WIMSE
Internet-Draft
Intended status: Standards Track
Expires: 29 August 2026

C. Nennemann
Independent Researcher

25 February 2026

Execution Context Tokens for Distributed Agentic Workflows
draft-nennemann-wimse-ect-00

Abstract

This document defines Execution Context Tokens (ECTs), a JWT-based extension to the WIMSE architecture that records task execution across distributed agentic workflows. Each ECT is a signed record of a single task, linked to predecessor tasks through a directed acyclic graph (DAG). ECTs reuse the WIMSE signing model and are transported in a new Execution-Context HTTP header field alongside existing WIMSE identity headers.

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 29 August 2026.

Copyright Notice

Table of Contents

1. Introduction

The WIMSE framework [I-D.ietf-wimse-arch] and its service-to- service protocol [I-D.ietf-wimse-s2s-protocol] authenticate workloads across call chains but do not record what those workloads actually did. This document defines Execution Context Tokens (ECTs), a JWT-based extension that fills the gap between workload identity and execution accountability. Each ECT is a signed record of a single task, linked to predecessor tasks through a directed acyclic graph (DAG).

1.1. Scope and Applicability

This document defines:

* The Execution Context Token (ECT) format (Section 3)

* DAG structure for task dependency ordering (Section 5)

* An HTTP header for ECT transport (Section 4)

* Audit ledger interface requirements (Section 7)

The following are out of scope and are handled by WIMSE:

* Workload authentication and identity provisioning

* Key distribution and management

* Trust domain establishment and management

* Credential lifecycle management

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, as defined by WIMSE [I-D.ietf-wimse-arch], that executes tasks within a workflow.

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 JSON Web Token [RFC7519] defined by this specification that records task execution details.

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

Workload Identity Token (WIT): A WIMSE credential proving a workload’s identity within a trust domain.

Workload Proof Token (WPT): A WIMSE proof-of-possession token used for request-level authentication.

Trust Domain: A WIMSE concept representing an organizational boundary with a shared identity issuer, corresponding to a SPIFFE [SPIFFE] trust domain.

3. Execution Context Token Format

An Execution Context Token is a JSON Web Token (JWT) [RFC7519] signed as a JSON Web Signature (JWS) [RFC7515]. 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.

ECTs reuse the WIMSE signing model. The ECT MUST be signed with the same private key associated with the agent's WIT. The JOSE header "kid" parameter MUST reference the public key identifier from the agent's WIT, and the "alg" parameter MUST match the algorithm used in the corresponding WIT. In WIMSE deployments, the ECT "iss" claim SHOULD use the WIMSE workload identifier format (a SPIFFE ID [SPIFFE]).

3.1. JOSE Header

The ECT JOSE header MUST contain the following parameters:

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

Figure 1: ECT JOSE Header Example

alg: REQUIRED. The digital signature algorithm used to sign the ECT. MUST match the algorithm in the corresponding WIT. 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.

typ: REQUIRED. MUST be set to "wimse-exec+jwt" to distinguish ECTs from other JWT types, consistent with the WIMSE convention for type parameter values.

kid: REQUIRED. The key identifier referencing the public key from the agent's WIT [RFC7517]. Used by verifiers to look up the correct public key for signature verification.

3.2. JWT Claims

3.2.1. Standard JWT Claims

An ECT MUST contain the following standard JWT claims [RFC7519]:

iss: REQUIRED. StringOrURI. A URI identifying the issuer of the ECT. In WIMSE deployments, this SHOULD be the workload's SPIFFE ID in the format spiffe://<trust-domain>/<path>, matching the "sub" claim of the agent's WIT. Non-WIMSE deployments MAY use other URI schemes (e.g., HTTPS URLs or URN:UUID identifiers).

aud: REQUIRED. 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 must 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".

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 "par"). 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.

3.2.2. Execution Context

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].

par: REQUIRED. Array of strings. Parent 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.

3.2.3. Data Integrity

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".

3.2.4. Extensions

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 "ext" object SHOULD NOT exceed 4096 bytes and SHOULD NOT exceed a nesting depth of 5 levels.

3.3. Complete ECT Example

The following is a complete ECT payload example:

```
{
  "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",
  "par": [],

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

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

Figure 2: Complete ECT Payload Example

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 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 the WIMSE Workload-Identity header. When a Workload Proof Token (WPT) is available per [I-D.ietf-wimse-s2s-protocol], agents SHOULD include it alongside the WIT and ECT.

```
GET /api/safety-check HTTP/1.1
Host: safety-agent.example.com
Workload-Identity: eyJhbGci...WIT...
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 in JWS Compact Serialization format.

When a receiver processes multiple Execution-Context headers, it MUST individually verify each ECT per the procedure in Section 6. 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.

5. DAG Validation

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

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 "par" 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.

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.

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. Signature and Token Verification

6.1. Verification Procedure

When an agent receives an 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 "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 a WIT within the trust domain.

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 trust domain's key lifecycle mechanism (e.g., certificate revocation list, OCSP, or SPIFFE trust bundle updates).

7. Verify the "alg" header parameter matches the algorithm in the corresponding WIT.

8. Verify the "iss" claim matches the "sub" claim of the WIT associated with the "kid" public key.

9. Verify the "aud" claim contains the verifier's own workload 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", "par") 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.

When ECT verification fails during HTTP request processing, the receiving agent SHOULD respond with HTTP 403 (Forbidden) if the WIT is valid but the ECT is invalid, and HTTP 401 (Unauthorized) if the ECT signature verification fails. 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.

7. Audit Ledger Interface

ECTs MAY be recorded in an immutable audit ledger for compliance verification and post-hoc analysis. A ledger is RECOMMENDED for regulated environments but is not required for point-to-point operation. 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.

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.

8. Security Considerations

8.1. 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.

8.2. 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. 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.

8.3. Signature Verification

ECTs MUST be signed with the agent’s private key using JWS [RFC7515]. The signature algorithm MUST match the algorithm specified in the agent’s WIT. Receivers MUST verify the ECT signature against the WIT 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 6).

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.

8.4. 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. 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 6).

8.5. Man-in-the-Middle Protection

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

8.6. 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.

8.7. 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 "par" to hide influences on its output; because ECTs are self-asserted (Section 8.2), 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.

8.8. 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 "par", as an authorization grant. Authorization decisions MUST remain with the identity and authorization layer (WIT, WPT, and deployment policy).

8.9. Denial of Service

Implementations SHOULD apply rate limiting to prevent excessive ECT submissions. DAG validation SHOULD be performed after signature verification to avoid wasting resources on unsigned tokens.

8.10. 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.

8.11. ECT Size Constraints

Implementations SHOULD limit the "par" array to a maximum of 256 entries. See Section 3.2.4 for "ext" size limits.

9. Privacy Considerations

9.1. 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)

9.2. 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 "ext" (Section 3.2.4) deserve particular attention: human-readable values risk exposing sensitive operational details. See Section 3.2.4 for guidance on using structured identifiers.

9.3. 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.

10. IANA Considerations

10.1. Media Type Registration

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

Type name: application

Subtype name: wimse-exec+jwt

Required parameters: none

Optional parameters: none

Encoding considerations: 8bit; 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.

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: Christi an Nennemann, ietf@nennemann.de

Intended usage: COMMON

Restrictions on usage: none

Author: Christian Nennemann

Change controller: IETF

10.2. 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

10.3. 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 3.2.2
exec_act	Action/Task Type	IETF	Section 3.2.2
par	Parent Task Identifiers	IETF	Section 3.2.2
inp_hash	Input Data Hash	IETF	Section 3.2.3
out_hash	Output Data Hash	IETF	Section 3.2.3
ext	Extension Object	IETF	Section 3.2.4

Table 1: JWT Claims Registrations

11. References

11.1. Normative References

[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-06, 30 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-wimse-arch-06>.

[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>.

[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>.

[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>.

11.2. Informative References

[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-07, 24 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-transaction-tokens-07>.

[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.oauth-transaction-tokens-for-agents]
Raut, A., "Transaction Tokens For Agents", Work in Progress, Internet-Draft, draft-oauth-transaction-tokens-for-agents-04, 10 February 2026, <https://datatracker.ietf.org/doc/html/draft-oauth-transaction-tokens-for-agents-04>.

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

[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>.

[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>.

[SPIFFE] "Secure Production Identity Framework for Everyone (SPIFFE)", <https://spiffe.io/docs/latest/spiffe-about/overview/>.

Use Cases

This section describes a representative use case demonstrating how ECTs provide structured execution records.

Note: task identifiers in this section are abbreviated for readability. In production, all "jti" values are required to be UUIDs per Section 3.2.2.

Cross-Organization Financial Trading

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.

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

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

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

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

Figure 4: Cross-Organization Trading Workflow

The resulting DAG:

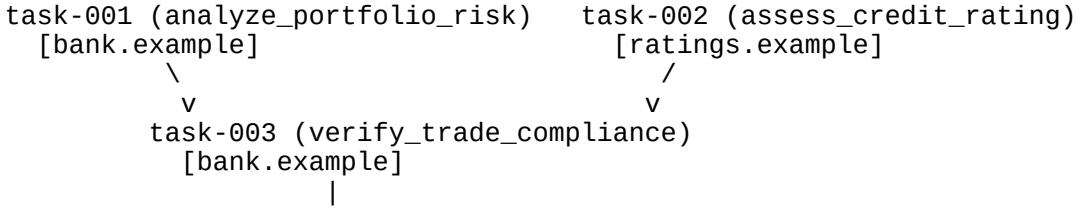

Figure 5: Cross-Organization DAG

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.

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 the identity foundation upon which ECTs are built. WIT/WPT answer "who is this agent?" and "does it control the claimed key?" while ECTs record "what did this agent do?" Together they form an identity-plus-accountability framework for regulated agentic systems.

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] 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. 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: 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 signed execution records. ECTs may reference OpenTelemetry trace identifiers in the "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 WIMSE workloads, and PROV's "wasInformedBy" relation corresponds to ECT "par" 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] 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.

Acknowledgments

The author thanks the WIMSE working group for their foundational work on workload identity in multi-system environments. The concepts of Workload Identity Tokens and Workload Proof Tokens provide the identity foundation upon which execution context tracing is built.

Author’s Address

Christian Nennemann
Independent Researcher
Email: ietf@nennemann.de