Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 29 October 2026

J. Miller

27 April 2026

Zero-Trust Negotiation Protocol (ZTNP)
draft-miller-ztnp-00

Abstract

The Zero-Trust Negotiation Protocol (ZTNP) specifies a session-scoped trust negotiation protocol for software-layer security and governance posture. ZTNP defines how a Relying Party challenges a counterparty for a signed posture credential, evaluates local policy against it, and issues a short-lived, channel-bound, scoped authorization artifact (a Permit) gating subsequent operations within the session.

ZTNP covers three phases:

* *Enrollment*, in which a Prover obtains a signed Posture Assertion from an Issuer that has assessed it against a publicly-identified security framework.

* *Negotiation*, in which the Requester issues a session challenge, receives the Prover’s Posture Assertion bound to that challenge, evaluates local policy against it, and either issues a Permit or returns a structured denial.

* *Validation*, in which subsequent operations within the session are gated against the Permit, with mandatory channel binding when the transport is TLS.

The protocol’s central design choice is to anchor the meaning of trust claims to a publicly-identified security framework (e.g., NIST AI RMF 1.0, ISO/IEC 42001) rather than to a specific Issuer. A policy expressing "require tier 3 against framework X" is portable across any Issuer that attests against framework X.

ZTNP is independent of any particular attestation pipeline. Posture Assertions MAY be derived from RATS Attestation Results [RFC9334], from compliance audit programs, from automated governance scanners, or from any other Issuer evaluation methodology that meets the format requirements of Section 5. ZTNP is *not* a RATS consumption profile; Section 1.5 describes how ZTNP composes with RATS-pipelined and non-RATS-pipelined deployments alike.

Multi-principal delegation, prompt-injection-resistant intent binding, and behavioral-claim extensions are specified in a companion document, the Zero-Trust Intent Protocol (ZTIP) [I-D.miller-ztip].

This draft is an individual submission. The author intends to progress this work in a venue scoped to attestation-gated session authorization; possibilities include OAuth, WIMSE, SAAG-routed new work, or an Independent Submission. Community guidance on venue is welcome.

Status of This Memo

This Internet-Draft will expire on 29 October 2026.

Copyright Notice

Table of Contents

1. Introduction

Existing identity and access management protocols establish _who_ a party is (identity) but do not bind the resulting authorization to that party's verified security posture, to the transport channel the negotiation occurred over, or to the specific assessment that justified issuance. A service may present a valid TLS certificate while having unpatched critical vulnerabilities, active security incidents, or no security or governance controls of relevance to the calling party's policy. A token-bearer may have been authorized at time T on the basis of attestation evidence the resource server has no way to verify at time T.

ZTNP defines a protocol by which two parties — services, agents, or any other principal capable of holding a signed credential — can negotiate trust at session establishment time based on cryptographically signed, machine-readable security posture claims, and gate the resulting authorization with verifiable constraints.

When the parties to a negotiation are themselves intermediaries acting on behalf of upstream principals (an intermediary acting on behalf of an orchestrator acting on behalf of a user), additional protections — delegation chains, intent-scoped Permits, and behavioral claims — are specified in a companion document, the Zero-Trust Intent Protocol (ZTIP) [I-D.miller-ztip]. This document and ZTIP are designed to be used together for multi-principal deployments and independently for two-party deployments.

1.1. Goals

ZTNP is designed to:

1. Allow any party to verify another party's trust posture at time T.

2. Bind that posture to a specific session via cryptographic challenge to prevent replay.

3. Enable local, deterministic policy evaluation without runtime calls to a central authority.

4. Support constrained authorization (Permits) with mandatory channel binding when TLS is the transport.

5. Make trust claims portable across Issuers by anchoring tier semantics to publicly-identified security frameworks rather than to vendor-specific tier scales.

6. Compose cleanly with existing IETF and industry work rather than replace it. In particular, ZTNP consumes RATS Attestation Results [RFC9334] and issues authorization artifacts compatible with OAuth 2.0 / DPoP [RFC9449] carriage patterns; it does not redefine either layer.

1.2. Relationship to Upstream Attestation Pipelines

ZTNP is *independent of any specific attestation pipeline*. A Posture Assertion is the output of an Issuer — an entity that has assessed the Prover and signed a structured claim set about its posture per Section 5. The Issuer’s evaluation methodology is unconstrained: deterministic compliance checklists, automated security scanners, human-led audit, RATS-style hardware-attestation evaluation, or LLM-mediated assessment all produce ZTNP-conformant Posture Assertions provided the resulting signed token meets the format requirements.

Three upstream pipelines are common in practice:

1. *Software-layer governance assessment* — compliance auditors, certification bodies, vendor self-assessment programs, automated governance scanners producing claims against frameworks such as NIST AI RMF 1.0, ISO/IEC 42001, or OWASP Top 10 for LLM Applications.

2. *RATS attestation pipeline [RFC9334]* — an Attester produces Evidence per a RATS attestation profile; a RATS Verifier evaluates Evidence and produces an Attestation Result; the Issuer is the RATS Verifier (or sits behind it) and emits a Posture Assertion derived from the Attestation Result. Natural where hardware roots of trust (TPM, TEE) or platform-rooted evidence (cgroups, sandbox attestation) are available.

3. *Hybrid* — software-layer governance claims composed with hardware-attested platform claims under a single Posture Assertion.

1.2.1. When RATS is the Upstream Pipeline

Where a deployment uses RATS as the attestation pipeline feeding its Issuer, the role correspondence is informational and follows the table below. ZTNP does *not* define itself as a RATS consumption profile, and the Posture Assertion is *not* a RATS Attestation Result: it is a separate, ZTNP-format signed token that may be derived from an Attestation Result via re-issuance under the Issuer's signing key.

ZTNP role	Corresponds to (when RATS is upstream)	Notes
Prover	RATS Attester	Source of Evidence in RATS-pipelined deployments.
Issuer	RATS Verifier (or downstream of one)	Evaluates upstream inputs and signs Posture Assertions. Publishes its public keys at an Issuer Key Set (IKS) endpoint.
Requester	RATS Relying Party	Consumes Posture Assertions, applies local policy, issues Permits.
Posture Assertion	(derived from) Attestation Result	A Posture Assertion MAY reference an upstream Attestation Result by hash or jti; it is not byte-equivalent to one.

Table 1

ZTNP's "Sponsor" role (Section 3, Section 10) is *not* a RATS Endorser. RATS Endorsers issue Endorsements about an Attester's intrinsic platform capabilities, consumed by the Verifier upstream of any ZTNP exchange. The Sponsor role in ZTNP is an OPTIONAL deployment-time vouching authority (orchestrator, CI/CD pipeline, owner) that participates in Assessed Enrollment (Section 10). Where a RATS Endorser is also present, it operates upstream of the ZTNP Issuer and is out of ZTNP scope.

1.2.2. What ZTNP Specifies that RATS Does Not

The aspects of session establishment that ZTNP specifies (Sections 7 and 8) are deliberately *outside* the RATS architecture's scope:

1. *Session negotiation.* RATS specifies how to _produce_ an Attestation Result; it does not specify a session negotiation protocol by which a Relying Party challenges a counterparty at session establishment, receives a bound credential, applies policy, and authorizes the session in real time.

2. *Channel binding to authorization.* RATS Attestation Results are not cryptographically bound to a TLS session by default. ZTNP Section 8.2 requires the issued Permit to include a TLS-exporter channel binding (per [RFC9266]), so that a Posture Assertion accepted on one connection cannot be lifted to another.

3. *Policy-driven authorization issuance.* RATS produces an Attestation Result and stops. ZTNP Section 8 defines a Permit — a short-lived, signed authorization artifact — that the Requester issues based on its policy evaluation of the Posture Assertion.

4. *Framework-anchored claim portability.* ZTNP Section 5 anchors claim semantics to a publicly-identified Framework URI (e.g., NIST AI RMF 1.0) so that policy expressions are portable across Issuers attesting against the same framework, regardless of which (if any) attestation pipeline produced the underlying evidence.

1.2.3. Deployments Not Using RATS

ZTNP does not require RATS. Posture Assertions are routinely produced by Issuers operating entirely outside any RATS profile (for example: a SOC 2 auditor's signed assessment artifact, a governance certification body's credential, an automated configuration scanner's signed report). The on-the-wire format, the Requester's verification path, and the Permit issuance and validation flow are identical.

This document does not specify any Evidence type, RATS attestation profile, or Verifier evaluation methodology. Those are deployment choices; the claim shapes ZTNP carries (Section 5) are valid regardless of how the upstream Issuer produced them.

1.3. Relationship to OAuth 2.0 and GNAP

ZTNP Permits are signed, scoped, time-bound, channel-bindable, optionally PoP-protected authorization artifacts. That description fits OAuth 2.0 access tokens [RFC6749] (especially in JWT form, with DPoP [RFC9449]) and GNAP access tokens [RFC9635]. A reviewer reading ZTNP after either of those will reasonably ask why the Permit is not simply one of those tokens. This subsection states what ZTNP requires that neither OAuth 2.0 nor GNAP enforces by themselves.

ZTNP requires a single cryptographic transaction that binds three things together:

1. The *Attestation Result* evaluated for the issuance decision (the Posture Assertion, which itself MUST bind to the negotiation's challenge_nonce per Section 7).

2. The *transport channel* the negotiation occurred on (the ch_binding.context_hash in the Permit, derived from TLS exporter material per [RFC9266], or a profile-defined transport-equivalent binding).

3. The *authorization artifact* itself (the Permit JWS).

Each of the three artifacts cryptographically references the others. A verifier presented with a Permit can prove, from the Permit alone plus the named Issuer's public key, that _the specific Attestation Result that justified this Permit was bound to the same channel the Permit was issued on, and that this all happened in a single negotiation event_. A Permit cannot be issued from a stale Attestation Result, lifted from a different channel, or detached from the posture decision that authorized it.

What OAuth 2.0 does not enforce: OAuth has no concept of attestation-gated issuance. An AS issues access tokens based on its grant flow (authorization code, client credentials, etc.); there is no standardized attestation evaluation step. DPoP binds tokens to a key the client proves possession of, not to an Attestation Result. An OAuth deployment could be configured to require attestation evidence at the token endpoint (via an extension grant type), but the resulting token does not carry a cryptographic reference to the attestation that justified its issuance, and a downstream resource server cannot verify that linkage from the token alone.

What GNAP does not enforce: GNAP is more flexible. The AS may evaluate arbitrary subject information during a grant request, and a deployment could in principle pass attestation evidence through GNAP's subject element. But:

1. *No normative attestation-to-token linkage.* GNAP does not require that the issued access token cryptographically reference the specific subject information that justified its issuance. A GNAP token issued at time T may have been authorized based on subject information presented at time T; the resource server has no way to verify, from the token alone, what posture the AS evaluated.

2. *No mandatory channel binding tied to the attestation flow.* GNAP supports proof-of-possession via key binding, but the bound key is not required to be the same key (or context) that bound the attestation evidence to the grant request. ZTNP requires the channel binding context in the Permit and the channel binding context the Posture Assertion was negotiated over to be the same value, derived from the same TLS exporter output.

3. *No standardized framework-anchored claim portability.* GNAP access tokens carry whatever the AS chose to put in them. ZTNP requires framework_id, tier, and flags as top-level claims on every Permit, anchored to a publicly-identified security framework (Section 5). A resource server applying policy to a Permit does not have to parse provider-specific claim soup; the policy language is portable across Issuers attesting against the same framework.

4. *AS-centric flow.* GNAP is designed around an AS that mediates between client and resource server. ZTNP supports peer-to-peer agent-to-agent flows where the Requester acts as both AS and resource server with no third party. While GNAP can be configured into this shape, the protocol's idioms (interact, continuation, multi-token grants) are heavyweight for the symmetric two-agent case.

Could ZTNP be a GNAP profile? Plausibly. A GNAP profile that mandated attestation-gated issuance, required the access token to cryptographically reference the evaluated subject information, mandated channel binding tied to the same TLS exporter output as the attestation, and standardized framework-anchored claim placement would express most of ZTNP's semantics. Such a profile would be a substantial document — comparable in length to ZTNP itself — and would inherit GNAP's negotiation overhead for the simple peer-to-peer case. The author considers a standalone protocol with explicit composition profiles (Section 13) clearer for the symmetric peer-to-peer attestation patterns that motivate this work, but does not consider the GNAP-profile path foreclosed; deployments comfortable with GNAP's machinery MAY treat ZTNP as a model for what such a profile would specify.

Composition with OAuth-issued tokens. Section 15 ("Relationship to Existing Work") sketches how ZTNP composes with OAuth deployments: the Permit JWS travels alongside an OAuth bearer token rather than being collapsed into ordinary JWT claims, so that ZTNP's three-way cryptographic linkage (Posture Assertion, channel binding, Permit) is preserved end-to-end. A future revision MAY normatively specify the wire format (e.g., a parallel HTTP field carrying the Permit) once deployment experience accumulates.

1.4. Non-Goals

ZTNP does not define:

* *Identity* — establishing who a subject is. ZTNP binds posture claims to a stable identifier; identity is established by complementary protocols (OAuth, mTLS, SPIFFE).

* *Multi-agent delegation semantics* — see ZTIP [I-D.miller-ztip].

* *A runtime envelope for agent communication* — protocols such as A2A address message exchange; ZTNP layers above such envelopes at the posture and authorization layer.

* *A tool capability protocol* — protocols such as MCP address tool semantics; ZTNP gates whether a tool may be called.

* *A transparency log* — SCITT addresses durable, auditable statement registration; ZTNP MAY use SCITT but does not require it.

* *A key management system* — ZTNP defines key discovery (Section 6) but defers key generation, custody, and rotation to deployment-specific infrastructure.

* *Risk taxonomy or assessment methodology* — Profiles (Section 13) define domain-specific semantics; ZTNP defines the carrier.

* *Standardized evaluation methodology* — different Issuers may use deterministic checklists, automated scanners, human review, large-language-model-based evaluation, or any combination. Two Issuers using different methodologies MAY produce different Posture Assertions for the same Prover, including different tier values; this is expected. The choice between Issuers — and the implicit choice of methodology — is a Requester policy decision, not a protocol guarantee. ZTNP standardizes the credential format and the trust model that lets Requesters select Issuers; it does not standardize what the Issuers do.

1.5. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Terminology

Term	Definition
Requester (R)	The party that initiates trust negotiation and consults its local policy to decide whether to proceed.
Prover (P)	The party that presents a Posture Assertion to demonstrate its security posture.
Issuer (I)	The entity that signs Posture Assertions after assessing a Prover. Publishes its public keys at an Issuer Key Set (IKS).
Sponsor (S)	An OPTIONAL principal (orchestrator, CI/CD pipeline, owner, human user) that vouches for a Prover’s identity at enrollment time (Section 10). The Sponsor is *not* a RATS Endorser; see Section 1.5.
Posture Assertion (PA)	A signed token containing machine-readable security posture claims about a Prover.
Permit	A short-lived, signed authorization artifact issued by a Requester after successful negotiation.
Issuer Key Set (IKS)	The public key endpoint published by an Issuer; analogous to a JWKS endpoint.
Framework	A registered security or governance framework (e.g., NIST AI RMF, ISO/IEC 42001) against which a Posture Assertion is made.
Framework	A URI identifying the security framework

Identifier	underlying a Posture Assertion’s claims. The URI SHOULD be a stable, publicly resolvable identifier published by the framework’s authoring organization.
Tier	An integer within a registered framework conveying assessment outcome. Tier semantics are defined by the framework, not by ZTNP.
Binding Mode	The direction in which Posture Assertions are exchanged during negotiation (Section 7).
Channel Binding	A cryptographic binding of a Permit to a specific TLS session, using TLS exporter material per [RFC9266].

Table 2

3. Architectural Overview

This section gives a high-level view of the protocol. Subsequent sections specify each element in detail.

3.1. Roles

ZTNP defines four roles. Where a deployment uses RATS as its upstream attestation pipeline, three of these roles correspond informally to RATS roles per Section 1.5. A single deployed entity MAY play more than one role.

* *Issuer* signs Posture Assertions and publishes its public keys at an Issuer Key Set (IKS) endpoint. In RATS-pipelined deployments, the Issuer is or sits behind a RATS Verifier.

* *Prover* holds a Posture Assertion issued to it and presents the assertion when requested. In RATS-pipelined deployments, the Prover is the RATS Attester.

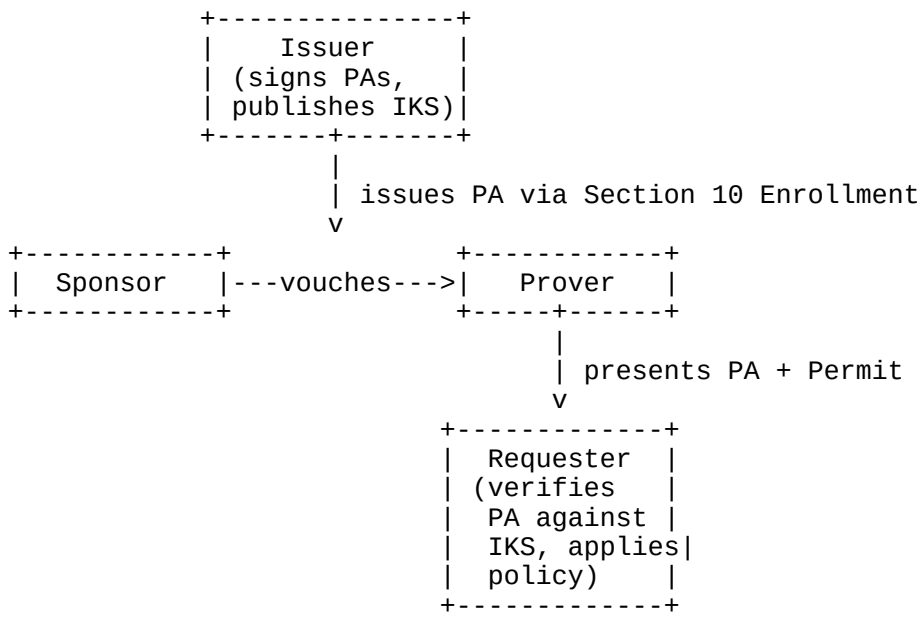
* *Requester* asks for a Posture Assertion and decides, based on local policy, whether to issue a Permit. In RATS-pipelined deployments, the Requester is the RATS Relying Party.

* *Sponsor* OPTIONALLY vouches for a Prover’s identity at enrollment time (Section 10). The Sponsor role is *not* a RATS Endorser: RATS Endorsers issue Endorsements about an Attester’s intrinsic

platform capabilities, consumed by the Verifier upstream of any ZTNP exchange. The Sponsor is a deployment-time onboarding authority (orchestrator, CI/CD pipeline, owner) that participates in Assessed Enrollment.

A single agent typically begins as a Prover when asked to demonstrate posture, and acts as a Requester when calling other services.

3.2. Trust Model



The trust relationships are:

* *Requester → Issuer:* The Requester has out-of-band trust in one or more Issuers, encoded in its policy. It verifies Posture Assertions against the Issuer's cached IKS.

* *Prover → Issuer:* The Prover obtained its Posture Assertion from the Issuer via Enrollment (Section 10).

* *Sponsor → Prover:* For Assessed Enrollment (Section 10), the Sponsor signs a vouching statement attesting to the Prover's identity.

ZTNP does not require any party to dial home to the Issuer at session establishment. Verification is local against a cached IKS.

3.3. Three Protocol Phases

Phase 1: Enrollment (per-agent, one-time). Section 10 specifies enrollment.

Phase 2: Negotiation (per-session). The Requester issues a challenge nonce. The Prover responds with a Posture Assertion bound to the nonce. The Requester verifies the signature, evaluates local policy against the assertion's framework_id, tier, and flags, and either issues a Permit or returns a structured DENY. When the transport is TLS, the Permit is bound to the TLS session via channel binding (Section 8). Section 7 specifies negotiation.

Phase 3: Validation (per-request). Each subsequent operation within the session presents the Permit. The receiving party validates the Permit signature, channel binding, scope, and (optionally) proof-of-possession. Section 8 specifies validation.

3.4. What ZTNP Provides

A successfully-negotiated ZTNP session provides the following machine-verifiable guarantees to the Requester:

* The Prover's posture has been assessed by a known Issuer against an identified framework.
* The posture meets the Requester's policy as of the negotiation timestamp.
* The Permit cannot be lifted to a different TLS session.
* The scope of permitted actions is bounded and signed.

When ZTIP [I-D.miller-ztip] is also in use, additional guarantees apply: each request is bound to a specific signed intent, and the call chain is tied to a delegation chain back to a trusted root principal.

3.5. Concrete Role Examples

The Roles defined in Section 3.1 (Requester, Prover, Issuer, Sponsor) are abstract. In practice the same entity may fill different roles in different interactions. Three concrete scenarios illustrate the pattern.

3.5.1. Scenario A: Agent Calls a Tool Server (PA-R / Resource-Presents)

An orchestrator agent wants to call an MCP tool server that exposes sensitive operations. The orchestrator wants to verify the tool server's posture against its policy before calling it.

Role	Played by	What it does
Requester	The orchestrator agent	Asks the tool server "show me your posture before I call you." Issues the Permit (or DENY) based on local policy.
Prover	The MCP tool server	Holds a Posture Assertion issued by an Issuer; presents it on request, bound to the orchestrator's challenge nonce.
Issuer	A third-party assessor (e.g., a vendor or compliance auditor)	Has previously assessed the tool server and signed its Posture Assertion. Publishes its IKS at a stable URL.

Table 3

This is analogous to TLS server-certificate verification: the client (Requester) verifies the server (Prover) before trusting it. Most MCP integrations fit this pattern.

3.5.2. Scenario B: Tool Server Gates Callers (PA-C / Caller-Presents)

A tool server holding sensitive data — say, an HR-records lookup — wants to gate which callers may invoke it. The server requires callers to present a Posture Assertion meeting its policy before serving any request.

Role	Played by	What it does
Requester	The tool server	Asks each calling agent "show me your posture before I serve you." Issues the Permit per local policy.
Prover	The calling agent	Holds a Posture Assertion proving its security and governance maturity; presents it on request.
Issuer	The agent's compliance assessor	Has previously assessed the calling agent and signed its Posture Assertion.

Table 4

This is analogous to mTLS client-certificate verification. The role assignment is the inverse of Scenario A.

3.5.3. Scenario C: Cross-Organization Agent Collaboration (Mutual)

Acme Corp's orchestrator agent needs to coordinate with a partner organization's data agent to complete a multi-step task. Both organizations want assurance about each other's posture.

Role	Played by	What it does
Requester (and Prover)	Acme's orchestrator	Verifies Partner's Posture Assertion AND presents its own.
Prover (and Requester)	Partner's data agent	Verifies Acme's Posture Assertion AND presents its own.
Issuer (Acme side)	Acme's compliance assessor	Has previously assessed Acme's orchestrator.
Issuer (Partner side)	Partner's compliance assessor (may differ)	Has previously assessed Partner's data agent.

Table 5

Both parties verify against the other's IKS. Acme might trust both its own assessor and a small set of approved third-party Issuers; Partner has its own trust list. Mutual mode requires both lists to overlap or both parties to accept the same Issuers.

In all three scenarios, the Sponsor role is OPTIONAL and only appears at enrollment time (Section 10) when an Issuer requires the Prover's identity be vouched for.

4. Trust Anchoring (Background)

This section is informative. It motivates the design choice that distinguishes ZTNP from earlier posture systems: anchoring tier semantics to an IANA-registered framework rather than to a specific Issuer.

4.1. The Failure Mode

A posture protocol must guarantee that the meaning of a claim is stable across the parties relying on it. Consider two agents calling a partner's data API. Agent A presents a Posture Assertion from Issuer X claiming tier: 3; Agent B presents one from Issuer Y, also tier: 3. The partner's policy engine sees two equally-acceptable agents.

In reality, Issuer X's tier 3 may mean "passed a 90-day continuous monitoring program against ISO/IEC 42001 controls" while Issuer Y's tier 3 means "self-attested at signup." The integer 3 is not a security guarantee; it is a marketing term mediated by Issuer documentation that the policy engine never reads. This is the *trust anchoring problem*: claims are only as portable as the semantic substrate they reference, and an integer without one is portable to nowhere.

4.2. Why Issuer-Only Anchoring Fails

A first-pass solution is to constrain policies on the Issuer:

```
{ "issuers": ["https://issuer-x.example"], "tier_min": 3 }
```

This eliminates cross-issuer tier confusion but at high cost: each Requester must explicitly trust each Issuer (vendor lock-in); a policy written for Issuer X cannot apply to Issuer Y (no portability); cross-organization deployments require bilateral trust meshes (no federation); and the semantic content of "tier 3" sits in PDF documentation outside the protocol's verification path (no semantic transparency). This is the failure mode that ailed early certificate-authority-centric PKI.

4.3. The Right Anchor: Frameworks

The substantive content of a posture claim is _which framework was assessed and to what tier within that framework_. The Issuer is the party that performed the assessment — important for liability, but not the semantic anchor. A Posture Assertion that names its framework via a stable URI resolves the problem:

```
{
  "framework_id": "https://nist.gov/airmf/1.0",
  "tier": 3,
  "iss": "https://issuer-x.example"
}
```

A Requester writes portable policy:

```
{
  "framework_id": "https://nist.gov/airmf/1.0",
  "tier_min": 3
}
```

This accepts credentials from any Issuer attesting against NIST AI RMF 1.0 at tier 3 or higher. Adding a new compliant Issuer requires no policy change by any Requester.

4.4. Framework Identifiers as URIs

ZTNP uses URIs as framework identifiers rather than allocating an IANA-managed integer registry. Three reasons:

1. *No appropriate gatekeeper exists.* Security and governance frameworks are authored by NIST, ISO, OWASP, AICPA, the European Commission, individual vendors, and individual enterprises. Asking IANA to maintain a directory of these external bodies' work is outside IANA's normal protocol-parameter scope.

2. *The flexibility is required for adoption.* Vendors publishing proprietary trust profiles, enterprises operating internal compliance frameworks, open-source projects publishing domain-specific assurance frameworks, and standards bodies publishing emerging frameworks all need a way to identify their work without waiting for a central registry to allocate them an integer. URI-based identifiers make all four cases first-class.

3. *The precedent exists in IETF practice.* OAuth scope strings (RFC 6749), JOSE algorithm identifiers when not in the registered list, and JWT private claim names (RFC 7519 Section 4.3) all use URIs or URI-like collision-free names without IANA gatekeeping.

4.4.1. URI Discipline

A framework_id URI SHOULD:

* Be *publicly resolvable* at the time of issuance, so verifiers can look up the framework's specification.

* *Encode version in the path* (https://nist.gov/airmf/1.0, not https://nist.gov/airmf) so that NIST AI RMF 1.0 and 2.0 are distinct identifiers.

* *Use a domain controlled by the framework's authoring organization*, so anyone reading the URI can identify the publisher.

* *Resolve to a stable document* (DOI, persistent URL, or organization's archival publication system).

Private and internal-only frameworks MAY use the urn: scheme: urn:org.acme:internal-trust-framework:v1. URIs not resolvable to a public document MUST NOT be presented in cross-organization deployments unless the parties have separately exchanged the framework's specification out-of-band.

4.4.2. Well-Known Framework URIs (Informative)

The following URIs identify widely-recognized public frameworks at the time of this specification's publication. Implementations SHOULD use these URIs when their assertions are made against the corresponding frameworks. This list is informative; the canonical authority for each URI is the framework's authoring organization.

Framework	Recommended framework_id URI
NIST AI Risk Management Framework 1.0	https://doi.org/10.6028/NIST.AI.100-1
ISO/IEC 42001:2023	https://www.iso.org/standard/81230.html
OWASP Top 10 for LLM Applications (2025)	https://genai.owasp.org/llm-top-10/2025
SOC 2 Trust Services Criteria (2017)	https://www.aicpa-cima.com/resources/landing/system-and-organization-controls-soc-suite-of-services
EU AI Act (Regulation (EU) 2024/1689)	https://eur-lex.europa.eu/eli/reg/2024/1689/oj

Table 6

These URIs SHOULD be updated by the relevant authoring organizations as their publications evolve. ZTNP implementations are not required to recognize this list; Requesters are required only to evaluate the URIs in their local policy.

4.5. What Issuer Trust Still Means

Anchoring on framework does not eliminate the need to trust Issuers. A Requester still needs to know whether the entity claiming framework_id: "https://nist.gov/airmf/1.0" actually performed an NIST AI RMF assessment (answered by the Issuer's signature and IKS lookup), and whether the Issuer's assessment process is worth trusting (answered by local policy). Both anchoring dimensions can appear in a single policy; either dimension can stand alone.

5. Posture Assertion Format

A Posture Assertion is a signed token containing machine-readable security posture claims about a Prover. ZTNP supports any signed token format meeting the requirements in this section. Implementations SHOULD use JWS [RFC7515] over JSON or COSE [RFC9052] over CBOR. JSON encoding is used in this specification for clarity.

When carried in ZTNP messages (Sections 7 and 9), the Posture Assertion appears in a JSON field named pa.

5.1. Required Claims

A Posture Assertion payload MUST include:

Claim	Type	Description
ver	string	Posture Assertion format version (e.g., "0.2")
iss	string	Issuer identifier (stable URI or DID)
sub	string	Subject identifier for the Prover
iat	integer	Issued-at time, Unix seconds
exp	integer	Expiration time, Unix seconds
jti	string	Unique Posture Assertion identifier
framework_id	string (URI)	URI identifying the security framework against which this assertion was made (Section 5.2).
tier	integer	Tier value within the named framework_id. Tier semantics are defined by the framework, not by ZTNP.
scope	object	What this assertion covers (Section 5.3)
claims	object	Posture-specific claim payload (Section 5.4)
bind	object	Challenge binding data (Section 5.5); REQUIRED unless mode is PA-R pre-fetch
enrollment_mode	string	One of self or assessed. Determines the tier cap a Requester applies; see Section 11.

Table 7

5.1.1. Private and Public Claims

Implementations MAY include additional top-level claims beyond those in the table above. To avoid collisions with future ZTNP-defined claims, additional claims SHOULD use one of the following naming conventions, mirroring [RFC7519] Section 4:

* *Public claims*: collision-free names registered with IANA in the ZTNP Posture Assertion Top-Level Claim Names registry (Section 17), or names that are URIs (e.g., https://acme.example/claims/internal-trust-level).

* *Private claims*: names agreed upon by the Issuer and the Requester out-of-band, and used only within deployments where both parties understand them.

Implementations MUST NOT redefine the meaning of any claim listed in the required-claims table above. Verifiers MUST ignore unrecognized claims they do not understand, except when the recognition of a claim is required by local policy.

5.2. Framework Identifier

The framework_id claim is the protocol's anchor for the meaning of tier and any framework-specific structure within claims.posture. Section 4 motivates the design choice; this section specifies normative verification rules.

framework_id MUST be a URI conforming to [RFC3986]. It identifies a particular versioned security or governance framework (e.g., NIST AI RMF 1.0, ISO/IEC 42001:2023, or a vendor-published or enterprise-internal framework).

Verification rules:

1. A Requester MUST treat a Posture Assertion whose framework_id is not in its policy's allowed-frameworks list as a non-match for that policy.

2. A Requester MUST NOT compare tier values across different framework_id values. The same integer tier: 3 under two different framework URIs describes two different things.

3. A Requester MAY accept Posture Assertions from any well-formed framework_id URI for opportunistic logging (mode PA-O), but MUST NOT make access decisions based on framework URIs its policy does not understand.

4. Verifiers SHOULD treat framework_id URIs as opaque identifiers for matching purposes — equality of URIs MUST be tested with byte-for-byte string comparison (case-sensitive, no normalization) to prevent ambiguity.

When a single assessment satisfies multiple frameworks (a common case — many ISO/IEC 42001 controls overlap with NIST AI RMF), the Issuer MAY include an additional_frameworks array:

```
{
  "framework_id": "https://doi.org/10.6028/NIST.AI.100-1",
  "tier": 3,
  "additional_frameworks": [
    { "framework_id": "https://www.iso.org/standard/81230.html", "tier": 3 },
    { "framework_id": "https://genai.owasp.org/llm-top-10/2025", "tier": 2 }
  ]
}
```

Each entry MUST itself contain a framework_id URI and a tier integer. Requesters MAY consult additional_frameworks for policy matching; if a Requester's policy specifies framework_id: X, a Posture Assertion whose primary framework_id is Y but whose additional_frameworks includes {framework_id: X, tier: T} SHOULD match the policy at tier T. The same per-framework constraints in this section apply to additional_frameworks entries.

5.2.1. Non-Integer Tier Outcomes

The tier field is normatively an integer in this specification. Some frameworks express assessment outcomes as letter grades (A, B, C, D), maturity-model levels with named labels ("Initial", "Defined", "Managed"), or other non-integer schemes. Frameworks using such outcomes MUST publish an integer mapping in their framework definition (e.g., A=4, B=3, C=2, D=1) and MUST publish that mapping at the framework_id URI alongside the framework's specification. Issuers attesting against such frameworks emit the mapped integer in the tier field; Requesters consulting the framework definition can reason about the mapping when needed for human-readable display, but compare tier values numerically.

This rule keeps the wire format simple (single integer comparison) while accommodating frameworks whose human-facing taxonomy is non-numeric.

5.3. Scope

scope MUST include:

Field	Type	Required	Description
kind	string	Yes	One of: agent, service, tool, model, environment
target	string	Yes	Canonical identifier of the subject instance
env	string	No	Environment label (e.g., prod, staging)
components	array	No	Component identifiers covered by this assertion

Table 8

A Requester MUST verify that the sub and scope.target match the entity it intended to interact with.

5.4. Claims Payload

claims is a flexible object whose structure is partially defined by the assertion's framework_id. Base ZTNP requires:

Field	Type	Required	Description
flags	object	Yes	Boolean flags for critical gating
posture	object	No	Framework-defined claim structure
assessment_method	string	SHOULD	Methodology used to produce this assertion (see below)
policy_id	string (URI)	No	OPTIONAL Issuer-published identifier for the specific appraisal policy applied (see below)

Table 9

flags SHOULD include, where applicable: critical_open, incident_open, pii_access_allowed, secrets_access_allowed, code_exec_allowed. Flag names are administered by the IANA ZTNP Posture Assertion Flag Names Registry (Section 17).

posture carries framework-specific structured data; the shape depends on the framework identified by framework_id. Profiles (Section 13) document the posture shape for their framework.

The claims.posture.ai_behavior sub-namespace within claims.posture is reserved for Behavioral Claims as defined in Section 5 of [I-D.miller-ztip]. Posture Assertions composed with ZTIP carry behavioral safety properties (e.g., prompt_injection_tested, tool_call_audit_logged) under this namespace. Verifiers that do not implement ZTIP MUST ignore unrecognized fields under claims.posture.ai_behavior per the unrecognized-claim rule below; ZTIP's composition profile (Section 6.1 of [I-D.miller-ztip]) gives the full schema.

5.4.1. Appraisal Policy Identifier

policy_id is an OPTIONAL URI naming the specific Issuer-published appraisal policy (rule set, control mapping, scoring rubric) applied to produce this Posture Assertion. It is finer-grained than assessment_method: where assessment_method describes the _kind_ of evaluation (deterministic, scanner-based, human, LLM, hybrid), policy_id identifies the _exact_ policy version applied.

A Requester's policy MAY require a specific policy_id (or set of acceptable policy_id values) for high-stakes interactions where the Issuer's published policy version is part of the trust decision. This is the natural integration point for deployments that wish to consume RATS Verifier Appraisal Policy identifiers, AR4SI policy identifiers, or any other Issuer-defined policy URI; ZTNP treats the value as opaque and matches by exact URI equality.

5.4.2. Assessment Method

assessment_method is a SHOULD-include claim describing the methodology by which the Issuer produced this Posture Assertion. It allows Requesters to filter on methodology when their policy requires (for example, a policy may decline to accept tier-3 assertions produced solely by an LLM evaluator).

Common values:

Value	Meaning
deterministic_checklist	Evidence checked against a fixed rule set with deterministic pass/fail.
automated_scan	Programmatic scanning (vulnerability scanners, configuration auditors, static analyzers).
human_review	Human-led assessment by qualified reviewers.
llm_evaluator	Evaluation conducted by a language model interpreting evidence.
hybrid	Combination of two or more of the above.
unspecified	Methodology not disclosed; Requesters SHOULD treat with greater caution.

Table 10

Profile documents MAY define additional values for their domain. The IANA ZTNP Assessment Method Names Registry (Section 17) administers cross-profile values.

A Requester's policy MAY include an assessment_method_allowed constraint listing acceptable values; Posture Assertions whose assessment_method is outside this list MUST be rejected with POLICY_METHOD_MISMATCH. Requesters concerned with non-determinism in evaluation (e.g., LLM-evaluator output varying between models or runs) SHOULD set this constraint explicitly.

5.5. Challenge Binding

To prevent replay, Posture Assertions issued during a ZTNP Negotiation exchange MUST be bound to the Requester's challenge nonce.

bind MUST include:

Field	Type	Required	Description
method	string	Yes	nonce_hash or nonce_sig
nonce	string	Yes	Hashed or plaintext challenge nonce
ctx	string	No	Session context string
aud	string	No	Requester identifier

Table 11

Method A: nonce_hash (REQUIRED to implement). bind.nonce MUST be base64url(SHA-256(challenge_nonce || ctx || aud)). Verification: the Requester recomputes and compares.

Method B: nonce_sig (OPTIONAL). bind.nonce contains the plaintext challenge_nonce. The Prover returns a detached signature binding_sig over (challenge_nonce || jti || sub) using its subject_key. Both signatures MUST verify.

Pre-fetched Posture Assertions (PA-R mode) MAY omit bind; Requesters MUST apply stricter freshness policy (RECOMMENDED max age 1 hour).

Freshness model. ZTNP’s bind.nonce mechanism corresponds to the Nonce-based freshness model described in [RFC9334] Section 10 (in deployments that use RATS as the upstream attestation pipeline; the same mechanism applies independently in non-RATS deployments). ZTNP does not currently specify an Epoch-ID or Time-of-Receipt freshness model; deployments that require those models SHOULD layer them in profile documents.

5.6. Algorithm Agility

ZTNP is algorithm-agnostic. Implementations MUST support at least one secure asymmetric signature scheme. Post-quantum signature schemes (e.g., ML-DSA per [FIPS204]) MAY be used and are RECOMMENDED for long-lived deployments.

Requesters MUST reject Posture Assertions that use an algorithm not in the allowed list, reference a kid not in the IKS, have exp in the past, have iat in the future beyond clock skew, carry an unregistered framework_id, or fail challenge-binding verification.

5.7. Versioning of ver

The ver claim uses semantic versioning: MAJOR.MINOR.PATCH. Implementations MUST reject Posture Assertions with an unsupported MAJOR version.

The MAJOR / MINOR / PATCH boundaries are defined as follows:

MAJOR version increment is REQUIRED when any of the following changes:

* A required claim is added, removed, or renamed.
* The canonical-form serialization for a signed payload changes (e.g., switching from JCS to a different canonicalization).
* A binding method (nonce_hash, nonce_sig, channel binding) is removed.
* The interpretation of a required claim’s value changes such that previously-valid Posture Assertions are now interpreted incorrectly.

MINOR version increment is REQUIRED when any of the following changes:

* A new optional claim is added.
* A new binding method is added (existing methods continue to work).
* A new flag is added to the IANA Posture Assertion Flag Names Registry.
* A new constraint type, reason code, or assessment-method value is added.
* An existing optional claim’s permitted value range is widened.

PATCH version increment is appropriate for editorial-only changes: prose clarifications, table reformatting, reference updates that do not change normative behavior.

Implementations encountering a Posture Assertion with a MAJOR they support but a MINOR newer than they recognize MUST process the assertion’s required claims normally and SHOULD ignore unrecognized optional claims (unless their policy depends on those claims). This rule preserves forward compatibility within a MAJOR version.

6. Issuer Key Set (IKS)

6.1. IKS Endpoint

Each Issuer MUST publish a key set endpoint containing active public keys and metadata. This endpoint is the Issuer Key Set (IKS) and is analogous to a JWKS endpoint [RFC7517].

The IKS MUST provide: Issuer identifier (iss); Key identifiers (kid); Algorithm identifiers (alg); Public key material; Key validity windows (RECOMMENDED).

IKS endpoints MUST be served over HTTPS [RFC9110].

6.2. Key Rotation and Caching

Requesters MUST verify Posture Assertion signatures using keys obtained from the IKS corresponding to the assertion’s iss claim.

Requesters MUST cache IKS responses with a bounded TTL (RECOMMENDED: 1 hour). Requesters MUST handle key rotation by periodically re-fetching the IKS and MUST NOT accept Posture Assertions whose kid is not present in a freshly-fetched IKS.

Issuers SHOULD support overlap windows during rotation: both the retiring and the incoming keys present in the IKS for a period sufficient to cover Requester cache TTLs.

7. Negotiation

ZTNP negotiation has four steps: Discovery, Challenge, Proof, and Decision. The selected Binding Mode (Section 7.5) determines which party plays Prover at which step. Wire-encoding details are in Section 9.

7.1. Discovery

R → P: DISCOVER (optional metadata hint)

P → R: DISCOVER_RESPONSE

Field	Type	Required	Description
ztnp_version	string	Yes	Highest ZTNP version supported by P
sub	string	Yes	Prover subject identifier
issuers	array	Yes	Issuer identifiers whose Posture Assertions P can present
frameworks	array	Yes	Framework identifiers (registered) P’s Posture Assertions cover
iks_urls	array	No	Direct links to Issuer Key Set endpoints
modes	array	Yes	Supported binding modes
features	array	No	Optional capability strings

Table 12

The frameworks field is REQUIRED.

7.2. Challenge

R → P: CHALLENGE

Field	Type	Required	Description
challenge_nonce	bytes	Yes	Cryptographically random, 16-32 bytes, base64url-encoded
ctx	string	No	Session context string
aud	string	No	Requester identifier
mode	string	No	Requested binding mode
framework_required	array	No	Frameworks the Requester’s policy will accept

Table 13

7.3. Proof

P → R: PROOF

Field	Type	Required	Description
pa	signed token	Yes	The challenge-bound Posture Assertion
binding_sig	signature	If nonce_sig	Detached subject signature for nonce_sig binding
prover_meta	object	No	Implementation info, declared capabilities

Table 14

When ZTIP is in use, the PROOF additionally carries a delegation_chain field; see [I-D.miller-ztip].

7.4. Decision

The Requester MUST respond with PERMIT or DENY.

R → P: PERMIT — Section 8 specifies the Permit format.

R → P: DENY — reasons (array of machine-readable codes plus human text), required_improvements (optional advisory).

7.5. Binding Modes

A Binding Mode specifies the *direction* of Posture Assertion exchange — which party presents, which party verifies, or both.

7.5.1. PA-R (Resource-Presents)

The Prover (typically a tool server) presents its Posture Assertion to the Requester before the Requester proceeds.

7.5.2. PA-C (Caller-Presents)

The Requester presents its Posture Assertion to the Prover.

7.5.3. Mutual

Both parties exchange and verify Posture Assertions before proceeding.

7.6. Adoption Posture

A Binding Mode specifies _who presents to whom_; the Adoption Posture specifies _what the Requester does when the expected Posture Assertion is absent or fails policy_. The two are orthogonal: any Binding Mode MAY be operated under any Adoption Posture.

Posture	Behavior on missing or failing PA
required	Requester denies the session. Default for production deployments.
monitor-only	Requester proceeds with the session, logs the failure, and emits a structured signal to its observability pipeline. No Permit-gated authorization is established for failed PAs; deployments using monitor-only accept that the session proceeds without ZTNP-level authorization.
opportunistic	Requester proceeds _and_ issues a Permit even when no Posture Assertion is presented. The Permit's claims.adoption_posture MUST record "opportunistic" so downstream auditors can identify sessions authorized without a posture credential. RECOMMENDED only during progressive adoption windows.

Table 15

The Adoption Posture is a Requester-side policy choice. It is announced (where relevant) in the Requester's CHALLENGE message via an OPTIONAL adoption_posture field, but the authoritative value is the one applied by the Requester at decision time. Provers cannot influence the Requester's Adoption Posture.

Default Adoption Posture is required. Implementations SHOULD log the Adoption Posture applied to every issued or denied Permit.

7.7. Policy Evaluation

Policies are local to the Requester. ZTNP standardizes inputs and outputs while leaving policy language implementation-defined.

Policies that gate on tier_min MUST specify either framework_id or a non-empty issuers_allowed set. Policies specifying tier_min alone are incomplete and SHOULD be rejected (reason: POLICY_INCOMPLETE).

Standard constraint types (administered by IANA, Section 15): data, actions, rate_limit, ttl, redaction, tools.

8. Permit Format and Validation

A Permit is a Requester-issued, short-lived authorization artifact returned after a successful allow or allow_with_constraints decision.

8.1. Required Fields

Field	Type	Description
iss	string	Requester identifier
sub	string	Prover identifier
iat	integer	Issued-at, Unix seconds
exp	integer	Expiration, Unix seconds
permit_id	string	Unique permit identifier
constraints	object	Enforced constraints (may be empty)
ch_binding	object	Channel binding (REQUIRED when transport is TLS, see Section 8.2)
cnf	object	OPTIONAL Proof-of-Possession key confirmation (Section 12)

Table 16

When ZTIP is in use, additional fields (intent_hash, intent_scope, chain_root_iss, chain_root_jti) appear; see [I-D.miller-ztip].

Permits MUST be signed by the Requester. Provers MUST include the Permit in subsequent requests. Permits MUST NOT be reused across sessions.

8.2. Channel Binding (Mandatory when TLS)

When the ZTNP exchange occurs over TLS, the Permit MUST include a channel binding tying it to the specific TLS session via TLS exporter material per [RFC9266]. This prevents a Permit obtained on one TLS connection from being used on a different connection.

ch_binding MUST include:

Field	Description
method	"tls-exporter"
label	Exporter label (REQUIRED to use "EXPORTER-ZTNP-permit-binding")
context_hash	base64url(SHA-256(tls_exporter_output)) per [RFC8446] Section 7.5

Table 17

Provers MUST recompute the exporter and verify the hash matches. Mismatch causes rejection with PERMIT_CHANNEL_MISMATCH.

If the transport does not provide TLS keying material, the Permit MUST still carry a ch_binding object. Three values for ch_binding.method are permitted in this case:

1. A profile-defined transport-equivalent binding mechanism registered in the IANA ZTNP Channel Binding Method Registry (Section 17). The MCP profile’s mcp-process-ephemeral-key (for stdio transport) is the first concrete instance.

2. The literal value "none", accompanied by a REQUIRED ch_binding.rationale string explaining why no binding is provided (e.g., "closed in-process IPC, single-tenant trust boundary"). Receivers MUST log the rationale and MAY reject Permits whose method is "none" per local policy.

3. Any other value defined by a profile document that publishes its semantics; such values MUST be registered before deployment in cross-organization use.

Silent omission of ch_binding is *not* permitted: a conforming Permit always carries an explicit binding declaration, even if that declaration is "none" plus rationale. This makes the absence of channel binding visible in audit logs rather than implicit in the wire format.

Implementations claiming "ZTNP" conformance MUST implement and enforce channel binding when their declared transport is TLS, and MUST emit either a registered transport-equivalent method or method: "none" with rationale otherwise.

8.3. Permit Validation

For each request carrying a Permit, the receiving party MUST validate:

1. The Permit signature is valid against the issuing Requester's key.
2. iat and exp are valid within bounded clock skew.
3. The current TLS session's exporter output matches ch_binding.context_hash (when present).
4. The requested operation is within the Permit's constraints.
5. (If cnf present) A valid Proof-of-Possession proof per Section 12.
6. (If ZTIP in use) The operation is within the Permit's intent_scope.

Validation is local; no central authority is consulted at request time.

9. Wire Encodings

ZTNP defines a transport-binding model. Specific transport profiles are published as separate documents; this section describes the binding shape and lists current profiles.

A wire-encoding profile MUST specify how the abstract messages of Section 7 (DISCOVER, DISCOVER_RESPONSE, CHALLENGE, PROOF, PERMIT, DENY) are carried over the transport, and MUST satisfy:

* All messages serialized in a deterministic, machine-parsable format.
* All endpoints serving over a confidential transport (typically TLS) when the deployment threat model warrants.
* Channel binding (Section 8.2) applied when the transport provides TLS keying material.

Current profiles:

Profile	Document	Status
HTTP	(Working notes in repository)	Working draft
MCP (Model Context Protocol)	draft-miller-ztnp-mcp-profile-00	Internet-Draft
A2A (Agent-to-Agent Protocol)	draft-miller-ztnp-a2a-profile-00	Internet-Draft

Table 18

Implementations MAY support additional transports by publishing a profile document.

10. Enrollment Requirements

ZTNP does not specify a new enrollment wire format. How a Prover obtains its first Posture Assertion is a deployment choice; existing IETF and industry mechanisms — Dynamic Client Registration [RFC7591]/[RFC7592], RATS attestation flows [RFC9334], and supply-chain transparency registries (e.g., SCITT [SCITT-ARCH]) — already cover the wire-level concerns and SHOULD be reused. This section defines only the abstract requirements that any enrollment mechanism MUST satisfy for the resulting Posture Assertions to be ZTNP-conformant, and a single security property (the Self-Enrollment tier cap) that ZTNP relies on for its overall trust model.

10.1. Abstract Requirements

Any enrollment mechanism that produces a Posture Assertion that will be presented over ZTNP MUST:

1. *Bind the assertion to a subject key.* The Issuer MUST bind the Posture Assertion to a public key held by the Prover. The Prover MUST be able to demonstrate possession of the corresponding private key during ZTNP negotiation (Section 7).

2. *Establish the Issuer’s signing key publicly.* The Issuer MUST publish its public keys at an Issuer Key Set (Section 6) so that Requesters can verify Posture Assertions offline.

3. *Record the enrollment mode.* The Posture Assertion MUST carry the enrollment_mode claim (Section 5) with a value of self or assessed. This is the only enrollment-related signal ZTNP requires inside the assertion; it is what enforces the tier cap below.

4. *Support revocation.* The Issuer MUST be able to revoke a Posture Assertion if the underlying enrollment is invalidated. The mechanism is unconstrained — short-lived assertions with natural expiry, OCSP-style status endpoints, and revocation lists are all acceptable.

10.2. Enrollment Modes and the Tier Cap

ZTNP distinguishes two enrollment modes, recorded in the enrollment_mode claim of every issued Posture Assertion:

* *self* (Self-Enrollment): the Issuer performed only proof-of-possession of the Prover's subject_key. No third party vouched for the Prover's identity or posture.

* *assessed* (Assessed Enrollment): the Issuer performed an actual posture assessment of the Prover, optionally consuming a Sponsor's vouching statement (Section 10.4) and any upstream attestation Evidence. Where RATS is the upstream pipeline, the Issuer is or sits behind a RATS Verifier; where it is not, the Issuer's evaluation is whatever methodology its assessment_method claim declares.

Tier cap (load-bearing security property): A Posture Assertion with enrollment_mode: "self" MUST NOT carry tier > 1. Issuers MUST NOT issue, and Requesters MUST reject, Self-Enrolled assertions claiming higher tier.

This cap is what allows Requesters to apply differentiated policy ("require tier 2") with confidence that no Self-Enrolled subject can clear the bar. The cap is enforced from the Posture Assertion alone — Requesters do not need to know which wire format was used at enrollment time.

10.3. Sponsor Role

When an enrollment uses the assessed mode, an OPTIONAL *Sponsor* MAY participate. The Sponsor's role is to vouch for the Prover's identity at enrollment time (typically: "this software-image hash corresponds to this organization's published artifact," "this CI/CD pipeline produced this build," "this owner authorizes this agent to enroll under this identity"), separate from the Issuer's role of

evaluating posture. The Sponsor's signature on a vouching statement is a deployment concern; ZTNP imposes no constraints on its format.

The Sponsor role is *distinct from the RATS Endorser role* [RFC9334]: a RATS Endorser issues Endorsements about an Attester's intrinsic platform capabilities, consumed by a Verifier upstream of any ZTNP exchange. Where a deployment uses both — for example, a hardware vendor's RATS Endorsement plus an organizational Sponsor's identity vouching — the RATS Endorsement flows into the Issuer's evaluation pipeline (out of ZTNP scope) and the Sponsor's statement participates in ZTNP enrollment as described in this section.

For deployments that wish to publish Sponsor keys in a discoverable way, an OPTIONAL *Sponsor Key Set (SKS)* endpoint MAY be served at /.well-known/ztnp-sponsor-keys as a JWKS document [RFC7517] with iss matching the Sponsor identifier. The SKS has the same operational properties (cache TTL, rotation) as the IKS (Section 6). Deployments using direct PKI configuration, hardware-attested workload identity, or manually-installed keys need not implement the SKS endpoint.

10.4. Recommended Wire Mechanisms

Implementers SHOULD adopt one of the following rather than invent a new enrollment flow:

* *OAuth Dynamic Client Registration ([RFC7591]/[RFC7592])* — for deployments already running OAuth infrastructure. The Prover registers with the Issuer's registration endpoint and receives an Issuer-signed Posture Assertion alongside (or as) the registration response. ZTNP-specific fields (enrollment_mode, subject_kind, framework requests, sponsorship reference) are carried as extension members per [RFC7591] Section 2.4.

* *RATS attestation flow ([RFC9334])* — for deployments where attestation Evidence is generated at the Prover and submitted to a RATS Verifier (the Issuer). The resulting Attestation Result is the Posture Assertion. This is the natural pattern when Evidence is hardware-rooted (TPM, TEE) or platform-rooted (cgroup, sandbox).

* *Supply-chain transparency registry (e.g., SCITT [SCITT-ARCH])* — for deployments that want enrollment events recorded in a tamper-evident, queryable log. The Issuer registers the enrollment as a signed statement; the Posture Assertion carries a reference to the statement's registry entry for downstream audit.

* *Direct provisioning* — for closed deployments (single operator, manually-managed agent fleet), the Issuer provisions Posture Assertions out-of-band. ZTNP imposes no wire-format requirements on this case beyond the Abstract Requirements above.

10.5. Self-Enrollment Anti-Abuse

When an Issuer offers Self-Enrollment, the registration endpoint becomes a target for Sybil attacks: an adversary that can self-enroll without limit can trivially create unbounded numbers of tier <= 1 identities. Deployments offering Self-Enrollment SHOULD apply rate limits at the registration endpoint (RECOMMENDED default: at most 10 successful registrations per minute per source identifier — IP address, OAuth client_id, or deployment-specific source). High-throughput deployments MAY relax these limits with documented compensating controls.

10.6. Revocation Linkage

A Posture Assertion’s sub is the unique join key between the enrollment record and the assertions issued under it. Issuers MUST NOT reissue the same sub to a different enrolled subject after revocation; if the same logical entity re-enrolls, it MUST receive a fresh sub. Issuers operating a revocation endpoint SHOULD publish revocations immediately; Issuers without one MUST rely on the natural exp of in-flight assertions, which may leave an accepted-but-revoked window. High-stakes deployments SHOULD operate a revocation endpoint for this reason.

11. Proof-of-Possession (Optional)

This section is OPTIONAL. Channel binding (Section 8.2, mandatory when TLS) addresses cross-connection-theft. PoP is a stricter defense for high-stakes operations: it binds the Permit to an ephemeral key the Prover proves possession of on every request. PoP follows the pattern of [RFC9449] (DPoP).

The Prover generates an ephemeral asymmetric key pair at session establishment, sends the public key thumbprint in PROOF metadata, and the Requester embeds key confirmation in the Permit:

```
{ "cnf": { "jkt": "<base64url(SHA-256(public_key_JWK_thumbprint))>" } }
```

For each request, the Prover includes a short-lived JWS containing htm, htu, iat, jti, permit_id. The Requester verifies that cnf.jkt matches the signing key, iat is fresh, and jti has not been seen within the proof TTL.

PoP is RECOMMENDED for capabilities that touch pii_access_allowed, secrets_access_allowed, or code_exec_allowed flagged operations.

12. Profile Extension Mechanism

Domain-specific semantics are defined in *Profiles* that extend ZTNP's base claim schema.

A Profile MAY define: additional structured claims under claims.posture; framework-specific tier semantics, registered with the IANA Framework Tier Semantics Registry (Section 15); standard policy packs; evidence-reference conventions; additional flags.

A Profile MUST NOT: alter the required claims in Section 5.1; redefine bind, scope, or signature requirements; introduce transport behaviors incompatible with Section 9.

Profile documents are published independently. Profiles that wish to be referenced by framework_id MUST register their framework in the IANA ZTNP Security Frameworks Registry (Section 15).

13. Relationship to Existing Work

ZTNP composes with several existing specifications. This section is informative.

13.1. ZTNP Composes With

RATS [RFC9334] — RATS specifies an attestation architecture up to and including the Attestation Result. ZTNP composes with RATS as one possible upstream attestation pipeline (Section 1.5) but is *not* a RATS consumption profile and does not claim to operate within RATS' charter. RATS is hardware-attestation-centric; ZTNP's primary problem domain is software-layer security and governance posture, session negotiation, and channel-bound authorization issuance.

SCITT [SCITT-ARCH] — A Posture Assertion MAY be encoded as a SCITT-compatible COSE_Sign1 Signed Statement and registered with a Transparency Service.

OAuth 2.0 [RFC6749] and GNAP [RFC9635] — A ZTNP Permit is structurally a scoped grant. Deployments MAY emit Permits in a GNAP-compatible format.

JOSE [RFC7515] / COSE [RFC9052] — Signed-token formats for Posture Assertions and Permits.

RFC 7591 / RFC 7592 [RFC7591][RFC7592] — Section 10 defines a compatibility profile.

DPoP [RFC9449] — Section 12 follows the DPoP pattern.

TLS Channel Bindings [RFC9266] — Section 8 uses the TLS exporter mechanism.

13.2. Adjacent Layers Out of Scope

ZTNP is intentionally narrow.

A2A [A2A] — runtime envelope for inter-agent communication. ZTNP layers a posture concern over A2A. ZTNP does not specify how agents address one another or exchange artifacts.

MCP [MCP] — tool-capability semantics. ZTNP gates whether and under what constraints an agent invokes a given tool. ZTNP does not specify tool semantics.

SPIFFE / SPIRE [SPIFFE] — cryptographic workload identity. ZTNP binds posture claims to a stable identifier that MAY itself be a SPIFFE ID.

TLS / mTLS — transport security and identity. ZTNP runs over TLS and binds to it but does not replace it.

Multi-agent delegation [I-D.miller-ztip] — out of scope for ZTNP; covered by ZTIP.

14. Security Considerations

This section is REQUIRED by [RFC3552].

14.1. Assumptions

#	Assumption
A1	Issuer signing keys are not known to any adversary.
A2	IKS endpoints are served over authenticated TLS.
A3	Implementations use a reliable time source (skew < 5 minutes).
A4	Cryptographically secure random number generators.

A5	Transport provides confidentiality where Posture Assertions or Permits contain sensitive details.
A6	Requesters correctly implement policy evaluation.
A7	Sponsor keys (where Sponsors are used; Section 10.4) are managed in hardware or otherwise protected.

Table 19

14.2. Adversaries

ADV-NET (Network adversary), ADV-RESOURCE (Malicious resource), ADV-CALLER (Malicious agent), ADV-COLLUDING, ADV-ISSUER (Compromised Issuer), ADV-PARTIAL-ISSUER (Semi-trusted Issuer).

14.3. Attack Surface and Mitigations

Attack	Adversary	Mitigation	Reference
PA forgery	ADV-NET, ADV-RESOURCE, ADV-CALLER	Signature verification against IKS	Section 5, 6
Replay of valid PA	ADV-NET	Challenge binding (bind.nonce)	Section 5
Substitution	ADV-RESOURCE, ADV-CALLER	Subject binding	Section 5
Stale PA	ADV-RESOURCE	exp enforcement + freshness	Section 5
Cross-vendor tier confusion	ADV-PARTIAL-ISSUER	Framework anchoring	Section 4, 5
Algorithm downgrade	ADV-NET, ADV-RESOURCE	Allowed-algorithm list	Section 5
Issuer key compromise	ADV-ISSUER	Short PA lifetimes; revocation; key rotation	Section 6
Permit theft / cross-connection	ADV-NET, ADV-CALLER	Mandatory channel binding when TLS	Section 8

replay			
Permit replay outside session	ADV-CALLER	Optional PoP	Section 12
Sybil enrollment	ADV-CALLER	Tier-1 ceiling on Self-Enrollment	Section 10
Sponsorship forgery	(Sponsor key compromise)	HSM-protected Sponsor keys	Section 10
Clock manipulation	ADV-NET	Reliable time source	Section 5
IKS poisoning	ADV-NET	TLS on IKS; cache TTL	Section 6

Table 20

For multi-agent delegation threats (prompt injection, confused deputy, intent substitution), see ZTIP [I-D.miller-ztip].

14.4. Out of Scope

Runtime posture drift; identity establishment; evidence quality grading; side-channel attacks on signature verification; denial-of-service; supply chain compromise of an Issuer; key compromise on the Prover (subject_key from enrollment).

14.5. Channel Binding Practical Considerations

Section 8.2 makes channel binding mandatory when the transport is TLS. Implementing it requires that the application have access to the TLS session's exporter material. This is straightforward in some environments and constrained in others; this section gives implementers the operational guidance they need.

14.5.1. TLS Library Availability

The TLS exporter mechanism is defined in [RFC9266] and built on [RFC8446] Section 7.5. As of this specification's publication, the API is available in:

* *OpenSSL* 1.1.1 and later: SSL_export_keying_material().

* *BoringSSL*: SSL_export_keying_material().

* *rustls* (Rust): Connection::export_keying_material().

* *Go*: tls.ConnectionState.ExportKeyingMaterial().

* *Node.js* (LTS): tls.TLSSocket.exportKeyingMaterial().

* *Python* 3.12 and later: ssl.SSLObject.export_keying_material().

* *Java* 13 and later (Conscrypt and recent JDK builds): exporter API exposed via SSLEngine.

Older runtimes or restricted embedded TLS stacks may not expose the exporter API. Implementations targeting such environments cannot implement channel binding directly and SHOULD either upgrade the TLS stack or delegate the channel-binding step to a frontend that does support it.

14.5.2. Termination by Intermediaries

A common deployment pattern places a TLS-terminating reverse proxy (a CDN, an L7 load balancer, an API gateway) in front of the resource server. In this pattern the proxy has the exporter material; the origin does not. Two strategies:

1. *Channel binding at the proxy.* The proxy performs ZTNP enforcement (verifies Posture Assertions, issues Permits, validates channel binding on each subsequent request) and forwards already-authorized requests to the origin. The origin sees no ZTNP machinery. This is the recommended pattern when the proxy is in the same trust domain as the origin.

2. *Re-establish TLS to the origin.* The proxy re-encrypts to the origin (mTLS or service-mesh sidecar TLS), and the origin performs ZTNP enforcement against the proxy-to-origin TLS session. The proxy must forward the Posture Assertion and the channel-binding values that match the proxy-to-origin session, not the client-to-proxy session. This requires the proxy to re-issue the Permit's ch_binding against the new exporter, which means the proxy is acting as a Requester even if the client thought it was negotiating with the origin.

Strategy (1) is operationally simpler and the recommended default. Strategy (2) is appropriate when the proxy is not trusted to make policy decisions on behalf of the origin.

14.5.3. Connection Migration and Resumption

Channel binding ties a Permit to a specific TLS session's exporter output. Connection migration (HTTP/3, QUIC) and TLS session resumption (Section 2.2 of [RFC8446]) introduce new keying material; the previously-issued Permit's ch_binding will no longer match. The Permit MUST be considered invalid in the new session, and a fresh ZTNP negotiation is REQUIRED.

This is a security feature, not a limitation: a session change is exactly when channel binding is supposed to invalidate a Permit.

14.5.4. What to Do When TLS Is Not Available

Some transports do not provide TLS keying material:

* MCP-over-stdio (in-process pipes, local IPC): no TLS. The MCP profile defines a transport-equivalent *Process Binding* mechanism using ephemeral-key exchange and per-call proof-of-possession.

* Service-mesh transports that explicitly opt out of mTLS in same-trust-domain communication.

For these transports, the Permit MAY either omit ch_binding (Section 8.2) — relying on the underlying transport's trust boundary — or include a profile-defined ch_binding.method value naming a transport-equivalent binding. *Concrete transport-equivalent binding mechanisms are intentionally deferred to profile documents* (Section 13) rather than being specified in this base specification. Each transport's natural binding substrate is sufficiently different (stdio's parent-process spawning, service-mesh sidecar identity, embedded environments) that a single ZTNP-level mechanism would be either too prescriptive for some or too permissive for others. The MCP profile (draft-miller-ztnp-mcp-profile-00) addresses the MCP-over-stdio case via Process Binding (ch_binding.method = "mcp-process-ephemeral-key"); future profiles for other non-TLS transports SHOULD define their binding mechanism similarly.

14.5.5. Reference Implementation Guidance

Implementers SHOULD consult the reference TypeScript verifier at the project's repository for an example of how to wrap a TLS library's exporter API and bind the result to Permit issuance. The reference implementation includes channel-binding helpers for Node.js TLS sockets.

15. Privacy Considerations

This section is informed by [RFC6973]. ZTNP carries data that can reveal organizational and operational details; deployments SHOULD apply data minimization.

Posture data sensitivity. Posture Assertions describe vulnerabilities, incidents, and framework controls. Aggregated, this leaks security maturity and incident history. Issuers SHOULD NOT include claims more granular than necessary.

Subject identifier privacy. The sub claim is stable and long-lived. Where Provers participate in many short interactions, a stable sub enables linkability. Issuers SHOULD support pseudonymous subject identifiers where deployment context permits.

Subject key linkability. A Prover’s subject_key (bound to its Posture Assertion at enrollment time, Section 11.1) is also stable and long-lived. The thumbprint of subject_key appears in nonce_sig bindings (Section 5.5) and in PoP cnf.jkt values (Section 12). An observer who can correlate signatures or PoP proofs across sessions can link sessions to the same subject even if the sub claim is rotated or pseudonymous. Provers concerned with cross-session unlinkability MAY use a per-session ephemeral key for PoP and rely on the stable subject_key only at enrollment and verification points where stability is required. Profile documents MAY specify rotation conventions for subject_key itself.

Issuer linkability. A Requester learns which Issuer assessed a given Prover. Provers MAY hold Posture Assertions from multiple Issuers and select per-session.

Evidence references. evidence_refs and posture claims SHOULD reference evidence by hash rather than embedding raw artifacts.

Discovery leakage. Discovery responses advertise issuers and frameworks. Provers SHOULD limit Discovery to authenticated callers when posture details are sensitive.

Logging. Logs containing Permit IDs and subject identifiers SHOULD be access-controlled and retention-bounded.

Cross-session correlation. Channel-bound Permits cannot be linked across sessions by an external observer.

Children and protected categories. Profiles operating in regulated environments (healthcare, education, financial services) MUST address regulatory privacy obligations.

16. IANA Considerations

This document requests the following IANA actions.

16.1. Registry Strategy

ZTNP defines several extension namespaces (flags, constraint types, assessment methods, reason codes, binding modes, adoption postures, channel binding methods, enrollment modes). Each is administered as a separate IANA registry rather than collapsed into a single "ZTNP parameters" registry, because the value spaces are semantically disjoint and a flat registry would invite name collisions across categories.

All ZTNP registries created by this document use the Specification Required policy ([RFC8126] Section 4.6). This is a deliberate choice over the heavier alternatives:

* _Standards Action_ (full IETF consensus per [RFC8126] Section 4.9) would require an RFC for every flag or constraint name, which is disproportionate to the typical extension — a single boolean flag, a single denial reason code — and would create a permanent backlog. Profile authors and deployment vendors would be unable to register their extensions in any reasonable timeframe.

* _IETF Review_ (Section 4.8) and _RFC Required_ (Section 4.7) have the same problem at smaller scale.

* _Expert Review_ alone (Section 4.5) does not require a written specification, which would risk under-documented entries that future implementers cannot interpret.

Specification Required strikes the right balance: anyone with a publicly accessible written specification — an RFC, an industry-body publication, or an Internet-Draft with a stable baseline — can register an extension after Designated Expert sign-off. The administrative load on IANA is modest because the gating work happens at the DE rather than through full WG processing.

16.2. Designated Expert Guidance

Designated Experts (DEs) appointed for ZTNP registries SHOULD evaluate registration requests against the following criteria:

* *Specification stability.* A registration MUST cite a publicly accessible, version-stable specification (an RFC, an ISO/IEC or NIST publication, a formally published industry specification, an Internet-Draft with a stable baseline, or equivalent).

Registrations that cite only blog posts, unversioned web pages, or specifications under active drafting without any published baseline SHOULD be rejected.

* *Identifiable change controller.* Each registration MUST name a change controller capable of authorizing future updates to the entry.

* *Non-overlap with existing entries.* Registrations whose semantics duplicate an existing entry under a different name SHOULD be rejected.

* *Interoperability impact.* For registries that affect protocol behavior (Constraint Type, Binding Mode, Reason Code, Channel Binding Method), DEs SHOULD assess whether the new value can be interpreted unambiguously by an implementation that has not previously seen it.

* *Protocol scope.* Registrations MUST NOT redefine ZTNP semantics; they extend them.

DEs SHOULD respond to registration requests within 30 days. Where a DE has a conflict of interest with a particular registration, the DE SHOULD recuse and request a substitute via IANA.

16.3. Well-Known URI Registrations

Register the following entries in the "Well-Known URIs" registry [RFC8615]:

URI Suffix	Change Controller	Reference	Status
ztnp	IETF	This document, Section 6	permanent
ztnp-sponsor-keys	IETF	This document, Section 10.4	permanent

Table 21

16.4. Media Type Registrations

Register the following media types in the "Media Types" registry per [RFC6838]:

Type	Description	Reference
application/posture-assertion+jwt	A Posture Assertion serialized as a JWS Compact Serialization [RFC7515]	This document, Section 5
application/posture-assertion+cwt	A Posture Assertion serialized as a COSE_Sign1 [RFC9052] CWT	This document, Section 5
application/ztnp-permit+jwt	A ZTNP Permit serialized as a JWS Compact Serialization	This document, Section 8
application/ztnp-permit+cwt	A ZTNP Permit serialized as a COSE_Sign1	This document, Section 8

Table 22

Each media type's registration template follows [RFC6838]; security considerations for each are those of this document (Section 15).

16.5. Note on Framework Identifiers

ZTNP does not request an IANA registry of security frameworks. Framework identifiers (framework_id claim, Section 5.2) are URIs published by the framework's authoring organization (NIST, ISO/IEC, OWASP, AICPA, the European Commission, vendor-publishers, or enterprises operating internal frameworks). Section 4.4 motivates this design choice; the informative table of well-known canonical URIs in Section 4.4 is non-normative guidance.

This explicit non-registration is itself an IANA action: this document requests that future ZTNP-related allocations avoid creating a centralized framework directory in IANA's namespace, leaving framework identification to the publishers of those frameworks.

16.6. ZTNP Posture Assertion Flag Names Registry

This document requests creation of a registry titled "ZTNP Posture Assertion Flag Names". It administers names usable in the claims.posture.flags field of Posture Assertions (Section 5).

Allocation Policy: Specification Required (per [RFC8126]).

Registration Template:

* *Name*: identifier matching [a-z][a-z0-9_]*

* *Description*: prose definition of the operational state the flag asserts

* *Reference*: specification defining the flag

* *Change Controller*

Initial Registrations:

Name	Description	Reference
critical_open	At least one open issue at "critical" severity per the Issuer’s severity scale	This document
incident_open	At least one ongoing security incident affects the subject	This document
pii_access_allowed	Subject is authorized to access personally identifiable information	This document
secrets_access_allowed	Subject is authorized to access cryptographic secrets	This document
code_exec_allowed	Subject is authorized to execute arbitrary code	This document

Table 23

16.7. ZTNP Constraint Type Registry

This document requests creation of a registry titled "ZTNP Constraint Types". It administers types usable as keys within Permit constraints objects (Section 8).

Allocation Policy: Specification Required (per [RFC8126]).

Registration Template:

* *Type*: identifier matching [a-z][a-z0-9_]*

* *Description*: prose semantics

* *Value Schema*: JSON-Schema fragment or prose schema for the constraint's value

* *Reference*: specification

* *Change Controller*

Initial Registrations:

Type	Description	Reference
data	Restricts the data classes the Permit covers	This document
actions	Restricts the action verbs the Permit covers	This document
rate_limit	Imposes a request-rate cap	This document
ttl	Bounds session duration beyond exp	This document
redaction	Specifies output redaction requirements	This document
tools	Restricts the tool identifiers the Permit covers	This document

Table 24

16.8. ZTNP Assessment Method Names Registry

This document requests creation of a registry titled "ZTNP Assessment Method Names". It administers values usable in the claims.assessment_method field of Posture Assertions (Section 5.4).

Allocation Policy: Specification Required (per [RFC8126]).

Registration Template:

* *Name*: identifier matching [a-z][a-z0-9_]*

* *Description*: prose description of the methodology

* *Determinism*: whether two assessments of the same Prover are expected to produce the same outcome (deterministic / non-deterministic / mixed)

* *Reference*: specification

* *Change Controller*

Initial Registrations:

Name	Description	Determinism	Reference
deterministic_checklist	Evidence checked against a fixed rule set with deterministic pass/fail	deterministic	This document
automated_scan	Programmatic scanning (vulnerability scanners, static analyzers, configuration auditors)	deterministic	This document
human_review	Human-led assessment by qualified reviewers	non-deterministic	This document
llm_evaluator	Evaluation conducted by a language model interpreting evidence	non-deterministic	This document
hybrid	Combination of two or more of the above	mixed	This document
unspecified	Methodology not disclosed	mixed	This document

Table 25

Profile-defined values that are not generally reusable across profiles SHOULD use a profile-prefixed name (e.g., acme_internal_v1) to avoid collisions.

16.9. ZTNP Denial Reason Code Registry

This document requests creation of a registry titled "ZTNP Denial Reason Codes". It administers reason-code strings returned by Requesters when a negotiation, validation, or enrollment step fails.

Allocation Policy: Specification Required (per [RFC8126]).

Registration Template:

* *Code*: identifier matching [A-Z][A-Z0-9_]*
* *Description*: prose definition of the failure condition
* *Reference*: specification
* *Change Controller*

Initial Registrations (all reference this document):

Code	Description
PA_MISSING	Required Posture Assertion was not presented
PA_EXPIRED	Posture Assertion is past its exp
PA_INVALID_SIG	Posture Assertion signature did not verify
PA_BINDING_FAILED	Posture Assertion binding (nonce, channel, or subject) did not match
PA_ISSUER_UNKNOWN	Issuer is not in the Requester’s trusted-issuer set
PA_FRAMEWORK_UNKNOWN	The Posture Assertion’s framework_id URI is not in the Requester’s allowed-frameworks list.
POLICY_TIER_LOW	Posture tier is below the policy minimum
POLICY_FLAG_BLOCKED	A blocking flag is asserted on the Posture Assertion

POLICY_FRESHNESS	Posture Assertion is older than the policy freshness window
POLICY_FRAMEWORK_MISMATCH	The Posture Assertion's framework_id is well-formed but is not in the (otherwise valid) policy's allowed-frameworks list. Distinct from POLICY_INCOMPLETE — the policy is fine; the assertion's framework is the problem.
POLICY_INCOMPLETE	The Requester's policy is malformed (e.g., specifies tier_min without either framework_id or issuers_allowed). A conforming policy engine SHOULD refuse to evaluate such a policy. Distinct from POLICY_FRAMEWORK_MISMATCH — the assertion is not the problem; the policy is.
POLICY_METHOD_MISMATCH	Posture Assertion's assessment_method is outside the policy's assessment_method_allowed list
SUBJECT_MISMATCH	Subject identifier does not match expectations
PERMIT_EXPIRED	Permit is past its exp
PERMIT_CHANNEL_MISMATCH	Permit channel binding did not match
PERMIT_SCOPE_VIOLATION	Operation is outside Permit scope
POP_PROOF_INVALID	Proof-of-possession proof did not verify
ENROLL_TIER_EXCEEDED	Posture Assertion's enrollment_mode is self but its tier exceeds the Self-Enrollment ceiling (Section 11.2). Returned by a Requester verifying the assertion, not by an enrollment server.

Table 26

ZTIP [I-D.miller-ztip] registers additional codes in this registry.

16.10. ZTNP Binding Mode Registry

This document requests creation of a registry titled "ZTNP Binding Modes". It administers identifiers describing how Posture Assertions and Permits are bound to a session (Section 8).

Allocation Policy: Specification Required (per [RFC8126]).

Registration Template:

* *Mode*: identifier matching [A-Za-z][A-Za-z0-9-]*
* *Description*: prose semantics — what is bound and how
* *Applicability*: which transports or contexts the mode applies to
* *Reference*: specification
* *Change Controller*

Initial Registrations:

Mode	Description	Applicability	Reference
PA-R	Resource-Presents: Prover presents Posture Assertion to Requester	Any transport	This document
PA-C	Caller-Presents: Requester presents Posture Assertion to Prover	Any transport	This document
Mutual	Both parties exchange and verify Posture Assertions	Any transport	This document

Table 27

16.11. ZTNP Adoption Posture Registry

This document requests creation of a registry titled "ZTNP Adoption Postures". It administers identifiers describing the Requester’s behavior when an expected Posture Assertion is missing or fails policy (Section 7.6).

Allocation Policy: Specification Required (per [RFC8126]).

Registration Template:

* *Posture*: identifier matching [a-z][a-z0-9_-]*

* *Description*: prose definition of behavior when a PA is missing or fails

* *Issues Permit on failure?*: yes / no

* *Reference*: specification

* *Change Controller*

Initial Registrations:

Posture	Description	Issues Permit on failure?	Reference
required	Deny on missing or failing PA	no	This document
monitor-only	Proceed with session, log failure, do not issue Permit	no	This document
opportunistic	Proceed and issue Permit even without a PA; record posture in Permit	yes	This document

Table 28

16.12. ZTNP Channel Binding Method Registry

This document requests creation of a registry titled "ZTNP Channel Binding Methods". It administers identifiers usable in ch_binding.method of Permits (Section 8.2).

Allocation Policy: Specification Required (per [RFC8126]).

Registration Template:

* *Method*: identifier matching [a-z][a-z0-9_-]*

* *Description*: how the binding is computed and verified

* *Transport*: which transport(s) the method applies to

* *Unforgeability rationale*: brief explanation of why a party outside the transport's natural trust boundary cannot forge the binding

* *Reference*: specification

* *Change Controller*

Initial Registrations:

Method	Description	Transport	Reference
tls-exporter	TLS exporter material per [RFC9266] (Section 8.2)	TLS	This document
none	No channel binding; REQUIRES rationale field	Any	This document

Table 29

Profile documents that define transport-equivalent bindings (e.g., the MCP profile's mcp-process-ephemeral-key) MUST register their method in this registry before deployment in cross-organization use.

16.13. ZTNP Enrollment Mode Registry

This document requests creation of a registry titled "ZTNP Enrollment Modes". It administers identifiers describing how Issuers issue Posture Assertions to subjects (Section 10).

Allocation Policy: Specification Required (per [RFC8126]).

Registration Template:

* *Mode*: identifier matching [a-z][a-z0-9_]*

* *Description*: prose definition of the enrollment process

* *Tier ceiling*: maximum tier issuable under this mode

* *Reference*: specification

* *Change Controller*

Initial Registrations:

Mode	Description	Tier ceiling	Reference
self	Self-Enrollment: Issuer performs proof-of-possession only; no Sponsor	1	This document, Section 10
assessed	Assessed Enrollment: Issuer performs posture assessment; OPTIONAL Sponsor vouching	unbounded	This document, Section 10

Table 30

17. Implementation Status

This section records known implementations per [RFC7942]. Per RFC 7942 guidance, this section is to be removed before publication as an RFC.

A reference TypeScript verifier covering the mandatory verification path is published at:

https://github.com/agent-trust-protocols/agent-trust-protocols/tree/main/reference/typescript/ztnp

A set of test vectors covering valid Posture Assertions, expired marks, signature failures, binding failures, channel-binding failures, subject mismatches, and unknown key identifiers is published at:

https://github.com/agent-trust-protocols/agent-trust-protocols/tree/main/test-vectors/ztnp

A Python reference implementation is in progress.

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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/rfc/rfc6838>.

[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>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/rfc7517>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/rfc/rfc7519>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[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>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[RFC9266] Whited, S., "Channel Bindings for TLS 1.3", RFC 9266, DOI 10.17487/RFC9266, July 2022, <https://www.rfc-editor.org/rfc/rfc9266>.

[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>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/rfc/rfc9449>.

18.2. Informative References

[A2A] Google et al., "Agent-to-Agent Protocol", 2025.

[FIPS204] NIST, "Module-Lattice-Based Digital Signature Standard (ML-DSA)", FIPS 204, August 2024, <https://csrc.nist.gov/pubs/fips/204/final>.

[I-D.miller-ztip] Miller, J., "Zero-Trust Intent Protocol (ZTIP)", Work in Progress, Internet-Draft, draft-miller-ztip-00, n.d., <https://datatracker.ietf.org/doc/html/draft-miller-ztip-00>.

[MCP] Anthropic, "Model Context Protocol Specification", n.d., <https://modelcontextprotocol.io>.

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

[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>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/rfc/rfc6973>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <https://www.rfc-editor.org/rfc/rfc7591>.

[RFC7592] Richer, J., Ed., Jones, M., Bradley, J., and M. Machulak, "OAuth 2.0 Dynamic Client Registration Management Protocol", RFC 7592, DOI 10.17487/RFC7592, July 2015, <https://www.rfc-editor.org/rfc/rfc7592>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/rfc/rfc7942>.

[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-ARCH] Birkholz, H., "An Architecture for Trustworthy and Transparent Digital Supply Chains", Work in Progress, Internet-Draft, draft-ietf-scitt-architecture, n.d., <https://datatracker.ietf.org/doc/html/draft-ietf-scitt-architecture>.

[SPIFFE] SPIFFE Project, "Secure Production Identity Framework For Everyone", n.d., <https://spiffe.io>.

Appendix A. Conformance Profiles

An implementation MAY claim conformance to one or more of the following profiles.

A.1. ZTNP

REQUIRED for any conformance claim. Sections 5 through 10, plus channel binding (Section 8.2) when transport is TLS.

A.2. ZTNP Hardened

ZTNP PLUS Proof-of-Possession (Section 12). RECOMMENDED for sensitive-capability deployments.

ZTNP Federated and ZTNP Agentic profiles are defined in companion documents (the latter requiring ZTIP [I-D.miller-ztip]).

Appendix B. Open Questions and WG Venue

WG venue. ZTNP's primary problem domain is attestation-gated session authorization at the software layer: a Relying Party challenges a counterparty for a signed posture credential, applies local policy, and issues a session-scoped, channel-bound Permit. The author's preferred venues, in order: a SAAG-routed BoF or new WG scoped to attestation-gated session authorization; OAuth (for the Permit-issuance and token-extension portions); WIMSE (for the enrollment-requirements portion as a workload-identity concern); the Independent Submission stream as a fallback. ZTNP is *not* a RATS work item: while RATS is one possible upstream attestation pipeline (Section 1.5), ZTNP's center of gravity — framework-anchored portability, session negotiation, channel-bound Permits — sits outside the RATS architecture. RATS reviewers and contributors are welcomed in any chosen venue.

Profile registration logistics. Relationship between profile authors and framework authors.

Channel binding for non-TLS transports. Future revision SHOULD specify a binding for at least MCP-over-stdio.

MAJOR vs MINOR version criteria. Future revision will provide concrete rules.

Appendix C. Example Policies and Payloads

C.1. Example Policy (Framework-Anchored)

```
{
  "require": {
    "framework_id": "https://doi.org/10.6028/NIST.AI.100-1",
    "tier_min": 3,
    "issuers_allowed": ["https://issuer-x.example", "https://issuer-y.example"],
    "freshness_seconds": 86400,
    "flags": {
      "critical_open": false,
      "incident_open": false
    }
  }
}
```

C.2. Example Posture Assertion Payload

```
{
  "ver": "0.2",
  "iss": "https://issuer.example",
  "sub": "agent:acme-corp/data-processor",
  "iat": 1745500800,
  "exp": 1745587200,
  "jti": "pa_01HVXYZ123ABC456DEF",
  "framework_id": "https://doi.org/10.6028/NIST.AI.100-1",
  "tier": 3,
  "scope": {
    "kind": "agent",
    "target": "https://api.acme.com/agents/data-processor",
    "env": "prod"
  },
  "claims": {
    "flags": {
      "critical_open": false,
      "incident_open": false,
      "pii_access_allowed": true
    }
  },
  "bind": {
    "method": "nonce_hash",
    "nonce": "dGhpcyBpcyBhIGhhc2hlZCBub25jZQ==",
    "ctx": "mcp",
    "aud": "agent:requester-corp/orchestrator"
  }
}
```


Acknowledgments

The author thanks the early reviewers for feedback during the draft cycle and welcomes broader community input as the work progresses toward IETF submission.

Author’s Address

Jake Miller
Email: jake@zivis.ai