Network Working Group
Internet-Draft
Intended status: Informational
Expires: 27 September 2026

C. Yakung
Attest

26 March 2026

Agent Credential Attestation Protocol (ACAP)
draft-yakung-oauth-agent-attestation-00

Abstract

This document defines the Agent Credential Attestation Protocol (ACAP), a cryptographic credentialing protocol for autonomous AI agent pipelines. An ACAP credential is a short-lived JSON Web Token (JWT) signed with RS256 that carries scope-limited permissions together with a SHA-256 hash of the original human instruction that initiated the task. Credentials may be delegated to child agents; each delegation narrows scope, cannot outlive its parent, increments a delegation depth counter, and extends a tamper-evident chain of token identifiers. Every lifecycle event is recorded in an append-only, hash-chained audit log.

This document specifies the credential format, issuance rules, delegation rules, verification algorithm, revocation semantics, human-in-the-loop approval protocol, and audit log structure.

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 27 September 2026.

Copyright Notice

Table of Contents

1. Introduction

1.1. Problem Statement

Contemporary authorization frameworks such as OAuth 2.0 [RFC6749] and OpenID Connect [OIDC] were designed for human-initiated, single-hop delegations: a resource owner grants a client access on behalf of themselves. AI agent pipelines violate this assumption in several important ways.

First, an agent pipeline may involve an arbitrary number of hops. A root agent receives a task from a human, decomposes it, and delegates sub-tasks to child agents, which may themselves delegate further. OAuth's two-party model has no native representation for this; each hop requires a fresh grant cycle or an out-of-band trust agreement.

Second, the originating human instruction -- the intent -- is not cryptographically bound to any OAuth token. An agent can receive a token whose original purpose has been transformed or corrupted by prompt injection at an intermediate step, and the verifying party has no way to detect this.

Third, scope in OAuth is flat and is not guaranteed to narrow monotonically through a delegation chain. A child agent can, in principle, present its parent token to a different authorization server and request broader access. Nothing in the wire format prevents scope creep.

Fourth, the delegation graph is not natively recorded. OAuth introspection surfaces information about a single token; it provides no view of the full task ancestry.

Recent IETF work -- notably the WIMSE working group [WIMSE] and the draft on AI agents acting on behalf of users [OBO01] -- acknowledges these gaps but stops short of specifying a compact, self-contained credential format with cryptographically enforced monotone scope reduction and intent binding.

1.2. What ACAP Adds

ACAP addresses the gaps above as follows.

Intent binding: Every ACAP credential carries att_intent, a hex-encoded SHA-256 hash of the original UTF-8 instruction text. This value is set at issuance and propagated unchanged through every delegation. A verifier can confirm that a presented credential descends from a specific human instruction by independently computing the hash.

Monotone scope reduction: At delegation time the issuer MUST verify that the child's requested scope is a subset of the parent's scope using the IsSubset algorithm defined in Section 5. Any request that fails this check MUST be rejected. Scope therefore only ever narrows; it can never widen through delegation.

Depth-limited delegation: Each credential carries att_depth, an

integer that starts at 0 for a root credential and increments by 1 at each delegation. Depth is bounded above by MaxDelegationDepth (10). A credential whose depth equals this limit MUST NOT be used as a parent for further delegation.

Tamper-evident chain: Each credential carries att_chain, an ordered list of JWT IDs from the root of the delegation tree to the current token. The verifier checks that the chain length equals att_depth + 1 and that the final element equals the token's own jti. Together these checks detect any tampering with the ancestry record.

Lifetime containment: A child credential's expiry is capped at the parent's expiry. Concretely, exp = min(requested_exp, parent_exp). This ensures that revoking a parent token effectively expires all descendants, even without an active revocation lookup.

Cascade revocation: The revocation store records revoked JTIs and cascades revocation to all descendants by inspecting the att_chain column in the credential store. Revocation is permanent and takes precedence over token expiry.

Append-only audit log: Every issuance, delegation, verification, revocation, and expiry event is recorded in a hash-chained log. Each entry commits to the previous entry's hash, the event type, the JTI, and the timestamp, forming a tamper-evident record of all credential lifecycle events within a task tree.

1.3. Relationship to Prior Art

ACAP is informed by but is not a profile of any existing specification. The Agentic JWT (A-JWT) paper proposes delegation chains for AI agents but does not specify exact claim semantics, scope subset enforcement, or audit log structure. The IETF draft [AGENTJWT] covers similar ground in the JWT format domain. The draft [OBO01] addresses delegation in the OAuth authorization code flow context. ACAP occupies a different point in the design space: it is a self-contained signed-token format with no external authorization server required at verification time.

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.

3. Terminology

Issuer: The service that creates and signs ACAP credentials. The Issuer holds an RSA private key and is identified by a URI recorded in the iss JWT claim. A single Issuer MAY serve multiple task trees.

Root Credential: An ACAP credential whose att_depth is 0 and whose att_pid is absent. A root credential is issued directly by the Issuer in response to a human principal's instruction. Its att_chain contains exactly one element: its own jti.

Delegated Credential: An ACAP credential whose att_depth is greater than 0. A delegated credential is derived from a parent credential and MUST have att_pid set to the parent's jti. Its scope MUST be a subset of the parent's scope and its expiry MUST NOT exceed the parent's expiry.

Task Tree: The complete graph of credentials that share the same att_tid. A task tree has exactly one root credential; all other credentials in the tree are delegated credentials descended from it.

Intent Hash: The value of the att_intent claim: a lowercase hex-encoded SHA-256 digest of the UTF-8 encoding of the original human instruction string. The intent hash is set at root issuance and propagated unchanged to all descendants.

Depth: The integer value of att_depth. Depth 0 identifies a root credential. Each delegation increments depth by exactly 1. The maximum permitted depth is MaxDelegationDepth (10).

Chain: The ordered list of JWT IDs in att_chain. For a credential at depth D, the chain contains exactly D+1 elements: the root credential's jti at index 0, followed by each intermediate credential's jti in delegation order, with the current credential's jti at the final position (index D).

Scope Entry: A string of the form resource:action where resource and action are non-empty strings. The wildcard character * MAY appear in either position.

4. Credential Format

4.1. Overview

An ACAP credential is a JSON Web Token [RFC7519] with:

* Header: { "alg": "RS256", "typ": "JWT" }

* Payload: the claims defined in Section 4.2 and Section 4.3

* Signature: RS256 over the ASCII representation of the header and payload

All implementations MUST use RS256 (RSASSA-PKCS1-v1_5 using SHA-256 [RFC7518]) as the signing algorithm. No other signing algorithm is permitted.

4.2. Standard JWT Claims

The following standard JWT claims are used. All are REQUIRED unless noted.

Claim	Type	Required	Description
iss	string	REQUIRED	Issuer URI
sub	string	REQUIRED	Subject. MUST be of the form agent:{agent_id}
iat	NumericDate	REQUIRED	Issued-at time (Unix epoch seconds, UTC)
exp	NumericDate	REQUIRED	Expiry time (Unix epoch seconds, UTC)
jti	string	REQUIRED	JWT ID. A UUID v4 [RFC9562] that uniquely identifies this credential

Table 1

The sub claim MUST match the pattern ^agent:[A-Za-z0-9_\-]+$. Implementations MUST reject credentials whose sub does not begin with the literal prefix agent:.

4.3. ACAP Extension Claims

All ACAP-specific claims are prefixed with att_. Implementations MUST ignore unknown att_* claims to allow forward compatibility.

Claim	Type	Required	Description
att_tid	string (UUID v4)	REQUIRED	Task tree identifier
att_pid	string (UUID v4)	CONDITIONAL	Parent credential identifier
att_depth	integer	REQUIRED	Delegation depth (0 for root)
att_scope	array of strings	REQUIRED	Permission set
att_intent	string (hex)	REQUIRED	SHA-256 of original instruction (64 hex chars)
att_chain	array of strings	REQUIRED	Ordered list of JTIs from root to current
att_uid	string	REQUIRED	Originating human user identifier
att_hitl_req	string (UUID v4)	OPTIONAL	HITL approval request ID
att_hitl_uid	string	OPTIONAL	Identity of the human who approved
att_hitl_iss	string	OPTIONAL	IdP issuer of the approving human
att_idp_iss	string	OPTIONAL	IdP issuer from root OIDC session
att_idp_sub	string	OPTIONAL	IdP subject from root OIDC session
att_ack	string	OPTIONAL	Agent binary checksum digest

Table 2

att_pid MUST be present when att_depth > 0 and MUST be absent when att_depth == 0.

att_tid, att_intent, and att_uid MUST be propagated unchanged through all delegations.

4.4. Concrete Examples

4.4.1. Root Credential Payload

```
{
  "iss": "https://attest.example.com",
  "sub": "agent:inbox-agent-v2",
  "iat": 1742386800,
  "exp": 1742473200,
  "jti": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "att_tid": "f9e8d7c6-b5a4-3210-fedc-ba9876543210",
  "att_depth": 0,
  "att_scope": ["email:read", "email:draft"],
  "att_intent": "3b4c2a1f8e7d6c5b4a3f2e1d0c9b8a7f...",
  "att_chain": ["a1b2c3d4-e5f6-7890-abcd-ef1234567890"],
  "att_uid": "user:alice"
}
```

Note that att_pid is absent because this is a root credential. The att_chain contains exactly one element -- the credential's own jti.

4.4.2. Delegated Credential Payload

```
{
  "iss": "https://attest.example.com",
  "sub": "agent:summariser-agent-v1",
  "iat": 1742387100,
  "exp": 1742473200,
  "jti": "c3d4e5f6-a7b8-9012-cdef-012345678901",
  "att_tid": "f9e8d7c6-b5a4-3210-fedc-ba9876543210",
  "att_pid": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "att_depth": 1,
  "att_scope": ["email:read"],
  "att_intent": "3b4c2a1f8e7d6c5b4a3f2e1d0c9b8a7f...",
  "att_chain": [
    "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    "c3d4e5f6-a7b8-9012-cdef-012345678901"
  ],
  "att_uid": "user:alice"
}
```

5. Scope Model

5.1. Scope Entry Format

A scope entry is a string conforming to the grammar:

```
scope-entry = resource ":" action
resource    = 1*( ALPHA / DIGIT / "_" / "-" / "*" )
action      = 1*( ALPHA / DIGIT / "_" / "-" / "*" )
```

Both resource and action MUST be non-empty. Implementations MUST reject invalid scope entries at all stages: issuance, delegation, and verification.

5.2. Wildcard Rules

The wildcard character * MAY appear as the entirety of either the resource or action component (or both). Coverage rules:

* P covers C if P.resource equals * OR P.resource equals C.resource

* AND P.action equals * OR P.action equals C.action

Consequently *:* covers every valid scope entry, email:* covers email:read, email:draft, etc.

5.3. NormaliseScope

Before storing scope in any credential or comparing scopes, implementations MUST apply the NormaliseScope procedure:

1. Trim leading and trailing whitespace from each entry.

2. Remove any empty entries that result from trimming.

3. Remove duplicate entries, preserving the first occurrence.

4. Return the resulting list in insertion order.

5.4. IsSubset Algorithm

```
function IsSubset(parentScope, childScope):
    for each entry CE in childScope:
        childEntry = ParseScope(CE)
        if childEntry is invalid:
            return false
        covered = false
        for each entry PE in parentScope:
            parentEntry = ParseScope(PE)
            if parentEntry is invalid:
                continue
            if EntryCovers(parentEntry, childEntry):
                covered = true
                break
        if not covered:
            return false
    return true

function EntryCovers(parent, child):
    resourceOK = (parent.Resource == "*") OR
                 (parent.Resource == child.Resource)
    actionOK   = (parent.Action  == "*") OR
                 (parent.Action  == child.Action)
    return resourceOK AND actionOK
```

6. Issuance

6.1. Overview

Issuance creates a root credential (depth 0) that anchors a new task tree. The Issuer MUST perform all validation checks before constructing the credential. The Issuer MUST sign the credential with its RSA private key using RS256.

6.2. Input Validation

The Issuer MUST reject the issuance request if any of the following conditions hold:

1. agent_id is absent or empty.

2. user_id is absent or empty.

3. scope is absent or contains no entries.

4. Any entry in scope is not parseable as a valid scope entry.

5. instruction is absent or empty.

6.3. Intent Hash Computation

The intent hash is computed as:

att_intent = lowercase-hex(SHA-256(UTF-8-encode(instruction)))

The instruction string MUST be encoded as UTF-8 before hashing. The resulting 32-byte SHA-256 digest MUST be encoded as 64 lowercase hexadecimal characters. No canonicalization is applied to the instruction before hashing; the raw UTF-8 bytes are hashed as-is.

6.4. Identifier Generation

The Issuer MUST generate two UUID v4 [RFC9562] values at issuance:

* jti: the unique identifier for this credential.

* att_tid: the task tree identifier.

Both values MUST be generated using a cryptographically secure random source.

6.5. TTL and Expiry

1. If ttl_seconds is zero, the TTL defaults to DefaultTTLSeconds (3600 seconds).

2. If ttl_seconds is negative, the issuance MUST be rejected.

3. If ttl_seconds exceeds MaxTTLSeconds (86400 seconds), the TTL is capped at MaxTTLSeconds.

4. exp = iat + ttl_seconds (after applying the rules above).

6.6. Root Credential Construction

Field	Value
iss	Issuer URI
sub	"agent:" + agent_id
iat	Current UTC time
exp	iat + ttl
jti	Freshly generated UUID v4

att_tid	Freshly generated UUID v4
att_pid	Absent
att_depth	0
att_scope	NormaliseScope(scope)
att_intent	hex(SHA-256(UTF-8(instruction)))
att_chain	[jti]
att_uid	user_id

Table 3

7. Delegation

7.1. Overview

Delegation creates a child credential derived from an existing parent credential. The result is a credential at att_depth + 1 with a scope that is a subset of the parent's scope and an expiry no later than the parent's expiry.

7.2. Input Validation

The Issuer MUST reject a delegation request if:

1. parent_token is absent or empty.

2. child_agent is absent or empty.

3. child_scope is absent or contains no entries.

7.3. Parent Token Verification

The Issuer MUST:

1. Verify the RS256 signature against the Issuer's public key.

2. Verify that the parent token has not expired (exp > now).

3. Verify that the signing algorithm is RS256.

7.4. Scope Subset Enforcement

The Issuer MUST apply NormaliseScope to the requested child_scope. The Issuer MUST then apply IsSubset(parentScope, childScope). If IsSubset returns false, the delegation MUST be rejected.

7.5. Depth Limit

The Issuer MUST check that parent.att_depth < MaxDelegationDepth (10). If the parent's depth equals or exceeds the limit, the delegation MUST be rejected.

7.6. Expiry Computation

```
parent_exp = parent.exp

if ttl_seconds > 0:
    requested_exp = now + ttl_seconds
    child_exp = min(requested_exp, parent_exp)
else:
    default_exp = now + DefaultTTLSeconds
    child_exp = min(default_exp, parent_exp)
```

The child's expiry MUST NOT exceed the parent's expiry.

7.7. Delegated Credential Construction

Field	Value
iss	Issuer URI
sub	"agent:" + child_agent
iat	Current UTC time
exp	min(now + ttl_seconds, parent.exp)
jti	Freshly generated UUID v4
att_tid	parent.att_tid (propagated)
att_pid	parent.jti
att_depth	parent.att_depth + 1
att_scope	NormaliseScope(child_scope)

att_intent	parent.att_intent (propagated)
att_chain	parent.att_chain + [jti]
att_uid	parent.att_uid (propagated)

Table 4

8. Verification

8.1. Overview

A credential is valid if and only if all of the following conditions are satisfied:

1. The RS256 signature verifies against the Issuer’s public key.
2. The current time is before exp (subject to clock-skew allowance).
3. The length of att_chain equals att_depth + 1.
4. The final element of att_chain equals jti.
5. The jti is not present in the revocation store.

8.2. Verification Algorithm

```
function Verify(tokenString, issuerPublicKey, revocationStore):

    // Step 1: Signature and expiry
    claims, err = RS256Parse(tokenString, issuerPublicKey)
    if err is not nil:
        return invalid("signature verification failed")

    // Step 2: Chain length invariant
    expectedLen = claims.att_depth + 1
    if len(claims.att_chain) != expectedLen:
        return invalid("chain length mismatch")

    // Step 3: Chain tail invariant
    if claims.att_chain[len(claims.att_chain) - 1] != claims.jti:
        return invalid("chain tail does not match jti")

    // Step 4: Revocation check
    if revocationStore.IsRevoked(claims.jti):
        return invalid("credential has been revoked")

    return valid(claims)
```

8.3. Warnings vs. Hard Failures

Chain length and chain tail inconsistencies are surfaced as warnings that cause the credential to be reported as invalid. The warning mechanism exists to distinguish between cryptographic failures (which may indicate active attack) and structural inconsistencies (which may indicate implementation bugs).

9. Revocation

9.1. Semantics

Revocation permanently invalidates a credential identified by its jti. Once revoked, a credential MUST be treated as invalid regardless of its exp claim. Revocation is irreversible.

9.2. Cascade Semantics

Revoking a credential with jti X automatically revokes every credential whose att_chain contains X:

```
Revoke(X):
    targets = { jti | jti == X OR X in credentials[jti].att_chain }
    for each T in targets:
        revocations.insert(T, now, revokedBy)
```

Cascade revocation SHOULD be performed atomically within a single database transaction.

9.3. Revocation Store

The revocation store records:

Field	Type	Description
jti	text (PK)	The revoked credential identifier
revoked_at	timestamptz	UTC timestamp of revocation
revoked_by	text	Agent or user ID that triggered revocation

Table 5

Duplicate revocation attempts MUST be treated as a no-op (idempotent).

9.4. Lifetime Interaction

A credential that has expired is invalid regardless of the revocation store. A credential that has been revoked is invalid regardless of whether it has expired. These are independent failure conditions.

10. Human-in-the-Loop Approval

10.1. Overview

Certain delegations require explicit human approval before being granted. ACAP defines an approval protocol that makes the human decision a cryptographic event embedded in the resulting credential.

10.2. Approval Lifecycle

The HITL approval flow consists of four phases:

1. *Request.* The agent sends an approval request containing the parent_token, requested child_scope, an intent description, and the agent_id. The Issuer creates a pending approval record and returns a challenge_id.

2. *Poll.* The agent polls for approval status using the challenge_id. Status is one of: pending, approved, rejected, or expired.

3. *Grant or Deny.* A human reviews the request via a dashboard or integration. To grant, the human authenticates via an OIDC Identity Provider and the Issuer verifies the id_token. The Issuer records the human's identity and marks the approval as granted.

4. *Credential Issuance.* Upon grant, the Issuer delegates a new credential from the parent token with the approved scope. The credential carries att_hitl_req, att_hitl_uid, and att_hitl_iss.

10.3. Approval Expiry

Pending approvals MUST expire after a bounded time window. Implementations MUST NOT allow approvals to remain pending indefinitely. An expired approval MUST be treated identically to a rejection.

10.4. Parent Token Re-verification

When an approval is granted, the Issuer MUST re-verify the parent token before issuing the delegated credential. If the parent token has expired or been revoked while the approval was pending, the Issuer MUST reject the grant.

10.5. HITL Claims Propagation

The HITL claims (att_hitl_req, att_hitl_uid, att_hitl_iss) from the most recent human approval propagate to all subsequent delegations. If a new HITL approval occurs at a deeper delegation, the new claims replace the inherited ones.

10.6. Multi-Tenant Isolation

Approval requests are scoped to the authenticated organization. An approval created by org A MUST NOT be resolvable by org B.

11. Audit Log

11.1. Structure

The audit log is an append-only, hash-chained record of credential lifecycle events. Each entry commits to the previous entry's hash, making the log tamper-evident.

The audit log is partitioned by att_tid. Each task tree has its own independent hash chain. The first event uses a genesis hash of 64 ASCII zero characters as the previous hash value.

11.2. Entry Hash Computation

```
entry_hash = lowercase-hex( SHA-256(
    prev_hash
    || event_type
    || jti
    || created_at_rfc3339nano
) )
```

where || denotes string concatenation. The genesis hash is:

```
"0000000000000000000000000000000000000000000000000000000000000000"
```

(64 ASCII zero characters.)

11.3. Event Types

Event Type	Trigger
issued	Root credential issued
delegated	Delegated credential issued
verified	Credential verified
revoked	Credential revoked (one event per cascade target)
expired	Credential reached exp time
hitl_granted	Human approved a HITL request
action	Agent executed a registered action
lifecycle	Agent lifecycle transition (started/completed/failed)

Table 6

11.4. Audit Entry Fields

Field	Type	Description
id	bigserial	Monotonically increasing row ID
prev_hash	text	Hash of the prior entry for same att_tid
entry_hash	text	SHA-256 of concatenated fields
event_type	text	One of the defined event types
jti	text	JWT ID of the credential involved
org_id	text	Tenant identifier
att_tid	text	Task tree identifier
att_uid	text	Human principal identifier
agent_id	text	Agent identifier
scope	jsonb	Scope array at time of event
meta	jsonb	Optional implementation-defined metadata
created_at	timestamptz	UTC timestamp

Table 7

11.5. Append-Only Enforcement

Implementations MUST enforce append-only semantics on the audit log.

11.6. Log Verification

To verify the integrity of the audit log for a given att_tid:

1. Retrieve all entries in ascending id order.

2. For each entry starting at the second, verify that entry.prev_hash equals the entry_hash of the preceding entry.

3. Recompute entry_hash from the raw fields and verify it matches.

Any discrepancy indicates tampering or corruption.

12. Security Considerations

12.1. Prompt Injection

ACAP credentials bind to the intent hash of the original instruction, but they do not prevent a legitimate credential from being used by a compromised agent. The narrow, monotone-reducing scope model limits the blast radius of a compromised agent. The intent hash enables post-hoc detection of misuse. The att_ack claim carries a checksum of the agent binary for agent substitution detection.

12.2. Replay Attacks

Credentials carry a unique jti. Verifiers MUST check the revocation store on every verification call. The exp claim bounds the replay window for non-revoked credentials. Verifiers SHOULD maintain a short-term cache of recently seen jti values.

12.3. Scope Creep

Scope creep is prevented by IsSubset enforcement at delegation time. Resource servers MUST verify that the presented credential’s att_scope covers the requested operation.

12.4. Clock Skew

Implementations SHOULD allow a clock-skew leeway of up to 60 seconds. Implementations MUST NOT allow a leeway of more than 300 seconds.

12.5. Key Management

The Issuer’s RSA private key is the root of trust. Compromise allows forging credentials. Implementations SHOULD use hardware security modules and short key rotation periods. The public key MUST be distributed via an out-of-band mechanism (e.g., a JWKS endpoint).

12.6. Credential Store Integrity

The revocation store and credential store are security-critical. Implementations MUST apply access controls preventing unauthorized modification.

12.7. Audit Log Integrity

Hash-chaining provides tamper-evidence but not tamper-prevention. The audit log SHOULD be replicated to a write-once store or transparency log for stronger guarantees.

13. IANA Considerations

This specification defines the following JWT claim names for registration in the IANA "JSON Web Token Claims" registry established by [RFC7519].

Claim Name	Description	Change Controller	Reference
att_tid	ACAP task tree identifier	IESG	This document
att_pid	ACAP parent credential identifier	IESG	This document
att_depth	ACAP delegation depth	IESG	This document
att_scope	ACAP permission scope	IESG	This document
att_intent	ACAP intent hash	IESG	This document
att_chain	ACAP delegation chain	IESG	This document
att_uid	ACAP originating user identifier	IESG	This document
att_hitl_req	ACAP HITL request ID	IESG	This document
att_hitl_uid	ACAP HITL approving user	IESG	This document
att_hitl_iss	ACAP HITL issuer authority	IESG	This document
att_idp_iss	ACAP IdP issuer	IESG	This document
att_idp_sub	ACAP IdP subject	IESG	This document
att_ack	ACAP agent checksum digest	IESG	This document

Table 8

14. References

14.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/rfc/rfc6749>.

[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>.

[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>.

14.2. Informative References

[AGENTJWT] Goswami, D., "Agentic JWT: Secure Delegation Protocol for AI Agent Pipelines", 2025, <https://datatracker.ietf.org/doc/draft-goswami-agentic-jwt/>.

[OBO01] "OAuth 2.0 for AI Agents Acting on Behalf of Users", 2025, <https://datatracker.ietf.org/doc/draft-oauth-ai-agents-on-behalf-of-user/>.

[OIDC] Sakimura, N., Bradley, J., Jones, M., Medeiros, B. de., and C. Mortimore, "OpenID Connect Core 1.0", November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

[WIMSE] "IETF Workload Identity in Multi System Environments (WIMSE) Working Group", n.d., <https://datatracker.ietf.org/wg/wimse/about/>.

Appendix A. Summary of Constants

Constant	Value	Description
MaxDelegationDepth	10	Maximum permitted att_depth
DefaultTTLSeconds	3600	Default credential lifetime (1 hour)
MaxTTLSeconds	86400	Maximum credential lifetime (24 hours)
Genesis hash	64 x "0"	Previous hash for first audit entry
Clock skew (SHOULD)	60s	Recommended leeway
Clock skew (MUST NOT exceed)	300s	Maximum leeway

Table 9

Appendix B. Implementation Status

A reference implementation is available as open source software:

* Server: Go, available at https://github.com/chudah1/attest-dev

* TypeScript SDK: @attest-dev/sdk on npm (0.1.0-beta.4)

* Python SDK: attest-sdk on PyPI (0.1.0b3)

* Framework integrations: Anthropic Claude, LangGraph, OpenAI Agents SDK, Model Context Protocol (MCP)

* Deployment: Docker Compose, Railway

All SDKs implement credential issuance, delegation, offline verification via JWKS, revocation, and audit trail retrieval.

Appendix C. Data Model

The following SQL schema is informative for implementations using relational storage:

```
CREATE TABLE IF NOT EXISTS credentials (
    jti         TEXT        PRIMARY KEY,
    org_id      TEXT        NOT NULL,
    att_tid     TEXT        NOT NULL,
    att_pid     TEXT,
    att_uid     TEXT        NOT NULL,
    agent_id    TEXT        NOT NULL,
    depth       INTEGER     NOT NULL DEFAULT 0,
    scope       TEXT[]      NOT NULL,
    chain       TEXT[]      NOT NULL,
    issued_at   TIMESTAMPTZ NOT NULL,
    expires_at  TIMESTAMPTZ NOT NULL
);

CREATE INDEX IF NOT EXISTS idx_credentials_chain
    ON credentials USING GIN (chain);

CREATE TABLE IF NOT EXISTS revocations (
    jti         TEXT        PRIMARY KEY,
    revoked_at  TIMESTAMPTZ NOT NULL,
    revoked_by  TEXT        NOT NULL
);

CREATE TABLE IF NOT EXISTS approvals (
    id              TEXT        PRIMARY KEY,
    org_id          TEXT        NOT NULL,
    agent_id        TEXT        NOT NULL,
    att_tid         TEXT        NOT NULL,
    parent_token    TEXT        NOT NULL,
    intent          TEXT        NOT NULL,
    requested_scope TEXT[]      NOT NULL,
    status          TEXT        NOT NULL DEFAULT 'pending',
    approved_by     TEXT,
    created_at      TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    resolved_at     TIMESTAMPTZ
);

CREATE TABLE IF NOT EXISTS audit_log (
    id           BIGSERIAL   PRIMARY KEY,
    org_id       TEXT        NOT NULL,
    prev_hash    TEXT        NOT NULL,
    entry_hash   TEXT        NOT NULL,
    event_type   TEXT        NOT NULL,
    jti          TEXT        NOT NULL,
    att_tid      TEXT        NOT NULL,
    att_uid      TEXT        NOT NULL,
    agent_id     TEXT        NOT NULL,
    scope        JSONB       NOT NULL DEFAULT '[]',
```

```
        meta         JSONB,
        idp_issuer   TEXT,
        idp_subject  TEXT,
        hitl_req     TEXT,
        hitl_issuer  TEXT,
        hitl_subject TEXT,
        created_at   TIMESTAMPTZ NOT NULL DEFAULT NOW()
    );

    CREATE OR REPLACE RULE audit_log_no_update AS
        ON UPDATE TO audit_log DO INSTEAD NOTHING;
    CREATE OR REPLACE RULE audit_log_no_delete AS
        ON DELETE TO audit_log DO INSTEAD NOTHING;
```

Acknowledgements

The authors acknowledge the contributions of the broader AI safety and identity communities, including the IETF WIMSE working group, the NIST AI Agent Standards Initiative, and the authors of the Agentic JWT proposal.

Author’s Address

Chudah Yakung
Attest
Email: ychudah@gmail.com