
Separate keys without DNSSEC	Limited trust; consider additional verification mechanisms

Table 8

9.2. SVCB Integrity Digest (Optional)

When svcb-digest is present in TXT, SVCB records can be cross-checked for integrity (for example, during HTTPS fallback reconciliation). This section defines the canonicalization and digest computation procedures.

9.2.1. SVCB Canonicalization

To compute the svcb-digest, SVCB records MUST be canonicalized as follows:

9.2.1.1. Step 1: Collect and Sort

1. Collect all SVCB records for the agent's _agent prefix.
2. Exclude AliasMode records (priority = 0).
3. Sort records by priority in ascending order (lowest first).
4. If priorities are equal, sort by TargetName lexicographically.

9.2.1.2. Step 2: Normalize Each Record

For each SVCB record, construct a canonical string in the following format:

<priority> <target> <params>

Where: - priority: Decimal integer with no leading zeros - target: Fully qualified domain name in lowercase, with trailing dot removed - params: SvcParams in sorted order by key number, formatted as key=value

9.2.1.3. Step 3: SvcParam Normalization

SvcParams MUST be normalized as follows:

1. Sort by SvcParamKey number (ascending).
2. Format each parameter as: key<number>=<value>
3. String values are enclosed in double quotes.
4. List values (e.g., alpn) use comma separation with no spaces.
5. Separate parameters with a single space.

9.2.1.4. Canonical Format Example

Original SVCB records:

```
_agent.translator.example.com. IN SVCB 2 agent-v2.example.com. (  
  alpn=h2 port=443 key65480="v2" key65481="a2a"  
)  
_agent.translator.example.com. IN SVCB 1 agent-v3.example.com. (  
  alpn=h2 port=443 key65480="v3" key65481="a2a,anp"  
)
```

Canonical representation (sorted by priority):

```
1 agent-v3.example.com key1=h2 key3=443 key65480="v3" key65481="a2a,anp"  
2 agent-v2.example.com key1=h2 key3=443 key65480="v2" key65481="a2a"
```

Note: alpn is SvcParamKey 1, port is SvcParamKey 3 as defined in [RFC9460].

9.2.2. Digest Computation

```
canonical_svcb = <line1> + "\n" + <line2> + "\n" + ...  
digest_bytes = SHA-256(UTF-8(canonical_svcb))  
svcb-digest = Base64Encode(digest_bytes)
```

The resulting svcb-digest is approximately 44 characters (32 bytes encoded in Base64).

9.3. Signature Specification

This section defines the signature mechanism when signature-based security is used.

9.3.1. Public Key Requirements

The pk field contains the public key used for signature verification. There are two options:

9.3.1.1. When Using TLS Certificate Keys (RECOMMENDED)

The pk field MUST contain the public key from the domain's TLS certificate:

1. Extract the SubjectPublicKeyInfo from the TLS certificate.
2. Encode using Base64 [RFC4648].
3. The certificate MUST be valid for the agent's domain name.

9.3.1.2. When Using Separate Key Pair

The pk field contains the agent provider's self-managed public key:

1. Generate a key pair using a supported algorithm.
2. Extract the public key in SubjectPublicKeyInfo format.
3. Encode using Base64 [RFC4648].

Note: When using separate keys, the public key is self-declared and lacks an independent trust anchor. See Security Model Overview for implications.

9.3.1.3. Supported Key Types

- * EC P-256 (for ES256 algorithm) - RECOMMENDED
- * Ed25519 (for Ed25519 algorithm)

9.3.2. Signature Input Construction

When signature-based TXT validation is used, the signature input MUST be constructed from TXT fields as follows:

1. Include required fields in this exact order: v, kid, alg, pk.
2. If present, append optional fields in this exact order: svcb-digest, agent-desc, agent-desc-sha256.
3. Use key=value pairs separated by semicolons, with no trailing semicolon.

```
signing_input = "v=" + v + ";kid=" + kid + ";alg=" + alg + ";pk=" + pk
if svcb-digest present: signing_input += ";svcb-digest=" + svcb-digest
if agent-desc present: signing_input += ";agent-desc=" + agent-desc
if agent-desc-sha256 present: signing_input += ";agent-desc-sha256="
                                + agent-desc-sha256
```

Example: ~~~ v=1;kid=key-2025-01;alg=ES256;pk=MFkwEwYHkoZI...;agent-desc=https://translator.example.com/.well-known/agent-descriptor.json
~~~~

### 9.3.3. Signature Generation

```
signature_bytes = Sign(private_key, UTF-8(signing_input))
sig = Base64Encode(signature_bytes)
```

Where `private_key` is either: - The TLS certificate's private key (Option 1, RECOMMENDED), or - The agent provider's separately managed private key (Option 2)

For ES256: signature is 64 bytes (r || s format), resulting in 88 Base64 characters. For Ed25519: signature is 64 bytes, resulting in 88 Base64 characters.

### 9.3.4. Signature Verification Procedure

Clients MUST perform the following steps:

1. Parse TXT record and extract `v`, `kid`, `alg`, `pk`, `sig`, and any optional signed fields present.
2. Reconstruct `signing_input` using the required/optional field ordering defined above.
3. Decode `pk` from Base64 to obtain the public key.
4. Decode `sig` from Base64 to obtain the signature bytes.
5. Verify the signature using the specified algorithm.
6. (RECOMMENDED for Option 1) Verify that `pk` matches the TLS certificate presented during connection.

If verification fails, the client MUST reject the TXT record.

When using Option 2 (separate key pair), clients should be aware that the signature only proves consistency between the TXT record content and the private key holder. Without DNSSEC or TLS binding, there is no external trust anchor to verify the key's authenticity.

### 9.3.5. TLS Certificate Binding Verification (Option 1 Only)

When TLS certificate keys are used (Option 1), clients SHOULD verify that the pk in the TXT record matches the server's TLS certificate:

1. Establish TLS connection to the agent's domain.
2. Extract the public key from the server's certificate.
3. Compare with the pk field in the TXT record.
4. If mismatch, treat as verification failure.

This binding ensures that the entity controlling the TLS private key is the same entity that published the DNS records.

Note: This verification is not applicable when separate key pairs are used (Option 2), as the pk in the TXT record will not match the TLS certificate.

## 10. Implementation Checklist

### 10.1. For Agent Publishers

1. Prepare domain name, configure HTTPS and TLS certificate
2. Configure DNS A/AAAA records (basic connectivity)
3. (OPTIONAL) Configure DNS TXT record (\_agent.xxx) for signature metadata (alg/pk/sig), svcb-digest, and/or descriptor pointer fields
4. Configure DNS SVCB records with endpoint, protocol/version, and (SHOULD) address hints
5. (OPTIONAL) Publish descriptor URI + digest in TXT (agent-desc, agent-desc-sha256) for heavy metadata externalization
6. (RECOMMENDED) Publish /.well-known/agent-dns.json fallback file
7. (RECOMMENDED for public deployments) Enable DNSSEC

### 10.2. For Client Developers

1. Query SVCB records, parse version and endpoint information
2. If SVCB is unavailable, use A/AAAA of the Agent Identifier as the default endpoint

3. Query TXT records (if present), parse optional fields (pk, sig, svcb-digest, agent-desc, agent-desc-sha256)
4. If descriptor fields are present and required by local policy, fetch descriptor and verify digest before use
5. (Fallback) If SVCB unavailable, fetch agent-dns.json when needed
6. Connect to endpoint per agent protocol (key65481) specification
7. Validate DNSSEC when present, and fail closed for bogus SVCB/TXT results that are part of the selected trust path

### 10.3. DNS Record Configuration Example

```
; Basic connectivity
translator.example.com.      IN A      203.0.113.50
translator.example.com.      IN AAAA    2001:db8::50

; Optional TXT identity/security/descriptor metadata
_agent.translator.example.com. IN TXT    "v=1;kid=key-2025-01;
                                     alg=Ed25519;pk=...;sig=...;
                                     svcb-digest=...;
                                     agent-desc=https://translator.example.com/
                                     .well-known/agent-descriptor.json;
                                     agent-desc-sha256=x48E9qOokqqr7kbu9DBPE="

; Version resolution (SVCB)
_agent.translator.example.com. IN SVCB 1 agent-v3.example.com. (
  alpn=h2 port=443
  ipv4hint=203.0.113.50 ipv6hint=2001:db8::50
  key65480="v3" key65481="a2a,anp"
)
_agent.translator.example.com. IN SVCB 2 agent-v2.example.com. (
  alpn=h2 port=443 ipv4hint=203.0.113.51 key65480="v2" key65481="a2a"
)
```

## 11. Security Considerations

This specification uses DNS as the authoritative source for agent resolution and identity information. Its security objectives are to ensure the authenticity, integrity, and verifiability of resolution results, rather than evaluating agent service quality or behavioral trustworthiness.

### 11.1. Threat Model

This specification primarily considers the following threats:

- \* DNS poisoning or cache pollution leading to incorrect endpoint resolution
- \* Tampering with resolution results to redirect clients to unintended endpoints
- \* Downgrade attacks inducing clients to use older versions or weaker protocols
- \* Trust violations caused by expired or replaced identity declarations

### 11.2. Mandatory Security Requirements

To address the above threats, this specification mandates:

- \* Clients MUST establish at least one validated integrity path before endpoint use: DNSSEC validation, or TXT signature verification when TXT signing fields are used
- \* Clients MUST perform TXT-SVCB consistency checks when svcb-digest is present and selected by local policy
- \* Clients MUST use TLS [RFC8446] and verify server certificates [RFC9525]
- \* Clients MUST NOT use endpoints that fail verification
- \* Agents that publish svcb-digest or TXT signatures over endpoint-related metadata MUST synchronously update TXT and SVCB information when versions or endpoints change

### 11.3. Deployment Recommendations

- \* For Internet-facing agent domains, authoritative operators SHOULD enable DNSSEC [RFC4033].
- \* If DNSSEC data is present and validates as bogus for SVCB (or TXT, when TXT is part of the selected trust path), clients MUST fail closed for that endpoint.
- \* For private/enterprise deployments without DNSSEC, clients SHOULD require TXT signature verification and TLS certificate validation as minimum controls.

- \* This specification does not require DNSSEC as the only trust mechanism; deployments MAY combine DNSSEC and signature-based protections.

#### 11.4. Specification Scope

This specification guarantees the following properties:

- \* Verifiability of agent identity
- \* Integrity and consistency of resolution results
- \* Encryption and tamper-proofing of connections

This specification does NOT attempt to address:

- \* Agent capability authenticity
- \* Service quality (SLA) or behavioral compliance
- \* Agent reputation or governance issues

These concerns should be handled by upper-layer protocols, operational frameworks, or governance mechanisms.

#### 12. IANA Considerations

This document requests IANA registration of the following SvcParamKeys:

| Number | Name            | Meaning                                           | Reference     |
|--------|-----------------|---------------------------------------------------|---------------|
| 65480  | agent-version   | Agent version identifier                          | This document |
| 65481  | agent-protocols | Comma-separated list of supported agent protocols | This document |

Table 9

Note: The values 65480-65481 are in the private use range (65280-65534) as defined in [RFC9460]. Upon publication, these should be replaced with IANA-assigned values from the Expert Review range.

#### 13. References

### 13.1. Normative References

- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/rfc/rfc1035>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/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>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/rfc/rfc4648>>.
- [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>>.
- [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>>.
- [RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <<https://www.rfc-editor.org/rfc/rfc8785>>.
- [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>>.
- [RFC9461] Schwartz, B., "Service Binding Mapping for DNS Servers", RFC 9461, DOI 10.17487/RFC9461, November 2023, <<https://www.rfc-editor.org/rfc/rfc9461>>.
- [RFC9525] Saint-Andre, P. and R. Salz, "Service Identity in TLS", RFC 9525, DOI 10.17487/RFC9525, November 2023, <<https://www.rfc-editor.org/rfc/rfc9525>>.

### 13.2. Informative References

[A2A] Google, "Agent2Agent Protocol (A2A)", 2025,  
<<https://google.github.io/A2A/>>.

[ANP] ANP Community, "Agent Network Protocol (ANP)", 2025,  
<<https://agent-network-protocol.com/>>.

#### Acknowledgments

#### Author's Address

Yong Cui  
Tsinghua University  
Beijing, 100084  
China  
Email: [cuiyong@tsinghua.edu.cn](mailto:cuiyong@tsinghua.edu.cn)  
URI: <http://www.cuiyong.net/>