# What Is AI Agent Identity? Why Agents Need the Discipline Service Accounts Never Had

Definitions / Last updated 2026-06-11 / https://www.scrambleid.com/learn/what-is-ai-agent-identity

**In one sentence:** AI agent identity is a class of non-human identity for autonomous and semi-autonomous AI systems that comprises a hardware-bound cryptographic key, an explicit human or team owner, short-lived sender-constrained access tokens, and an immutable audit trail, designed to give an AI agent the same identity discipline that workforce humans get and the same revocation, attribution, and least-privilege properties that traditional service accounts conspicuously fail to provide.

## TL;DR (canonical)

- **An AI agent is not a service account.** Service accounts assume predictable behavior. AI agents make runtime authorization-relevant decisions. The control architecture has to reflect that.
- **The five properties of a properly-architected agent identity:**
  1. **Hardware-bound asymmetric key pair** in TPM, HSM, or OS keystore. No client secrets.
  2. **Explicit owner mapping** (human or team) at registration. No orphaned agents.
  3. **Short-lived sender-constrained tokens** (≤ 300 second TTL, bound via mTLS or DPoP). No refresh tokens for non-human clients.
  4. **Immutable audit** of every issuance, validation, and revocation event.
  5. **Standards-based foundation** (RFC 7523, RFC 8705, RFC 9449, RFC 8693) so the architecture extends naturally as the standards landscape matures.
- **The architecture is built on the M2M control plane** (What Is M2M Authentication?) with agent-specific governance and telemetry layered on top.
- **The bot-versus-AI distinction is preserved at the auth surface.** Headless services and most AI agents use the M2M flow. UI-driven RPA bots and the rare AI agent that operates a human login screen use the ScrambleID Web flow with an automation account governed exactly like a human identity. We don't pretend a UI-driven bot is a backend service.

## Why AI agents need their own identity story

For two decades, machine-to-machine authentication was a back-office concern. Backend services exchanged shared secrets, stored them in vaults, and called it done. The model worked because machines were predictable, did what they were programmed to do, and had stable access patterns.

AI agents broke that model.

An AI agent makes decisions at runtime about what to call, when to call it, and on whose behalf. The agent might compose new tool sequences that no human anticipated. It might delegate to other agents. It might encounter an authorization boundary it cannot articulate in advance. And it might do all of this hundreds or thousands of times per second.

Treating an AI agent like a traditional service account fails for at least four reasons:

1. **Static credentials are an unbounded liability.** A leaked API key for an AI agent gives the attacker unbounded autonomous capability, not just access to a single API.
2. **Shared service accounts destroy attribution.** If multiple agents share a credential, you cannot tell which agent took which action. Forensics is impossible.
3. **Long-lived tokens have unbounded blast radius.** A stolen long-lived token for an AI agent means the attacker can run an autonomous workload on the victim's behalf, possibly for days or weeks before detection.
4. **Human-on-behalf-of OAuth semantics break down.** Traditional OAuth assumes a human delegated to an application. With agent chains, the question is which human, with which assurance level, authorized which agent, which then delegated to which other agent, which then called which tool. Every hop deserves an answer.

The architectural choice that flows from these failure modes is straightforward: AI agents must have cryptographic, revocable, attributed identity that mirrors the discipline applied to human identity.

## The five properties in detail

### 1. Hardware-bound asymmetric key pair

Every agent has its own asymmetric key pair. The private key is generated and held in hardware-backed storage on the agent's runtime environment: TPM (Windows servers, modern Linux), HSM (high-security deployments, cloud KMS-backed signing), OS keystore (macOS Secure Enclave, modern Linux key-protected storage). The public key is registered with the identity provider. There is no shared secret to leak, by construction.

The cryptographic algorithms supported are standard: ES256 and RS256 for JWT signing, with curves and parameters per JOSE specifications. The key-pair generation happens at agent

provisioning; the private key never leaves the hardware-backed environment, and a compromised agent runtime cannot exfiltrate it.

### 2. Explicit owner mapping at registration

Every M2M client identity (including every agent) must map to a human owner or team. Lifecycle APIs enforce ownership at registration time. Orphaned service accounts, the historical bane of identity governance, cannot exist by construction.

Concretely, the registration API requires an `ownerId` (typically a user identity or team identity in the IDP). Without it, registration is rejected. Owner coverage is tracked as a KPI; target is 100%. Reports surface clients without recent owner attestation, allowing periodic recertification.

This matters because: the audit trail tying an agent's actions back to "the team that built and operates it" is essential for forensics, for capacity planning, and for the operational question "should this still exist?"

### 3. Short-lived sender-constrained tokens

The default access token TTL for non-human clients is ≤ 300 seconds. Refresh tokens are not issued to non-human clients. The cost of a stolen token is bounded to minutes, and the agent re-authenticates from its hardware-bound private key for every new TTL window.

Tokens are sender-constrained, meaning they can only be used by the original requester:

- **mTLS binding** (RFC 8705) using the `cnf` claim with `x5t#S256` (the thumbprint of the client's TLS certificate). The resource server validates the incoming TLS client certificate against this thumbprint at every request.
- **DPoP binding** (RFC 9449) for app-layer proof of possession where mTLS is impractical. The client signs each request with a key whose thumbprint is in the token's `cnf.jkt`.

A stolen bearer token without the matching mTLS client certificate or DPoP key cannot be used at the resource server. The token returns `invalid_token` with a `cnf` mismatch error.

### 4. Immutable audit

Every authorization event is captured in an append-only audit stream:

- Registration of the agent (who, when, owner)
- Token issuance (when, scope, audience, PoP binding mode)
- Token validation (when, resource, scope used, PoP verification result)
- Key operations (rotation, addition, revocation)
- Revocation events (when, why, by whom)

Logs are tenant-isolated and retained per organizational policy. The audit trail is the basis for incident response, compliance attestation, and the operational question "is this agent doing what we expected it to do?"

ScrambleID Overwatch (in development) defines a metric, A11 (JWT Replay Attempts), designed to track distinct `jti` collisions per client per 24 hours and surface high-severity alerts; this is one example of how the audit stream is designed to feed operational risk telemetry.

### 5. Standards-based foundation

The architecture sits on:

- **RFC 7523**: JWT profile for OAuth 2.0 client authentication and authorization grants. The `client_assertion` mechanism replaces shared client secrets.
- **RFC 8705**: Mutual-TLS client authentication and certificate-bound access tokens. The transport-layer foundation for sender constraint.
- **RFC 9449**: DPoP, app-layer proof of possession. The mTLS-impractical alternative.
- **RFC 8693**: OAuth 2.0 Token Exchange. The `subject_token` and nested `act` claim semantics needed for delegation chains.
- **OAuth 2.0 / OIDC** for the authorization-server core: token endpoint, JWKS distribution, introspection, well-known configuration.
- **OWASP API Security Top 10** as the reference for BOLA/BOA defenses via scope and PoP.

The architecture intentionally does not invent proprietary primitives. As IETF standards converge on agentic identity (Agentic JWT, A2A Transaction Token profile, Identity Chaining), the existing foundation extends without disruption.

---

## What an agent identity is not

**Not a service account with a chatbot in front.** A traditional service account uses a shared client secret, has no owner attestation, issues long-lived tokens, and lacks per-action attribution. Calling that arrangement "agent identity" is a marketing exercise, not an architecture.

**Not a user-on-behalf bearer token.** When an agent reuses the user's bearer token at full scope, prompt-injection or tool misuse becomes a full account-takeover-equivalent compromise. The agent must have its own identity distinct from the user delegating to it. (See Prompt Injection Defense Through Identity Controls.)

**Not a single shared agent identity for a fleet.** Each agent instance gets its own identity. Shared agent identities destroy attribution.

**Not a hardcoded credential in code.** ScrambleID agents do not have a credential to hardcode. The asymmetric key is generated in hardware-backed storage at agent provisioning time. There is nothing to commit to a public repository.

**Not a workaround for human authentication.** UI-driven RPA bots that operate a human login screen authenticate via the ScrambleID Web flow (QR DID or WebAuthn) using an automation account. We

don't pretend a UI-driven bot is a backend service. We give it the right primitive for its actual behavior.

## How agent identity differs from human identity

The architecture preserves the principle that non-human and human identity get the same discipline. The differences are in the surface, not the principles:

| Property | Human identity | AI agent identity |
|---|---|---|
| Key location | Apple Secure Enclave / Android StrongBox / Windows TPM | TPM / HSM / OS keystore on agent runtime |
| Authentication ceremony | WebAuthn UV (biometric, PIN) | JWT client assertion (RFC 7523) |
| Token TTL | Configurable, often hours | ≤ 300 seconds default |
| Refresh tokens | Permitted | Not issued to non-human clients |
| Sender constraint | WebAuthn origin binding | mTLS or DPoP ( `cnf` claim) |
| Owner | Self | Mapped human or team |
| Audit | Per-event signed | Per-event signed |
| Revocation | User-initiated or admin | Admin-initiated, immediate |
| Recovery | Identity proofing + new device | Re-provision in hardware-backed storage |

The same cryptographic foundation, applied to different surfaces, with different operational properties.

## How agent identity differs from traditional service accounts

| Property | Traditional service account | AI agent identity |
|---|---|---|
| Credential type | Long-lived shared secret | Hardware-bound asymmetric key |
| Storage | Vault, env var, config file | Hardware-backed (TPM, HSM, keystore) |
| Ownership | Often "the team that ran it" (informal) | Mandatory at registration; KPI-tracked |
| Token TTL | Often hours to days | ≤ 300 seconds |
| Refresh | Common | Not issued |
| Sender constraint | Rare | Default (mTLS or DPoP) |
| Audit per-action | Optional | Mandatory |

| Property | Traditional service account | AI agent identity |
|---|---|---|
| Revocation | Manual, often slow | Immediate via API |
| Failure mode if credential leaks | Indefinite attacker access | Bounded to TTL plus revocation latency |

## Where agent identity sits in the broader ScrambleID architecture

Agent identity is delivered through the M2M control plane (see M2M Authentication Without Secrets and Sender-Constrained Tokens (mTLS, DPoP)), with three agent-specific layers added:

1. **Agent governance.** Owner mapping is enforced. Agent inventory is queryable by team. Compliance attestations are surfaced per-owner.
2. **Agent-specific telemetry.** KPIs include static-credential surface shrink (target ≥ 95% within 90 days), low-latency token issuance and validation, PoP failure rate (any > 0 alerted, > 0.05% critical), JWT replay attempts (Overwatch metric A11, in development), mean time to rotate keys (≤ 24 hours), owner coverage (target 100%).
3. **Multi-hop delegation primitives.** RFC 8693 Token Exchange with `subject_token` and nested `act` claims provides the foundation for Human → Agent A → Agent B → Tool C chains. (See Multi-Hop Agent Delegation Chains.)

For specific scenarios:

- **Service-to-service API call:** internal workload obtains a token via JWT client assertion, calls a target service, resource server validates audience, issuer, expiration, and PoP binding.
- **AI agent calling a tools API:** agent process signs a JWT, requests a scoped token (e.g., `tools:read`), invokes the API with proof-of-possession. Telemetry ties the agent to the owning team. Scope enforcement is verified.
- **Cross-organization machine trust:** partner registers a key. Tokens are limited to the partner audience. PoP enforced at the gateway.
- **Batch job in air-gapped environment:** client agent and server agent exchange a Dynamic Identifier within a time window. The verifier posts the DID to ScrambleID. ScrambleID confirms the match and authorizes the action out-of-band. Patent-protected pattern (US 12,388,656 B2).

## Standards alignment

| Standard | Relevance |
|---|---|
| RFC 7523 | JWT client assertion for agent-to-AS authentication |
| RFC 8705 | mTLS for sender-constrained tokens |
| RFC 9449 | DPoP for app-layer proof of possession |

| Standard | Relevance |
| --- | --- |
| RFC 8693 | Token Exchange foundation for multi-hop delegation |
| RFC 9700 | OAuth 2.0 Security Best Current Practice |
| NIST SP 800-207 | Zero Trust Architecture: strong identity, continuous evaluation, least privilege, short-lived tokens |
| OWASP API Security Top 10 | BOLA/BOA defenses via scope and PoP |
| SOC 2 / ISO/IEC 27001 | Operational controls for key management, logging, access reviews |

## Key Takeaway

AI agent identity is a class of non-human identity for autonomous and semi-autonomous AI systems, comprising five properties: a hardware-bound asymmetric key pair (TPM, HSM, or OS keystore) with no shared client secret; explicit human or team owner mapping enforced at registration; short-lived sender-constrained access tokens (≤ 300 second TTL, bound via mTLS per RFC 8705 or DPoP per RFC 9449) with no refresh tokens for non-human clients; immutable audit of every issuance, validation, and revocation event; and a standards-based foundation (RFC 7523 JWT client assertions, RFC 8705 mTLS, RFC 9449 DPoP, RFC 8693 Token Exchange) that extends naturally as agentic-identity standards mature. AI agents cannot be authenticated like traditional service accounts because they make runtime decisions, compose new tool sequences, and operate at high frequency; static credentials are unbounded liability, shared service accounts destroy attribution, and long-lived tokens have unbounded blast radius. The architecture is delivered through the M2M control plane with agent-specific governance and telemetry layered on top.

## FAQ

### What is AI agent identity?

AI agent identity is a class of non-human identity for autonomous and semi-autonomous AI systems that make runtime decisions about what tools to call, when, and on whose behalf. A properly-architected agent identity comprises an asymmetric key pair generated and stored in hardware-backed storage on the agent's runtime (TPM, HSM, or OS keystore), a public-key registration with the identity provider, an explicit human or team owner, short-lived sender-constrained access tokens (≤ 300 second TTL, bound via mTLS per RFC 8705 or DPoP per RFC 9449), and an immutable audit trail of every issuance, validation, and revocation event.

### Why can't AI agents use traditional service accounts?

Service accounts assume predictable, deterministic behavior. AI agents make runtime decisions, compose new tool sequences, delegate to other agents, and operate at hundreds-to-thousands of actions per second. Traditional service-account patterns fail because: static credentials are an unbounded liability (a leaked API key gives the attacker autonomous capability), shared service accounts destroy attribution (which agent took which action?), long-lived tokens have unbounded blast radius (days or weeks of attacker dwell time), and human-on-behalf-of OAuth semantics break down across multi-agent chains.

### Is an AI agent the same thing as a service account?

Both are non-human identities (NHI), but the threat models and operational characteristics differ. Service accounts represent specific predictable workloads and traditionally use long-lived shared secrets. AI agents make runtime authorization-relevant decisions and require ephemeral, sender-constrained, audit-rich identity. The architectural pattern that fits AI agents (no client secrets, short TTLs, hardware-bound keys, sender-constrained tokens, mandatory ownership, immutable audit) is also the right pattern for modern service accounts, which is why the two converge on the same control plane. See What Is Non-Human Identity (NHI)?.

### What's a "sender-constrained" token and why does it matter for agents?

A sender-constrained token is an access token that is only valid when presented by the original requester, as proven by a separate cryptographic operation. ScrambleID supports two binding methods: TLS client certificate (mTLS, RFC 8705) using `cnf` claim with `x5t#S256` thumbprint, or DPoP (RFC 9449) for app-layer proof of possession where mTLS is impractical. A stolen bearer token is useless without the corresponding proof-of-possession material. For AI agents, this matters because token theft from agent runtimes is a real attack surface; sender constraint bounds the blast radius.

### Why does every agent need an owner?

Without ownership mapping, an agent identity becomes orphaned. Nobody knows who is responsible. Nobody can attest to whether the agent should still exist. Nobody is accountable for the agent's actions. ScrambleID enforces ownership at registration time: every M2M client (including every agent) must map to a human owner or team. Lifecycle APIs prevent registration without ownership. Owner coverage is a tracked KPI; target is 100%.

### How long should an agent's tokens last?

Default access token TTL for non-human clients in ScrambleID is ≤ 300 seconds. Refresh tokens are not issued to non-human clients. The cost of a stolen token is bounded to one TTL window, and the agent re-authenticates from its hardware-bound private key for every new TTL window. This is a deliberate departure from human-user OAuth patterns where refresh tokens enable long-lived

sessions; for non-human identities the long-lived session is exactly what creates unbounded blast radius.

### What happens if the agent's runtime is compromised?

If an agent's runtime is compromised, the attacker still cannot extract the private key from hardware-backed storage. The attacker can use the agent's identity for the duration of the compromise, but cannot exfiltrate the credential, and revocation immediately stops authorization. Compare this to a stolen API key, which the attacker can copy and use indefinitely from anywhere on the internet. The agent-identity model fails closed; the API-key model fails open.

---

## References (public)


- RFC 7523 (JWT Profile for OAuth 2.0 Client Authentication): 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 8693 (OAuth 2.0 Token Exchange): https://www.rfc-editor.org/rfc/rfc8693.html
- RFC 9700 (OAuth 2.0 Security BCP): https://www.rfc-editor.org/rfc/rfc9700.html
- NIST SP 800-207 (Zero Trust): https://csrc.nist.gov/publications/detail/sp/800-207/final
- OWASP API Security Top 10: https://owasp.org/API-Security/editions/2023/en/0x00-introduction/


---

## Related reading


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