WG Working Group
Internet-Draft
Intended status: Informational
Expires: 25 June 2026

Y. Cui
Tsinghua University

22 December 2025

DNS-Native AI Agent Naming and Resolution
draft-cui-dns-native-agent-naming-resolution-00

Abstract

This document specifies a DNS-native naming and resolution mechanism for AI agents. Building upon the DNS specification (RFC 1035) and leveraging Service Binding (SVCB) records (RFC 9460, RFC 9461), it defines how AI agents are identified using Fully Qualified Domain Names (FQDNs), how their identity and cryptographic keys are published via DNS TXT records, and how multi-version, multi-protocol service resolution is achieved using DNS SVCB records. The design philosophy emphasizes DNS as the authoritative source, protocol autonomy, and graceful degradation for legacy clients. When version is not explicitly specified, the highest priority (default) version is provided. This approach maximizes reuse of existing Internet infrastructure while enabling the rapid evolution of AI agent ecosystems.

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 WG Working Group mailing list (mailto:WG@example.com), which is archived at https://example.com/WG.

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.

Copyright Notice

Table of Contents

1. Introduction

The emergence of AI agents as autonomous software entities capable of communicating with each other and with humans creates new requirements for naming, resolution, and identity verification. Existing approaches often rely on centralized registries or application-layer discovery mechanisms that introduce single points of failure and do not leverage the globally distributed, well-understood DNS infrastructure.

This document proposes a DNS-native approach that builds upon the foundational DNS specification [RFC1035] and leverages the Service Binding (SVCB) and HTTPS DNS Resource Records defined in [RFC9460] and [RFC9461] for advanced service resolution. The core design principles are:

* *DNS is the authoritative source*: DNS TXT and SVCB records serve as the single source of truth for agent identity and resolution, while HTTP-based mechanisms serve only as fallback mirrors.

* *Protocols are autonomous*: Agent interaction protocols (e.g., [A2A], [ANP]) are decoupled from transport, each defining its own interaction semantics.

* *Default availability*: Basic A/AAAA records [RFC1035] ensure minimum connectivity, while enhanced resolution capabilities through SVCB [RFC9460] are optional but recommended. When version is not explicitly specified, the highest priority (default) version is provided.

The key innovations of this specification include:

1. Use of SVCB records [RFC9460] [RFC9461] for multi-version distribution and protocol negotiation

2. Use of TXT records [RFC1035] as identity trust anchors with public key publication and SVCB integrity protection

3. An HTTPS-based fallback mechanism (agent-dns.json) for clients without SVCB support

4. Flexible security model supporting both DNSSEC and signing

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 four core principles:

Principle	Description
DNS-First	DNS is the sole authoritative source for agent resolution; HTTP serves only as a readable mirror
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. Naming and Resource Location

4.1. 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.

4.1.1. 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

4.1.2. Naming Examples

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

4.2. 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. Query DNS SVCB records -> obtain version, endpoint, and protocol information

2. Query DNS TXT records -> obtain public key and verify identity

3. Connect to the TargetName:port specified in SVCB

4. Interact according to the protocol specification

5. DNS Record Design

This specification uses three types of DNS records [RFC1035], each with distinct responsibilities. The SVCB record type is defined in [RFC9460] and [RFC9461]:

Record Type	Responsibility	Requirement
A / AAAA	Basic connectivity	REQUIRED - ensures minimum availability
TXT	Identity declaration, public key publication, SVCB integrity (svcb-digest), signature	REQUIRED - identity trust anchor
SVCB	Version distribution, endpoint resolution, protocol negotiation	RECOMMENDED - enhanced resolution

Table 2

5.1. TXT Record: Identity Trust Anchor

TXT records [RFC1035] serve as the sole trust anchor for agent identity and integrity. Their responsibilities are strictly limited to four functions:

1. Declare agent identity

2. Publish public key / fingerprint

3. Protect SVCB record integrity via digest (svcb-digest)

4. Sign identity and integrity metadata (tamper-proof)

5.1.1. TXT Record Format

```
_agent.translator.example.com. IN TXT (
  "v=1;"
  "kid=key-2025-01;"
  "alg=Ed25519;"
  "pk=base64-encoded-public-key;"
  "svcb-digest=base64-encoded-sha256-digest;"
  "sig=base64-encoded-signature"
)
```

5.1.2. 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
pk	Base64-encoded public key
svcb-digest	Base64-encoded SHA-256 digest of canonicalized SVCB records (OPTIONAL but RECOMMENDED when signing is used)
sig	Signature over record content (OPTIONAL but RECOMMENDED)

Table 3

5.2. 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

Table 4

5.2.1. SVCB Record Example

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

5.3. Version and Protocol Resolution

5.3.1. SVCB Private Parameters

This specification introduces two SVCB private parameters (SvcParam) as defined in [RFC9460] to declare version and protocol:

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

Table 5

5.3.1.1. 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)

5.3.2. 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.

5.3.3. 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. HTTPS Fallback Mechanism

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

6.1. 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

6.1.1. 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.

6.1.2. 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).

6.1.3. 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)"
      }
    }
  }
```

6.1.4. 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..."
  }
```

6.1.5. 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)
```

6.1.6. 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.

6.2. 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

6.3. Applicable Scenarios

* Clients that do not support SVCB queries

* Browsers / debugging tools

* Early ecosystem transition period

* Environments where DNS resolution is limited

7. Security

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

7.1. Security Model Overview

This specification provides two approaches for ensuring the integrity and authenticity of agent resolution data:

1. *DNSSEC (RECOMMENDED)*: Protocol-level cryptographic authentication of DNS data

2. *Signature-based Security (OPTIONAL but RECOMMENDED)*: Signing of TXT content, SVCB digest, and JSON fallback

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

Table 7

7.1.1. DNSSEC-based Security (RECOMMENDED)

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

* All DNS records are signed by the zone’s DNSSEC keys

* Clients with DNSSEC validation can verify record authenticity

* No application-layer signatures are required

DNSSEC is the RECOMMENDED approach for environments where it is fully deployed.

7.1.2. 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:

7.1.2.1. 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. The TXT record contains a digest of SVCB records (svcb-digest)
3. A signature covers the TXT content including the SVCB digest
4. Clients can verify the signature using the public key from the TLS certificate chain

7.1.2.2. 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

7.1.3. 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	Signing required for JSON integrity	
	Using separate keys without	Limited trust; consider additional verification

	DNSSEC	mechanisms

Table 8

7.2. SVCB Integrity Protection

When signature-based security is used, SVCB records are protected via a digest stored in the TXT record. This section defines the canonicalization and digest computation procedures.

7.2.1. SVCB Canonicalization

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

7.2.1.1. 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.

7.2.1.2. 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

7.2.1.3. 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.

7.2.1.4. 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].

7.2.2. 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).

7.3. Signature Specification

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

7.3.1. Public Key Requirements

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

7.3.1.1. 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.

7.3.1.2. 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.

7.3.1.3. Supported Key Types

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

* Ed25519 (for Ed25519 algorithm)

7.3.2. Signature Input Construction

The signature input MUST be constructed by concatenating the following fields in order, separated by semicolons, with no trailing semicolon:

```
signing_input = "v=" + v + ";kid=" + kid + ";alg=" + alg + ";
                  pk=" + pk + ";svcb-digest=" + svcb-digest
```

Example: ~~~ v=1;kid=key-2025-01;alg=ES256;pk=MFkwEwYHKoZI...;svcb-digest=K7gNU3sdo+OL... ~~~

7.3.3. 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.

7.3.4. Signature Verification Procedure

Clients MUST perform the following steps:

1. Parse TXT record and extract v, kid, alg, pk, svcb-digest, sig fields.

2. Reconstruct signing_input from v, kid, alg, pk, svcb-digest.

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.

7.3.5. 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.

8. Implementation Checklist

8.1. For Agent Publishers

1. Prepare domain name, configure HTTPS and TLS certificate
2. Configure DNS A/AAAA records (basic connectivity)
3. Configure DNS TXT record (_agent.xxx), publish public key
4. Configure DNS SVCB records, declare version and protocols
5. (RECOMMENDED) Publish /.well-known/agent-dns.json fallback file
6. (RECOMMENDED) Enable DNSSEC

8.2. For Client Developers

1. Query SVCB records, parse version and endpoint information
2. Query TXT records, extract public key
3. (Fallback) If SVCB unavailable, fetch agent-dns.json
4. Connect to endpoint per agent protocol (key65481) specification
5. (RECOMMENDED) Validate DNSSEC

8.3. DNS Record Configuration Example

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

; Identity trust anchor (with SVCB integrity protection)
_agent.translator.example.com. IN TXT "v=1;kid=key-2025-01;
                    alg=Ed25519;pk=...;svcb-digest=...;sig=..."

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

9. 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.

9.1. 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

9.2. Mandatory Security Requirements

To address the above threats, this specification mandates:

* Clients MUST complete TXT verification before using any SVCB resolution results
* Clients MUST perform TXT-SVCB consistency checks
* Clients MUST use TLS [RFC8446] and verify server certificates [RFC9525]
* Clients MUST NOT use endpoints that fail verification
* Agents MUST synchronously update TXT and SVCB information when versions or endpoints change

9.3. DNSSEC Usage

* Agent operators SHOULD enable DNSSEC [RFC4033] for their domains to enhance DNS-layer data integrity
* Clients MAY verify DNSSEC signatures

* This specification MUST NOT rely on DNSSEC as the sole security mechanism; its absence SHOULD NOT cause system unavailability

9.4. 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.

10. IANA Considerations

This document requests IANA registration of the following SVCB 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 and 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.

11. References

11.1. Normative References

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

11.2. 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
URI: http://www.cuiyong.net/