Workgroup:	WIMSE
Internet-Draft:	draft-nennemann-wimse-ect-02
Published:	12 April 2026
Intended Status:	Standards Track
Expires:	14 October 2026
Author:	C. Nennemann *Independent Researcher*

Execution Context Tokens for Distributed Agentic Workflows

Abstract

This document defines Execution Context Tokens (ECTs), a JWT-based mechanism for recording task execution across distributed agentic workflows. Each ECT captures a single task, linked to predecessor tasks through a directed acyclic graph (DAG). ECTs are transported in a new Execution-Context HTTP header field. While designed for use with the WIMSE architecture, ECTs are identity-framework agnostic and can operate with any asymmetric key infrastructure including WIMSE WIT/WPT, X.509 certificates, OAuth-based credentials, or plain JWK sets.

ECTs support three assurance levels — Level 1 (unsigned JSON), Level 2 (JOSE asymmetric signing), and Level 3 (JOSE signing with audit ledger) — allowing deployments to choose the appropriate trade-off between simplicity and regulatory compliance.

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 14 October 2026.

Copyright Notice

Table of Contents

1. Introduction

Frameworks such as WIMSE [I-D.ietf-wimse-arch] authenticate workloads across call chains but do not record what those workloads actually did. This document defines Execution Context Tokens (ECTs), a JWT-based mechanism that fills the gap between workload identity and execution accountability. Each ECT captures a single task, linked to predecessor tasks through a directed acyclic graph (DAG). ECTs are designed for use with the WIMSE signing model but can operate with any identity framework that provides asymmetric key pairs with verifiable binding to agent identity.

1.1. Scope and Applicability

This document defines:

- The Execution Context Token (ECT) format (Section 3)
- Three assurance levels for ECT production and verification (Section 3.3, Section 3.4, Section 3.5)
- DAG structure for task dependency ordering (Section 5)
- An HTTP header for ECT transport (Section 4)
- Audit ledger interface requirements (Section 6)

The following are out of scope and are handled by the deployment's identity framework (e.g., WIMSE, X.509 PKI, OAuth):

- Workload authentication and identity provisioning
- Key distribution and management
- Trust domain establishment and management
- Credential lifecycle management

1.2. Assurance Levels and Applicability

ECTs support three assurance levels that govern how tokens are produced, transported, and verified. Each level builds on the payload structure defined in Section 3.1.

Level 1 (L1) — Unsigned JSON: ECTs are plain JSON objects with no cryptographic signature. Integrity depends on transport security (TLS). L1 is intended for internal deployments within a single trust domain where non-repudiation is not required, such as development environments or internal microservice meshes.

Level 2 (L2) — JOSE Asymmetric Signing: ECTs are signed as JWS Compact Serialization tokens using asymmetric keys bound to the agent's identity credential. L2 provides non-repudiation and tamper detection. This is the baseline assurance level for cross-organization and peer-to-peer deployments. L2 corresponds to the behavior defined in Version -00 of this specification.

Level 3 (L3) — JOSE Signing with Audit Ledger: L3 extends L2 by requiring that every ECT be recorded in an audit ledger with cryptographic commitment (hash chain, Merkle tree, or equivalent). L3 is intended for regulated environments that require hash-committed audit trails with tamper-evident history.

Assurance level selection is orthogonal to human-in-the-loop (HITL) policy: any level may be combined with HITL requirements. Level selection guidance is provided in Section 3.6.

1.3. Relationship to Agent Context Tokens (ACT)

The Agent Context Token (ACT) [I-D.nennemann-act] defines a two-phase authorization and accountability mechanism for agentic workflows. In the first phase an ACT Mandate authorizes an agent to perform a bounded set of actions with explicit capability constraints and delegation chains. In the second phase an ACT Record captures what the agent actually did, enabling post-hoc comparison between authorized and observed behavior.

ECTs and ACTs are complementary. ACTs answer "was this agent authorized to act, and what did it do relative to that authorization?" ECTs answer "which workload executed this task, in which trust domain, and at what assurance level?" The two tokens serve different accountability layers and a deployment **MAY** carry both simultaneously: an ACT for capability-scoped authorization and audit, and an ECT for workload-identity-bound execution recording with DAG ordering and assurance levels.

The following claims have identical semantics in both specifications: "exec_act", "jti", "wid", "inp_hash", "out_hash", and "pred". Implementations that produce both token types **MUST** use consistent values for these claims when they refer to the same task.

2. Conventions and Definitions

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

The following terms are used in this document:

Agent: An autonomous workload that executes tasks within a workflow. In WIMSE deployments, this corresponds to a WIMSE workload [I-D.ietf-wimse-arch].

Task: A discrete unit of agent work that consumes inputs and produces outputs.

Directed Acyclic Graph (DAG): A graph structure representing task dependency ordering where edges are directed and no cycles exist.

Execution Context Token (ECT): A token defined by this specification that records task execution details. At Level 2 and Level 3, an ECT is a signed JSON Web Token [RFC7519]. At Level 1, an ECT is an unsigned JSON object with the same payload structure.

Identity Credential: A credential that proves an agent's identity, such as a WIMSE Workload Identity Token (WIT), an X.509 certificate, an OAuth client credential, or a JWK with out-of-band binding. See Section 3.8.

Assurance Level: One of three tiers (L1, L2, L3) governing how an ECT is produced, transported, and verified. See Section 1.2.

Audit Ledger: An append-only, immutable log of all ECTs within a workflow or set of workflows, used for audit and verification.

Trust Domain: An administrative boundary within which agents share a common identity authority. In WIMSE deployments, this corresponds to a SPIFFE [SPIFFE] trust domain.

3. Execution Context Token Format

An Execution Context Token records a single task in a distributed workflow. The ECT payload structure is common across all assurance levels; only the envelope and verification requirements differ.

3.1. ECT Payload Structure

The ECT payload is a JSON object containing the claims defined in this section.

3.1.1. Standard Claims

The following standard JWT claims [RFC7519] are defined for ECTs:

iss: **OPTIONAL** at L1; **REQUIRED** at L2 and L3. StringOrURI. A URI identifying the issuer of the ECT. The value **MUST** correspond to the agent's identity as asserted by its identity credential (see Section 3.8). In WIMSE deployments, this **SHOULD** be the workload's SPIFFE ID in the format `spiffe://<trust-domain>/<path>`. Other deployments **MAY** use HTTPS URLs, URN:UUID identifiers, or other URI schemes appropriate to the identity framework in use. L1 deployments are encouraged to include "iss" for consistency but it is not required. See Section 3.4.3 and Section 3.5.4 for the L2/L3 verification requirements.

aud: **RECOMMENDED**. StringOrURI or array of StringOrURI. The intended recipient(s) of the ECT. The "aud" claim **SHOULD** contain the identifiers of all entities that will verify the ECT. When an ECT is to be verified by both the next agent and the audit ledger independently, "aud" **MUST** be an array containing both identifiers. Each verifier checks that its own identity appears in "aud". The "aud" claim is **REQUIRED** for L2 and L3 deployments (see Section 3.4.3 and Section 3.5.4).

iat: **REQUIRED**. NumericDate. The time at which the ECT was issued.

exp: **REQUIRED**. NumericDate. The expiration time of the ECT. Implementations **SHOULD** set this to 5 to 15 minutes after "iat".

jti: **REQUIRED**. String. A unique identifier for both the ECT and the task it records, in UUID format [RFC9562]. The "jti" serves as both the token identifier (for replay detection) and the task identifier (for DAG parent references in "pred"). Receivers **MUST** reject ECTs whose "jti" has already been seen within the expiration window. When "wid" is present, uniqueness is scoped to the workflow; when "wid" is absent, uniqueness **MUST** be enforced globally across the ECT store.

3.1.2. Execution Context Claims

The following claims are defined by this specification:

wid: **OPTIONAL**. String. A workflow identifier that groups related ECTs into a single workflow. When present, **MUST** be a UUID [RFC9562].

exec_act: **REQUIRED**. String. The action or task type identifier describing what the agent performed (e.g., "process_payment", "validate_safety"). This claim name avoids collision with the "act" (Actor) claim registered by [RFC8693].

pred: **REQUIRED**. Array of strings. Predecessor task identifiers representing DAG dependencies. Each element **MUST** be the "jti" value of a previously verified ECT. An empty array indicates a root task with no dependencies. A workflow **MAY** contain multiple root tasks. The "pred" claim is always required (rather than optional with absence meaning "root task") to simplify validation logic and eliminate ambiguity between a root task and a claim accidentally omitted.

3.1.3. Data Integrity Claims

The following claims provide integrity verification for task inputs and outputs without revealing the data itself:

inp_hash: **OPTIONAL**. String. The base64url encoding (without padding) of the SHA-256 hash of the input data, computed over the raw octets of the input. SHA-256 is the mandatory algorithm with no algorithm prefix in the value, consistent with [RFC9449] and [I-D.ietf-wimse-s2s-protocol].

out_hash: **OPTIONAL**. String. The base64url encoding (without padding) of the SHA-256 hash of the output data, using the same format as "inp_hash".

3.1.4. Extension Claims

ect_ext: **OPTIONAL**. Object. A general-purpose extension object for domain-specific claims not defined by this specification. Implementations that do not understand extension claims **MUST** ignore them. Extension key names **SHOULD** use reverse domain notation (e.g., "com.example.custom_field") to avoid collisions. The serialized "ect_ext" object **MUST NOT** exceed 4096 bytes and **MUST NOT** exceed a nesting depth of 5 levels. Receivers **MUST** reject ECTs whose "ect_ext" object exceeds these limits.

3.2. Complete ECT Payload Example

The following is a complete ECT payload example. This payload structure is the same at all assurance levels; only the transport envelope differs.

```
{
  "iss": "spiffe://example.com/agent/clinical",
  "aud": "spiffe://example.com/agent/safety",
  "iat": 1772064150,
  "exp": 1772064750,
  "jti": "550e8400-e29b-41d4-a716-446655440001",

  "wid": "a0b1c2d3-e4f5-6789-abcd-ef0123456789",
  "exec_act": "recommend_treatment",
  "pred": [],

  "inp_hash": "n4bQgYhMfWWaL-qgxVrQFaO_TxsrC4Is0V1sFbDwCgg",
  "out_hash": "LCa0a2j_xo_5m0U8HTBBNBNCLXBkg7-g-YpeiGJm564",

  "ect_ext": {
    "com.example.trace_id": "abc123"
  }
}
```

Figure 1: Complete ECT Payload Example

3.3. Level 1: Unsigned JSON

At Level 1, an ECT is a plain JSON object carrying the payload defined in Section 3.1. No cryptographic signature is applied.

3.3.1. L1 Transport

L1 ECTs are transported as base64url-encoded JSON (without padding). Two mechanisms are defined:

HTTP Header: The Execution-Context header field (Section 4) carries the base64url-encoded JSON payload (without padding). The header value is NOT raw JSON; the receiver **MUST** base64url-decode the value before parsing.

HTTP Body: When the ECT is the primary request payload, the ECT **MAY** be carried in the HTTP request body with Content-Type `application/json`.

L1 ECTs **MUST** be transmitted over TLS or mTLS connections. Transport security is the sole integrity mechanism at this level.

3.3.2. L1 Verification

When an agent receives an L1 ECT, it **MUST** perform the following verification steps:

1. Parse the JSON object.
2. Verify that all required claims ("jti", "iat", "exp", "exec_act", "pred") are present and well-formed.
3. Verify that the "jti" has not been previously seen within the expiration window, consistent with the replay detection requirement in Section 3.1.2.

4. Verify the "exp" claim indicates the ECT has not expired.
5. Verify the "iat" claim is not unreasonably far in the past (**RECOMMENDED** maximum: 15 minutes) and is not unreasonably far in the future (**RECOMMENDED**: no more than 30 seconds ahead of the verifier's current time).
6. Perform DAG validation per Section 5.

If any verification step fails, the ECT **MUST** be rejected and the failure **MUST** be logged.

3.3.3. L1 Security Properties and Applicability

L1 provides the following security properties:

- Structural integrity via JSON schema validation
- Temporal ordering via "iat" and "exp" claims
- DAG integrity via parent reference validation
- Transport integrity via TLS

L1 does NOT provide:

- Non-repudiation (no signature binds the ECT to its issuer)
- Tamper detection after delivery (the recipient cannot prove the ECT was not modified after receipt)
- Issuer authentication at the ECT layer (identity depends entirely on the transport layer)

Note: At L1, without a deployed audit ledger, DAG parent existence validation (Section 5 step 2) is limited to parent ECTs received inline with the current request. Parents from prior requests that were not forwarded inline cannot be verified.

L1 **MUST NOT** be used across trust domain boundaries. Deployments using L1 **SHOULD** restrict it to internal environments where all agents are operated by the same organization and transport security is considered sufficient.

3.4. Level 2: JOSE Asymmetric Signing

At Level 2, an ECT is a JSON Web Token (JWT) [RFC7519] signed as a JSON Web Signature (JWS) [RFC7515]. L2 corresponds to the signing behavior defined in Version -00 of this specification.

ECTs **MUST** use JWS Compact Serialization (the base64url-encoded `header.payload.signature` format) so that they can be carried in a single HTTP header value.

The ECT **MUST** be signed with a private key that is verifiably bound to the agent's identity credential (see Section 3.8). The JOSE header "kid" parameter **MUST** reference the public key identifier from the agent's identity credential, and the "alg" parameter **MUST** match the algorithm associated with that credential.

3.4.1. L2 JOSE Header

The ECT JOSE header **MUST** contain the following parameters:

```
{
  "alg": "ES256",
  "typ": "exec+jwt",
  "kid": "agent-a-key-id-123"
}
```

Figure 2: ECT JOSE Header Example

alg: **REQUIRED**. The digital signature algorithm used to sign the ECT. **MUST** match the algorithm associated with the agent's identity credential. Implementations **MUST** support ES256 [RFC7518]. The "alg" value **MUST NOT** be "none". Symmetric algorithms (e.g., HS256, HS384, HS512) **MUST NOT** be used, as ECTs require asymmetric signatures for non-repudiation. To support algorithm agility, deployments **SHOULD** maintain an allowlist of accepted signing algorithms and **SHOULD** plan for migration to stronger algorithms as cryptographic requirements evolve. The algorithm is signaled in-band via the "alg" header parameter, enabling verifiers to support multiple algorithms during a transition period. Deployments **SHOULD** document their algorithm migration strategy and **SHOULD NOT** assume that ES256 will remain sufficient indefinitely.

typ: **REQUIRED**. **MUST** be set to "exec+jwt" to distinguish ECTs from other JWT types. WIMSE deployments **MAY** use "wimse-exec+jwt" for backward compatibility with Version -00 of this specification. Verifiers **MUST** accept both values.

kid: **REQUIRED**. The key identifier referencing the public key from the agent's identity credential [RFC7517]. Used by verifiers to look up the correct public key for signature verification. In WIMSE deployments, this references the WIT public key. In X.509 deployments, this **MAY** be the certificate thumbprint or subject key identifier.

3.4.2. L2 Transport

L2 ECTs are transported in the Execution-Context HTTP header field (Section 4) as JWS Compact Serialization. The header field value consists of three Base64url-encoded parts separated by period (".") characters.

L2 ECTs **MUST** be transmitted over TLS or mTLS connections.

3.4.3. L2 Verification

When an agent receives an L2 ECT, it **MUST** perform the following verification steps in order:

1. Parse the JWS Compact Serialization to extract the JOSE header, payload, and signature components per [RFC7515].
2. Verify that the "typ" header parameter is "exec+jwt" or "wimse-exec+jwt".
3. Verify that the "alg" header parameter appears in the verifier's configured allowlist of accepted signing algorithms. The allowlist **MUST NOT** include "none" or any symmetric algorithm (e.g., HS256, HS384, HS512). Implementations **MUST** include ES256 in the allowlist; additional asymmetric algorithms **MAY** be included per deployment policy.

4. Verify the "kid" header parameter references a known, valid public key from an identity credential within the trust domain (see Section 3.8).
5. Retrieve the public key identified by "kid" and verify the JWS signature per [RFC7515] Section 5.2.
6. Verify that the signing key identified by "kid" has not been revoked within the trust domain. Implementations **MUST** check the key's revocation status using the identity framework's key lifecycle mechanism (e.g., certificate revocation list, OCSP, SPIFFE trust bundle updates, or JWK set rotation).
7. Verify the "alg" header parameter matches the algorithm associated with the agent's identity credential.
8. Verify the "iss" claim is present and matches the agent identity asserted by the identity credential associated with the "kid" public key (see Section 3.8).
9. Verify the "aud" claim is present and contains the verifier's own identity. When "aud" is an array, it is sufficient that the verifier's identity appears as one element; the presence of other audience values does not cause verification failure. When the verifier is the audit ledger, the ledger's own identity **MUST** appear in "aud".
10. Verify the "exp" claim indicates the ECT has not expired.
11. Verify the "iat" claim is not unreasonably far in the past (implementation-specific threshold, **RECOMMENDED** maximum of 15 minutes) and is not unreasonably far in the future (**RECOMMENDED**: no more than 30 seconds ahead of the verifier's current time, to account for clock skew).
12. Verify all required claims ("jti", "exec_act", "pred") are present and well-formed.
13. Perform DAG validation per Section 5.
14. If all checks pass and an audit ledger is deployed, the ECT **SHOULD** be appended to the ledger.

If any verification step fails, the ECT **MUST** be rejected and the failure **MUST** be logged for audit purposes. Error messages **SHOULD NOT** reveal whether specific parent task IDs exist in the ECT store, to prevent information disclosure.

3.4.4. L2 Security Properties and Applicability

L2 provides the following security properties:

- Non-repudiation (the ECT is cryptographically bound to the issuing agent via asymmetric signature)
- Tamper detection (any modification invalidates the signature)
- Issuer authentication (the "kid" links to the agent's identity credential)
- Audience restriction (the "aud" claim limits valid verifiers)
- All L1 properties (structural, temporal, DAG, transport integrity)

L2 is suitable for cross-organization deployments, peer-to-peer agent communication, and any deployment requiring cryptographic non-repudiation. L2 is the **RECOMMENDED** default for production deployments.

3.5. Level 3: JOSE Signing with Audit Ledger

Level 3 extends Level 2 by requiring that every ECT be recorded in an audit ledger that provides cryptographic commitment. L3 transport and JWS signing are identical to L2; the additional requirements apply to ledger recording and verification.

3.5.1. L3 Transport

L3 ECTs use the same JWS Compact Serialization transport as L2 (see Section 3.4.2).

3.5.2. L3 Audit Ledger Requirements

At Level 3, the audit ledger **MUST** satisfy the baseline requirements defined in Section 6 and the following additional requirements:

1. Hash Chain: Each ledger entry **MUST** include a cryptographic hash of the previous entry, forming a hash chain. The hash algorithm **MUST** be SHA-256 or stronger.
2. Cryptographic Commitment: The ledger **MUST** provide a mechanism for verifiable cryptographic commitment over its entries, such as Merkle tree proofs, hash chains with signed roots, or equivalent constructions. The commitment **MUST** allow a verifier to confirm that a specific ECT is included in the ledger at a specific position without trusting the ledger operator.
3. Receipts: Upon successful append, the ledger **MUST** return a receipt to the submitting agent. The receipt **MUST** contain the ledger sequence number, the hash of the appended ECT entry, and the cryptographic commitment proof (e.g., Merkle inclusion proof) at the time of recording.
4. Availability: The ledger **MUST** be accessible for verification queries by authorized entities within the trust domain.

3.5.3. L3 Recording Semantics

An agent producing an L3 ECT **MUST** attempt to record the ECT in the audit ledger. Two recording modes are defined:

Synchronous Recording: The agent submits the ECT to the ledger and waits for a receipt before transmitting the ECT to the next agent. Synchronous recording provides the strongest guarantee: the ECT is committed to the ledger before any downstream processing occurs.

Asynchronous Recording: The agent submits the ECT to the ledger and transmits the ECT to the next agent without waiting for a receipt. The agent **MUST** subsequently verify that the ledger accepted the ECT and **MUST** retain the receipt. If the ledger rejects the ECT, the agent **MUST** alert the workflow coordinator or log a critical error. If asynchronous ledger recording fails, the producing agent **MUST** notify downstream agents that the ECT's L3 status is unconfirmed. Downstream agents **SHOULD** treat such an ECT as L2-verified (not L3-verified) until ledger confirmation is independently obtainable.

Deployments **SHOULD** use synchronous recording unless latency constraints make it impractical. The recording mode **SHOULD** be documented in the deployment's security policy.

3.5.4. L3 Verification

L3 verification consists of L2 verification (Section 3.4.3) followed by ledger verification:

Note: The "iss" and "aud" claims, which are **RECOMMENDED** at L1 and verified at L2 (steps 8-9 of Section 3.4.3), are **REQUIRED** at L3.

1. Perform all L2 verification steps (steps 1 through 14 of Section 3.4.3).
2. Verify that the ECT has been recorded in the audit ledger by querying the ledger for the ECT's "jti". If the ECT is found, verify that the ledger entry's receipt contains a valid sequence number, ECT hash, and cryptographic commitment proof. Note: the producing agent is responsible for recording the ECT per Section 3.5.3; the verifier checks that recording has occurred.
3. If the ECT is not yet present in the ledger (e.g., due to asynchronous recording), the verifier **MAY** retry after a short delay. If the ledger does not contain the ECT within the configured timeout, the verifier **MUST** either reject the ECT or downgrade to L2 verification per deployment policy.
4. If the ledger is unavailable, the verifier **SHOULD** retry with exponential backoff. If the ledger remains unavailable after a deployment-configured number of retries, the verifier **MUST** either reject the ECT or downgrade to L2 verification per deployment policy.

If any L3-specific verification step fails, the ECT **MUST** be rejected even if L2 verification succeeded.

3.5.5. L3 Security Properties and Applicability

L3 provides the following security properties beyond L2:

- Tamper-evident history (the hash chain detects any modification, insertion, or deletion of ledger entries)
- Cryptographic commitment (Merkle proofs or equivalent allow independent verification of ledger inclusion)
- Non-repudiation of recording (the receipt proves the ECT was committed at a specific time and position)
- All L2 properties

L3 is intended for regulated environments requiring auditable, tamper-evident execution records, such as healthcare (FDA 21 CFR Part 11), financial services (MiFID II), and environments subject to the EU AI Act.

3.6. Level Selection

Deployments **SHOULD** select the assurance level based on the following criteria:

- Use L1 when all agents operate within a single trust domain, non-repudiation is not required, and transport security (TLS) is considered sufficient. Typical environments: development, testing, internal microservice meshes.
- Use L2 when agents cross trust domain boundaries, when non-repudiation is required, or when agents communicate peer-to-peer across organizations. L2 is the **RECOMMENDED** default for production deployments.
- Use L3 when regulatory requirements mandate tamper-evident audit trails with cryptographic commitment, or when the deployment needs to demonstrate compliance with frameworks such as FDA 21 CFR Part 11, MiFID II, or the EU AI Act.

A deployment **MAY** use different assurance levels for different workflows within the same infrastructure. When agents at different levels interact, the higher level's verification requirements apply to the receiving agent. Specifically, an ECT at a higher assurance level **MAY** reference parent ECTs at a lower assurance level in its "pred" claim. In this case, the receiving agent applies its own level's verification to the current ECT and the parent's level verification to each parent ECT. For example, an L2 agent receiving an L1 parent ECT verifies the L1 parent per Section 3.3.2 and its own L2 ECT per Section 3.4.3. Whether cross-level parent references are permitted is a deployment policy decision; deployments **MAY** reject ECTs whose parents are below a minimum assurance level.

This specification does not define a level negotiation mechanism. Deployments configure the required assurance level out of band. A future extension **MAY** define in-band level signaling.

3.7. Backward Compatibility and Level Detection

A verifier determines the assurance level of a received ECT as follows:

1. If the raw Execution-Context header value (or body content) contains exactly two period (".") characters separating three non-empty segments, attempt to parse the value as JWS Compact Serialization. If the first segment base64url-decodes to a JSON object containing an "alg" field, the ECT is L2 or L3.
2. Otherwise, base64url-decode the header value (without padding) and attempt to parse the result as JSON. If successful, the ECT is L1.
3. If neither parse succeeds, the ECT **MUST** be rejected.

L2 and L3 use the same JWS format. Differentiation between L2 and L3 is a matter of deployment policy: L3 deployments require ledger recording (Section 3.5.2), while L2 deployments treat ledger recording as optional.

Implementations compliant with Version -00 of this specification are L2-compatible. No changes to existing -00 implementations are required for L2 interoperability.

3.8. Identity Binding

At L2 and L3, the ECT signing key **MUST** be verifiably bound to the agent's identity. This specification defines the abstract requirements for identity binding; concrete bindings are defined for specific identity frameworks.

An identity binding **MUST** satisfy the following requirements:

1. Key Association: The "kid" value in the ECT JOSE header **MUST** resolve to a public key through the identity framework's key discovery mechanism.
2. Issuer Correspondence: The "iss" claim in the ECT payload **MUST** correspond to the agent identity asserted by the identity credential containing the "kid" public key.
3. Algorithm Consistency: The "alg" value in the ECT JOSE header **MUST** match the algorithm associated with the identity credential.
4. Revocation Checking: The verifier **MUST** be able to determine whether the signing key has been revoked using the identity framework's revocation mechanism.

3.8.1. WIMSE Binding

When the deployment uses the WIMSE framework [I-D.ietf-wimse-arch]:

- The ECT "kid" references the public key from the agent's Workload Identity Token (WIT).
- The ECT "iss" **SHOULD** be the workload's SPIFFE ID [SPIFFE], matching the "sub" claim of the WIT.
- The ECT "alg" **MUST** match the algorithm in the WIT.
- Revocation is checked via the SPIFFE trust bundle or the trust domain's key lifecycle mechanism.
- ECTs are transported alongside the WIT and WPT ([I-D.ietf-wimse-s2s-protocol]) in HTTP requests.

ECT defines its own `Execution-Context` HTTP header field (Section 4) and does not rely on WIMSE HTTP message signature machinery. Deployments that additionally apply WIMSE HTTP message signatures [I-D.ietf-wimse-http-signature] to protect requests should note that as of draft-ietf-wimse-http-signature-03 the audience value is conveyed via the `wimse-aud` signature metadata parameter (per the HTTP Message Signatures framework [RFC9421]) rather than a dedicated HTTP header. This change does not affect the ECT payload's own `aud` claim or the `Execution-Context` header defined in this document.

3.8.2. X.509 Binding

When the deployment uses X.509 certificates:

- The ECT "kid" **MAY** be the certificate thumbprint (x5t) or subject key identifier.
- The ECT "iss" **SHOULD** be a URI derived from the certificate subject or subject alternative name.
- Revocation is checked via CRL or OCSP.

3.8.3. JWK Set Binding

When the deployment uses plain JWK sets [RFC7517] with out-of-band trust establishment:

- The ECT "kid" **MUST** match the "kid" parameter in a JWK from the trusted JWK set.
- The ECT "iss" **MUST** match a value associated with the JWK through the deployment's configuration.
- Revocation is handled by removing the key from the published JWK set.

4. HTTP Header Transport

4.1. Execution-Context Header Field

This specification defines the Execution-Context HTTP header field [RFC9110] for transporting ECTs between agents.

The format of the header field value depends on the assurance level:

- At Level 1, the header field value is the base64url-encoded JSON payload (without padding).
- At Level 2 and Level 3, the header field value is the ECT in JWS Compact Serialization format [RFC7515]. The value consists of three Base64url-encoded parts separated by period (".") characters.

An agent sending a request to another agent includes the Execution-Context header alongside any identity headers required by the deployment's identity framework. In WIMSE deployments, agents include the Workload-Identity header and, when available, the Workload Proof Token (WPT) per [I-D.ietf-wimse-s2s-protocol].

```
GET /api/safety-check HTTP/1.1
Host: safety-agent.example.com
Authorization: Bearer <identity-token>
Execution-Context: eyJhbGci...ECT...
```

Figure 3: HTTP Request with ECT Header

When multiple parent tasks contribute context to a single request, multiple Execution-Context header field lines **MAY** be included, each carrying a separate ECT. This applies to all assurance levels.

When a receiver processes multiple Execution-Context headers, it **MUST** individually verify each ECT per the applicable verification procedure (Section 3.3.2, Section 3.4.3, or Section 3.5.4). If any single ECT fails verification, the receiver **MUST** reject the entire request. The set of verified parent task IDs across all received ECTs represents the complete set of parent dependencies available for the receiving agent's subsequent ECT.

4.2. HTTP Error Handling

When ECT verification fails during HTTP request processing, the receiving agent **SHOULD** respond with HTTP 403 (Forbidden). This applies regardless of whether the failure is due to an invalid ECT payload, a signature verification failure, or a missing ECT when one is required by deployment policy. HTTP 401 (Unauthorized) **SHOULD NOT** be used for ECT failures, as 401 conventionally indicates that authentication credentials are missing or invalid and requires a WWW-Authenticate header per [RFC9110]. The response body **SHOULD** include a generic error indicator without revealing which specific verification step failed. The receiving agent **MUST NOT** process the requested action when ECT verification fails.

5. DAG Validation

ECTs form a Directed Acyclic Graph (DAG) where each task references its parent tasks via the "pred" claim. DAG validation is performed against the ECT store — either an audit ledger or the set of parent ECTs received inline.

DAG validation applies regardless of assurance level.

When receiving and verifying an ECT, implementations **MUST** perform the following DAG validation steps:

1. Task ID Uniqueness: The "jti" claim **MUST** be unique within the applicable scope (the workflow identified by "wid", or the entire ECT store if "wid" is absent). If an ECT with the same "jti" already exists, the ECT **MUST** be rejected.
2. Parent Existence: Every task identifier listed in the "pred" array **MUST** correspond to a task that is available in the ECT store (either previously recorded in the ledger or received inline as a verified parent ECT). If any parent task is not found, the ECT **MUST** be rejected.
3. Temporal Ordering: The "iat" value of every parent task **MUST NOT** be greater than the "iat" value of the current task plus a configurable clock skew tolerance (**RECOMMENDED**: 30 seconds). That is, for each parent: `parent.iat < child.iat + clock_skew_tolerance`. The tolerance accounts for clock skew between agents; it does not guarantee strict causal ordering from timestamps alone. Causal ordering is primarily enforced by the DAG structure (parent existence in the ECT store), not by timestamps. If any parent task violates this constraint, the ECT **MUST** be rejected.
4. Acyclicity: Following the chain of parent references **MUST NOT** lead back to the current ECT's "jti". If a cycle is detected, the ECT **MUST** be rejected. Note: because the Parent Existence check (step 2) requires that all parents already exist in the ECT store, and an ECT cannot reference itself or a future ECT, cycles are prevented by construction. This explicit check serves as defense in depth against implementation errors or store corruption.
5. Trust Domain Consistency: Parent tasks **SHOULD** belong to the same trust domain or to a trust domain with which a federation relationship has been established.

6. Workflow Consistency: When "wid" is present, verifiers **SHOULD** validate that the "wid" of each parent ECT matches the "wid" of the current ECT. Cross-workflow parent references (where a parent's "wid" differs from the child's "wid") **MUST** be rejected unless explicitly permitted by deployment policy. See also Section 7.9 for the security rationale.

To prevent denial-of-service via extremely deep or wide DAGs, implementations **SHOULD** enforce a maximum ancestor traversal limit (**RECOMMENDED**: 10000 nodes). If the limit is reached before cycle detection completes, the ECT **SHOULD** be rejected.

In distributed deployments, a parent ECT may not yet be available locally due to replication lag. Implementations **MAY** defer validation to allow parent ECTs to arrive, but **MUST NOT** treat the ECT as verified until all parent references are resolved.

6. Audit Ledger Interface

ECTs **MAY** be recorded in an immutable audit ledger for compliance verification and post-hoc analysis. A ledger is **OPTIONAL** for L1 and L2 deployments but is **REQUIRED** for L3 deployments. This specification does not mandate a specific storage technology. Implementations **MAY** use append-only logs, databases with cryptographic commitment schemes, distributed ledgers, or any storage mechanism that provides the required properties.

6.1. Baseline Ledger Requirements

When an audit ledger is deployed, the implementation **MUST** provide:

1. Append-only semantics: Once an ECT is recorded, it **MUST NOT** be modified or deleted.
2. Ordering: The ledger **MUST** maintain a total ordering of ECT entries via a monotonically increasing sequence number.
3. Lookup by ECT ID: The ledger **MUST** support efficient retrieval of ECT entries by "jti" value.
4. Integrity verification: The ledger **SHOULD** provide a mechanism to verify that no entries have been tampered with (e.g., hash chains or Merkle trees).

The ledger **SHOULD** be maintained by an entity independent of the workflow agents to reduce the risk of collusion.

6.2. L3 Ledger Requirements

At Level 3, the audit ledger **MUST** satisfy the baseline requirements above and the additional requirements defined in Section 3.5.2. In particular:

- Each entry **MUST** include a cryptographic hash of the previous entry, forming a verifiable hash chain.
- The ledger **MUST** provide cryptographic commitment proofs (Merkle tree proofs, signed hash chain roots, or equivalent).
- The ledger **MUST** return a receipt upon successful append, containing the sequence number, entry hash, and commitment proof.

6.3. Ledger Independence

The audit ledger **SHOULD** be operated by an entity that is independent of the workflow agents. When the ledger operator and the workflow agents are controlled by the same organization, additional safeguards **SHOULD** be implemented, such as separation of duties between ledger administrators and agent operators, or independent ledger replicas maintained by third parties.

6.4. L3 Ledger Verification

Auditors verifying L3 ledger integrity **SHOULD** perform the following checks:

1. Verify the hash chain: for each entry, recompute the hash of the previous entry and confirm it matches the recorded value.
2. Verify the cryptographic commitment: validate Merkle inclusion proofs (or equivalent) for a sample or all entries.
3. Verify ECT signatures: for each ECT in the ledger, perform the L2 verification procedure (Section 3.4.3) using the public keys valid at the time of recording.
4. Verify DAG consistency: confirm that all parent references in all ECTs resolve to valid entries in the ledger.

7. Security Considerations

7.1. Threat Model

The threat model considers: (1) a malicious agent that creates false ECT claims, (2) an agent whose private key has been compromised, (3) a ledger tamperer attempting to modify recorded entries, and (4) a time manipulator altering timestamps to affect perceived ordering.

7.2. Level-Specific Security Properties

7.2.1. Level 1

L1 provides no cryptographic binding between the ECT and its issuer. A compromised or malicious intermediary with access to the transport channel can modify L1 ECTs without detection after delivery. L1 does not provide non-repudiation: the issuer can deny having produced an ECT, and the receiver cannot prove otherwise.

L1 **MUST NOT** be used across trust domain boundaries. L1 is appropriate only when all agents are operated by the same organization, the transport channel is fully trusted (e.g., service mesh with mTLS), and the deployment does not require non-repudiation or tamper evidence beyond transport security.

7.2.2. Level 2

L2 inherits all security properties of the JWS-based ECT mechanism defined in this document. The existing security analysis (signature verification, replay prevention, key compromise, collusion) applies directly to L2.

7.2.3. Level 3

L3 provides all L2 security properties plus tamper-evident history via the audit ledger's hash chain and cryptographic commitment. Modifications to ledger entries are detectable through hash chain verification. Deletions or insertions are detectable through Merkle proof validation.

L3 does not prevent a compromised ledger operator from refusing to record entries (censorship), but the submitting agent's receipt mechanism (Section 3.5.2) provides evidence of submission. Deployments concerned about ledger censorship **SHOULD** use multiple independent ledger replicas.

7.3. Assurance Level Downgrade Attacks

The assurance level of an ECT is determined by its format: unsigned JSON indicates L1, while a JWS compact serialization indicates L2 or L3. This format-based detection determines what was sent, not what was expected by the verifier.

A man-in-the-middle or compromised proxy could strip the JWS signature from an L2 or L3 ECT and re-encode the payload as an unsigned L1 JSON object. Because the resulting ECT is syntactically valid at L1, a verifier that accepts any assurance level would process it without detecting the downgrade.

Verifiers **MUST** be configured with a minimum acceptable assurance level and **MUST** reject ECTs whose detected level falls below that minimum. Format-based level detection alone is insufficient without a policy-enforced minimum level requirement.

7.4. Self-Assertion Limitation

ECTs are self-asserted by the executing agent. The agent claims what it did, and this claim is signed with its private key (at L2 and L3). A compromised or malicious agent could create ECTs with false claims (e.g., claiming an action was performed when it was not).

ECTs do not independently verify that:

- The claimed execution actually occurred as described
- The input/output hashes correspond to the actual data processed
- The agent faithfully performed the stated action

The trustworthiness of ECT claims depends on the trustworthiness of the signing agent and the integrity of the broader deployment environment. ECTs provide a technical mechanism for execution recording; they do not by themselves satisfy any specific regulatory compliance requirement.

At Level 1, the self-assertion limitation is compounded by the absence of any cryptographic binding; any entity with access to the transport channel can create ECTs claiming any identity and any action.

7.5. Signature Verification

For L2 and L3 deployments, ECTs **MUST** be signed with the agent's private key using JWS [RFC7515]. The signature algorithm **MUST** match the algorithm associated with the agent's identity credential (see Section 3.8). Receivers **MUST** verify the ECT signature against the corresponding public key before processing any claims. Receivers **MUST** verify that the signing key has not been revoked within the trust domain (see step 6 in Section 3.4.3).

If signature verification fails or if the signing key has been revoked, the ECT **MUST** be rejected entirely and the failure **MUST** be logged.

Implementations **MUST** use established JWS libraries and **MUST NOT** implement custom signature verification. Implementations **SHOULD** follow the JWT security best practices defined in [RFC8725].

The prohibition of "alg": "none" (see Section 3.4.3) also serves as defense against level-confusion attacks: an L1 payload wrapped in a JWS with alg=none would be detected as L2 by format and rejected at the algorithm allowlist check, preventing an attacker from bypassing L2 security requirements.

7.6. Replay Attack Prevention

ECTs include short expiration times (**RECOMMENDED**: 5-15 minutes) and audience restriction via "aud" to limit replay attacks. Implementations **MUST** maintain a cache of recently-seen "jti" values and **MUST** reject ECTs with duplicate "jti" values. For L2 and L3, each ECT is cryptographically bound to the issuing agent via "kid"; verifiers **MUST** confirm that "kid" resolves to the "iss" agent's key (step 8 in Section 3.4.3).

7.7. Man-in-the-Middle Protection

ECTs **MUST** be transmitted over TLS or mTLS connections. When used with WIMSE [I-D.ietf-wimse-s2s-protocol] or similar service mesh protocols, transport security is already established.

7.8. Key Compromise

If an agent's private key is compromised, an attacker can forge ECTs that appear to originate from that agent. Mitigations:

- Implementations **SHOULD** use short-lived keys and rotate them frequently.
- Private keys **SHOULD** be stored in hardware security modules or equivalent secure key storage.
- Trust domains **MUST** support rapid key revocation.

ECTs recorded before key revocation remain valid historical records but **SHOULD** be flagged for audit purposes. New ECTs **MUST NOT** reference a parent ECT whose signing key is known to be revoked at creation time.

7.9. Collusion and DAG Integrity

A single malicious agent cannot forge parent task references because DAG validation requires parent tasks to exist in the ECT store. However, multiple colluding agents could create a false execution history. Additionally, a malicious agent may omit actual parent dependencies from "pred" to hide influences on its output; because ECTs are self-asserted (Section 7.4), no mechanism can force complete dependency declaration.

Mitigations include:

- The ledger **SHOULD** be maintained by an entity independent of the workflow agents.
- Multiple independent ledger replicas can be compared for consistency.
- External auditors can compare the declared DAG against expected workflow patterns.

Verifiers **SHOULD** validate that the declared "wid" of parent ECTs matches the "wid" of the child ECT, rejecting cross-workflow parent references unless explicitly permitted by deployment policy.

7.10. Privilege Escalation via ECTs

ECTs record execution history; they do not convey authorization. Verifiers **MUST NOT** interpret the presence of an ECT, or a particular set of parent references in "pred", as an authorization grant. Authorization decisions **MUST** remain with the deployment's identity and authorization layer.

7.11. Denial of Service

Implementations **SHOULD** apply rate limiting to prevent excessive ECT submissions. For L2 and L3, DAG validation **SHOULD** be performed after signature verification to avoid wasting resources on unsigned or incorrectly signed tokens. L1 ECTs are cheaper to validate (no signature verification) but are correspondingly easier for an attacker to generate; L1 deployments **SHOULD** apply stricter rate limits to compensate.

7.12. Timestamp Accuracy

Implementations **SHOULD** use synchronized time sources (e.g., NTP) and **SHOULD** allow a configurable clock skew tolerance (**RECOMMENDED**: 30 seconds). Cross-organizational deployments **MAY** require a higher tolerance and **SHOULD** document the configured value.

7.13. ECT Size Constraints

Implementations **SHOULD** limit the "pred" array to a maximum of 256 entries. See Section 3.1.4 for "ect_ext" size limits.

When ECTs are transported via HTTP headers, the total encoded size of the Execution-Context header value is subject to practical limits imposed by HTTP servers and intermediaries. Many implementations enforce header size limits of 8 KB or 16 KB. Implementations **SHOULD** ensure that the total size of an ECT (including JWS overhead for L2/L3) does not exceed 8 KB when transported via HTTP header. ECTs that exceed HTTP header size limits **SHOULD** be transported in the HTTP request body instead (see Section 3.3.1 and Section 3.4.2). Deployments **SHOULD** monitor ECT sizes and alert when ECTs approach transport limits.

7.14. Identity Binding Security

7.14.1. JWK Set Binding

When identity is bound via JWK Set URI (see Section 3.8), there is a time-of-check/time-of-use gap between JWK set refreshes. A key that has been removed from the JWK set may still be accepted by a verifier whose cached copy has not yet been refreshed. Implementations **SHOULD** refresh JWK sets at configurable intervals (**RECOMMENDED**: no longer than 5 minutes).

7.14.2. X.509 Binding

When identity is bound via X.509 certificates, revocation checking depends on OCSP responder or CRL distribution point availability. If the revocation source is unreachable, the verifier needs to decide whether to accept or reject the ECT. Implementations **SHOULD** hard-fail for L3 (reject the ECT if revocation status cannot be determined), as L3 workflows require the strongest assurance. Implementations **MAY** soft-fail for L2 with logging, accepting the ECT but recording the revocation check failure for subsequent audit review.

7.14.3. WIMSE Binding

When identity is bound via WIMSE trust bundles, the same time-of-check/time-of-use concern applies to trust bundle refreshes. Implementations **SHOULD** refresh trust bundles frequently to minimize the window during which a revoked or rotated identity remains accepted.

7.15. Audit Ledger Threats

7.15.1. Availability

If the audit ledger is unavailable in synchronous recording mode (see Section 3.5.2), all L3 workflows halt because agents cannot obtain ledger receipts. Deployments **SHOULD** implement ledger redundancy (e.g., multiple ledger replicas behind a load balancer) to prevent the ledger from becoming a single point of failure.

7.15.2. Split-View Attacks

A compromised ledger could present different views to different verifiers (equivocation), causing inconsistent audit state across the deployment. Deployments **SHOULD** use multiple independent ledger replicas and **SHOULD** periodically compare their state to detect divergence.

7.15.3. Receipt Authenticity

If the ledger's signing key is compromised, an attacker can generate fake receipts for entries that were never recorded. Ledger signing keys **SHOULD** be stored in hardware security modules (HSMs) and **SHOULD** be rotated regularly.

7.15.4. Asynchronous Recording Gap

In asynchronous recording mode (see Section 3.5.2), downstream agents act on ECTs before ledger confirmation is received. During this gap, an ECT that will ultimately fail ledger recording may already have influenced downstream workflow steps. Deployments using asynchronous recording **SHOULD** implement reconciliation procedures to detect and handle ECTs that fail ledger confirmation after downstream processing has begun.

8. Privacy Considerations

8.1. Data Exposure in ECTs

ECTs necessarily reveal:

- Agent identities ("iss", "aud") for accountability purposes
- Action descriptions ("exec_act") for audit trail completeness
- Timestamps ("iat", "exp") for temporal ordering

ECTs are designed to NOT reveal:

- Actual input or output data values (replaced with cryptographic hashes via "inp_hash" and "out_hash")
- Internal computation details or intermediate steps
- Proprietary algorithms or intellectual property
- Personally identifiable information (PII)

Privacy exposure is equivalent across all assurance levels: the ECT payload contains the same claims regardless of whether the ECT is unsigned (L1), signed (L2), or signed with ledger recording (L3). L3 ledger recording increases the durability of the privacy-relevant data but does not increase its scope.

8.2. Data Minimization

Implementations **SHOULD** minimize the information included in ECTs. The "exec_act" claim **SHOULD** use structured identifiers (e.g., "process_payment") rather than natural language descriptions. Extension keys in "ect_ext" (Section 3.1.4) deserve particular attention: human-readable values risk exposing sensitive operational details. See Section 3.1.4 for guidance on using structured identifiers.

8.3. Storage and Access Control

ECTs stored in audit ledgers **SHOULD** be access-controlled so that only authorized auditors can read them. Implementations **SHOULD** consider encryption at rest for ledger storage. ECTs provide structural records of execution ordering; they are not intended for public disclosure.

Full input and output data (corresponding to the hashes in ECTs) **SHOULD** be stored separately from the ledger with additional access controls, since auditors may need to verify hash correctness but general access to the data values is not needed.

8.4. Workflow Topology Leakage

The DAG structure of ECTs reveals workflow topology: which agents interact, fan-out and fan-in patterns, sequential versus parallel execution, and organizational structure. At L3, this topology is permanently recorded in the audit ledger. Deployments **SHOULD** consider whether workflow topology constitutes sensitive information and apply appropriate access controls to ECT stores and ledgers.

8.5. Cross-Workflow Correlation

Stable agent identifiers in the "iss" claim enable cross-workflow activity correlation: an observer with access to ECTs from multiple workflows can track which agents participate in which workflows and how frequently. Deployments with privacy requirements **MAY** use per-workflow or rotating agent identifiers where feasible to limit cross-workflow correlation.

9. IANA Considerations

9.1. Media Type Registrations

This document requests registration of the following media types in the "Media Types" registry maintained by IANA:

Note: The media type "application/exec+jwt" uses the "+jwt" structured syntax suffix. While "+jwt" is widely used in practice, it is not yet a formally registered structured syntax suffix per [RFC6838]. Registration of the "+jwt" suffix is the subject of ongoing work in the IETF.

9.1.1. application/exec+jwt

Type name: application

Subtype name: exec+jwt

Required parameters: none

Optional parameters: none

Encoding considerations: 8bit; at Level 2 and Level 3, an ECT is a JWT that is a JWS using the Compact Serialization, which is a sequence of Base64url-encoded values separated by period characters. At Level 1, this media type is not used; L1 ECTs use application/json.

Security considerations: See the Security Considerations section of this document.

Interoperability considerations: none

Published specification: This document

Applications that use this media type: Applications that implement agentic workflows requiring execution context tracing and audit trails.

Additional information: Magic number(s): none File extension(s): none Macintosh file type code(s): none

Person and email address to contact for further information: Christian Nennemann, ietf@nennemann.de

Intended usage: COMMON

Restrictions on usage: none

Author: Christian Nennemann

Change controller: IETF

9.1.2. application/wimse-exec+jwt

This document also registers "application/wimse-exec+jwt" as an alias for backward compatibility with Version -00 of this specification. WIMSE deployments **MAY** use either media type; new deployments **SHOULD** prefer "application/exec+jwt".

Type name: application

Subtype name: wimse-exec+jwt

Required parameters: none

Optional parameters: none

Encoding considerations: 8bit; at Level 2 and Level 3, an ECT is a JWT that is a JWS using the Compact Serialization, which is a sequence of Base64url-encoded values separated by period characters. At Level 1, this media type is not used; L1 ECTs use application/json.

Security considerations: See the Security Considerations section of this document.

Interoperability considerations: none

Published specification: This document

Applications that use this media type: Applications that implement agentic workflows requiring execution context tracing and audit trails.

Additional information: Magic number(s): none File extension(s): none Macintosh file type code(s): none

Person and email address to contact for further information: Christian Nennemann, ietf@nennemann.de

Intended usage: COMMON

Restrictions on usage: none

Author: Christian Nennemann

Change controller: IETF

9.2. HTTP Header Field Registration

This document requests registration of the following header field in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" maintained by IANA:

Field name: Execution-Context

Status: permanent

Specification document: This document, Section 4

9.3. JWT Claims Registration

This document requests registration of the following claims in the "JSON Web Token Claims" registry maintained by IANA:

Claim Name	Claim Description	Change Controller	Reference
wid	Workflow Identifier	IETF	Section 3.1.2
exec_act	Action/Task Type	IETF	Section 3.1.2
pred	Predecessor Task Identifiers	IETF	Section 3.1.2
inp_hash	Input Data Hash	IETF	Section 3.1.3
out_hash	Output Data Hash	IETF	Section 3.1.3
ect_ext	Extension Object	IETF	Section 3.1.4

Table 1: JWT Claims Registrations

10. References

10.1. Normative References

[I-D.nennemann-act] Nennemann, C., "Agent Context Token (ACT)", Work in Progress, Internet-Draft, draft-nennemann-act-01, 2026, <https://datatracker.ietf.org/doc/draft-nennemann-act/>.

[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>.

[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>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/rfc/rfc7518>.

[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>.

[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>.

[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>.

[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>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/rfc/rfc9562>.

10.2. Informative References

[AgentRFC] Shenghan, Z. and Z. Qifan, "AgentRFC: Security Design Principles and Conformance Testing for Agent Protocols", 2026, <https://arxiv.org/abs/2603.23801>.

[I-D.draft-emirdag-scitt-ai-agent-execution] Emirdag, "SCITT Profile for AI Agent Execution", 2026, <https://datatracker.ietf.org/doc/draft-emirdag-scitt-ai-agent-execution/>.

[I-D.draft-king-dawn-requirements] King and Farrel, "Requirements for Discovery of AI Agents and Workloads Across Network Boundaries", 2026, <https://datatracker.ietf.org/doc/draft-king-dawn-requirements/>.

[I-D.ietf-oauth-transaction-tokens] Tulshibagwale, A., Fletcher, G., and P. Kasselman, "Transaction Tokens", Work in Progress, Internet-Draft, draft-ietf-oauth-transaction-tokens-08, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-transaction-tokens-08>.

[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-wimse-arch] Salowey, J. A., Rosomakho, Y., and H. Tschofenig, "Workload Identity in a Multi System Environment (WIMSE) Architecture", Work in Progress, Internet-Draft, draft-ietf-wimse-arch-07, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-wimse-arch-07>.

[I-D.ietf-wimse-http-signature] "HTTP Message Signatures for Workloads", Work in Progress, Internet-Draft, draft-ietf-wimse-http-signature-03, 7 April 2026, <https://datatracker.ietf.org/doc/draft-ietf-wimse-http-signature-03/>.

[I-D.ietf-wimse-s2s-protocol] Campbell, B., Salowey, J. A., Schwenkschuster, A., and Y. Sheffer, "WIMSE Workload-to-Workload Authentication", Work in Progress, Internet-Draft, draft-ietf-wimse-s2s-protocol-07, 16 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-wimse-s2s-protocol-07>.

[I-D.oauth-transaction-tokens-for-agents] Bertocci, V., "Transaction Tokens for Agentic AI Systems", Work in Progress, Internet-Draft, draft-oauth-transaction-tokens-for-agents-00, 2025, <https://datatracker.ietf.org/doc/draft-oauth-transaction-tokens-for-agents-00/>.

[MIGT] Andrew, K. and K. Klaudia, "Who Governs the Machine? A Machine Identity Governance Taxonomy", 2026, <https://arxiv.org/abs/2604.06148>.

[NIST-NCCoE-AI-Agents] NIST, "Accelerating the Adoption of Software and AI Agent Identity and Authorization", 2026, <https://www.nccoe.nist.gov/projects/ai-agent-identity-authorization>.

[OPENTELEMETRY] Cloud Native Computing Foundation, "OpenTelemetry Specification", <https://opentelemetry.io/docs/specs/otel/>.

[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>.

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <https://www.rfc-editor.org/rfc/rfc8693>.

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/rfc/rfc8725>.

[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>.

[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>.

[SPIFFE] "SPIFFE ID", <https://spiffe.io/docs/latest/spiffe-about/spiffe-concepts/>.

Use Cases

This section describes representative use cases demonstrating how ECTs provide structured execution records at different assurance levels.

Note: task identifiers in this section are abbreviated for readability. In production, all "jti" values are required to be UUIDs per Section 3.1.2.

Cross-Organization Financial Trading (L3)

In a cross-organization trading workflow, an investment bank's agents coordinate with an external credit rating agency. The agents operate in separate trust domains with a federation relationship. The DAG records that independent assessments from both organizations were completed before trade execution.

This workflow uses Level 3 (JOSE signing with audit ledger) because regulatory requirements (e.g., MiFID II) mandate tamper-evident, auditable execution records.

```
Trust Domain: bank.example
  Agent A1 (Portfolio Risk):
    jti: task-001    pred:[]
    iss: spiffe://bank.example/agent/risk
    exec_act: analyze_portfolio_risk

Trust Domain: ratings.example (external)
  Agent B1 (Credit Rating):
    jti: task-002    pred:[]
    iss: spiffe://ratings.example/agent/credit
    exec_act: assess_credit_rating

Trust Domain: bank.example
  Agent A2 (Compliance):
    jti: task-003    pred:[task-001, task-002]
    iss: spiffe://bank.example/agent/compliance
    exec_act: verify_trade_compliance

  Agent A3 (Execution):
    jti: task-004    pred:[task-003]
    iss: spiffe://bank.example/agent/execution
    exec_act: execute_trade
```

Figure 4: Cross-Organization Trading Workflow (L3)

The resulting DAG:

```
task-001 (analyze_portfolio_risk)   task-002 (assess_credit_rating)
  [bank.example]                      [ratings.example]
          \                              /
           v                            v
         task-003 (verify_trade_compliance)
           [bank.example]
                    |
                    v
         task-004 (execute_trade)
           [bank.example]
```

Figure 5: Cross-Organization DAG

Task 003 has two parents from different trust domains, demonstrating cross-organizational fan-in. The compliance agent verifies both parent ECTs — one signed by a local key and one by a federated key from the rating agency's trust domain.

Multi-Vendor SaaS Integration (L2)

In a document processing pipeline, a customer's orchestration agent coordinates with third-party vendor agents across organizational boundaries. The customer uploads documents that pass through an OCR vendor for text extraction, then fan out to a translation vendor for multi-language output, before results converge back at the customer's storage agent.

This workflow uses Level 2 (JOSE signing without audit ledger) because the cross-organization boundary requires non-repudiation — each vendor must prove it performed its step — but no regulatory audit ledger is mandated.

```
Trust Domain: customer.example
  Agent C1 (Orchestrator):
    jti: task-201    pred:[]
    iss: spiffe://customer.example/agent/orchestrator
    exec_act: initiate_document_pipeline

Trust Domain: ocr-vendor.example (external)
  Agent V1 (OCR Extractor):
    jti: task-202    pred:[task-201]
    iss: spiffe://ocr-vendor.example/agent/ocr
    exec_act: extract_text

Trust Domain: translate-vendor.example (external)
  Agent V2a (Translator EN→DE):
    jti: task-203    pred:[task-202]
    iss: spiffe://translate-vendor.example/agent/translate
    exec_act: translate_de

  Agent V2b (Translator EN→FR):
    jti: task-204    pred:[task-202]
    iss: spiffe://translate-vendor.example/agent/translate
    exec_act: translate_fr

Trust Domain: customer.example
  Agent C2 (Storage):
    jti: task-205    pred:[task-203, task-204]
    iss: spiffe://customer.example/agent/storage
    exec_act: store_results
```

Figure 6: Multi-Vendor SaaS Document Pipeline (L2)

The resulting DAG:

```
task-201 (initiate_document_pipeline)
  [customer.example]
           |
           v
task-202 (extract_text)
  [ocr-vendor.example]
        /            \
       v              v
task-203             task-204
(translate_de)       (translate_fr)
[translate-vendor]   [translate-vendor]
        \              /
         v            v
task-205 (store_results)
  [customer.example]
```

Figure 7: Multi-Vendor SaaS DAG

Task 202 fans out to two parallel translation tasks (203 and 204) at the translation vendor, demonstrating cross-vendor fan-out. Task 205 performs fan-in, requiring both translations to complete before storing the combined results. Each vendor's ECT is signed with that vendor's private key, providing cross-organizational non-repudiation without requiring an external audit ledger.

Internal Microservice Mesh (L1)

In an internal AI platform, multiple microservices coordinate to process requests. All agents operate within a single trust domain behind a service mesh with mTLS. Non-repudiation is not required; the ECTs are used for observability and debugging.

This workflow uses Level 1 (unsigned JSON) because all agents are internal, the transport is fully trusted, and the overhead of cryptographic signing is not justified.

```
Trust Domain: internal.example
  Agent S1 (Preprocessor):
    jti: task-101    pred:[]
    exec_act: preprocess_input

  Agent S2 (Model Inference):
    jti: task-102    pred:[task-101]
    exec_act: run_inference

  Agent S3 (Postprocessor):
    jti: task-103    pred:[task-102]
    exec_act: format_output
```

Figure 8: Internal Microservice Workflow (L1)

Related Work

WIMSE Workload Identity

The WIMSE architecture [I-D.ietf-wimse-arch] and service-to- service protocol [I-D.ietf-wimse-s2s-protocol] provide one identity foundation with which ECTs can operate. WIT/WPT answer "who is this agent?" and "does it have authority?" while ECTs record "what did this agent do?" Together they form an identity-plus-accountability framework for regulated agentic systems. ECTs define an explicit WIMSE identity binding (see Section 3.8.1) but are not limited to WIMSE deployments.

Section 3.3.9 of the WIMSE architecture [I-D.ietf-wimse-arch] explicitly names "AI and ML-Based Intermediaries as autonomous agents propagating security context downstream" as an in-scope architectural case but does not itself specify a format for that propagated execution context. ECTs provide the standardized execution-context format that this architectural section requires: a JWT-based per-task record that an AI/ML intermediary can produce, sign, and propagate downstream to preserve accountability across the agent chain. In this sense, ECTs directly realize a requirement surfaced by the WIMSE charter itself, and the Execution-Context HTTP header defined in Section 4 is the concrete on-the-wire encoding for the §3.3.9 propagation model.

ECTs are also designed to compose with the HTTP message signing profile defined in [I-D.ietf-wimse-http-signature]: an Execution-Context header carrying an L2 or L3 ECT can be covered by a WIMSE HTTP message signature over the same request, so that integrity protection of the ECT and of its transport binding are aligned under a single signing model.

Composition Safety for Agent Protocols

Recent analysis of agent protocol security ([AgentRFC]) establishes that security properties which hold for individual agent protocols can break when those protocols are composed through shared infrastructure, because assumptions made by one protocol are not necessarily preserved by adjacent hops. This provides theoretical motivation for tracking execution context at each hop in an agent chain rather than relying solely on end-to-end authorization tokens, since the boundary where composition fails is generally not observable from any single endpoint. ECTs record execution context per-task with a cryptographic binding to the issuing agent, so that composition-induced failures become detectable during post-hoc audit even when they were not prevented in-band.

Machine Identity Governance (MIGT)

The Machine Identity Governance Taxonomy [MIGT] catalogues risk categories for enterprise machine identities and documents that AI agents and automated workflows now outnumber human identities in enterprise environments by ratios exceeding 80 to 1. The taxonomy identifies record-keeping, traceability, and non-repudiation of automated actions as primary risk categories under regulatory regimes such as EU AI Act Article 12 on record-keeping, which ECT

execution records are specifically designed to address. ECTs provide the per-task signed artifact that such governance frameworks require as evidence that a given automated action was performed by a specific agent identity at a specific time.

NIST/NCCoE AI Agent Identity

The NIST/NCCoE concept paper on AI agent identity and authorization [NIST-NCCoE-AI-Agents] is the first US government standards-body document to treat AI agent identity as an enterprise identity management concern, explicitly building on OAuth, OIDC, and SCIM rather than proposing a parallel stack. This validates ECT's standards-first approach of layering accountability on existing IETF credentials and JOSE signing primitives, and ECTs are positioned to serve as a referenced execution-record format for the NCCoE demonstration project alongside the identity and authorization primitives it enumerates.

SCITT AI Agent Execution Profile

The SCITT profile for AI agent execution [I-D.draft-emirdag-scitt-ai-agent-execution] defines an AgentInteractionRecord (AIR) with COSE_Sign1 payloads intended for anchoring into SCITT Transparency Services. ECTs and AIR are complementary along the in-transit vs. at-rest dimension: ECTs carry execution context in transit, embedded in a JWT and propagated through the Execution-Context HTTP header defined in Section 4, while AIR anchors records into a SCITT transparency service for long-term tamper-evident storage. Higher-assurance ECT deployments operating at Level 3 (Section 3.5) **MAY** use AIR as the SCITT payload format when the configured audit ledger is a SCITT Transparency Service, with the ECT's signed payload converted into the COSE_Sign1 envelope expected by AIR.

DAWN: Discovery of Agents and Workloads

The proposed DAWN working group and its requirements draft [I-D.draft-king-dawn-requirements] define requirements for discovering AI agents, workloads, and named entities across organizational boundaries. ECTs are identity-framework agnostic by design (Section 3.8) and therefore compose cleanly with any discovery mechanism DAWN may produce, regardless of the underlying credential type (WIMSE WIT/WPT, X.509, OAuth, or JWK sets). If DAWN charters, the workload and agent bindings recorded in an ECT are directly usable as discoverable execution-context metadata for agents located through DAWN discovery, without requiring changes to the ECT format itself.

OAuth 2.0 Token Exchange and the "act" Claim

[RFC8693] defines the OAuth 2.0 Token Exchange protocol and registers the "act" (Actor) claim in the JWT Claims registry. The "act" claim creates nested JSON objects representing a delegation chain: "who is acting on behalf of whom." While the nesting superficially resembles a chain, it is strictly linear (each "act" object contains at most one nested "act"), represents authorization delegation rather than task execution, and carries no task identifiers or input/output integrity data. The "act" chain cannot represent branching (fan-out) or convergence (fan-in) and therefore cannot form a DAG.

ECTs intentionally use the distinct claim name "exec_act" for the action/task type to avoid collision with the "act" claim. The two concepts are orthogonal: "act" records "who authorized whom," ECTs record "what was done, in what order."

Transaction Tokens

OAuth Transaction Tokens [I-D.ietf-oauth-transaction-tokens] (currently at version -08 and in IETF Last Call; the normative reference will be updated to the published RFC) propagate authorization context across workload call chains. The Txn-Token "req_wl" claim accumulates a comma-separated list of workloads that requested replacement tokens, which is the closest existing mechanism to call-chain recording.

However, "req_wl" cannot form a DAG because:

- It is linear: a comma-separated string with no branching or merging representation. When a workload fans out to multiple downstream services, each receives the same "req_wl" value and the branching is invisible.
- It is incomplete: only workloads that request a replacement token from the Transaction Token Service appear in "req_wl"; workloads that forward the token unchanged are not recorded.
- It carries no task-level granularity, no parent references, and no execution content.
- It cannot represent convergence (fan-in): when two independent paths must both complete before a dependent task proceeds, a linear "req_wl" string cannot express that relationship.

Extensions for agentic use cases ([I-D.oauth-transaction-tokens-for-agents]) add agent identity and constraints ("agentic_ctx") but no execution ordering or DAG structure.

ECTs and Transaction Tokens are complementary: a Txn-Token propagates authorization context ("this request is authorized for scope X on behalf of user Y"), while an ECT records execution accountability ("task T was performed, depending on tasks P1 and P2"). An agent request could carry both a Txn-Token for authorization and an ECT for execution recording. In WIMSE deployments, the WPT "tth" claim defined in [I-D.ietf-wimse-s2s-protocol] can hash-bind a WPT to a co-present Txn-Token; a similar binding mechanism for ECTs is a potential future extension.

Distributed Tracing (OpenTelemetry)

OpenTelemetry [OPENTELEMETRY] and similar distributed tracing systems provide observability for debugging and monitoring. ECTs differ in several important ways: at L2 and L3, ECTs are cryptographically signed per-task with the agent's private key; ECTs are tamper-evident through JWS signatures; ECTs enforce DAG validation rules; and ECTs are designed for regulatory audit rather than operational monitoring. OpenTelemetry data is typically controlled by the platform operator and can be modified or deleted without detection. ECTs and distributed traces are complementary: traces provide observability while ECTs provide execution records. ECTs may reference OpenTelemetry trace identifiers in the "ect_ext" claim for correlation.

W3C Provenance Data Model (PROV)

The W3C PROV Data Model defines an Entity-Activity-Agent ontology for representing provenance information. PROV's concepts map closely to ECT structures: PROV Activities correspond to ECT tasks, PROV Agents correspond to ECT-issuing agents, and PROV's "wasInformedBy" relation corresponds to ECT "pred" references. However, PROV uses RDF/OWL ontologies designed for post-hoc documentation, while ECTs are runtime-embeddable JWT tokens with cryptographic signatures. ECT audit data could be exported to PROV format for interoperability with provenance-aware systems.

SCITT (Supply Chain Integrity, Transparency, and Trust)

The SCITT architecture [I-D.ietf-scitt-architecture] (version -22, currently in AUTH48 / RFC Editor queue and about to become an RFC; readers should use the RFC number once assigned) defines a framework for transparent and auditable supply chain records. ECTs and SCITT are complementary: the ECT "wid" claim can serve as a correlation identifier in SCITT Signed Statements, linking an ECT audit trail to a supply chain transparency record.

There is a notable parallel between SCITT's Transparency Service and ECT's Level 3 audit ledger: both use append-only logs with cryptographic commitment to provide tamper-evident recording. SCITT Signed Statements use COSE for their envelope format while ECTs use JOSE, but the architectural pattern — transparent, verifiable recording of statements about artifacts or actions — is shared. A deployment requiring L3 assurance could use a SCITT Transparency Service as the audit ledger backend, recording ECTs as supply chain statements about agent execution.

RATS (Remote Attestation Procedures)

RATS [RFC9334] defines an architecture for conveying attestation evidence about platform trustworthiness. RATS attests "is this platform in a trustworthy state?" while ECTs record "what did this agent do?" — both deal with claims about entities but at different layers. RATS operates at the platform and firmware layer, establishing that a workload's execution environment has not been tampered with, whereas ECTs operate at the application layer, recording the logical sequence of tasks performed by agents. ECTs could complement RATS by recording execution context on platforms whose trustworthiness has been established through RATS attestation.

Emerging Agent Protocol Frameworks

Several emerging frameworks address agent-to-agent communication, including Google's Agent-to-Agent Protocol (A2A), Anthropic's Model Context Protocol (MCP), and orchestration frameworks such as LangChain and LangGraph. These frameworks primarily address agent discovery, message routing, and tool invocation but do not provide cryptographically verifiable execution records or DAG-based audit trails. ECTs complement these frameworks by adding an execution accountability layer: agents communicating via any of these protocols can produce and verify ECTs to record what was done, regardless of the communication mechanism used.

Acknowledgments

The author thanks the WIMSE working group for their foundational work on workload identity in multi-system environments. The WIMSE concepts of Workload Identity Tokens and Workload Proof Tokens provided the original motivation for execution context tracing and remain a primary identity binding for ECTs.

Author's Address

Christian Nennemann
Independent Researcher
Email: ietf@nennemann.de