# ExecutionRank: Verified Multi-Agent Execution via Weighted Attestation and Threshold Acceptance

Nikko Ambroselli
daybed-wile-0e@icloud.com

February 14, 2026

**Abstract**

Autonomous agent systems increasingly delegate tool invocation to remote agents discovered through registries. While this enables scale and specialization, it introduces a missing primitive: *verifiable execution trust*. Most systems accept a tool's return value as evidence of correctness or rely on an LLM to judge outputs, neither of which constitutes independent verification. We present *ExecutionRank*, a trust layer for autonomous tool invocation that combines (i) verified multi-agent execution (worker + independent verifiers), (ii) a weighted attestation graph built from auditable positive attestations, and (iii) deterministic threshold acceptance to gate consumption of results (and optionally, authorization of side effects). ExecutionRank yields a policy-tunable assurance signal for tool calls under uncertainty.

**Keywords:** autonomous agents; tool invocation; verification; attestation; reputation; EigenTrust; risk gating; audit receipts.

## 1 Introduction and Problem Context

Autonomous agent ecosystems delegate tool invocation to remote agents discovered via registries (e.g., ERC-8004). A host (H) selects an agent, invokes a tool, and consumes the result. This model lacks verifiable execution trust.

**Unverified execution.** Most systems treat "tool returned successfully" as evidence of correctness. A faulty or malicious worker can return arbitrary data.

**LLM-as-judge insufficiency.** Semantic plausibility checks are subjective and vulnerable to prompt injection and distribution shift.

**Absence of independent attestation.** Traditional distributed systems introduce independent witnesses (replication, quorum, BFT [1, 4]). Agent tool invocation typically lacks such witnesses.

**Risk of unverified invocation.** Tools may have side effects (payments, state mutation, external API calls). Accepting unverified results risks incorrect state and financial loss.

**Contributions.** ExecutionRank introduces (1) a verified execution protocol, (2) a weighted attestation graph, and (3) deterministic threshold acceptance.

## 2 System Model

### 2.1 Entities and Roles

We consider a host H, a control plane CP, agents exposing tools, and a receipt store RS.

| Role | Identity | Responsibility |
|---|---|---|
| Worker ($W$) | Agent ID | Executes tool and produces outputs |
| Verifier ($V$) | Agent ID | Independently validates outputs |
| Invoker ($I$) | Principal | Consumes result; may attest semantics (post-hoc via a |

Table 1: Roles in verified multi-agent execution.

### 2.2 Assumptions and Threat Model (Outline)

We assume RSis append-only (or tamper-evident) and that verifiers can be sampled from a pool not fully controlled by a single worker. We consider adversaries controlling subsets of workers, verifiers, or invokers (Section 7).

## 3 Protocol Overview

### 3.1 Verified Multi-Agent Execution

Given a request to invoke tool $T$ on worker $W$ with args $x$: (i) CPinvokes $T$ on $W$ to obtain $y$, (ii) computes digests $d_x \leftarrow H(x)$ and $d_y \leftarrow H(y)$, (iii) samples $k$ verifiers $V_1..V_k$ with $V_i \neq W$, (iv) collects verification receipts

$\langle$verdict, confidence, mode$\rangle$, (v) stores receipts, and (vi) decides acceptance (Section 6).

# 4 Verification Modes

Modes are ordered by strength:

- REPLAY: re-execute and compare outputs (or validate invariants for non-deterministic tools).
- CROSSCHECK: validate via alternate logic or independent sources.
- HEURISTIC: rule-based or approximate checks.

# 5 Weighted Attestation Graph

## 5.1 Graph Structure and Rationale

We construct a bipartite graph with edges from attestors to workers; only PASSattestations create edges. Worker→worker edges are disallowed to prevent mutual inflation loops.

## 5.2 Edge Weight

For attestor $A$ attesting PASSto worker $W$, define:

$$t(A, W) = \lambda_{\text{role}} \cdot \text{conf}(A, W) \cdot s(\text{mode}) \cdot g(\text{agree}) \cdot \delta(\Delta t). \tag{1}$$

Implementation: $\lambda_{\text{verifier}} = 1$, $\lambda_{\text{invoker}} = 0.5$; $s(\text{REPLAY}) = 1$, $s(\text{CROSSCHECK}) = 0.8$, $s(\text{HEURISTIC}) = 0.6$; $g$ boosts when multiple attestors concur on the same task (capped 1.2); $\delta = \exp(-\ln 2 \cdot \Delta t/\mathit{halfLife})$ with half-life 90 days.

## 5.3 Normalized Matrix C

$$\mathbf{C}_{A,W} = \frac{\max(t(A, W), 0)}{\sum_{W'} \max(t(A, W'), 0)}. \tag{2}$$

### 5.3.1 Attestor–Attestor Flow Matrix $M$ (Derived from $C$)

Let $\mathcal{A}$ be the set of attestors ($|\mathcal{A}| = n$) and $\mathcal{W}$ the set of workers ($|\mathcal{W}| = m$). Let $C \in \mathbb{R}^{n\times m}$ be the row-stochastic attestor→worker matrix defined in Section 5.3.

Define the worker column-sums:

$$s_w \;=\; \sum_{a\in\mathcal{A}} C_{a,w} \quad \forall w \in \mathcal{W}.$$

Define a worker→attestor redistribution matrix $Q \in \mathbb{R}^{m\times n}$:

$$Q_{w,b} \;=\; \begin{cases} \frac{C_{b,w}}{s_w}, & s_w > 0, \\ \frac{1}{n}, & s_w = 0, \end{cases} \qquad \forall w \in \mathcal{W},\ b \in \mathcal{A}.$$

Then the induced attestor→attestor flow matrix is:

$$\begin{aligned} M &= CQ \\ &= CD^{-1}C^{\top}. \end{aligned}$$

where $D = \text{diag}(s_1, \ldots, s_m)$ (and $D^{-1}$ is interpreted with the $s_w = 0$ convention above).

Equivalently, entrywise:

$$M_{a,b} = \sum_{w\in\mathcal{W}} C_{a,w} \frac{C_{b,w}}{\sum_{a'\in\mathcal{A}} C_{a',w}}$$
(with 0/0 handled by the $s_w = 0$ case).

**Row-stochasticity.** If $C$ is row-stochastic and $Q$ is defined as above, then $M$ is row-stochastic: $\sum_b M_{a,b} = 1$ for all $a$.

## 5.4 Reputation via EigenTrust-Style Iteration

Following the EigenTrust approach [3]:

$$\mathbf{R} = (1 - \alpha)\mathbf{p} + \alpha\, \mathbf{M}\, \mathbf{R}. \tag{3}$$

# 6 Acceptance Threshold Logic

Let $\mathcal{A}$ be PASS attestors for a task, and $R_A$ their reputations.

**Definition 1** (Acceptance Predicate)**.**

$$\textsf{accepted} \iff \left(\sum_{A\in\mathcal{A}} R_A \geq \tau\right) \wedge (|\mathcal{A}| \geq 1)\,. \tag{4}$$

*$\mathcal{A}$ is the set of* verifier *attestors that returned PASSfor the task (invoker attestation is asynchronous and does not affect immediate acceptance). Unknown attestors receive baseline rank 0.2. Default $\tau = 0.5$, $\alpha = 0.85$.*

# 7 Security and Adversarial Considerations

## 7.1 Sybil Attacks

Sybil attacks [2] allow a single entity to control multiple identities. Attack / mitigation / residual risk.

## 7.2 Collusion Rings

Attack / mitigation / residual risk.

## 7.3 Low-Risk Farming

Attack / mitigation / residual risk.

**Algorithm 1** VerifiedExecution (Control Plane)

**Require:** Worker $W$, tool $T$, args $x$, verifier count $k$, policy $\Pi$
**Ensure:** Decision accepted/rejected, receipts

$y \leftarrow \text{CP.}\mathsf{Invoke}(W, T, x)$
$d_x \leftarrow H(x)$; $d_y \leftarrow H(y)$
$\mathcal{V} \leftarrow \text{CP.}\mathsf{SelectVerifiers}(k, \Pi, \text{exclude} = W)$
**for all** $V \in \mathcal{V}$ **do**
    $(v, c, m) \leftarrow \text{CP.}\mathsf{Invoke}(V, \mathsf{verify\_result}, \langle \text{taskId}, W, T, d_x, d_y \rangle)$ ▷ $d_x = H(x)$, $d_y = H(y)$; verifier returns $\{\text{verdict}, \text{confidence}, \text{mode}\}$
    $\text{CP.}\mathsf{AppendVerificationReceipt}(V, W, T, d_x, d_y, v, c, m)$
**end for**
$\text{CP.}\mathsf{AppendExecutionReceipt}(W, T, d_x, d_y)$
**return** $\text{CP.DecideAcceptance(taskId)}$

**Algorithm 2** ComputeAttestorReputation

**Require:** PASS receipts, damping $\alpha$, seed $\mathbf{p}$, tolerance $\varepsilon$, maxIters
**Ensure:** Attestor reputation vector $\mathbf{R}$

Build $t(A, W)$ from PASS receipts
Normalize rows to form $\mathbf{C}$
Derive $\mathbf{M}$ from $\mathbf{C}$ (attestor agreement flow through shared workers)
$\mathbf{R} \leftarrow \mathbf{p}$
**for** $i = 1$ to maxIters **do**
    $\mathbf{R}_{\text{next}} \leftarrow (1 - \alpha)\mathbf{p} + \alpha(\mathbf{MR})$
    **if** $\|\mathbf{R}_{\text{next}} - \mathbf{R}\|_1 < \varepsilon$ **then break**
    **end if**
    $\mathbf{R} \leftarrow \mathbf{R}_{\text{next}}$
**end for**
Normalize $\mathbf{R}$ (e.g., $\max(\mathbf{R}) = 1$)
**return R**

### 7.4 Invoker-Only Inflation

Attack / mitigation / residual risk.

### 7.5 Receipt Integrity and Omission

Attack / mitigation / residual risk.

## 8 Deployment and Integration Models

Centralized control plane; decentralized receipt sets; enterprise tool invocation; payment-gated execution.

## 9 Implementation Notes

The attest-substrate control plane implements ExecutionRank: discovery via ERC-8004 subgraph (search_agents, list_tools); policy filters (MIN_REPUTATION, ALLOW_RISK_CLASS); invoke always runs worker + verifiers; receipts stored in SQLite; ranking refreshed periodically (default 60s). Verifier selection: allowlist (capped at 10 agents) or search-based; worker excluded. Verifier tool: `verify_result` (fallback `validate`); input `{taskId, workerAgentId, workerTool, workerArgsHash, workerOutputHash}`; output `{verdict, confidence, mode}`. Digests: SHA-256 of canonical JSON.

## 10 Evaluation Plan

Fault injection; lazy verifiers; collusion simulation; latency/cost vs assurance; convergence under sparsity.

## 11 Limitations and Future Work

Negative evidence; risk-class reputation; verifier diversity guarantees; receipt consensus; privacy; incentives.

## A Notation

## B Illustrative Receipt Schemas

### B.1 Execution Receipt (Example)

```
{
  "receipt_id": "uuid",
  "task_id": "uuid",
  "principal": "anonymous",
  "agent_id": 12345,
  "tool_name": "coinbase_price",
  "tool_id": "12345::coinbase_price",
  "args_digest": "sha256-hex...",
  "result_digest": "sha256-hex...",
  "outcome": "ok",
  "started_at": "ISO8601",
  "ended_at": "ISO8601",
  "latency_ms": 42
}
```


Figure 1: Execution Receipt (attest-substrate schema). Digests are SHA-256 of canonical JSON.

| Symbol | Meaning |
|---|---|
| H | Host |
| CP | Control plane |
| RS | Receipt store |
| $W$ | Worker agent |
| $V$ | Verifier agent |
| $A$ | Attestor |
| $\mathbf{C}$ | Attestor→worker matrix |
| $\mathbf{M}$ | Attestor→attestor flow matrix |
| $\mathbf{R}$ | Attestor reputation vector |
| $\mathbf{p}$ | Seed distribution |
| $\alpha$ | Damping |
| $\tau$ | Acceptance threshold |

Table 2: Notation summary.

# References


[1] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22–25, 1999*, pages 173–186. USENIX Association, 1999.

[2] John R. Douceur. The sybil attack. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems: First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7–8, 2002, Revised Papers*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer, 2002.

[3] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20–24, 2003*, pages 640–651. ACM, 2003.

[4] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.