Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: October 6, 2026

R. Sharif
CyberSecAI Ltd

April 6, 2026

Agent Identity Framework: Trust and Identity for Autonomous AI Agents
draft-sharif-agent-identity-framework-00

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 October 6, 2026.

Copyright Notice

Abstract

Autonomous artificial intelligence (AI) agents are increasingly performing actions that were previously the exclusive domain of authenticated human users: initiating financial transactions, querying regulated data, invoking external tools, and coordinating with other agents. Internet protocols designed for human-operated clients lack primitives to answer three fundamental questions about any autonomous action: which agent performed it, whether the agent was authorized to perform it, and whether the resulting evidence is independently verifiable.

This document defines a framework for agent identity and trust enforcement on the Internet. It enumerates the gaps between current Internet standards and the requirements of autonomous agent systems, introduces a five-layer model (identity, authorization, attestation, evidence, trust) that separates concerns that are currently conflated, and outlines mechanisms to close specific gaps. The framework is intended to guide future Standards Track work and to provide a common vocabulary for researchers, implementers, and regulators.

This document is informational. It does not define a wire protocol. It references existing Internet-Drafts and specifications that instantiate individual mechanisms within the framework.

1. Introduction

1.1. The Emergence of Autonomous Agents

Large language models (LLMs) and related machine learning systems have made it practical to deploy software agents that operate with minimal human supervision. A modern autonomous agent is typically composed of a language model that determines what action to take next, a tool invocation layer that enables the model to execute external operations, and a scheduling or orchestration layer that runs the agent in a continuous loop. Agents of this form can hold long-running sessions with multiple services, initiate network connections to arbitrary endpoints, make financial transactions, read and write persistent data, and spawn or delegate to other agents.

The protocols that carry agent traffic on the Internet are the same protocols developed for human users of web applications: HTTP [RFC9110], OAuth 2.0 [RFC6749], OpenID Connect, and related mechanisms. These protocols assume a human user behind a browser or mobile application, authenticating interactively at the start of a session and supervising the actions that follow. Autonomous agents violate every assumption in that model. They are not human. They do not authenticate interactively. They do not supervise themselves.

This document does not debate whether autonomous agents should exist. They exist, they are deployed at scale, and they are already performing consequential actions. This document addresses the narrower question of what identity and trust primitives the Internet requires to make agent actions safe, auditable, and accountable.

1.2. Scope

This document is a framework. It enumerates problems, organises them into layers, and describes mechanisms that close specific gaps. It does not define a wire protocol. Individual mechanisms referenced in this document are specified in separate documents, some of which are also referenced here as work in progress.

This document does not propose a new identity system to replace OAuth 2.0, OpenID Connect, or FAPI [FAPI2]. It proposes a layer of agent-specific identity and trust primitives that compose with existing standards. Where existing standards are adequate, this document says so. Where they are inadequate, this document describes the gap and outlines how it may be addressed.

This document is not specific to any AI model architecture, agent framework, or vendor. The framework applies equally to agents built on LLMs, reinforcement learning systems, symbolic AI, or hybrid architectures.

1.3. Requirements Language

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.

1.4. Document Structure

Section terminology defines terms used throughout this document.

Section problem states the agent identity problem and enumerates gaps between current Internet standards and the requirements of autonomous agent systems.

Section framework introduces the five-layer framework: identity, authorization, attestation, evidence, and trust.

Sections Section identity-layer through Section trust-layer describe each layer in detail.

Sections Section mechanism-agent-identity-credentials through Section mechanism-compliance-evidence-export describe specific mechanisms that close individual gaps.

Section threat-model develops a threat model for autonomous agents.

Section security-considerations discusses security considerations.

Section privacy-considerations discusses privacy considerations.

Section integration describes how the framework integrates with existing Internet standards.

2. Terminology

For the purposes of this document, the following terms apply. These definitions are scoped to the agent identity and trust domain. Where a term is also defined in an existing RFC, the meaning given here is additive, not contradictory.

Autonomous Agent (or "Agent"):
: A software system that executes tasks on behalf of a principal without continuous interactive human supervision for each action. An agent typically consists of a decision-making component (which may be a machine learning model), a tool invocation layer, and a persistence or scheduling layer. Agents are distinguished from ordinary software clients by their capacity to decide, within policy constraints, which action to take next.

Principal:
: The human, organisation, or service on whose behalf an agent acts. A principal authorises an agent to perform actions within a defined scope. Delegation from principal to agent is the foundation of accountability.

Agent Identity:
: A cryptographically verifiable credential that uniquely identifies a specific agent instance. Agent identity is non-transferable, non-repudiable within its validity period, and distinct from the identity of the principal.

Agent Instance:
: A specific running instance of an agent, distinguished from the agent implementation or the agent template. Two instances of the same implementation, running at the same time, have distinct agent identities.

Agent Passport:
: A self-contained, signed credential that binds an agent identity to its public key, trust level, issuing authority, and lifecycle state. An agent passport is designed to be verified offline by any party holding the issuing authority’s public key.

Issuing Authority:
: An entity that issues agent passports and vouches for the identity of the agents to which it issues credentials. An issuing authority may be internal to an organisation (a private trust root), a commercial service, or a public infrastructure. This document does not mandate a single issuing authority model.

Trust Level:
: A discrete classification of the assurance an issuing authority can provide about an agent identity. This document defines a five-level hierarchy (L0 through L4) ranging from unsigned to fully audited, described in Section mechanism-trust-levels.

Trust Gate:
: A decision point in an agent-to-service interaction where the service verifies the agent’s passport, checks its trust level, evaluates policy, and either allows or denies the requested operation before execution. The Trust Gate is a complement to, not a replacement for, traditional authorisation checks.

Per-Operation Signing:
: Cryptographic signing applied to every individual agent operation, rather than once per session or once per client authentication. Each operation carries its own signature, nonce, and timestamp, making every action independently verifiable and non-repudiable.

Delegation Chain:
: The ordered sequence of identities that authorise a specific action. For example: Principal (human user) -> Agent A (primary agent) -> Agent B (sub-agent invoked by Agent A) -> Target Service. Each link in the chain represents an authorisation grant.

Tamper-Evident Audit Trail:
: A sequence of log records where each record cryptographically references the hash of the previous record, such that undetected modification of any earlier record is computationally infeasible. See also Certificate Transparency [RFC6962] for a related construction applied to TLS certificates.

Attestation:
: A signed assertion by some party about a state of affairs. In this document, attestations are issued about agent identities, agent code, agent trust levels, and agent actions. Attestation is distinguished from evidence (below): attestation is a claim, evidence is an artifact that can be verified independently of the claim.

Evidence:
: A cryptographically verifiable artifact that records a specific state of affairs. Evidence is verifiable without reference to the identity or trustworthiness of the party that produced it. For example, a SHA-256 hash of a file is evidence of the file's contents; it does not depend on who computed the hash.

Trust:
: A policy decision by one party to accept the claims or actions of another. Trust is distinct from identity (which is what a party is) and from attestation (which is what a party says about itself or another party). Trust incorporates historical behaviour, peer attestations, and policy.

MCP:
: Model Context Protocol, an open protocol for connecting large language models to external tools, data sources, and services. MCP defines a JSON-RPC 2.0 message format carried over HTTP or stdio. MCP does not define cryptographic identity, signing, or trust enforcement; those are addressed by the mechanisms referenced in this document.

MCPS:
: Model Context Protocol Secure, a cryptographic security profile for MCP that adds per-message signing, passport-based identity, replay protection, trust levels, and tool integrity verification to the baseline MCP wire format [MCPS].

3. The Agent Identity Problem

3.1. Why Existing Standards Are Insufficient

The Internet has mature identity and authorization standards. OAuth 2.0 [RFC6749] handles delegated authorization for client applications. OpenID Connect extends OAuth for user authentication. JSON Web Tokens (JWTs) [RFC7519] provide a format for conveying claims. FAPI 2.0 [FAPI2] profiles these for high-assurance use cases such as banking. Mutual TLS [RFC8705] provides sender-constrained tokens. These standards are well understood, widely deployed, and suitable for their intended purpose.

Their intended purpose does not include autonomous agents.

The gap can be stated precisely. Existing identity standards answer the question "is this client authorised to make this request on behalf of this user?" They do not answer the question "is this specific autonomous agent, acting within its policy envelope, authorised to take this specific action, and can the action be independently verified after the fact?"

The difference is not semantic. It is structural. OAuth's model assumes a known, pre-registered client. An autonomous agent is not a pre-registered client; it is software running inside a pre-registered client, making decisions that the pre-registered client did not explicitly enumerate. OAuth's scopes are pre-declared at registration time. An agent's actions are selected at runtime by a language model. OAuth's audit trail identifies the client, not the agent instance. OAuth's revocation affects the client, not individual agents running under it.

In short: OAuth authenticates the container. It does not authenticate what runs inside the container and makes decisions.

Similar gaps exist throughout the identity stack. TLS client certificates identify the machine, not the process. API keys identify the application, not the runtime instance. Session cookies [RFC6265] bind to a browser, not to an autonomous decision loop. Every mechanism in current use assumes that the decision to take an action is made by a human, supervised by a human, or at least authorised in advance by a human for a narrow class of actions. None of these assumptions hold for autonomous agents.

3.2. Why This Problem Cannot Be Solved at the Application Layer

A natural objection to this framing is that agent identity should be handled by agent developers in application code. Each agent can sign its actions, maintain its own audit log, and self-attest its trust level. No new standards are required.

This objection is incorrect for the same reasons that TLS could not have been solved by each application implementing its own encryption. There are four specific failure modes.

First, inconsistency. If agent identity is handled at the application layer, every agent developer makes independent choices about signing algorithms, key rotation, revocation mechanisms, and audit formats. Interoperability is impossible. A service that receives a signed action from an agent built on framework A cannot verify an action from an agent built on framework B, because the signature formats and verification paths differ.

Second, adoption. Application-layer solutions depend on developer action. Every existing application-layer security initiative that required developer action has had inconsistent adoption: CSRF tokens, subresource integrity, security headers, content security policies. HTTPS became near-universal only when the platform (web browsers, cloud providers, CAs issuing free certificates) enforced it. Agent identity will follow the same path. Platform-level enforcement is the only mechanism that achieves consistent adoption.

Third, trust root. Application-layer solutions do not provide a shared trust root. An agent that signs its actions with a key it generated at runtime is making a claim that the verifying party has no basis to accept. A trust root establishes the chain from agent identity to a verifiable issuing authority. Trust roots cannot be bootstrapped purely in application code.

Fourth, audit. Application-layer audit is not tamper-evident by default. An agent that maintains its own audit log can be modified to write whatever log entries serve the agent operator. Only cryptographic chaining (references to prior records) and external anchoring provide tamper evidence. These primitives must be consistent across agents to be useful.

3.3. Quantifying the Problem

As of early 2026, the Model Context Protocol ecosystem comprises thousands of publicly addressable MCP servers. A survey of 1,900+ indexed MCP servers conducted by the author in April 2026 found that 99.4 percent of these servers implement no cryptographic identity verification, no message signing, and no replay protection. Agents connecting to these servers have no means of verifying that a response is genuine, that a tool definition has not been tampered with, or that a server is the one it claims to be.

During the same period, the author filed six Common Vulnerabilities and Exposures (CVEs) against components of the MCP ecosystem, covering resource exhaustion, unbounded memory allocation, and related issues affecting packages with tens of millions of weekly downloads. These are not exceptional findings. They reflect the state of an ecosystem that has grown faster than its security primitives.

The quantitative point is not that the MCP ecosystem is uniquely insecure. The quantitative point is that the ecosystem has adopted the conventions of unauthenticated HTTP services because the alternative (developers implementing cryptographic identity independently at the application layer) is not feasible at scale.

3.4. The Cost of Inaction

The cost of the status quo is not limited to the MCP ecosystem. Autonomous agents are being deployed in financial services, healthcare, government, and critical infrastructure. Each deployment extends the attack surface of the organisation that hosts it, in ways that existing identity and authorization systems were not designed to contain.

Specific costs include:

Financial exposure: an agent making unauthorised payments cannot be halted mid-transaction by revoking a user session, because the agent does not depend on a user session for authorisation.

Regulatory exposure: compliance frameworks such as the EU AI Act [EU-AI-Act] (Articles 12, 13, 14, 15, and 50) require logging, transparency, human oversight, accuracy, and identification for AI systems. Current logging infrastructure does not distinguish between actions taken by an agent and actions taken by the user on whose behalf the agent is running. This undermines the audit requirements that regulators rely on.

Forensic exposure: when an incident occurs involving an agent, investigators must reconstruct what the agent did, when, on whose behalf, and with what authorisation. Current logs answer these questions for human users. They do not answer them for agents.

Systemic risk: as agents proliferate and begin invoking other agents, cascades of automated action can propagate errors or attacks faster than human operators can respond. Without identity and trust primitives that work at agent granularity, containment is impossible.

The cost of inaction is not measured in a single incident. It is measured in the progressive loss of the ability to answer, after any incident, the question of what happened and who was responsible.

4. Framework Overview: Five Layers of Agent Trust

4.1. Separation of Concerns

Current discussions of agent security often conflate concerns that should be distinct. This document separates them into five layers.

The layers form a stack. Each layer depends on the ones below it and provides services to the ones above it. The figure below shows the stack and the question each layer answers.

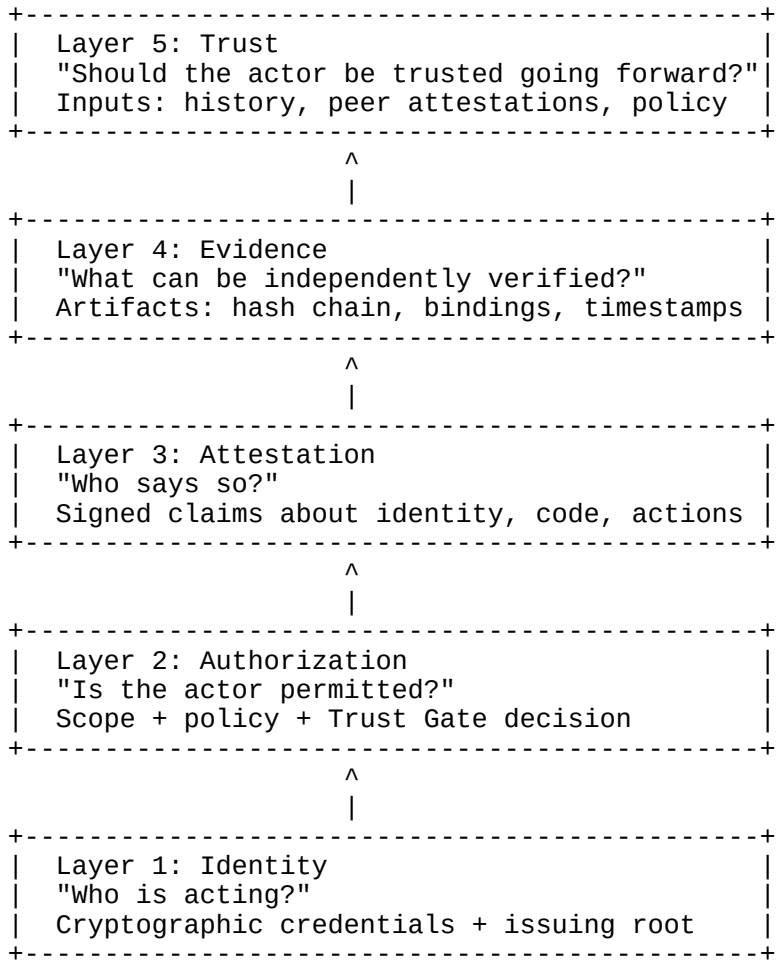


Figure 1: The Five-Layer Agent Trust Stack

Each layer is a distinct concern and can be reasoned about independently of the others. However, a complete system composes all five; skipping a layer creates a gap that adversaries will find and exploit. Each layer answers a different question. Each layer requires different primitives. Each layer can be implemented, deployed, and reasoned about independently of the others, though composition across layers is essential for a complete system.

The five layers, in the order in which they apply to any action, are:

1. Identity: who is acting? 2. Authorization: is the actor permitted? 3. Attestation: who says so? 4. Evidence: what can be independently verified about this action? 5. Trust: should the actor be trusted going forward?

The layers are presented in order because they reflect the sequence in which any decision about an agent action must be made. First, the verifying party must determine who is acting. Then it must determine whether the actor is permitted to take this action. If permitted, the action proceeds and produces evidence. The outcome feeds back into a trust judgement that informs future decisions.

4.2. The Importance of Separating Evidence from Attestation

A consequence of the layered model is that evidence and attestation must not be confused. Attestation is a claim by some party; its value depends on the trustworthiness of the claimant. Evidence is an artifact that can be verified independently of the claimant’s trustworthiness; its value depends on the verifiability of the artifact itself.

For example, an agent’s self-report that it executed a database query successfully is an attestation. A cryptographic hash of the exact request and response, signed at the time the action occurred, bound to a hash of the previous such record, is evidence. The attestation can be revised later. The evidence cannot.

This distinction is critical because systems that conflate the two concepts end up with neither. A system that accepts attestations as evidence cannot prove anything after the fact. A system that demands evidence for every attestation is unusable in practice. The correct approach is to provide both, at different layers, with clear interfaces between them.

4.3. Why Five Layers

The choice of five layers is not arbitrary. It reflects five questions that any complete agent trust system must answer, and each question requires different technical primitives.

Identity requires cryptographic credentials, a key management system, and an issuing authority model.

Authorization requires a policy language, a decision point, and a mechanism for expressing and evaluating scopes.

Attestation requires a signing key and a format for asserting claims.

Evidence requires tamper-evident storage, time ordering, and a mechanism for external verification.

Trust requires reputation accumulation, peer attestations, and a policy mechanism for combining historical signals into a current decision.

These five primitive sets are necessary and collectively sufficient for the agent trust problem. Fewer layers conflate concerns that must be separated. More layers over-decompose and make reasoning harder.

5. The Identity Layer

5.1. Purpose

The identity layer answers the question: who is acting? For an autonomous agent, this question has several parts.

Which agent instance is making this request? Two instances of the same agent implementation, running simultaneously, are distinct actors. They MUST have distinct identities.

Which agent implementation is this instance running? The implementation is the code and model that define the agent’s capabilities and behaviour.

Which principal authorised this agent? The principal is the human, organisation, or service on whose behalf the agent is acting. Agent identity does not replace principal identity; it supplements it.

Who issued the agent’s identity? The issuing authority vouches for the agent’s identity. Verification of agent identity requires verifying the issuing authority’s assertion.

5.2. Requirements

An identity layer for autonomous agents MUST satisfy the following requirements.

R1: Non-transferability. An agent identity MUST be bound to a specific agent instance. It MUST NOT be possible to transfer an agent identity to a different instance without the issuing authority’s consent.

R2: Non-repudiation. An action signed with an agent identity MUST NOT

be later disavowable by the agent or its operator. The signature itself provides cryptographic evidence of the action.

R3: Time-bound validity. Agent identities MUST have explicit issuance and expiration times. An identity that has no expiration time is equivalent to a permanent credential, which violates the principle of least privilege.

R4: Offline verifiability. It MUST be possible to verify an agent identity without contacting the issuing authority for every verification. A design that requires online verification on every action creates a dependency and a bottleneck that are incompatible with distributed systems.

R5: Distinct from principal identity. Agent identity MUST be distinct from the identity of the principal. The system MUST NOT conflate "the user logged in" with "the agent is acting on the user's behalf."

R6: Revocability. It MUST be possible to revoke a specific agent identity without revoking the principal's session or the identity of other agents running under the same principal.

These requirements collectively rule out several approaches that are sometimes proposed. They rule out using the principal's OAuth token as the agent identity (fails R5, R6). They rule out using a static API key (fails R1, R3). They rule out using a server-side session cookie (fails R4). They rule out using TLS client certificates without modification (fails R1 at the instance granularity, fails R6).

5.3. Cryptographic Basis

An agent identity credential is, at minimum, a public key, a binding to an identifier, a signature by an issuing authority, and associated metadata (issuance time, expiration time, trust level, principal). The public-private key pair used to generate signatures MUST be generated in a manner consistent with current cryptographic best practice; at the time of writing, ECDSA over NIST P-256 [FIPS186-5] is recommended for new deployments due to its balance of security, performance, and interoperability. Implementations MAY support additional algorithms.

Signatures SHOULD use IEEE P1363 fixed-length format (64 bytes for P-256), with low-S normalisation to prevent signature malleability. JSON payloads to be signed SHOULD be canonicalised per [RFC8785] before signing, so that verification is deterministic regardless of JSON serialisation choices.

Private keys corresponding to agent identities MUST be protected against exfiltration. In high-assurance deployments, private keys SHOULD be held in a hardware security module (HSM), trusted execution environment (TEE), or equivalent. In lower-assurance deployments, private keys MAY be held in the agent's memory, but the operator SHOULD ensure that memory dumps cannot be obtained by unauthorised parties.

5.4. The Agent Passport

An agent passport is the concrete artifact that carries agent identity. This framework does not mandate a single passport format, but describes the minimum content.

The passport is a self-contained signed document. It travels with the agent and can be verified offline by any party that holds the issuing authority's public key. The figure below illustrates the structure.

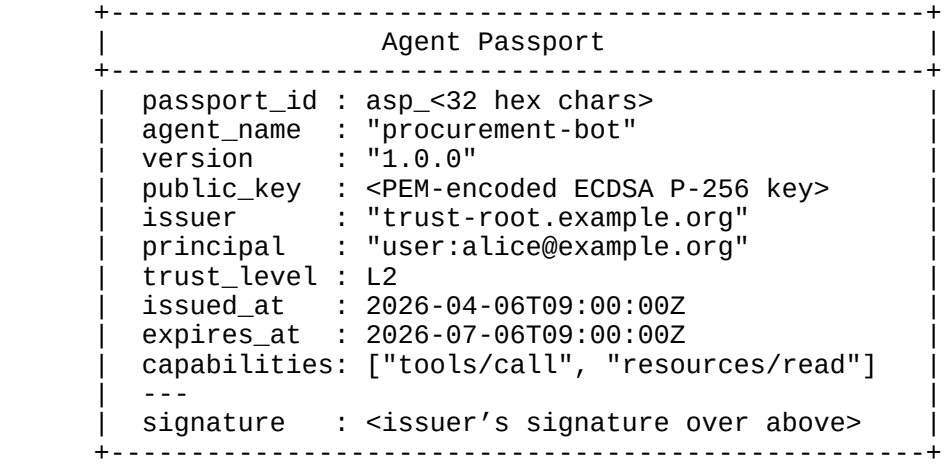


Figure 2: Agent Passport Structure

The passport contains everything a verifier needs: the public key used to verify the agent's signatures, the identity of the authority that issued the passport, the principal whose authority the agent operates under, the assurance level, and the validity window.

An agent passport is a signed data structure containing:

* A unique passport identifier.
* The agent's public key.
* The agent's name or label (human-readable).
* The agent's implementation version.
* The issuing authority's identifier.
* Issuance timestamp.
* Expiration timestamp.
* Trust level (see Section mechanism-trust-levels).
* Optional capability declarations.
* A signature by the issuing authority over the preceding fields.

A passport identifier format that has been used in deployed implementations is the prefix "asp_" followed by 32 hexadecimal characters, providing 128 bits of entropy. This document does not mandate this format, but notes that a prefix-based scheme makes passport identifiers self-describing and facilitates log analysis.

The passport is designed to be carried with every agent action. A verifier receiving an action signed by an agent can extract the passport, verify the issuing authority's signature over the passport, extract the agent's public key from the passport, verify the signature over the action, and proceed.

5.5. Issuing Authority Models

This framework supports multiple issuing authority models, corresponding to different deployment contexts.

Self-issued. An agent generates its own key pair and produces a self-signed passport. This is the simplest model and provides cryptographic non-repudiation but no third-party trust. It is appropriate for development, testing, and isolated deployments. Trust level L0 (see Section mechanism-trust-levels) caps self-issued passports.

Private trust root. An organisation operates its own issuing authority, issuing passports to agents under its control. Verification requires the organisation’s public key (the trust root) to be distributed to verifiers in advance. This is appropriate for internal deployments within an enterprise.

Federated trust. Multiple organisations form a trust federation with cross-signed trust roots, allowing agents from one organisation to be verified by verifiers in another. This model is compatible with existing federation standards.

Public trust root. A public service operates an issuing authority that any party can use. Verification requires the public service’s trust root, which is distributed through well-known infrastructure analogous to how Certificate Authority (CA) roots are distributed in the WebPKI.

This framework does not mandate any single model. It does require that the model in use MUST be discoverable by verifying parties.

6. The Authorization Layer

6.1. Purpose

The authorization layer answers the question: is the actor permitted? It takes as input an agent identity (from the identity layer) and a proposed action, and produces a decision: ALLOW or DENY.

In the human-user case, authorization is typically based on roles and scopes. A user has a role (e.g., "administrator"), a role has permissions, permissions are checked at runtime. OAuth 2.0 scopes generalise this: a client is granted specific scopes by the user at consent time, and the service checks the scopes on each request.

Agent authorization requires more than this. Because agents select their actions at runtime, coarse pre-granted scopes cannot adequately express the boundary between permitted and forbidden behaviour. An agent granted "read access to the user’s calendar" is either granted too much (any calendar action) or is blocked from actions the user would have approved in advance (creating calendar entries on the user’s behalf). Neither extreme is satisfactory.

6.2. Requirements

An authorization layer for autonomous agents MUST satisfy the following requirements.

A1: Fine granularity. Authorization decisions MUST be possible at the level of individual operations (e.g., specific tool invocations, specific API endpoints, specific data scopes), not only at the level of entire resources or client applications.

A2: Runtime evaluation. Authorization MUST be evaluable at the time of each action, not only at session establishment. An authorization decision made at session start cannot account for the actions the agent chooses to take later.

A3: Contextual parameters. Authorization decisions MUST be able to incorporate parameters beyond identity: time of day, spending limits, counterparty identity, request size, cumulative usage since a previous baseline.

A4: Delegation awareness. When agent A invokes agent B on behalf of principal P, the authorization decision on agent B’s action MUST incorporate the delegation chain: whether P authorised A to delegate to B, the limits of that delegation, and whether B’s action is within those limits.

A5: Policy expression in a standard form. Authorization policies MUST be expressible in a portable format so that they can be audited, versioned, reviewed, and transferred between systems without reverse engineering.

A6: Fail-closed default. In the absence of an explicit allow decision, the default MUST be deny. This is the standard fail-closed pattern and protects against policy oversight.

6.3. Beyond Pre-granted Scopes

Existing OAuth-style scopes do not satisfy A1 through A5. OAuth scopes are typically coarse-grained ("read", "write"), are granted at consent time (not runtime), and cannot easily express contextual parameters such as spending limits or delegation depth. Extending OAuth scopes to cover these cases has been attempted (RAR, Rich Authorization Requests; XACML, eXtensible Access Control Markup Language; ALFA; OPA, Open Policy Agent) with mixed adoption.

This framework takes the view that a complete solution requires three distinct components: a scope expression language (for describing what an agent may do), a consent mechanism (for translating a principal’s intent into concrete scopes), and a decision point (for evaluating actions against scopes at runtime). Individual mechanisms for these components are discussed in Section mechanism-agent-scope-expression and Section mechanism-fine-grained-consent.

6.4. The Trust Gate

A specific decision point pattern that this framework names explicitly is the Trust Gate.

The Trust Gate sits at the boundary of a service that accepts agent traffic. Incoming requests pass through the gate before reaching business logic. The gate performs identity verification, trust level check, policy evaluation, and contextual parameter evaluation. It produces a structured allow or deny decision and logs its reasoning.

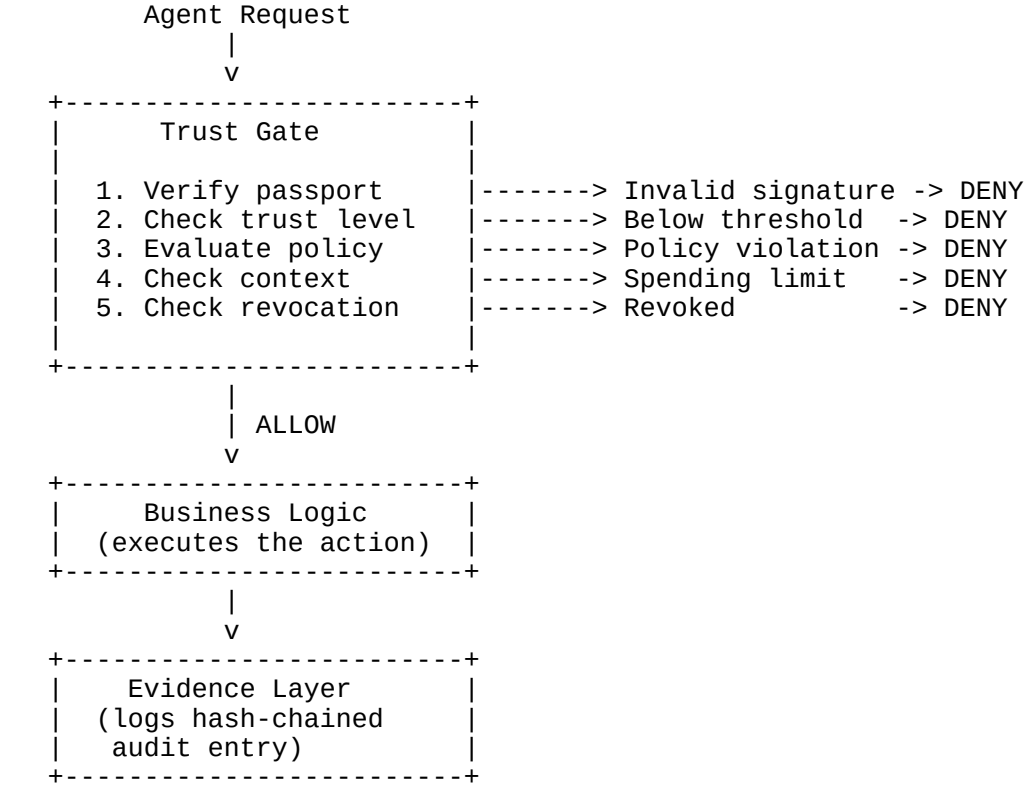


Figure 3: Trust Gate Decision Flow

The Trust Gate is a reusable pattern. Services that accept agent traffic should implement it as middleware, not inline in business logic. Centralising the decision point ensures consistent policy enforcement and provides a single point at which auditing, logging, and rate limiting are applied. The Trust Gate is a decision point at the boundary of a service, invoked before any requested operation is executed, that takes as input the agent’s passport, the requested operation, and contextual parameters, and produces an allow/deny decision.

The Trust Gate is distinct from traditional authorization in one important respect: it evaluates trustworthiness in addition to permission. A traditional authorization check asks "is this identity permitted to perform this operation?" The Trust Gate additionally asks "is this identity’s current trust level sufficient for this operation?" An agent that is permitted to execute a payment but has accumulated anomalous behaviour signals may be rejected by the Trust Gate even though it would be permitted by a role-based check. This permits policy decisions that blend identity, authorization, and behavioural signals.

Trust Gates are intended to be implemented as middleware in services that accept agent traffic, analogous to how rate limiting or HMAC verification are implemented today. They operate before business logic and return a structured decision that business logic may consult.

7. The Attestation Layer

7.1. Purpose

The attestation layer answers the question: who says so? When an agent makes a claim about itself (its identity, its code, its policy, its prior actions), the claim must be attributable to someone whose trustworthiness can be evaluated.

Attestation is distinguished from evidence at the next layer. Attestation is a claim made by a party: "I am agent X", "I executed action Y", "my code hash is Z". Evidence is a verifiable artifact that does not depend on a claim. A signed passport is an attestation by the issuing authority. A hash-chained log record is evidence.

7.2. Categories of Attestation

This framework distinguishes four categories of attestation that are relevant to agents.

Identity attestation. The issuing authority attests that a given public key belongs to a given agent identity. This is carried in the passport.

Code attestation. A party (the agent operator, or a trusted verifier such as a remote attestation service) attests that the agent’s code is a specific known version, with a specific known hash. Code attestation supports the claim that the agent has not been tampered with or replaced since it was audited.

Action attestation. The agent signs each action it takes. The signature is an attestation by the agent (under the identity vouched for by its issuing authority) that the specific action, with specific parameters, at a specific time, was performed by this agent.

Trust attestation. A third party (another agent, a peer service, a reputation system) attests to the trustworthiness of a given agent. Peer attestations form the basis of distributed trust scoring.

7.3. Requirements

An attestation layer for autonomous agents MUST satisfy the following requirements.

AT1: Cryptographic signing. Every attestation MUST be cryptographically signed. Unsigned claims are not attestations in

this framework’s sense.

AT2: Inclusion of identity. Every attestation MUST include a reference to the identity of the attesting party. An attestation that does not identify its source cannot be evaluated for trustworthiness.

AT3: Inclusion of time. Every attestation MUST include a timestamp. The timestamp is a claim by the attester about when the attestation was made; it is not itself evidence (unless anchored externally, as discussed in Section evidence-layer).

AT4: Scope. Every attestation MUST state clearly what is being attested. A signed blob without a clear statement of meaning is not an attestation; it is a signature with no semantic content.

AT5: Relation to prior attestations. Where applicable, attestations SHOULD reference prior related attestations. This allows a chain of attestations to be constructed (for example, a sequence of actions by the same agent over time).

7.4. Attestation Is Not Enough

A system that relies only on attestations (without the evidence layer below) has an obvious failure mode: a compromised agent, or an agent whose operator is malicious, can issue attestations about anything. The attestations will be cryptographically valid but semantically false.

This is why the framework separates attestation from evidence. Attestation at minimum provides non-repudiation (a party cannot later disavow a signed claim). But non-repudiation alone is insufficient when the adversary is the attester. Evidence, as discussed in the next layer, provides a verification path that does not depend on the attester’s honesty.

8. The Evidence Layer

8.1. Purpose

The evidence layer answers the question: what can be independently verified about this action? Evidence is an artifact that, once produced, can be checked for correctness without reference to the party that produced it.

The canonical example is a cryptographic hash. A SHA-256 hash of a file is evidence of the file’s contents: anyone with the file can recompute the hash and confirm that the hash matches. The party who computed the hash originally need not be trusted. The hash itself is the evidence.

Evidence differs from attestation in that evidence does not require the producer to be trusted. It is a deliberately lower bar than attestation, and therefore a stronger guarantee.

8.2. Minimal Evidence Bundle for an Agent Action

This framework proposes that every agent action SHOULD produce a minimal evidence bundle containing at least the following.

The evidence bundle structure is shown below. It is deliberately minimal: it contains hashes and identifiers, not payloads. This reduces the privacy surface of the audit trail while preserving the ability to verify that a specific action occurred.

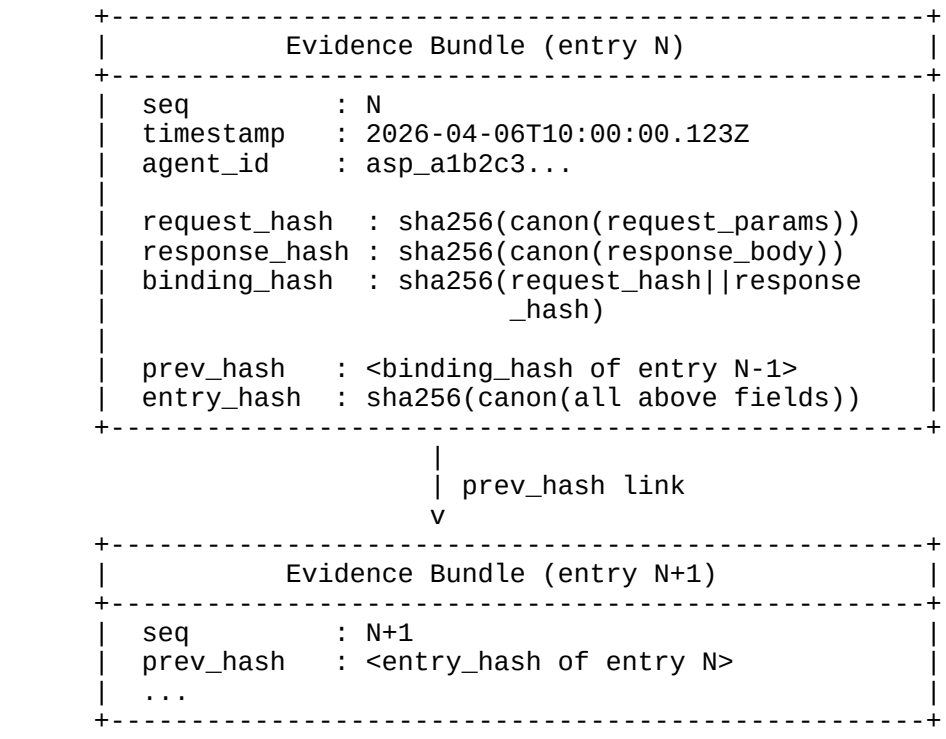


Figure 4: Evidence Bundle Chain

The hash chain is the heart of the tamper-evident property. Modifying entry N requires recomputing entryhash for N, which breaks the prevhash reference in entry N+1, which breaks N+1’s entry_hash, and so on to the end of the chain. Undetected modification requires rewriting every subsequent entry; detecting the modification requires only comparing a known root hash against a recomputed one.

Request hash. A cryptographic hash of the canonicalised request payload (the parameters of the action).

Response hash. A cryptographic hash of the canonicalised response payload (the result of the action, or the error).

Binding hash. A cryptographic hash of the canonicalised concatenation of the request hash and the response hash. This binds the request and response together, such that it is infeasible to substitute either one independently without detecting the substitution.

Timestamp. The time at which the action was performed, ideally

anchored against an external time source.

Prior hash. A reference to the binding hash of the previous action by the same agent, forming a hash chain.

These fields, and only these fields, provide the minimal guarantee that a specific request and response pair occurred at a specific time, in a specific order relative to other actions by the same agent. They do not prove that the action was authorised, or that the agent was legitimate, or that the action was the one the principal intended. Those claims are at higher layers. The evidence layer provides only the narrow technical claim that a specific artifact pair is bound together and ordered.

The evidence bundle is deliberately minimal. Additional metadata (the agent identity, the issuing authority, policy context) can be added at higher layers and verified separately. Keeping the evidence layer minimal ensures that the evidence remains verifiable even when higher layers are compromised or unavailable.

8.3. Tamper-Evident Audit Trail

Individual evidence bundles are valuable. A sequence of evidence bundles, chained together, is more valuable. A tamper-evident audit trail is a sequence of records where each record’s hash includes a reference to the previous record’s hash, such that modification of any record invalidates all subsequent records.

This is the construction used by Certificate Transparency [RFC6962] to provide public audit of TLS certificate issuance. The same construction applies to agent actions. A verifier can check that an audit trail has not been tampered with by recomputing the chain from a known good root to the current record.

An audit trail alone does not prove that records have not been added or that early records are genuine. It only proves that the sequence has not been modified since a known point in time. To achieve stronger guarantees, the audit trail should be externally anchored: root hashes should be committed to an external witness (for example, a blockchain, a trusted time-stamping service, or a public append-only log) at regular intervals. External anchoring converts a local audit trail into a globally verifiable record.

8.4. Integration with SIEM and Forensic Systems

Agent audit trails SHOULD be exportable in standard formats consumable by Security Information and Event Management (SIEM) systems and forensic analysis tools. Common formats include syslog [RFC5424], Common Event Format (CEF), JSON Lines, and vendor-specific formats. The export format is a presentation concern and does not affect the underlying audit guarantees.

9. The Trust Layer

9.1. Purpose

The trust layer answers the question: should the actor be trusted going forward? Trust is a policy decision that takes as input an actor's identity, attestations about the actor, evidence of the actor's past behaviour, and peer assessments, and produces a trust score or trust level that informs future decisions about the actor.

Trust is the most subjective of the five layers, and the layer where policy choices must be made by the deploying organisation. This framework does not prescribe a specific trust function. It describes the inputs, the properties the function should have, and the ways trust interacts with the other layers.

9.2. Inputs to Trust

A trust function operates on at least the following inputs.

Identity history. How long has this agent been known to the verifying party? Newly issued identities are inherently less trusted than identities with a track record.

Behavioural history. What has the agent done? Have its actions been consistent with stated policy? Are there anomalies (unusual operation counts, unusual hours, unusual counterparties)?

Peer attestations. Have other parties in the trust ecosystem issued attestations about this agent? Negative attestations (reports of policy violations) reduce trust; positive attestations (successful transactions, audit passes) increase trust.

Code attestation. Does the agent run a known, attested version of the implementation? An agent whose code has not been verified is less trusted than one whose code has.

Issuing authority standing. Is the issuing authority that vouched for this agent itself trusted? Trust propagates through the issuance chain.

9.3. Properties of a Good Trust Function

Without mandating any specific implementation, this framework identifies properties that a trust function should have.

Monotonicity under good behaviour. Actions consistent with policy should increase trust (with diminishing returns). Actions inconsistent with policy should decrease trust.

Asymmetry. Trust should decrease faster than it increases. A single serious violation should reduce trust more than it could be built up by a single positive action.

Transparency. The inputs and outputs of the trust function should be

transparent to the agent operator. Opaque trust systems cannot be audited or contested.

Due process. An agent that is about to be denied based on trust should have the opportunity to learn why (at some level of granularity) and to dispute the decision. This is analogous to the due process requirements on credit scoring in many jurisdictions.

Resistance to manipulation. The trust function should be robust against collusion (multiple agents issuing positive attestations about each other to inflate trust) and Sybil attacks (a single operator creating many agents to dilute negative signals).

9.4. Trust Levels

This framework defines a discrete five-level trust hierarchy (L0 through L4) described in Section mechanism-trust-levels. Discrete levels are easier to reason about than continuous scores and are easier to express in policy. Continuous scores may be appropriate for internal trust calculations but should be mapped to discrete levels for external consumption.

9.5. Trust Is Not a Moral Judgement

A frequent misunderstanding of trust systems is that they are making a moral judgement about the agent. They are not. Trust in this framework is a policy decision by one party about whether to accept actions from another, based on observable behaviour. It is a technical construct, not a verdict.

This matters because trust decisions must be contestable. An agent that is denied at a Trust Gate because of low trust should have recourse: the ability to ask why, the ability to challenge false signals, the ability to accumulate positive signals that restore trust over time. A trust system without recourse is a blacklist.

10. Mechanisms

The preceding sections describe the framework in the abstract. This section describes specific mechanisms that close specific gaps within the framework. Each mechanism is described at the level of requirements and interfaces; detailed wire formats are in separate documents.

10.1. Agent Identity Credentials

An agent identity credential is the concrete artifact carrying an agent passport. The credential contains the agent’s public key, metadata, and a signature by the issuing authority.

OpenID Connect provides a foundation for extending user identity to agents. An extension (see [I-D.sharif-openid-agent-identity]) defines additional claims that identify an actor as an agent, bind the agent to a principal, and carry agent-specific attributes.

The mechanism integrates with OAuth 2.0 token endpoints, OIDC Discovery, and JWKS endpoints. Existing identity infrastructure can issue agent credentials with minimal modification; the primary change is the addition of new claim types and the agent-specific token profile.

10.2. Delegation Chains

When agent A invokes agent B on behalf of principal P, the resulting action must be attributable to all three parties (P, A, and B) and bounded by the authority each granted to the next.

The figure below illustrates a delegation chain from a human principal through two agents to a target service. Each arrow carries a signed authorisation grant. The target service reconstructs the chain from the bottom up and verifies that each link is signed by the party named in the previous link.

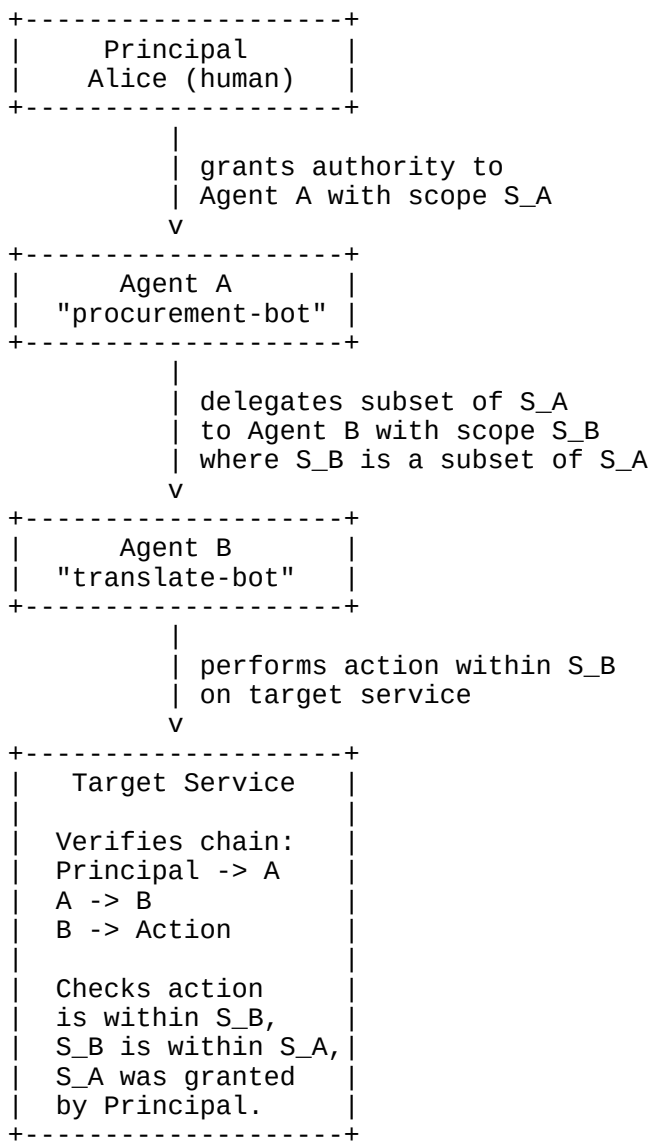


Figure 5: Delegation Chain

A delegation chain MUST be bounded. Infinite or unbounded delegation creates a denial of service risk and makes auditing impractical. A maximum delegation depth (this framework recommends 5 hops as a default, configurable per deployment) limits the chain.

Each link in the chain MUST carry its own cryptographic binding: the delegating party signs an assertion that includes the delegatee's identity, the scope of the delegation, the expiration time, and the prior link in the chain. Verification proceeds from the terminal action backwards through the chain to the principal.

OAuth 2.0 Token Exchange [RFC8693] provides a starting point. It defines a flow in which one token can be exchanged for another that is narrower in scope and represents a new actor. For agent delegation, the delegation chain is explicit: a delegated token carries references to its antecedents, so that the full chain can be reconstructed.

A delegation chain MUST be bounded. Infinite or unbounded delegation creates a denial of service risk and makes auditing impractical. A maximum delegation depth (this framework recommends 5 hops as a default, configurable per deployment) limits the chain.

Each link in the chain MUST carry its own cryptographic binding: the delegating party signs an assertion that includes the delegatee's identity, the scope of the delegation, the expiration time, and the prior link in the chain. Verification proceeds from the terminal action backwards through the chain to the principal.

10.3. Per-Operation Signing

Per-operation signing requires each agent action to carry a cryptographic signature over the action's parameters, along with replay-protection metadata (nonce and timestamp).

MCPS [MCPS] defines a specific instantiation of per-operation signing for the Model Context Protocol. The JSON-RPC message is wrapped in an envelope that contains the signature, the passport identifier, a nonce, and a timestamp. Canonical JSON serialisation [RFC8785] is used to make verification deterministic.

The performance cost of per-operation signing is typically negligible for interactive traffic (under one millisecond per operation on modern hardware for P-256 signing), but it is not zero. High-throughput deployments may require batching, signature caching, or hardware acceleration to avoid throughput loss.

10.4. Trust Levels (L0 through L4)

This framework defines five discrete trust levels. The levels are hierarchical: L4 is the highest, L0 is the lowest. Each level adds specific requirements on top of the level below it.

Trust Level	Criteria	Typical Use

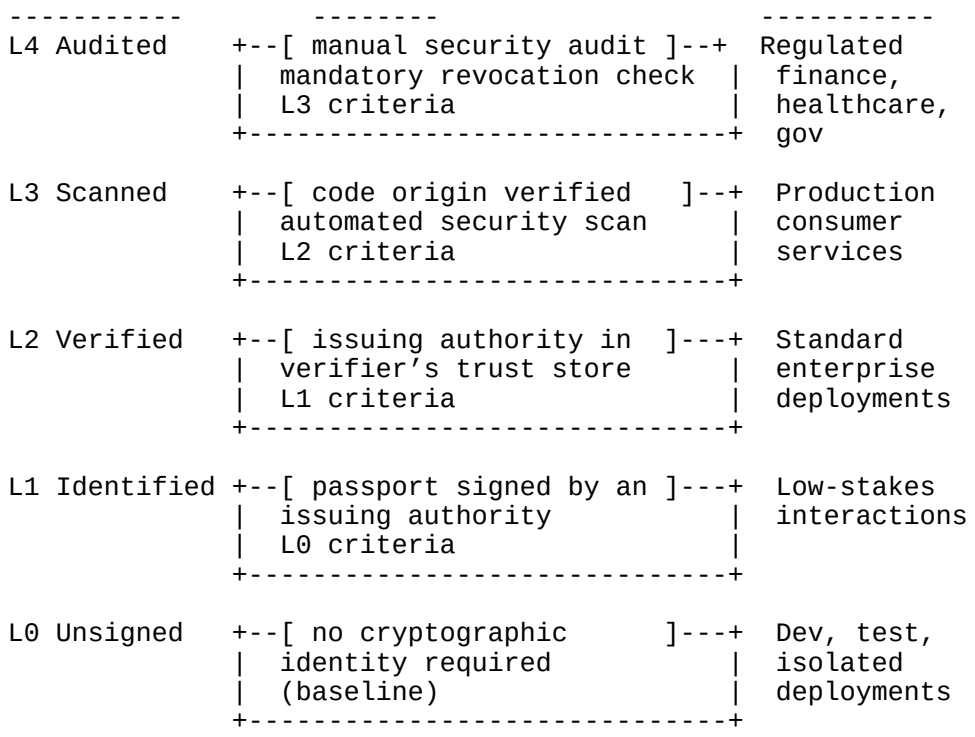


Figure 6: Trust Level Hierarchy

Verifying parties declare a minimum trust level as policy. An agent connecting at a level below the minimum is rejected. The numeric levels provide a simple ordering that can be compared directly ("mintrust <= agenttrust").

L0: Unsigned. The agent has no cryptographic identity. Actions are unsigned. This level is appropriate for development, testing, and isolated deployments where cryptographic identity is not required. Self-issued passports are capped at L0 unless explicitly trusted by a verifier.

L1: Identified. The agent has a passport signed by an issuing authority. Actions are signed with the agent's private key. The verifying party can confirm that the actions are attributable to the specific agent identity. No additional vetting is assumed.

L2: Verified. In addition to L1, the issuing authority's root of trust is in the verifier's trust store. The verifier has an explicit policy decision to accept this issuing authority. This is the level at which most production agent deployments should operate.

L3: Scanned. In addition to L2, the issuing authority has verified the agent's code origin (e.g., through code signing or provenance attestation), and the agent has passed automated security scanning against a standard set of checks. The OWASP MCP Security Cheat Sheet [OWASP-MCP] is an example of such a check set.

L4: Audited. In addition to L3, the agent has undergone manual security audit by a qualified party. Mandatory revocation signals are published for L4 agents, and verifiers check revocation before accepting L4 actions.

Verifying parties declare a minimum trust level as policy. An agent connecting at a level below the minimum is rejected.

The numeric levels are deliberate. They provide a simple ordering that can be compared directly ("mintrust <= agenttrust"). They do not imply that L4 is infinitely better than L1; they imply that L4 meets more specific criteria.

10.5. Trust Gate

The Trust Gate is a middleware component that implements the authorization layer’s decision point. It accepts incoming agent requests and produces an allow/deny decision based on identity, trust level, policy, and contextual parameters.

A Trust Gate implementation typically exposes an interface such as:

‘‘" decide(agent_passport, operation, parameters, context) -> decision "’’

where "decision" is a structured value indicating ALLOW or DENY, a reason code, and (in the DENY case) whether the decision is appealable.

Trust Gates SHOULD be implemented as reusable middleware rather than inline in business logic, so that policy changes do not require modifying business code and so that decisions are consistently logged.

10.6. Agent Audit Trail

The agent audit trail is a sequence of evidence records (see Section evidence-layer) that records every action taken by the agent. See [I-D.sharif-agent-audit-trail] for a detailed specification of the record format and chaining rules.

Key requirements: records are append-only, hash-chained, signed, and exportable in standard SIEM formats. Root hashes are anchored externally at regular intervals to protect against unilateral modification of the log.

10.7. Agent Payment Trust

Agent payment trust is a specific application of the framework to financial transactions. An agent initiating a payment goes through a sequence of checks: trust level verification, sanctions screening against external lists, spending limit enforcement, per-operation signing of the payment request, and non-repudiable receipt generation. See [I-D.sharif-agent-payment-trust] for detailed specification.

This mechanism integrates with FAPI 2.0 [FAPI2] for high- assurance

use cases. The per-operation signing requirement fits naturally with FAPI's existing sender-constrained token model, extending it from "the client that received this token" to "the specific agent operating within this client".

10.8. Agent Transport Protocols

Two transport protocols support agent-specific communication patterns.

The Agent Transport Protocol (ATP) provides asynchronous store-and-forward messaging between agents that may not be continuously online. Messages are queued at the destination and delivered when the destination is available. See [I-D.sharif-agent-transport-protocol].

The Agent Trust Transport Protocol (ATTP) provides synchronous agent-to-server communication with mandatory signing on every request. ATTP introduces the "attp://" URL scheme to mark endpoints that require trust-enforced transport. See [I-D.sharif-attp-agent-trust-transport].

Both protocols are complementary to HTTP and coexist with it; they do not replace existing transport.

10.9. Model Integrity Verification

Model integrity verification ensures that an AI model file loaded by an agent has not been tampered with between publication and use. A signed manifest containing file hashes and metadata travels with the model and is verified at load time. Supported model formats include SafeTensors, PyTorch, GGUF, ONNX, and TensorFlow.

This mechanism addresses a threat that is specific to ML systems: an attacker modifies a model file to introduce a backdoor or bias, and the modification is not detected at load time. Without integrity verification, the agent acts on a compromised model. With integrity verification, the load fails and the agent does not run.

10.10. Trust-Gated Networking

Trust-gated networking is a network enforcement pattern in which every outbound connection from an agent is evaluated against a trust policy before the connection is permitted. Destinations below a minimum trust threshold are blocked at the socket layer, preventing agents from contacting untrusted services.

The mechanism requires a trust score or trust level associated with each destination (typically an external service or peer agent) and a policy specifying the minimum level required for each type of action. Integration with a trust index service (analogous to a URL reputation service) provides the destination-side input.

10.11. Agent Scope Expression

Agent scope expression is a format for declaring what an agent may do, at finer granularity than OAuth scopes. A scope expression specifies the allowed tools or methods, the allowed parameter values (e.g., specific counterparties, specific amount ranges), time windows, and cumulative usage limits.

Scope expressions are intended to be verifiable: a decision point can take a scope expression and a proposed action and determine in bounded time whether the action is within the scope. This rules out Turing-complete policy languages for production use; the expression language should be expressive enough for common cases but tractable to evaluate and analyse.

10.12. Agent Revocation

Agent revocation invalidates a specific agent identity without invalidating other identities issued by the same authority or the principal’s session. Two mechanisms are required.

Pull-based revocation. Verifiers check a revocation list or status endpoint before accepting an action. This is the OCSP model for X.509 certificates. The limitation is that verifiers must check, and they must trust the revocation source.

Push-based revocation. The issuing authority publishes a signed revocation notice to all subscribed verifiers. Verifiers receive and cache the notice, and reject subsequent actions by the revoked agent. This is more responsive but requires a subscription mechanism.

In practice, high-assurance deployments use both. A status endpoint is consulted for the current revocation list, and push notifications are used for rapid propagation of urgent revocations.

10.13. Cross-Domain Trust Portability

Cross-domain trust portability is the ability for an agent’s trust standing in one domain to be verified in another. An agent with trust level L3 in its home organisation should be able to prove that status to an external verifier without the external verifier having to re-evaluate the entire trust history.

This mechanism requires:

* A standard format for portable trust assertions.
* A mechanism for federated verification (the home organisation vouches for the trust level, and a verifier in a foreign domain can verify the home organisation’s signature).
* Agreement on what each trust level means across domains.

Federation is not mandatory; deployments may choose to operate in isolation. But for cross-organisational agent commerce, some form of portable trust is necessary.

10.14. Fine-Grained Consent

Fine-grained consent allows a principal to grant an agent authority for specific actions, not only for broad categories of actions. Instead of "access to the user's calendar", consent might be "create and modify events in the user's calendar during business hours, not involving external participants, up to 10 events per day."

Fine-grained consent requires a user interface that can express the granted authority in terms the user understands, a storage format for the granted scope, and a decision point that can evaluate actions against the stored scope (see Section mechanism-agent-scope-expression).

Existing OAuth consent flows can be extended to capture fine- grained consent, but doing so adds complexity to the user experience. Careful interface design is required to keep consent comprehensible while capturing enough detail to be useful.

10.15. Agent Discovery

Agent discovery is the process by which one agent finds another to invoke. Discovery mechanisms should support finding agents by capability ("an agent that can translate documents"), by identity (a specific known agent), or by affiliation ("an agent operated by organisation X").

DNS-based discovery using SRV records is a natural starting point and aligns with existing Internet infrastructure. A capability registry (analogous to an OpenAPI catalogue) provides richer discovery at the cost of additional infrastructure.

This framework does not mandate a single discovery mechanism. It does recommend that any discovery mechanism include trust information as a first-class result, so that an agent discovering another agent immediately has access to the other agent's trust level and can apply policy accordingly.

10.16. Agent Capability Negotiation

Agent capability negotiation is the handshake process by which two agents establish what they can do together: which protocols they support, which cryptographic algorithms, which trust levels, which API versions. This is analogous to TLS cipher suite negotiation.

The negotiation should be signed, bound to the session, and resistant to downgrade attacks. A downgrade attack in the agent context would persuade one agent that the other supports only a weaker protocol, and cause both parties to operate at a lower trust level than necessary. Standard mitigations (signed handshake transcripts, explicit minimum-level requirements) apply.

10.17. Compliance Evidence Export

Compliance evidence export is the ability to produce, on demand, a verifiable bundle of evidence demonstrating compliance with a specific regulatory framework. For the EU AI Act [EU-AI-Act], relevant articles include Article 12 (record-keeping), Article 13 (transparency), Article 14 (human oversight), Article 15 (accuracy, robustness, cybersecurity), and Article 50 (identification of AI systems).

An evidence export is a structured document that references entries in the agent audit trail, groups them by the requirement they satisfy, and includes cryptographic evidence that the entries have not been tampered with. It is produced for a specific time range and can be verified by an auditor using the audit trail root hashes.

11. Threat Model

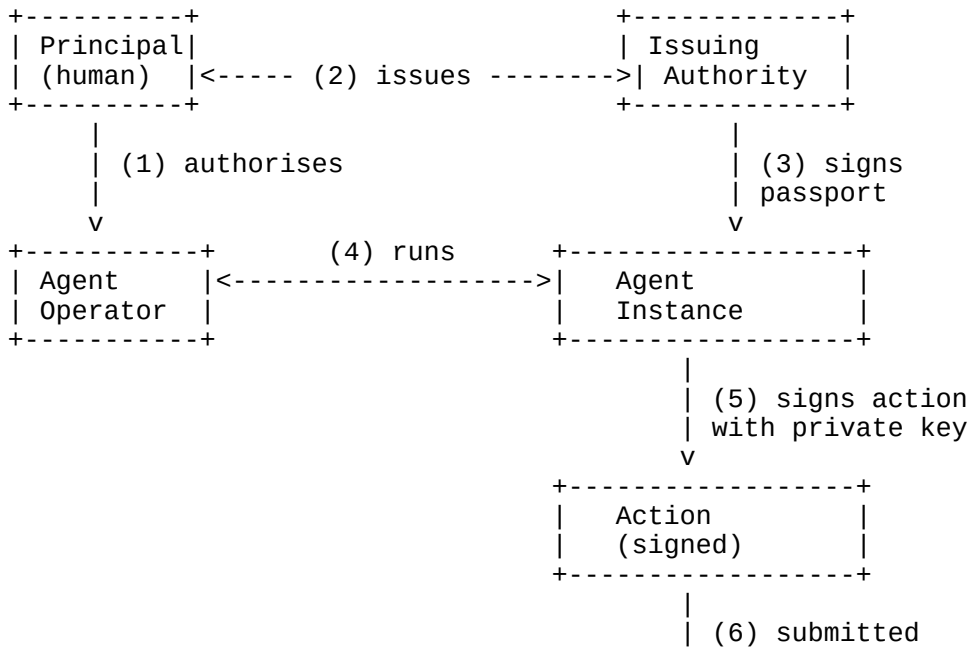
11.1. Scope of the Threat Model

This threat model covers the autonomous agent itself, its interaction with services, and the infrastructure that supports agent identity and trust. It does not cover the security of underlying LLM training data, prompt injection attacks against the LLM as a reasoning system, or the operational security of the agent operator's facilities. Those are important and related topics, but they are out of scope for this document.

Following the guidance of [RFC3552], the threat model identifies assets, threats, attackers, and mitigations, then analyses how the mechanisms of this framework address each threat.

11.2. System Model and Attack Surface

The figure below shows the system model assumed by this threat model. Arrows denote trust boundaries that an attacker may attempt to cross. Each boundary is a candidate attack surface.



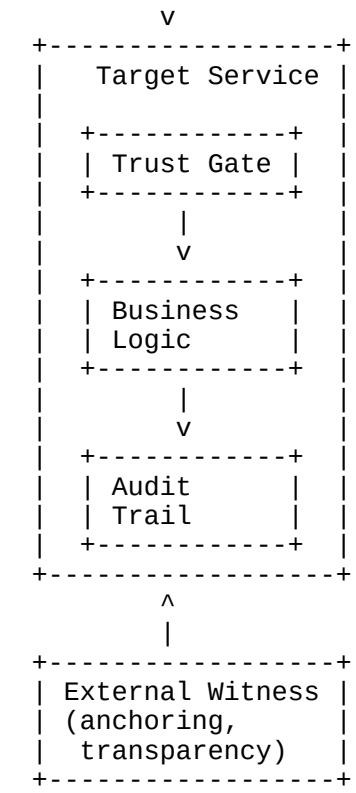


Figure 7: System Model and Attack Surfaces

The numbered arrows mark trust boundaries where an attacker can attempt a compromise:

(1) Principal-to-operator: compromise of the principal allows unauthorised delegation to agents. (2) Principal-to-authority: compromise of the authority- principal binding allows passport misissuance. (3) Authority-to-agent: compromise of the passport issuance process allows forged or misrouted credentials. (4) Operator-to-agent: compromise of the agent runtime allows key extraction, code substitution, or behavioural tampering. (5) Agent-to-target: compromise of the transport allows message tampering, replay, or impersonation. (6) Target-to-witness: compromise of the audit anchoring allows retroactive log tampering.

Each of the threats below maps to one or more of these boundaries.

11.3. Assets

The following assets require protection.

Agent identity credentials. The private key of an agent’s identity and the passport issued by the authority. Compromise allows impersonation of the agent.

Audit trails. The sequence of evidence records produced by an agent. Modification allows deniability or framing.

Authorization policies. The rules that determine what an agent may

do. Modification allows privilege escalation.

Trust scores. The accumulated behavioural and peer attestations about an agent. Manipulation allows unjustified trust elevation or unjustified denial of service.

Model files. The ML model weights and configuration used by the agent. Modification allows behavioural manipulation (backdoors, biased outputs).

Delegation chains. The records of authority passed from principal to agent to sub-agent. Modification allows unauthorised delegation.

Action evidence. The bundles that record individual actions. Modification or fabrication undermines forensic analysis.

11.4. Attackers

This framework assumes the following attacker capabilities.

External attacker. An attacker with Internet-level access, capable of intercepting unencrypted traffic, initiating connections to public endpoints, and executing computationally bounded attacks. This is the standard adversary assumed in TLS [RFC8446] and related protocols.

Malicious server. A server that an agent connects to, which may attempt to deceive the agent by returning fabricated responses, claiming identities it does not hold, or providing tool definitions that differ from the published versions.

Compromised agent. An agent whose operational security has been breached, either through software compromise, key extraction, or insider action. A compromised agent can sign arbitrary actions within its authority.

Malicious agent operator. The operator of an agent that is adversarial to the verifying party. The operator controls the agent’s code, keys, and outputs. This is a stronger threat than a compromised agent, because the operator has insider access to the agent by design.

Colluding agents. A set of agents, possibly operated by the same or cooperating operators, that coordinate to manipulate trust scoring, fabricate peer attestations, or launder malicious actions through intermediaries.

Pervasive passive surveillance. A large-scale adversary capable of passive monitoring of Internet traffic [RFC7258]. This adversary does not inject traffic but observes and correlates flows to learn about agents and their principals.

11.5. Threats

The following threats are analysed.

Agent impersonation. An attacker obtains or forges credentials to present themselves as a legitimate agent.

Replay attack. An attacker captures a signed action by a legitimate agent and replays it to the same or another verifier.

Message tampering. An attacker intercepts a signed action and modifies parameters before the action reaches the verifier.

Passport forgery. An attacker creates a passport that is not issued by a legitimate authority but is accepted as legitimate by verifiers.

Key extraction. An attacker obtains an agent’s private key through memory disclosure, side channels, or insider action.

Log tampering. An attacker modifies or deletes entries in an agent’s audit trail.

Code substitution. An attacker substitutes a modified version of the agent’s code for the legitimate version, while the identity credentials remain valid.

Model tampering. An attacker modifies an ML model file loaded by the agent, introducing backdoors or biased outputs.

Delegation abuse. An attacker causes an agent to delegate to an unauthorised sub-agent, or escalates privilege through a crafted delegation chain.

Trust manipulation. An attacker inflates an agent’s trust score through colluding attestations, or deflates a competitor’s trust score through fabricated negative reports.

Revocation bypass. An attacker continues to use a revoked identity by preventing the verifier from learning about the revocation.

Policy bypass. An attacker crafts requests that evade the authorization policy while still achieving the attacker’s goal (for example, by chaining multiple permitted actions to produce a forbidden outcome).

Resource exhaustion. An attacker submits large numbers of requests to exhaust the verifier’s capacity to check signatures, evaluate policy, or write audit entries.

Side-channel leakage. An attacker observes timing, memory, or other side channels during signature verification to extract information about the verifier or the signing key.

Downgrade attack. An attacker persuades two parties to negotiate a weaker protocol than both support, reducing the security of their interaction.

Cross-domain confusion. An attacker exploits ambiguity in how trust levels are interpreted across organisational boundaries to gain access they would not have in a single domain.

Sybil attack. An attacker creates many identities to dilute the effect of negative signals or to manipulate trust scoring.

Principal confusion. An attacker causes an action to be attributed to the wrong principal, either by tampering with the delegation chain or by exploiting ambiguity in the principal identifier.

11.6. STRIDE Analysis

The threats above are grouped below using the STRIDE taxonomy (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege). The mechanism column lists the primary framework mechanism that mitigates the threat.

STRIDE	Threat	Mechanism
S	Agent impersonation	Cryptographic identity, passport verification, per-operation signing
S	Passport forgery	Issuing authority signature, trust root verification
S	Principal confusion	Principal binding in passport, delegation chain verification
S	Sybil attack	Trust levels require history and attestations; new identities capped low
T	Message tampering	Per-operation signing over canonicalised payload
T	Log tampering	Hash-chained audit trail, external anchoring
T	Code substitution	Code attestation, model integrity verification
T	Model tampering	Signed model manifest verified at load time
R	Action repudiation	Per-operation signing, non-repudiable receipts
R	Delegation repudiation	Signed delegation chain with bounded depth
I	Passport content leak	Opaque identifiers, pseudonymous options
I	Audit trail leak	Encryption at rest, hash-only minimal bundles
I	Trust level leak	Coarse-grained or ephemeral trust assertions for cross-domain use
D	Resource exhaustion	Rate limiting at Trust Gate, pre-verification filtering
D	Trust Gate unavailability	Fail-closed default, high availability design

D	Audit trail flooding	Quota per agent, eviction policy, backpressure
E	Policy bypass	Tractable scope language, cumulative usage limits
E	Delegation abuse	Bounded delegation depth, signed scope inheritance
E	Revocation bypass	Pull + push revocation, level-specific check requirements
E	Downgrade attack	Signed capability negotiation, minimum-level enforcement

Figure 8: STRIDE Threat Analysis

11.7. Attack Tree for Agent Action Forgery

The attack tree below shows what an attacker must accomplish to successfully cause an unauthorised action to be accepted by a verifier. Each leaf is a specific capability the attacker must gain. Each interior node is a disjunction (OR) of the paths beneath it, unless marked AND.

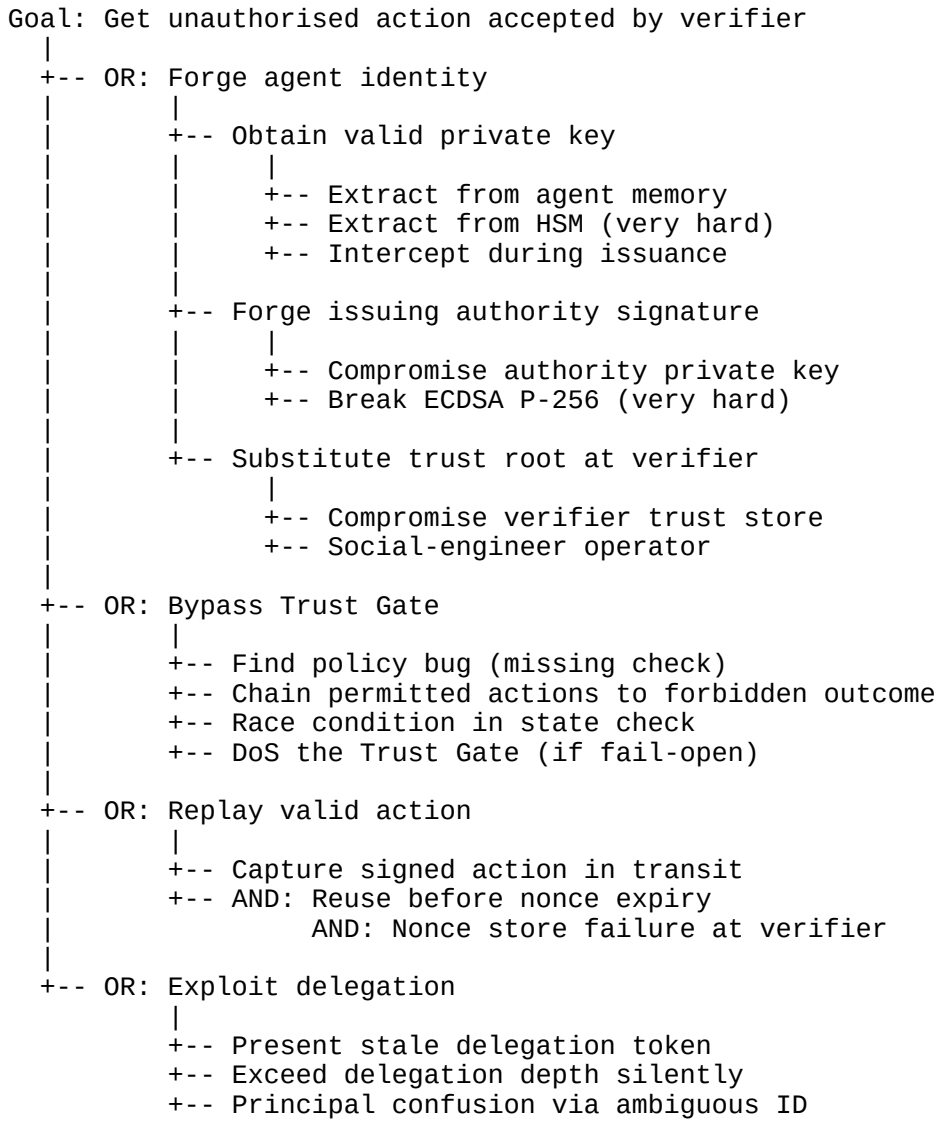


Figure 9: Attack Tree for Action Forgery

The attack tree shows that forging an accepted action requires the attacker to successfully complete at least one full path from root to leaf. The framework is designed so that each leaf is individually expensive or impossible:

* Extracting private keys from hardware-backed storage is computationally and physically expensive.
* Breaking ECDSA P-256 is computationally infeasible with classical computers.
* Compromising an issuing authority is detectable via transparency logs (if used) and attributable.
* Policy bugs are findable and fixable; cumulative usage limits and fail-closed defaults limit blast radius.
* Replay attacks require defeating both nonce stores and timestamp windows simultaneously.
* Delegation exploits are bounded by the cryptographic chain and the depth limit.

A security argument for the framework is that an attacker must either break cryptography, compromise a well-defended root of trust, or find an implementation bug. The framework does not guarantee the absence of implementation bugs, but it minimises their impact and makes them detectable.

11.8. How the Framework Addresses These Threats

Each threat above is addressed by one or more mechanisms in the framework. The correspondence is summarised here.

Agent impersonation is addressed by cryptographic identity and signing. An attacker cannot sign actions as a legitimate agent without the agent’s private key. If the key is properly protected, impersonation requires key extraction, which is a separate threat.

Replay attacks are addressed by per-operation signing, which includes a nonce and timestamp in every signature. A verifier rejects signatures with nonces it has already seen or timestamps outside an acceptance window.

Message tampering is addressed by per-operation signing. Modification of any parameter invalidates the signature.

Passport forgery is addressed by the issuing authority signature. A forged passport that is not signed by a recognised authority is rejected.

Key extraction is addressed by recommending hardware-backed key storage for high-assurance deployments. The framework does not eliminate key extraction as a threat, but it provides a strict upper bound on the damage an extracted key can do: at most, an attacker can

impersonate the specific agent until the key is revoked.

Log tampering is addressed by hash-chained audit trails with external anchoring. Modification of a past entry is detectable because it invalidates the chain. Anchoring to an external witness protects against the operator of the audit system rewriting history unilaterally.

Code substitution is addressed by code attestation. A deployment that requires code attestation as a precondition for accepting actions will reject an agent whose code hash does not match the attested value.

Model tampering is addressed by model integrity verification. An agent that verifies its model before loading will refuse to run with a tampered model.

Delegation abuse is addressed by explicit delegation chains with bounded depth. An attacker cannot escalate through delegation because each link is cryptographically bound and the chain has a policy-enforced maximum length.

Trust manipulation is addressed partially. A trust system cannot be made fully resistant to manipulation by colluding participants; however, the framework recommends asymmetric trust functions (harder to inflate than to lose), peer attestation weighting by attester reputation, and detection of anomalous attestation patterns.

Revocation bypass is addressed by using both pull-based and push-based revocation, and by requiring verifiers to check revocation at the policy-specified granularity for each trust level. High-trust operations (L3, L4) require revocation checks; lower-trust operations may skip them to improve latency.

Policy bypass is addressed by analysis of the scope expression language and by cumulative usage limits. The framework recommends that scope languages be tractable (not Turing complete) so that reachability analysis is possible.

Resource exhaustion is addressed by rate limiting at the Trust Gate and by placing expensive operations (signature verification, policy evaluation) behind rate limits at the service boundary.

Side-channel leakage is addressed by using constant-time signature verification implementations. The framework recommends specific algorithms (ECDSA over P-256 with constant-time implementations) but implementation-specific mitigations are the responsibility of deployers.

Downgrade attacks are addressed by requiring signed capability negotiation in handshake protocols. A transcript of the negotiated parameters is included in the session key derivation, so that a downgrade alters the session key and is detected.

Cross-domain confusion is addressed by requiring explicit mapping of

trust levels between domains in any federation agreement, and by including the issuing authority identifier in every passport.

Sybil attacks are addressed partially by the requirement that trust requires observable, attested, historical behaviour. Creating many new identities does not gain trust, because new identities are capped at low levels until they accumulate history.

Principal confusion is addressed by requiring the principal identifier to be cryptographically bound to the agent passport at issuance time, and to be verified against the delegation chain by the verifier.

11.9. Threats Not Addressed

The framework does not address the following threats and explicitly declares them out of scope.

Prompt injection against the LLM. If an attacker can cause an agent's LLM to select an undesired action, the framework will faithfully record and authorise that action within the agent's policy envelope. The framework does not prevent the LLM from being deceived. It ensures that the resulting action is auditable and bounded by the pre-agreed policy.

Compromise of the principal's credentials. If the principal's account is compromised, the attacker can issue agents with the full authority of the principal. The framework protects the agents but cannot protect against a compromised principal.

Insider attacks by the issuing authority. A malicious issuing authority can issue passports to agents it should not, or revoke passports it should not. Mitigation requires transparency mechanisms (similar to Certificate Transparency [RFC6962]) so that issuing authority actions are publicly auditable. The framework recommends such transparency but does not mandate a specific transparency mechanism.

Side-channel attacks on the LLM itself. Training data extraction, membership inference, and model extraction are machine learning research topics that are out of scope.

Quantum adversaries. All cryptographic recommendations in this document are based on classical cryptographic assumptions (elliptic curve discrete logarithm). A quantum adversary with sufficient resources would undermine these assumptions. Post- quantum migration is a separate concern shared with all of Internet security and is out of scope here.

12. Security Considerations

12.1. Cryptographic Choices

The framework recommends ECDSA over NIST P-256 with SHA-256 for

signing, per [FIPS186-5]. This choice balances security (128-bit level), performance (roughly 50 microseconds for signing on modern hardware), compact signatures (64 bytes), and ubiquitous implementation support.

Implementations MUST use constant-time code paths for signature verification to prevent timing side-channel attacks. Implementations SHOULD use low-S normalisation (BIP-0062) to prevent signature malleability. Canonical JSON serialisation [RFC8785] SHOULD be used before signing so that verification is deterministic across serialisation differences.

Future versions of this framework may recommend additional or alternative algorithms. Implementations SHOULD be designed for algorithm agility: the algorithm identifier is carried in the signature envelope, and new algorithms can be added without protocol changes.

12.2. Key Management

Agent private keys are high-value targets. Key management practice has a larger effect on real-world security than algorithm choice.

Recommendations for key management.

Keys SHOULD be generated inside the environment that will use them, not imported from outside. This minimises the window during which the key exists in transferable form.

Keys SHOULD be stored in hardware-backed storage (HSM, TEE, secure enclave, TPM) where available. Where hardware backing is not available, keys SHOULD be protected with file permissions, memory protection, and careful handling of process memory dumps.

Keys SHOULD be rotated on a policy-defined schedule. Rotation limits the window during which a compromised key can be used. The framework does not mandate a specific rotation interval; typical values range from days (high assurance) to months (standard).

Key compromise SHOULD trigger immediate revocation of the affected agent identity. The revocation mechanism MUST support rapid propagation; high-assurance deployments SHOULD use push-based revocation.

12.3. Trust Level Calibration

The numeric trust levels (L0 through L4) are intentionally abstract. Deployments that use these levels MUST calibrate them to their specific risk tolerance. A trust level that is appropriate for internal enterprise agent communication may be inappropriate for consumer-facing financial transactions.

Calibration decisions are policy decisions, not technical decisions. The framework provides the vocabulary and the ordering; deployments

provide the semantics.

12.4. Denial of Service

Per-operation signing is a computational cost. A malicious actor submitting large numbers of signature verification requests can exhaust verifier resources. Mitigations include rate limiting at the Trust Gate, caching of verified passports, rejection of requests with obviously invalid signatures before cryptographic verification, and scaling of verification infrastructure.

Trust Gates themselves can be targets of denial of service. A Trust Gate that is slow or unavailable blocks legitimate traffic. Trust Gate implementations MUST be designed to fail-closed (deny on failure) by default, but deployments may configure fail-open behaviour for specific non-critical operations.

12.5. Interaction with Existing Security Mechanisms

The framework is designed to compose with existing mechanisms, not replace them. TLS still provides transport security. OAuth still provides client authorization for the principal. WAFs still block malformed traffic. The framework adds an agent- specific layer on top of these.

A common mistake is to assume that an agent identity replaces these other mechanisms. It does not. An agent with a valid passport still needs TLS to protect its signatures in transit. An agent acting on a user’s behalf still needs the user’s OAuth consent. An agent’s traffic can still be blocked by a WAF rule that is unrelated to identity.

13. Privacy Considerations

13.1. Attribution and Pseudonymity

Agent identities can reveal information about the principal. An identifier like "payment-bot-alice" allows an observer to infer that Alice is using an automated payment system. In contexts where this inference is a privacy concern, identifiers SHOULD be opaque (random strings rather than descriptive names).

The framework’s recommendation of an "asp_" prefix followed by random hexadecimal characters is compatible with opaque identifiers: the prefix identifies the format but conveys no information about the principal.

13.2. Linkability Across Actions

An agent that signs every action with the same identity creates a cryptographic link between those actions. An observer with access to the signatures can determine that they were produced by the same agent. In some contexts (auditability, compliance) this is desirable. In others (user privacy against observers with traffic access) it is

not.

Deployments that require unlinkability SHOULD consider techniques such as one-time identities (a new passport per action), blind signatures, or zero-knowledge proofs. These are research topics with limited standardised support; the framework does not mandate a specific approach.

13.3. Audit Trail Contents

Audit trails contain detailed records of agent activity. Depending on what the agent does, the audit trail may contain sensitive personal data, financial information, or confidential business data. Protecting the audit trail is not only a security concern but a privacy concern.

Recommendations:

Audit trails SHOULD be encrypted at rest, with access controls limiting who can read them.

Audit trails SHOULD be retained for a policy-defined period and then deleted, in accordance with applicable data protection regulation (for example, the General Data Protection Regulation in the European Union).

Audit trails SHOULD NOT include full content of requests or responses where hashes would suffice. The minimal evidence bundle described in Section evidence-layer contains only hashes, not contents, precisely to reduce the privacy surface.

13.4. Cross-Organisational Trust Portability

Cross-domain trust portability (see Section mechanism-trust-portability) can leak information. An agent’s trust level in one organisation reveals something about that organisation’s policies when presented to another organisation. Deployments SHOULD consider whether exposing trust levels across boundaries is compatible with their privacy requirements.

Possible mitigations include coarse-grained trust portability (only expose a binary trusted/untrusted decision, not the level), ephemeral trust assertions (valid only for a single verifier for a short time), and selective disclosure (expose only the minimum information needed).

14. Integration with Existing Standards

14.1. OAuth 2.0 and OpenID Connect

The framework is designed to extend OAuth 2.0 [RFC6749] and OpenID Connect, not to replace them. An agent passport can be carried as an extension claim in an OIDC ID Token or in a JWT access token. The

agent’s actions can be signed with a key that is distinct from the OAuth client credentials but referenced from the OAuth token.

Specifically, an agent identity can be integrated with OAuth as follows. The principal authenticates to an OAuth authorization server and grants authority to a client (the agent runtime). The authorization server issues an access token as normal. The agent runtime then issues an agent passport from its own internal issuing authority (or from an external issuing authority) and presents both the OAuth token and the passport to the target service. The target service verifies the OAuth token (the principal’s authorization) and the passport (the agent’s identity) independently.

This composition preserves the separation of concerns: OAuth handles authorization, the framework handles agent identity and trust. Neither replaces the other.

14.2. FAPI 2.0

FAPI 2.0 [FAPI2] defines a security profile for high-assurance APIs (banking, government, healthcare). FAPI 2.0 provides sender-constrained tokens, strong client authentication, and protection against a specific set of attacks.

FAPI 2.0’s current profile is human-centric. The sender constraint binds the token to a client, not to an agent instance. The framework described in this document complements FAPI 2.0 by adding agent identity at the instance granularity and per-operation signing on top of FAPI’s session-level guarantees.

A deployment that uses FAPI 2.0 for high-assurance API access and this framework for agent identity can satisfy both the session-level guarantees FAPI provides and the per-action agent-level guarantees the framework provides. The combination is additive.

14.3. mTLS and Sender-Constrained Tokens

Mutual TLS [RFC8705] provides a strong binding between a client certificate and the requests made over a TLS session. Sender-constrained tokens bind an access token to the mTLS certificate of the client.

These mechanisms operate at the transport layer and bind to a specific TLS endpoint. An agent’s identity needs to bind to the specific agent instance, which may share a TLS endpoint with other agents. The framework’s per-operation signing provides this finer granularity while still allowing mTLS to secure the overall transport.

14.4. Certificate Transparency

Certificate Transparency [RFC6962] provides public, append- only logs of TLS certificate issuance. The goal is to detect misissuance by certificate authorities. The same construction applies to agent passports: public transparency logs of passport issuance allow

detection of malicious or compromised issuing authorities.

A transparency mechanism for agent passports is compatible with the framework and is recommended for high-assurance deployments. Specification is out of scope for this document but is identified as a topic for future work.

14.5. OWASP AI and MCP Security Guidance

OWASP has published security guidance for AI and for the Model Context Protocol, including the OWASP MCP Security Cheat Sheet [OWASP-MCP]. The guidance covers specific controls (input validation, resource limits, authentication, logging) that deployments should implement.

The framework in this document is complementary to OWASP guidance. OWASP describes what to do; the framework describes how to structure the system so that those controls compose coherently. A deployment that follows OWASP guidance and uses the framework has both the specific controls and the structural model.

14.6. NIST AI Risk Management Framework

The NIST AI RMF [NIST-AI-RMF] provides a risk management framework for AI systems. It defines functions (Govern, Map, Measure, Manage) and outcomes for responsible AI development.

This framework is more narrow in scope than the NIST AI RMF but complements it. The NIST AI RMF’s outcomes for logging, transparency, and accountability are supported by the evidence layer and audit trail mechanisms described here. A deployment that uses this framework to implement the technical primitives can satisfy many of the technical requirements implied by the NIST AI RMF’s "Manage" function.

14.7. EU AI Act

The EU AI Act [EU-AI-Act] imposes specific requirements on providers and deployers of AI systems. The relevant articles include Article 12 (record keeping), Article 13 (transparency), Article 14 (human oversight), Article 15 (accuracy, robustness, and cybersecurity), and Article 50 (identification of AI systems).

The framework’s mechanisms map to these requirements as follows.

Article 12 (record keeping) is supported by the agent audit trail mechanism. The hash-chained, tamper-evident log with external anchoring provides the record-keeping foundation.

Article 13 (transparency) is supported by agent identity and attestation. Users and auditors can determine which agent took which action.

Article 14 (human oversight) is supported by the Trust Gate mechanism, which provides a decision point at which human policy (expressed in rules) can be enforced.

Article 15 (accuracy, robustness, cybersecurity) is supported by per-operation signing (integrity), model integrity verification (robustness of the underlying model), and trust-gated networking (cybersecurity of outbound connections).

Article 50 (identification) is supported by agent identity credentials. Each agent has a verifiable identity that can be presented to users or auditors.

Compliance evidence export (see Section mechanism-compliance-evidence-export) produces structured artifacts that map directly to these requirements for regulatory inspection.

15. IANA Considerations

This document does not request IANA action. Specific mechanisms referenced in this document may request IANA actions in their respective specifications.

1. Acknowledgements

The author thanks contributors and reviewers in the OWASP Foundation, the CIS Benchmarks working group, the IETF community, and the open-source ecosystem around the Model Context Protocol for discussions that shaped this framework.

2. Author Credentials

Raza Sharif is the Founder and Lead AI Architect of CyberSecAI Ltd. He is a Fellow of the British Computer Society (FBCS), a Certified Information Systems Security Professional (CISSP), and a Certified Secure Software Lifecycle Professional (CSSLP). He is a contributor to the OWASP MCP Security Cheat Sheet, an invited contributor to the CIS MCP Benchmark working group, and the author of multiple IETF Internet-Drafts on autonomous agent identity, transport, payment trust, and audit trail.

3. Relation to Existing Internet-Drafts

This framework references and is supported by the following Internet-Drafts, each of which specifies a particular mechanism in more detail.

* draft-sharif-mcps-secure-mcp: MCPS specification for MCP security.

* draft-sharif-openid-agent-identity: OpenID Connect claims for agent identity.

* draft-sharif-agent-payment-trust: Protocol for trust-aware agent-initiated payments.

* draft-sharif-agent-audit-trail: Format and rules for the agent audit trail.

* draft-sharif-agent-transport-protocol: Asynchronous agent messaging.

* draft-sharif-attp-agent-trust-transport: Synchronous agent-to-server transport with mandatory signing.

These drafts are individual submissions by the same author and may be adopted, merged, or revised as this framework matures.

4. Document History

draft-sharif-agent-identity-framework-00: Initial submission.

Authors' Addresses

Raza Sharif (FBCS, CISSP, CSSLP)
CyberSecAI Ltd
London
United Kingdom

Email: contact@agentsign.dev
URI: https://cybersecai.co.uk