Network Working Group
Internet-Draft
Intended status: Informational
Expires: 15 November 2026

C. Munoz
Keel API, Inc.

14 May 2026

A SCITT Profile for Pre-Execution AI Action Authorization Records
draft-munoz-scitt-permit-profile-00

Abstract

This document specifies a SCITT (Supply Chain Integrity, Transparency, and Trust) profile for pre-execution authorization records of AI agent actions. The profile defines a Signed Statement type, the "Pre-Execution Authorization Record" (also called a Permit), that records a policy-evaluated decision to allow, deny, or challenge an AI agent action before that action is dispatched to a model provider, tool, or service. The profile cryptographically binds the authorization decision to the canonical bytes of the request that is subsequently dispatched, enabling independently verifiable "authorized request equals dispatched request" assertions. The profile composes with adjacent profiles for human-authority binding, post-execution material-action evidence, and content-refusal events, referenced rather than replicated.

Status of This Memo

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

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

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

This Internet-Draft will expire on 15 November 2026.

Copyright Notice

Table of Contents

1. Introduction

The SCITT architecture [I-D.ietf-scitt-architecture] defines an abstract framework for the production, registration, and verification of signed statements made about supply-chain artifacts. Pre-execution authorization decisions for AI agent actions are a class of statement that fits within this architecture but that none of the currently active SCITT profile drafts addresses directly.

This document defines such a profile. The profile's central artifact is a "Pre-Execution Authorization Record" (referred to throughout as a "Permit"), which is a signed statement that records (a) the policy that was evaluated, (b) the decision reached, (c) the subject of the decision, (d) the resource and action authorized, and (e) a commitment to the canonical bytes of the request body that will subsequently be dispatched.

The pre-execution-to-dispatch cryptographic binding is the distinctive technical contribution of this profile. A Permit is not merely a record that an authorization decision was made; it is a commitment to a specific canonical request, such that any modification of the dispatched bytes between authorization and dispatch is detectable by any third party.

As of 2026, several distinct categories of work are converging on runtime AI governance. These include governance capabilities native to application-delivery platforms, security and containment tooling for AI execution environments, enterprise organizational-governance frameworks for AI accountability, and cryptographic execution-trust primitives at the execution boundary. These categories operate at different layers and are largely complementary rather than mutually exclusive.

This profile addresses a gap that none of those categories fills directly: the pre-execution decision record. A Permit is a signed, independently verifiable record of the authorization decision reached before an AI agent action is dispatched. This document offers the

Permit as a candidate canonical decision artifact for AI execution. Canonical status is earned through profile adoption and interoperable implementation; it is not asserted here. Framing the Permit as a candidate, rather than as the established canonical artifact, preserves the openness expected of standards-track work.

1.1. Scope

This profile specifies:

* The Permit object as a SCITT Signed Statement
* The COSE_Sign1 [RFC9052] envelope binding for Permits and paired closure records
* The Receipt type used to demonstrate inclusion of a Permit in a hash-chain transparency log
* The canonicalization rules applied to request bytes for digest commitment
* Composition with the WIMSE pre-execution authorization profile [I-D.klrc-aiagent-auth], the SCITT AI agent execution profile [I-D.emirdag-scitt-ai-agent-execution], the SCITT refusal events profile [I-D.kamimura-scitt-refusal-events], and the OMP human authority binding profile [I-D.veridom-omp]

This profile does not specify:

* A policy language or evaluation engine
* A runtime, gateway, or proxy for emitting Permits
* An identity or RBAC model for subjects
* Live runtime API envelopes or network protocols
* Storage, indexing, or query semantics for Permits

These remain implementation-defined.

1.2. Relationship to Existing Work

A complete reference specification for the Permit object as deployed by the reference implementation is published at [KEEL-PERMIT]. This document profiles that specification for SCITT consumption. Implementations that conform to the Permit specification at [KEEL-PERMIT] also conform to this profile when they additionally satisfy the COSE_Sign1 envelope and receipt rules in this document.

1.3. Terminology

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.

This document uses the following SCITT terms as defined in [I-D.ietf-scitt-architecture]: Signed Statement, Receipt, Transparent Statement, Issuer, Transparency Service, Verifier.

Additional terms defined in this document:

Permit: A Signed Statement of type application/permit-v1+json that records a pre-execution authorization decision and a commitment to the canonical request bytes that will subsequently be dispatched.

Closure Record: A Signed Statement, paired with a Permit, that records the post-dispatch outcome of an authorized AI agent action, including digests of the bytes received from the provider and the bytes delivered to the client.

binding_request_hash: A SHA-256 digest committed inside a Permit, computed over the canonical wire-body bytes of the request that will be dispatched. See Section 4.

dispatch_request_digest_v1: A SHA-256 digest committed inside a Closure Record, equal to the corresponding Permit’s binding_request_hash when no modification of the request body occurred between authorization and dispatch.

Authorized Request: The canonical bytes committed by binding_request_hash.

Dispatched Request: The canonical bytes committed by dispatch_request_digest_v1.

2. Background: The Permit Object

A Permit is a JSON object that records the following information at minimum:

* An identifier and a project (tenancy) scope
* A decision: one of "allow", "deny", "challenge"
* A subject (subject_type plus subject_id)
* A resource (resource_provider and resource_model) and an action label
* A policy identifier and a policy version
* A request fingerprint (a SHA-256 derived from a stripped form of the request semantics; used for replay correlation, not for byte-level commitment)
* A binding_request_hash (the SHA-256 commitment to canonical request bytes; see Section 4)
* A creation timestamp

A Permit MAY additionally carry decision details, constraints, budgets, routing metadata, and post-execution accounting fields. These are descriptive and do not affect the cryptographic shape of the artifact.

A complete normative specification of the Permit object is in [KEEL-PERMIT].

3. The Permit Profile of SCITT

3.1. Signed Statement

A Permit, when its reserved signature field is populated, is a Signed Statement in the sense of [I-D.ietf-scitt-architecture]. The Signed Statement structure is a COSE_Sign1 envelope [RFC9052].

The COSE_Sign1 structure MUST be constructed as follows:

* The payload is the canonical JSON serialization of the Permit object, encoded as UTF-8 bytes.
* The protected header MUST contain at minimum:

  - The algorithm identifier (alg). Implementations MUST support EdDSA (alg -8) [RFC9053]. Implementations MAY support ES256 (alg -7).

  - A key identifier (kid) resolvable through the Issuer's published key manifest.

  - A content type indicating the payload media type: application/permit-v1+json.

* The unprotected header MAY contain implementation-specific fields. These MUST NOT affect verification semantics.

A Permit Signed Statement is the cryptographic commitment of the Issuer to the authorization decision recorded by the Permit object.

3.2. Paired Closure Record

For Permits whose decision is "allow" and whose binding_request_hash is non-null, the Issuer MUST produce a paired Closure Record after the authorized request has been dispatched. The Closure Record is a separate Signed Statement that commits to:

* The dispatch_request_digest_v1: equal to the Permit's binding_request_hash

* The provider_response_digest_v1: a SHA-256 over the raw bytes received from the provider or tool

* The client_response_digest_v1: a SHA-256 over the raw bytes delivered to the client response writer

* Status, timing, and accounting fields

The Closure Record's COSE_Sign1 envelope follows the same rules as the Permit's, with the content type application/closure-v2+json.

A Permit and its paired Closure Record are cryptographically linked and jointly required for verification. Verifiers MUST check that the Closure Record's dispatch_request_digest_v1 equals the Permit's binding_request_hash. A mismatch indicates that the request authorized by the Permit was not the request that was dispatched; this is the canonical "approved-but-modified" tampering signature.

3.3. Receipt

This profile uses a linked-chain Receipt format rather than a Merkle tree inclusion proof. The Transparency Service maintains a per-scope hash chain where each entry's record_hash incorporates the previous entry's record_hash, providing append-only tamper-evidence.

A Receipt for a Permit consists of:

* The chain segment from a known signed checkpoint to (and including) the entry that records the Permit's identifier

* The signed checkpoint itself (a COSE_Sign1 over the latest record_hash, signed by the Transparency Service)

Verification of a Receipt requires:

* Recomputing each entry's record_hash in the supplied segment per the chain entry algorithm specified in [KEEL-PERMIT]

* Verifying each entry's prev_hash equals the previous entry's record_hash

* Verifying the checkpoint signature against the Transparency Service's published key

Verification time is O(n) in the size of the supplied chain segment, where n is the distance from the supplied checkpoint to the entry under verification. Implementations MAY publish checkpoints periodically to bound n.

Discussion of the trade-offs between linked-chain Receipts and Merkle-tree Receipts appears in Section 7.

3.4. Transparent Statement

A Transparent Statement, in the sense of [I-D.ietf-scitt-architecture], consists of a Permit (Signed Statement) accompanied by its Receipt and, for "allow" decisions, the paired Closure Record (a second Signed Statement) and its Receipt.

The full delivery envelope for one or more Transparent Statements is an audit-export bundle, specified normatively in [KEEL-PERMIT]. Each record in the bundle contains the Permit, the Closure Record (when applicable), and the chain entries that constitute the Receipts.

3.5. Transparency Service Role

The Issuer and the Transparency Service MAY be the same operator. The reference implementation in [KEEL-PERMIT] combines both roles; this is permitted by the SCITT architecture.

Issuers operating in the dual role MUST document this in their published key manifest and Transparency Service operating specification, including the keys used for each role.

Issuers MAY externalize the Transparency Service to a third party. In that case, the Permit Signed Statement is registered with the external Transparency Service, which returns a Receipt that the Issuer attaches to the Permit before delivering the Transparent Statement to a verifier.

3.6. Verifier Behavior

A conforming Verifier MUST:

1. Verify the COSE_Sign1 signature on the Permit against the Issuer’s public key, resolved via the kid header.

2. Verify the Receipt: recompute each chain entry’s record_hash, verify prev_hash continuity within the supplied segment, verify the checkpoint signature.

3. For Permits with decision "allow" and non-null binding_request_hash: verify the existence and validity of the paired Closure Record. Verify the Closure Record’s COSE_Sign1 signature. Verify that the Closure Record’s dispatch_request_digest_v1 equals the Permit’s binding_request_hash.

4. For Closure Records with status "closed": verify that provider_response_digest_v1 and client_response_digest_v1 are present and that they match the corresponding chain event payloads.

A Verifier MUST emit a stable failure code on any integrity violation. The failure-code taxonomy specified in [KEEL-PERMIT] is the normative output of a conforming Verifier for this profile.

4. Canonicalization

The binding_request_hash is computed over canonical bytes derived from the request payload via a documented canonicalization pipeline.

The pipeline applies the following steps in order:

1. Strip volatile observability metadata keys from the payload (request IDs, trace IDs, timestamps, idempotency keys). The normative list is in [KEEL-PERMIT].

2. Strip sensitive credential keys from the payload (authorization headers, API keys). The normative list is in [KEEL-PERMIT].

3. Canonicalize the resulting payload by sorting object keys lexicographically, removing insignificant whitespace, and encoding as UTF-8 bytes.

The pre-canonicalization stripping steps are forensic-safety properties of this profile. Stripping volatile metadata makes the digest stable across retries and observability variation, supporting idempotency-correlated forensic analysis. Stripping sensitive credential keys prevents long-lived hashes from being credential brute-force targets.

The canonicalization step in [KEEL-PERMIT] is JCS-inspired [RFC8785] but does not currently claim strict JCS compliance. The documented deviations (number serialization edge cases, certain control-character escapes, Unicode normalization stance) are enumerated in [KEEL-PERMIT]. Implementations relying on cross-implementation byte-equivalence MUST validate against the published test vectors in [KEEL-PERMIT].

Future versions of this profile MAY align the canonicalization step with strict RFC 8785 JCS, while preserving the pre-canonicalization stripping steps as profile-specific input transformations. Such a migration would be accompanied by a new chain format version identifier; existing Permits and Receipts remain valid under the older canonicalization indefinitely.

5. COSE_Sign1 Envelope Binding

The reference Permit object specification in [KEEL-PERMIT] signs over the hexadecimal string representation of SHA-256(canonical_json(payload)). The COSE_Sign1 envelope [RFC9052] signs over a CBOR Sig_structure. These produce different signed bytes.

This profile addresses the difference through a dual-signature envelope. Conforming Issuers MAY emit Permits in either of two modes:

* Mode A (legacy envelope only): The Permit carries the Ed25519 signature over hex(SHA-256(canonical_json(payload))) as defined in [KEEL-PERMIT]. The Permit is not directly consumable as a SCITT Signed Statement under this profile.

* Mode B (dual envelope): The Permit carries both the legacy signature and a COSE_Sign1 signature over the same canonical payload bytes. The COSE_Sign1 signature is the Signed Statement for SCITT purposes; the legacy signature provides backward compatibility with non-SCITT-aware Verifiers.

Conforming Verifiers MUST accept Mode B Permits and verify the COSE_Sign1 signature. Verifiers MAY accept Mode A Permits in mixed-deployment environments where SCITT compatibility is not required.

A future version of this profile MAY deprecate Mode A. This profile-00 does not.

6. Composition with Adjacent Profiles

This profile composes with four adjacent IETF profiles, each addressed in a subsection below. Composition is one-directional in each case: this profile defines reference mechanisms by which a Permit may point to artifacts produced under the adjacent profile. This profile does not require modifications to any adjacent profile. A final subsection situates the profile against the broader category of execution-boundary cryptographic trust primitives.

6.1. Composition with WIMSE AI Agent Authentication and Authorization

The WIMSE draft [I-D.klrc-aiagent-auth] specifies how an AI agent obtains an identity and a runtime authorization grant. That draft does not define a signed evidence record of the authorization decision.

A Permit emitted by a WIMSE-authorized agent provides such a record. Issuers SHOULD set the Permit’s subject_type to "spiffe" and subject_id to the agent’s SPIFFE URI when the agent identity is established via WIMSE. The OAuth access token, when present at dispatch time, MAY be referenced through an extension claim in the Permit’s decision_details, though this profile does not require it.

A companion profile that specifies WIMSE-side integration in detail has been prepared as separate work.

6.2. Composition with SCITT AI Agent Execution

The SCITT AI agent execution profile [I-D.emirdag-scitt-ai-agent-execution] defines an AgentInteractionRecord (AIR) for post-execution evidence of agent actions. AIR's existing bridge fields (parent_record_id, workflow_id, trace_id, external_refs) carry the linkage to pre-execution authorization records.

Issuers that emit both Permits and AIRs SHOULD populate the AIR's parent_record_id with the corresponding Permit's identifier. When a Closure Record paired with the Permit is also emitted, Issuers SHOULD additionally populate the AIR's external_refs with a reference to the Closure Record's identifier. The mapping makes the pre-execution authorization, the dispatch binding, and the post-execution material-action evidence into a continuous verifiable chain.

This profile does not specify any modification to AIR.

6.3. Composition with SCITT Refusal Events

The SCITT refusal events profile [I-D.kamimura-scitt-refusal-events] defines four event types (ATTEMPT, DENY, GENERATE, ERROR) for content-generation refusal at the AI system level. A Permit's decision field is more general than refusal-events' event-type field: a Permit decision of "deny" covers content refusal as a special case but also covers policy-level denial outside the content-safety context.

Issuers that emit both refusal events and Permits SHOULD reference the corresponding Permit's identifier in the refusal event's external claims. The completeness invariant in [I-D.kamimura-scitt-refusal-events] composes naturally: every ATTEMPT has exactly one outcome, and the corresponding Permit captures the outcome's authorization context.

6.4. Composition with OMP Human Authority Binding

The OMP profile [I-D.veridom-omp] defines a human-authority binding artifact that records whether a named Accountable Officer held valid delegated authority for a regulated AI-assisted decision. OMP's central artifact is the authority_binding object, with results BOUND, AUTHORITY_UNBOUND, or EXEMPT.

A Permit MAY reference an OMP authority_binding artifact through an optional authority_context field carrying a URI and digest pointer. The reference is informational; this profile does not interpret OMP semantics within the Permit. Verifiers of this profile do not

validate the referenced OMP artifact; they verify only that the reference is well-formed and that the digest matches if the artifact is retrieved.

The composition pattern: in a regulated AI-assisted decision, the OMP authority_binding artifact records whether the human had authority; the Permit records what the AI was authorized to do. Both records are required for full evidentiary coverage; this profile delivers the AI-action-authorization layer and points to the human-authority layer.

6.5. Relationship to Execution-Boundary Trust Primitives

Separately from the four adjacent profiles above, a broader category of cryptographic trust primitives operates at the execution boundary itself. Primitives in this category verify the authenticity of an individual execution payload, for example by checking a cryptographic signature over the bytes of a single request or response as that payload crosses the boundary.

Such primitives operate at a different layer than the one this profile fills. A Permit is a pre-execution decision record: it captures the authorization decision reached before an AI agent action is dispatched, and this profile binds that decision to the canonical bytes of the dispatched request. An execution-boundary trust primitive instead attests to the authenticity of a payload at the boundary. The two layers compose: they are not the same slot. A deployment may emit a Permit as the pre-execution decision record and also apply an execution-boundary primitive to the payload, with each providing assurance the other does not.

This profile neither specifies nor requires an execution-boundary trust primitive. The relationship is noted here as a layering observation, so that implementers positioning a Permit-emitting deployment alongside such a primitive recognize the pre-execution decision record and the execution-boundary authenticity check as distinct and composable layers.

7. Canonicalization and Receipt Choices

The profile makes two implementation choices that diverge from the most common SCITT conventions to date: a JCS-inspired (rather than strict-JCS) canonicalization, and a linked-chain (rather than Merkle-tree) Receipt format. This section names the trade-offs.

7.1. Linked-Chain vs. Merkle-Tree Receipts

The reference implementation in [KEEL-PERMIT] uses a per-scope linked-list hash chain. Each entry's record_hash incorporates the previous entry's record_hash. Tamper-evidence is established by recomputing the chain segment and verifying continuity.

Merkle-tree-based transparency logs (as exemplified by Certificate Transparency [RFC9162] [RFC6962]) produce O(log n) inclusion proofs. The linked-chain construction produces O(n) inclusion proofs where n is the distance from the supplied checkpoint to the entry under verification.

The SCITT architecture [I-D.ietf-scitt-architecture] does not mandate Merkle-tree-based receipts. It mandates the integrity property: append-only, tamper-evident, verifiable inclusion. Both constructions satisfy that property.

The trade-offs:

* The linked-chain construction is structurally simpler and matches the reference implementation as deployed today.

* Inclusion proofs are larger and verification is linear in chain segment size. Periodic checkpoints bound this size.

* Migration to a Merkle-tree-based transparency log is a separate consideration not addressed in this profile.

7.2. Canonicalization

The canonicalization pipeline in Section 4 composes volatile-key stripping and sensitive-key stripping with a JCS-inspired serialization. The stripping steps are forensic-safety properties of this profile and MUST NOT be omitted.

The serialization step's deviations from strict RFC 8785 JCS [RFC8785] are enumerated in [KEEL-PERMIT]. Implementations producing or consuming Permits across language boundaries MUST validate against the published test vectors.

A future revision of this profile MAY adopt strict RFC 8785 JCS for the serialization step, while preserving the stripping steps. Such a transition would be accompanied by a new chain format version identifier; legacy Permits remain valid under the older serialization indefinitely.

8. Security Considerations

8.1. Omission Attacks

A Verifier consuming a Transparent Statement learns only about events that appear in the supplied chain. Events that were never recorded are not detectable by this profile in isolation. Architectural deployments that route all AI dispatch through a Permit-emitting layer reduce the risk of unlogged events; architectural review of the deployment is required.

8.2. Log Equivocation

A Transparency Service operator may, in principle, present different chain views to different Verifiers. This profile does not by itself defend against log equivocation. Deployments requiring such defense SHOULD anchor checkpoints via independent witnesses (RFC 3161 timestamp tokens [RFC3161], externally-anchored notary services, or multi-witness anchoring patterns).

8.3. Approval-Dispatch Divergence

The two-sided byte binding between Permit.binding_request_hash and the paired Closure Record's dispatch_request_digest_v1 is the canonical defense against modification of the request body between authorization and dispatch. Verifiers MUST check this equality.

Equality across two independently signed records (Permit and Closure Record) makes after-the-fact substitution of the authorized artifact detectable: an attacker who modifies the Permit must re-sign it, and an attacker who modifies the Closure Record must re-sign it, and any modification creates a verifiable mismatch.

8.4. Canonicalization Brittleness

Byte-level canonicalization is sensitive to floating-point representation, number serialization, and Unicode handling. This profile addresses brittleness by:

* Documenting the canonicalization rules explicitly (Section 4, [KEEL-PERMIT])

* Requiring volatile-key and sensitive-key stripping before canonicalization

* Recommending cross-implementation test vectors

Implementations relying on cross-language byte-equivalence MUST validate against the published test vectors.

8.5. Credential Containment

The sensitive-key stripping step in Section 4 removes common credential headers (authorization, apikey, x-api-key, and provider-specific variants) from the payload before canonicalization. This prevents long-lived hashes from being credential brute-force targets.

Implementations MUST NOT skip the stripping step. The stripping step is a forensic-safety property of this profile, not an optimization.

8.6. Subject Identifier Privacy

When subject_type is "spiffe" and subject_id is a SPIFFE URI, the subject is identified by trust domain and workload path. When subject_type identifies a human user, the subject_id may directly or indirectly identify a person. Issuers SHOULD consider whether the subject_id requires pseudonymization for the audience consuming the Transparent Statement.

8.7. Hashes of Prompt Content

Hashes of LLM prompts can be subject to dictionary attacks if the prompt space is small and predictable. The request_fingerprint defined in [KEEL-PERMIT] is computed over a stripped, canonical form, not the raw prompt; it is intended for replay correlation, not for prompt-content confidentiality. The binding_request_hash is computed over canonical request bytes (after stripping) and similarly does not commit the raw prompt.

Issuers deploying this profile in contexts where prompt-content confidentiality matters MAY supplement the digests defined here with salted or HMAC-based commitments.

9. Privacy Considerations

9.1. Sensitive Data in Wire Bodies

The bytes committed by binding_request_hash and dispatch_request_digest_v1 are the canonical bytes of the request, after stripping volatile and sensitive keys. Implementations MUST verify that no sensitive data outside the stripped key set is present in the request body before computing the hash.

9.2. Cross-Border Considerations

When Permits and the artifacts they reference cross jurisdictional boundaries, the data minimization properties of the profile (no raw prompt, no raw credential, no raw provider response) apply. Issuers SHOULD nevertheless consider whether the structured fields of the Permit (subject identifiers, policy identifiers, resource identifiers) contain regulated data that requires additional handling.

9.3. Logged Identifiers

Subject identifiers, policy identifiers, and request fingerprints in the Permit may, in aggregate, support re-identification of end-users or correlation across requests. Issuers SHOULD apply appropriate access controls to the Transparency Service log and audit-export bundles.

10. IANA Considerations

This document requests the registration of a media type for the Permit object: application/permit-v1+json.

The proposed registration template:

* Type name: application
* Subtype name: permit-v1+json
* Required parameters: none
* Optional parameters: none
* Encoding considerations: 8bit. JSON; UTF-8 encoded; see [KEEL-PERMIT] for canonicalization rules used in hash-input contexts.
* Security considerations: see Section 8 of this document.
* Interoperability considerations: see [KEEL-PERMIT].
* Published specification: this document and [KEEL-PERMIT].
* Applications that use this media type: SCITT-aware verifiers, AI governance evidence systems, audit log processors.
* Fragment identifier considerations: as per RFC 6838 for +json structured syntax suffix.

* Person and email address to contact for further information: Christian Munoz christian@keelapi.com

* Intended usage: COMMON

* Restrictions on usage: none

* Author/Change controller: IETF

* Provisional registration: no

A future revision of this profile MAY request additional registrations for the closure record media type (application/closure-v2+json) and for COSE header parameters specific to this profile.

11. Implementation Status

This section is to be removed before publication as an RFC.

A reference Issuer implementation, a reference Verifier implementation (pip-installable as keel-verifier), and a complete Permit specification (including JSON schemas, canonical-JSON rules, chain entry algorithm, and an audit-export bundle format) are published at [KEEL-PERMIT] under the Apache License 2.0. The specification is at version 1.0.0 as of this writing.

The implementation does not currently emit COSE_Sign1 envelopes; adding the dual-signature envelope described in this profile is work in progress.

A 14-framework control mapping artifact (CCPA, CPPA ADMT, EU AI Act, GDPR Art 17, AICPA SOC 2, NIST AI RMF, ISO/IEC 42001:2023, OWASP LLM Top 10 (2025), MITRE ATLAS, OWASP API Security Top 10 (2023), OWASP ASVS v5.0.0, FedRAMP / NIST SP 800-53 Rev 5, CIS Controls v8.1, PCI DSS v4.0.1) is also published in the same repository.

12. Acknowledgments

The author thanks the SCITT working group, the authors of [I-D.emirdag-scitt-ai-agent-execution], [I-D.kamimura-scitt-refusal-events], [I-D.klrc-aiagent-auth], and [I-D.veridom-omp] for their work on adjacent profiles.

13. References

13.1. Normative References

[I-D.ietf-scitt-architecture]
Birkholz, H., Delignat-Lavaud, A., Fournet, C., Deshpande, Y., and S. Lasker, "An Architecture for Trustworthy and Transparent Digital Supply Chains", Work in Progress, Internet-Draft, draft-ietf-scitt-architecture-22, 10 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-scitt-architecture-22>.

[I-D.ietf-scitt-scrapi]
Birkholz, H., Geater, J., and A. Delignat-Lavaud, "SCITT Reference APIs", Work in Progress, Internet-Draft, draft-ietf-scitt-scrapi-09, 21 April 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-scitt-scrapi-09>.

[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>.

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <https://www.rfc-editor.org/rfc/rfc3161>.

[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>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <https://www.rfc-editor.org/rfc/rfc8785>.

[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>.

[RFC9053] Schaad, J., "CBOR Object Signing and Encryption (COSE): Initial Algorithms", RFC 9053, DOI 10.17487/RFC9053, August 2022, <https://www.rfc-editor.org/rfc/rfc9053>.

13.2. Informative References

[I-D.emirdag-scitt-ai-agent-execution]
Emirdag, P., "AI Agent Execution Profile of SCITT", Work in Progress, Internet-Draft, draft-emirdag-scitt-ai-agent-

execution-00, 11 April 2026,
<https://datatracker.ietf.org/doc/html/draft-emirdag-scitt-ai-agent-execution-00>.

[I-D.kamimura-scitt-refusal-events]
Tokachi, K., "Verifiable AI Refusal Events using SCITT", Work in Progress, Internet-Draft, draft-kamimura-scitt-refusal-events-02, 29 January 2026,
<https://datatracker.ietf.org/doc/html/draft-kamimura-scitt-refusal-events-02>.

[I-D.klrc-aiagent-auth]
Kasselman, P., Lombardo, J., Rosomakho, Y., Campbell, B., and N. Steele, "AI Agent Authentication and Authorization", Work in Progress, Internet-Draft, draft-klrc-aiagent-auth-01, 30 March 2026,
<https://datatracker.ietf.org/doc/html/draft-klrc-aiagent-auth-01>.

[I-D.veridom-omp]
Adebayo, T. and O. Apalowo, "Operating Model Protocol (OMP) Core -- Version 01: Invariant 3 -- Verifiable Delegation Binding", Work in Progress, Internet-Draft, draft-veridom-omp-01, 1 May 2026,
<https://datatracker.ietf.org/doc/html/draft-veridom-omp-01>.

[KEEL-PERMIT]
Keel API, Inc., "Keel Permit Specification", 2026,
<https://github.com/keelapi/keel-permit/blob/1818c3e04eddf9a2ab6231486ca2cdb2d250ec74/spec/permit-chain-v1.md>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013,
<https://www.rfc-editor.org/rfc/rfc6962>.

[RFC9068] Bertocci, V., "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, DOI 10.17487/RFC9068, October 2021, <https://www.rfc-editor.org/rfc/rfc9068>.

[RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, DOI 10.17487/RFC9162, December 2021, <https://www.rfc-editor.org/rfc/rfc9162>.

[RFC9421] Backman, A., Ed., Richer, J., Ed., and M. Sporny, "HTTP Message Signatures", RFC 9421, DOI 10.17487/RFC9421, February 2024, <https://www.rfc-editor.org/rfc/rfc9421>.

Appendix A. Examples

A.1. Example Permit (informative)

The following is an informative example of a Permit object in its JSON form, before COSE_Sign1 wrapping:


```
{
  "id": "9c8b7a6e-5d4c-3b2a-1f0e-d9c8b7a6e5d4",
  "project_id": "0a1b2c3d-4e5f-6a7b-8c9d-0e1f2a3b4c5d",
  "decision": "allow",
  "reason": "policy-eval-pass",
  "actions_json": [],
  "subject_type": "spiffe",
  "subject_id": "spiffe://example.org/agent/x123",
  "action_name": "chat.completions.create",
  "resource_provider": "example-llm",
  "resource_model": "example-model-1",
  "estimated_input_tokens": 1024,
  "estimated_output_tokens": 512,
  "request_fingerprint":
    "3b8d6e0e7f4c2a1d5b9e0c3a7f1d4e6b8a2c5f0d3e6b9a1c4f7d0a3e6b9c2f5d",
  "idempotency_key": "req-2026-05-14-abc",
  "policy_id": "default-allow-policy",
  "policy_version": "v3",
  "created_at": "2026-05-14T10:15:30Z",
  "binding_request_hash":
    "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0a1b2"
}
```


Figure 1: Example Permit JSON

A.2. Example Composition Reference (informative)

A Permit referencing an OMP authority_binding artifact:

```
{
  "id": "9c8b7a6e-5d4c-3b2a-1f0e-d9c8b7a6e5d4",
  "decision": "allow",
  "subject_type": "spiffe",
  "subject_id": "spiffe://bank.example/agent/loan-decision",
  "action_name": "loan.decision.assess",
  "resource_provider": "internal-llm",
  "resource_model": "loan-model-3",
  "decision_details": {
    "decision": "allow",
    "code": "policy.allow",
    "authority_context": {
      "uri":
       "https://example.bank/authority/officer-12345/2026-05-14",
      "digest":
       "sha256:b7d1c0e5a4f6b2d3c8e9a0f1b4c5d6e7f8a9b0c1d2e3f4a5",
      "signed_by": "bank.example.authority.root.2026"
    }
  },
  "binding_request_hash":
    "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0a1b2"
}
```


Figure 2: Example Permit with OMP authority_context reference

Appendix B. Open Issues for -01 and Beyond

This section is to be removed before publication as an RFC.

The following issues are open as of this -00 revision:

1. Whether to retain the dual-signature transition pattern (Mode A plus Mode B) or to deprecate Mode A in a future revision.

2. Whether to migrate the canonicalization step to strict RFC 8785 JCS, while preserving the stripping steps. Discussed in Section 7.

3. The exact normative format of the linked-chain Receipt: field layout, checkpoint signature binding, partial-segment encoding for transport.

4. Whether the COSE Sign1 protected header should carry chain integrity claims directly (sequence_number, prev_hash, record_hash) as an alternative to relying on a separate Receipt.

5. IANA registration timing: in parallel with this draft, or after WG adoption.

6. Whether to define a strict-mode profile-02 that requires Mode B only and aligns canonicalization with RFC 8785 strictly.

7. Several normative elements are currently specified in [KEEL-PERMIT] rather than in this document: the Permit object's complete field-level specification, the canonical-JSON rules and their deviations from RFC 8785, the chain-entry record_hash algorithm, the Verifier failure-code taxonomy, and the audit-export bundle format. This externalization is intentional for this -00 revision. A future revision is expected to migrate these elements into this document, or into an adopted companion specification, so that the profile is interoperable without dependence on an external document.

Feedback on any of these is welcome on the SCITT mailing list.

Author's Address

Christian Munoz
Keel API, Inc.
Email: christian@keelapi.com
URI: https://keelapi.com