Multi-Hop Agent Delegation Chains: Identity Propagation Across Human → Agent → Agent → Tool → Resource

Machine Identity / Last updated 2026-06-11 / https://www.scrambleid.com/learn/multi-hop-agent-delegation-chains

In one sentence: The hardest agent identity problem in production today is not authenticating a single agent; it is propagating authorization across a chain of agents, tools, and resources while preserving original-caller attribution and enforcing scope contraction at every hop, and the right architecture is standards-based (RFC 8693 OAuth 2.0 Token Exchange) so it extends naturally as agentic-identity standards mature.

TL;DR (canonical)

- **The pattern:** Human → Agent A → Agent B → Tool C → Resource D. Each hop has its own identity. Each hop carries the chain. The resource sees the full delegation history.
- **Why it's hard:** Traditional OAuth assumes single human-to-app delegation. Multi-hop requires preserving original-delegator attribution, contracting scope at each hop, and auditing the full chain.
- **The standards-based answer:** RFC 8693 OAuth 2.0 Token Exchange with `subject_token` (the original delegator) and nested `act` claims (the actors at each hop). The architecture sits on standards, so as Agentic JWT, A2A Transaction Tokens, and Identity Chaining converge, the foundation extends naturally.
- **The threat model:** scope explosion (a chain that grants broader authority than the original delegator), attribution loss (the resource can't tell who originally authorized), chain depth attacks (deeply nested chains used to confuse policy), and cross-tenant escape (a chain that crosses tenant boundaries inappropriately).
- **The policy controls:** chain depth limits, scope-subset enforcement at each hop, audit trail preservation, cross-tenant trust boundaries explicit in policy.
- **Procurement implication:** agent identity is a multi-year architectural decision. Architectures that ignore delegation chains require rebuild; architectures that invent proprietary primitives diverge from where standards land; standards-based architectures extend.

The shape of the problem

Single-agent authentication is solvable with the patterns covered in What Is AI Agent Identity?. The hard problem is what happens when agents call each other, or when an agent calls a tool that calls another tool that touches a resource. The chain:

```
[Human]
   ↓ authenticates with phishing-resistant MFA
[Agent A] (e.g., a research agent)
   ↓ delegates a sub-task
[Agent B] (e.g., a database-query agent)
   ↓ invokes a tool
[Tool C] (e.g., a customer-records query)
   ↓ accesses
[Resource D] (e.g., the customer database)
```

At Resource D, three questions need cryptographic answers:

1. **Who originally authorized this?** Resource D's policy is "only customers, not staff, can read their own records via this path." The original human matters; the chain of agents in between is operational context.
2. **Is this specific chain allowed?** Maybe Agent A is allowed to delegate to Agent B for read scopes but not write scopes. The chain of actors matters.
3. **What scope is appropriate at this resource?** Maybe the chain has been issued tokens with `customer-records:read`, but Resource D's policy enforces additional checks (e.g., only the customer's own records, only during business hours, only from trusted networks).

Single-agent identity does not answer these. Chain-aware delegation does.

Why this is the frontier problem

Agentic systems in production today are increasingly multi-hop. A single user request to an AI assistant frequently triggers:

- A planner agent that decomposes the request into sub-tasks.
- Multiple specialist agents (search, retrieval, summarization, writing, code execution).
- Tool invocations across different MCP servers.
- Calls to backend resources that themselves may call further resources.

Each hop is an authorization decision. Each hop preserves or loses attribution. Each hop contracts or (worryingly) expands scope.

Most current production deployments handle this poorly:

1. **The user's bearer token is reused at every hop.** The agent presents the user's token to the next service, which presents it again. Audit attribution at the resource is "the user," and the entire chain has the user's full scope. Prompt-injection at any agent in the chain compromises the full session.
2. **Each hop has its own service-account credential.** Audit attribution is lost (the resource sees "agent B," not the original user). Scope is per-service-account, often broad.
3. **Custom delegation tokens.** Some teams roll their own delegation primitives. The result is parallel control planes, no compliance story, and rebuild risk as standards mature.

The standards-based pattern is materially better, and the standards exist.

RFC 8693 and the chain-aware token

RFC 8693 defines OAuth 2.0 Token Exchange. The relevant primitives:

`subject_token`: identifies the original delegator (or the most-recent subject in the chain, depending on profile). Carries the original authorization context.

`actor_token` **(and** `act` **claim)**: identifies the actor performing the request on the subject's behalf. In a chain, `act` can be nested: `act` of `act` represents Agent B acting on behalf of Agent A acting on behalf of the human.

`requested_token_type`: specifies whether the requestor wants an access token, a refresh token, or another type of artifact.

`scope`: the requested scope, which the authorization server validates against policy and constrains to a subset of what's permissible.

`audience`: where the resulting token will be presented.

A concrete chain via RFC 8693:

Step 1: Human authenticates and delegates to Agent A

The human authenticates with phishing-resistant MFA. The IDP issues an access token to Agent A:

```
{
  "iss": "https://id.scrambleid.com/",
  "sub": "human-user-12345",
  "aud": "agent-a",
  "scope": "research customer-data:read",
  "exp": 1715040300,
  "act": null,
  "cnf": { "x5t#S256": "<agent-a-cert-thumbprint>" }
}
```

Agent A receives a token where the subject is the human and Agent A is the bearer (sender-constrained via mTLS). The scope is what the human authorized.

Step 2: Agent A delegates to Agent B

Agent A wants to delegate part of the work to Agent B. Agent A calls the Token Exchange endpoint:

```
POST /oauth2/token
grant_type=urn:ietf:params:oauth:grant-type:token-exchange
subject_token=<agent-a's current token>
subject_token_type=urn:ietf:params:oauth:token-type:access_token
audience=agent-b
scope=customer-data:read
```

The IDP validates: Agent A is allowed to delegate this scope to Agent B (per policy), the scope requested is a subset of Agent A's scope, the chain depth is within bounds. It issues a new token:

```
{
  "iss": "https://id.scrambleid.com/",
  "sub": "human-user-12345",
  "aud": "agent-b",
  "scope": "customer-data:read",
  "exp": 1715040600,
  "act": {
    "sub": "agent-a"
  },
  "cnf": { "x5t#S256": "<agent-b-cert-thumbprint>" }
}
```

The new token preserves the original `sub` (the human). It adds `act` indicating Agent A as the immediate actor. The scope is contracted (only `customer-data:read`, not `research`). The audience is Agent B.

Step 3: Agent B invokes Tool C

Agent B calls the MCP server for Tool C with this token. The MCP server validates:

- Signature, audience (`agent-b`), expiration, PoP binding.
- Scope (`customer-data:read`).
- The chain: original `sub` is human-user-12345, current `act` is agent-a.

The MCP server may apply additional policy on the basis of the chain. For example: "this tool can be invoked by agents acting on behalf of the customer, but only if the chain originated from the customer themselves and not from a customer-service agent acting on behalf of the customer."

Step 4: MCP server calls Resource D

The MCP server can either present its own credentials to Resource D (Sub-pattern A in What Is MCP Server Authentication?) or perform a further Token Exchange to preserve the caller chain (Sub-pattern B). For sensitive resources, Sub-pattern B is preferred:

```
POST /oauth2/token
grant_type=urn:ietf:params:oauth:grant-type:token-exchange
subject_token=<agent-b's current token>
audience=resource-d
scope=customer-records:read-self
```

The new token issued for Resource D:

```
{
  "iss": "https://id.scrambleid.com/",
  "sub": "human-user-12345",
  "aud": "resource-d",
  "scope": "customer-records:read-self",
  "exp": 1715040900,
  "act": {
    "sub": "mcp-server-tool-c",
    "act": {
      "sub": "agent-b",
      "act": {
        "sub": "agent-a"
      }
    }
  },
  "cnf": { "x5t#S256": "<mcp-server-cert-thumbprint>" }
}
```

Resource D sees the full chain: original subject is the human, immediate actor is the MCP server, prior actors include Agents A and B. Resource D can apply granular policy on the basis of this chain.

The threat model and the policy controls

Threat 1: scope explosion

An attacker compromises Agent A and attempts to issue itself broader scope at the next hop.

Control: the Token Exchange endpoint enforces `scope_requested ⊆ scope(subject_token)`. Agent A's authorized scope at the time of exchange is the upper bound; the requested scope cannot exceed it. The IDP rejects exchange requests for scopes outside the subject token's authorization.

Threat 2: attribution loss

A chain that doesn't preserve original-delegator attribution lets a compromised intermediate agent pretend the original was something else.

Control: `subject_token` (the original delegator) is preserved across hops. `act` claims are nested, not overwritten. The full chain is cryptographically signed and tamper-evident. Resource D applies policy on the basis of the full chain.

Threat 3: chain depth attacks

Excessively deep chains are sometimes used to confuse policy engines or exhaust resources.

Control: chain depth is bounded by policy (typical depth limits 3-5). Chains exceeding max depth are rejected at the Token Exchange endpoint. This is a standard policy control in the RFC 8693 pattern, which ScrambleID's architecture follows; depth limits are a deployment policy choice that can vary by resource sensitivity.

Threat 4: cross-tenant escape

A chain that crosses tenant boundaries inappropriately can be used to escalate from one customer's environment to another.

Control: tenant-scoped JWKS, audience binding (`aud` claim is checked at every hop), and per-tenant policy enforcement. Cross-tenant federation is an explicit trust relationship, not an inadvertent one.

Threat 5: stolen intermediate token

An attacker steals an intermediate token (Agent B's token in our example).

Control: sender-constrained tokens (mTLS or DPoP) bind the token to the agent's runtime credentials. A stolen bearer token without the matching PoP material returns `invalid_token`. Short TTLs (≤ 300 seconds) further bound blast radius. Immediate admin-initiated revocation of the client stops the chain.

Threat 6: prompt injection at any chain hop

An indirect prompt injection at any agent in the chain attempts to misuse the chain.

Control: scope-per-tool at the resource boundary (covered in Prompt Injection Defense Through Identity Controls) bounds what each chain hop can do. The compromised agent cannot escalate scope beyond what was delegated. Dual control at irreversible actions catches misuse of legitimate scope; Lockstep (in development) is ScrambleID's design for enforcing it.

What this looks like in production

A practical agent platform implementing chain-aware delegation:

1. **Agent registration includes delegation policy.** Each agent declares what scopes it can delegate to which other agents. The IDP enforces this at Token Exchange time.
2. **Resource policies are chain-aware.** Resources express policy in terms of original subject, chain depth, and actor properties: "this action requires original subject of type customer, chain depth ≤ 3, all actors must have human owners."
3. **Audit is end-to-end.** The signed chain is captured at every hop. Forensic reconstruction across the chain is possible from the audit stream.

4. **Telemetry surfaces anomalies.** Unusual chain shapes (depth spikes, novel actor combinations, cross-tenant escapes) feed the SOC, and will feed the ScrambleID Overwatch risk engine (in development) once it ships.
5. **Revocation cascades.** Revoking an agent's identity revokes its outstanding tokens, which invalidates every chain that depended on it. Resources can re-validate chains in real time.

The standards landscape

The architecture sits on standards that exist today and extends naturally as standards mature:

Standard	Status	Relevance
RFC 8693 (Token Exchange)	Standard (RFC 8693, Jan 2020)	Foundation for the chain
RFC 7523 (JWT Profile)	Standard (RFC 7523, May 2015)	Client authentication at each hop
RFC 8705 (mTLS)	Standard (RFC 8705, Feb 2020)	Sender constraint
RFC 9449 (DPoP)	Standard (RFC 9449, Sep 2023)	Sender constraint where mTLS impractical
Agentic JWT (draft)	IETF draft	Adds agent-specific claims (model used, agent purpose)
A2A Transaction Token (draft)	Industry draft	Formalized agent-to-agent transaction semantics
Identity Chaining (working-group track)	Emerging	Formalized multi-hop identity propagation

Architectures sitting on RFC 8693 today extend cleanly as Agentic JWT, A2A Transaction Token, and Identity Chaining converge. The Token Exchange endpoint absorbs additional agent-specific claims without protocol break.

This is the procurement implication. A CISO evaluating agent identity today is making a multi-year architectural decision. Architectures that pretend delegation chains do not exist will require rebuild. Architectures that invent proprietary primitives will diverge from where the standards land. Standards-based architectures extend.

Anti-patterns

1. **Reusing the user's bearer token at every hop.** No chain. Full user scope at every hop. Prompt injection at any agent compromises the full session.
2. **Each hop has its own service-account credential, with no chain.** Attribution loss. Scope is per-service-account, broad.

3. **Custom proprietary delegation tokens.** Parallel control plane. No compliance story. Rebuild risk.
4. **No chain depth limits.** Vulnerable to chain-explosion attacks.
5. **Scope can expand at exchange.** Scope explosion threat materializes.
6. **No `act` chain preservation.** Attribution loss.
7. **No revocation cascade.** Revoking an agent doesn't invalidate dependent chains.
8. **Cross-tenant federation by default.** Enables cross-tenant escape.

Operational and standards considerations

Implementation maturity. RFC 8693 is well-implemented in modern OAuth 2.0 authorization servers. Adoption in production agent systems is growing but still uneven. Vendor-side compatibility varies; verify Token Exchange support specifically (some IDPs implement only the basic profile).

Performance. Token Exchange adds latency at every hop. Cached JWKS, short-lived tokens, and resource-side validation optimization all matter. ScrambleID's architecture is built for low-latency issuance and validation of short-lived credentials.

Compliance posture. Chain-aware delegation supports audit posture for SOC 2, ISO 27001, FedRAMP, and emerging AI governance frameworks (NIST AI RMF, EU AI Act). The audit chain is the compliance evidence.

Standards convergence. Stay close to IETF and IETF-adjacent draft work on Agentic JWT, A2A Transaction Token, and Identity Chaining. The ScrambleID architectural commitment is to track these standards rather than invent proprietary alternatives.

Standards alignment

Standard	Relevance
RFC 8693	Foundation of chain-aware tokens
RFC 7523	Client authentication at each hop
RFC 8705	Sender constraint via mTLS
RFC 9449	Sender constraint via DPoP
Agentic JWT (draft)	Future extension for agent-specific claims
A2A Transaction Token (draft)	Future formalization of agent-to-agent semantics
OAuth 2.0 / OIDC	Authorization-server foundation
OWASP API Security Top 10	BOLA/BFLA defenses applied at the chain
NIST SP 800-207	Zero Trust applied to agent chains

Key Takeaway

Multi-hop agent delegation is the pattern where authorization propagates across a chain (Human → Agent A → Agent B → Tool C → Resource D), with each hop carrying its own identity, the original delegator preserved, and scope contracting at each hop. The standards-based foundation is RFC 8693 OAuth 2.0 Token Exchange, which provides `subject_token` (original delegator) and nested `act` (chain of actors) semantics. Each hop authenticates via JWT client assertion (RFC 7523) and presents sender-constrained tokens (RFC 8705 mTLS or RFC 9449 DPoP). Policy controls bound the chain: scope-subset at every exchange (no scope explosion), chain depth limits (typically 3-5), audience binding at every hop, tenant-scoped trust boundaries, and immediate revocation cascade. Architectures sitting on RFC 8693 today extend naturally as Agentic JWT, A2A Transaction Token, and Identity Chaining mature, without requiring rebuild. The buyer-side argument: agent identity is a multi-year decision; architectures that ignore delegation chains, reuse user bearer tokens, or invent proprietary primitives will rebuild later. Standards-based foundations extend.

FAQ

What is multi-hop agent delegation?

Multi-hop agent delegation is the pattern where authorization propagates across a chain: Human → Agent A → Agent B → Tool C → Resource D. The original human authenticates and delegates to Agent A. Agent A delegates to Agent B for a sub-task. Agent B invokes Tool C. Tool C calls Resource D. At each hop, the resource needs to know who originally authorized this, whether this specific actor is allowed to act in this chain, and what scope of access is appropriate at this resource. Single-agent authentication does not solve this; chain-aware delegation does.

Why is delegation across agent chains hard?

Three reasons. First, traditional OAuth assumes a single human-to-application delegation. Multi-hop chains have multiple actors and the delegation context must be preserved across hops. Second, scope must contract (or at least not expand) at each hop; without this property, an attacker who compromises any agent in the chain could escalate to the full original scope. Third, audit attribution must travel with the chain so the resource can apply policy on the basis of the original delegator, not just the immediate caller.

What is RFC 8693 and how does it help?

RFC 8693 is the OAuth 2.0 Token Exchange standard. It provides the `subject_token` and nested `act` claim semantics needed for delegation chains. When Agent A delegates to Agent B, Agent A's authorization server exchanges Agent A's token for a new token where `subject_token` identifies the original human, `act` identifies Agent A as the actor, and the scope is constrained to Agent A's

authorized delegation. Resource D receives the chain in the token and can apply policy: "the original human is X, the chain went through agents A and B, and only this combination is authorized to perform this action." RFC 8693 is the standards-based foundation for the entire pattern.

What about emerging standards like Agentic JWT and A2A Transaction Tokens?

These are draft IETF specifications that build on RFC 8693 to address agent-specific delegation patterns. Agentic JWT proposes additional claims for agent context (the LLM model used, the agent's purpose, the specific user delegation context). A2A Transaction Token profiles propose specific patterns for agent-to-agent calls with formalized chain-of-actors semantics. Identity Chaining is an emerging working-group track formalizing the multi-hop semantics. The architecture should be standards-based: implementations sitting on RFC 8693 can extend naturally as these later specifications converge. Architectures that invent proprietary primitives now will require rebuild later.

How do you bound chain depth and scope explosion?

Two policy controls. First, chain depth is bounded by policy: chains exceeding maximum depth are rejected at the token exchange endpoint. Typical depth limits are 3-5; deeper chains are rare in legitimate scenarios and are common in attempted scope-escalation attacks. Second, scope at each hop must be a subset of the prior hop's scope. Agent B cannot have broader scope than Agent A delegated, and Agent A cannot have broader scope than the human authorized. The token exchange endpoint enforces both properties at issuance time; runtime checks at the resource enforce them at consumption time.

What if part of the chain is a third-party agent?

Cross-organization delegation is supported via federated trust between IDPs. The third-party agent's IDP issues a token signed by its issuer; the receiving IDP validates against the federation trust relationship and applies tenant-scoped policy. This is OAuth 2.0 federation extended to multi-hop scenarios. The receiving organization can require specific properties from the third-party agent's identity (owner, attestation, scope) and reject chains that don't meet policy. The Token Exchange flow continues to work; it just crosses an organizational trust boundary.

References (public)

- RFC 8693 (OAuth 2.0 Token Exchange): https://www.rfc-editor.org/rfc/rfc8693.html
- RFC 7523 (JWT Profile for OAuth 2.0): https://datatracker.ietf.org/doc/html/rfc7523
- RFC 8705 (OAuth 2.0 Mutual-TLS): https://datatracker.ietf.org/doc/html/rfc8705
- RFC 9449 (OAuth 2.0 DPoP): https://www.rfc-editor.org/rfc/rfc9449.html
- RFC 9700 (OAuth 2.0 Security BCP): https://www.rfc-editor.org/rfc/rfc9700.html
- OpenID Connect Core: https://openid.net/specs/openid-connect-core-1_0.html

- NIST SP 800-207 (Zero Trust): https://csrc.nist.gov/publications/detail/sp/800-207/final

Related reading

- What Is AI Agent Identity?
- What Is MCP Server Authentication?
- Prompt Injection Defense Through Identity Controls
- What Is Non-Human Identity (NHI)?
- AI Agent Authentication
- AI Agent Tool Access Playbook
- M2M Authentication Without Secrets
- Sender-Constrained Tokens (mTLS, DPoP)