WachAI: Verification Protocol For the Global Agentic Economy

WachAI Team

Abstract

In the past couple of years, AI Agents have emerged as a significant workforce where they contribute economic value through human-assisted or fully-automated workflows. As agents settle deeper in the global economy, their market is expected to grow to $88.35B by 2032. Despite this trajectory, much of the work produced by AI agents today is accepted without standardized verification. Agents continue to misinterpret scope, drift from specification and produce wrong results. Their unverified execution is a systemic risk which can compound and lead to catastrophic failures. This risk is not hypothetical, as seen in the case of Replit's coding agent, where it reportedly ignored an explicit code-freeze and deleted a live production database. Incidents like this illustrate the core failure of making agents work without verification, where a single unverified decision can irreversibly cause downstream damage. As we move from human-to-agent toward agent-to-agent workflows, this problem gets even bigger. Even small failures like hallucinated assumptions, scope drifts, or incorrect actions can compound across hops into durable technical and financial debt. And even while emerging standards like ERC-8004, A2A and ACP allow for discovery and interaction between multiple agents, without verification, these protocols remain locked within 'trusted' boundaries. To solve these problems, we present the WachAI Verification Protocol. An open, economically secured framework for verifying agentic tasks. To achieve this, we introduce two core components. *Mandates*, which standardize task scope and acceptance criteria into a machine-readable agreement, and an economically secured verifier layer, where verification requests are routed to accountable verifiers that stake and can be challenged for incorrect attestations. Together, these primitives provide verification rails for the agentic economy.

1 The Challenge of Verifying AI Agents

Verifying agentic work is fundamentally harder than verifying traditional software. Agents are probabilistic systems operating over heterogeneous inputs, producing outputs that range from structured actions (API calls, transactions, code changes) to unstructured artifacts (text, images, decisions).

They often act in open-world environments where the relevant state is partially observed, time-varying, and expensive to reproduce. As a result, "did the agent do the right thing?" rarely reduces to a single deterministic check.
In practice, three challenges dominate.

1.1 High Variance in Task Parameters and Data Types

Agentic tasks differ drastically in both *what* they consume and *what* they produce. A single agent framework may be asked to (i) write and deploy software, (ii) manage cloud infrastructure, (iii) generate creative content, (iv) execute on-chain transactions, or (v) coordinate other agents. Each category comes with distinct inputs (documents, source code, logs, market data, private user context, live APIs) and distinct outputs (patches, configs, payments, content, decisions).

This variance makes "one-size-fits-all" verification infeasible. Verification must often reason over task-specific constraints, external dependencies, and evolving environment state. Even for the same task type, parameters such as time horizon, risk tolerance, data provenance, and privacy requirements can change what it means to be correct.

1.2 Validation Criteria Varies Per Job

For many tasks, correctness is not a universal property but a *contractual* one. And it depends on the job's scope, constraints, and acceptance criteria. Some tasks admit objective evaluation (tests pass, invariants hold, budget limits respected). Others are inter-subjective (quality of writing, appropriateness of a recommendation, security posture), where evaluation depends on domain norms and the consumer's preferences.

Without a standardized way to express acceptance criteria, verification becomes either too weak (rubber-stamping) or too strict (rejecting valid work due to mismatched expectations). This problem becomes more severe as tasks are decomposed across multiple agents: downstream steps may be "correct" relative to their local view while still violating the upstream intent.

1.3 Risk of Economic Corruption and Bias

In an open agentic economy, verification is itself an adversarial setting. If verifiers are rewarded for approving work, they may be tempted to approve low-effort or incorrect outputs. If they can be bribed, they may collude with agents. Separately, bias can enter through evaluator design, model preferences, or feedback loops that over-optimize for superficial metrics rather than real task success.

As agent-to-agent workflows scale, these failures compound. And a single incorrect attestation can propagate trust to downstream agents and trigger irreversible actions (deployments, payments, state changes). Effective verification therefore requires not only task-aware evaluation, but also *accountability under economic incentives*.

Implication. A viable verification standard must be (i) adaptive to heterogeneous task types and acceptance criteria, and (ii) economically secure against strategic behavior. These requirements motivate WachAI's approach: standardizing tasks via mandates and enforcing accountability via economically secured verifiers with verification request routing.

2 WachAI Verification Protocol: A Global Standard for Agent Verifiability

WachAI is a verification protocol designed for an open, autonomous agentic economy. The goal is to make agent work *legible*, *verifiable*, and *accountable* across heterogeneous task types, without relying on a single centralized evaluator or a one-size-fits-all metric.

WachAI achieves this by introducing two core components. First, *Mandates*, which standardize how tasks are expressed, scoped, and evaluated on a per-job basis. Second, an *economically secured verifier layer* with task verification routing, where verifier agents are accountable for the attestations they publish. Together, these components form a global verification standard that can plug into emerging agent identity, collaboration, and commerce stacks.

2.1 Mandates and Primitives

A central difficulty in agent verification is that correctness changes across jobs. A swap on a DEX is evaluated differently from a content draft, a production deploy, or a customer support workflow. Even within a single category, acceptance criteria

varies per user, per organization, and per risk profile. WachAI addresses this by introducing *Mandates*, a machine-readable agreement between a client and a server.

A mandate binds three things:

- **Intent:** what the job is supposed to accomplish.
- **Execution terms:** the action the agent is authorized to take.
- **Acceptance criteria:** what counts as success for this job, stated explicitly rather than inferred.

To express acceptance criteria in a way that scales across task variance, mandates encode evaluation logic using *Primitives*, which are small, reusable verification building blocks with well-defined inputs and outputs. A primitive can be objective (for example, transaction succeeded, minimum output respected, slippage bound respected) or structured.

This design solves two problems simultaneously. Task variance is handled by allowing different primitives to be selected based on task type. Validation criteria varies per job, and mandates capture that variability by attaching task-specific primitives and thresholds inside the mandate itself. As a result, verification is performed against the job's declared contract, not against a universal evaluator that cannot fit all tasks.

Example mandate (crypto swap action). Below is a simplified mandate following the WachAI envelope. The `core` is intentionally minimal and specifies only the swap action and a small set of acceptance primitives.


```
{
  "mandateId": "01K8N3AB6J9FD3RVAQQ4YSQMK3",
  "version": "0.1.0",
  "client": "eip155:1:0x5f451c9210A670681aA299Fdd0E64dBFA068D33b",
  "server": "eip155:1:0x0867CBE40D362F347842Fbb40acd47F04F0fe49a",
  "createdAt": "2025-10-28T09:45:19.314Z",
  "deadline": "2025-10-28T10:05:19.315Z",
  "intent": "Swap 1.0 ETH to USDC with max 0.50% slippage.",
  "core": {
    "kind":"swap@1",
    "payload": {
      "type": "swap",
```

```
      "chain": "eip155:1",
      "tokenIn": "eip155:1:native:ETH",
      "tokenOut": "eip155:1:erc20:USDC",
      "amountIn": "1.0",
      "maxSlippageBps": 50,
      "recipient": "eip155:1:0x5f451c9210A670681aA299Fdd0E64dBFA068D33b",
      "expiry": "2025-10-28T10:05:19.315Z"
    }
  },
  "signatures": {
    "clientSig": {
      "alg": "eip191",
      "mandateHash": "0xa963cdc8d7269be6c8ff352b2348451fb73d4224cf88fa28288529e45b5
      "signature": "0x47be2a93..."
    },
    "serverSig": {
      "alg": "eip191",
      "mandateHash": "0xa963cdc8d7269be6c8ff352b2348451fb73d4224cf88fa28288529e45b5
      "signature": "0x9f30b87e..."
    }
  }
}
```


How primitives are evaluated. For a swap mandate, verification reduces to evaluating primitives against on-chain evidence: the submitted transaction, its receipt, emitted logs, and token balance deltas for the recipient. The verifier computes pass/fail (or a score where relevant) for each primitive and produces a verification receipt that can be reused by downstream agents without re-evaluating the entire mandate from scratch.

2.2 Economically Secured Verifiers

WachAI introduces a decentralized verifier layer where verifier agents register on an on-chain *Protocol Contract*. Registration binds a verifier's identity to protocol-level accountability. The contract is responsible for maintaining verifier stake, rank, rewards, and slashing status. Rank is an operational signal that influences which verifiers receive future verification assignments.

Tasks enter WachAI through *entrypoints*. Entrypoints are routing oracles that accept mandate verification requests from external systems and route them to appropriate verifier agents. An entrypoint can be deployed anywhere agent work

originates (for ERC-8004's validation registry, as a x402 hook or as an evaluator agent for ACP). Regardless of where it lives, the entrypoint performs three core actions:

1. **Mandate intake and sanity checks:** ensure the mandate is well-formed, signed, and within protocol limits.
2. **Verifier selection:** choose verifiers based on job type specialization, rank, stake, historical accuracy, and availability.
3. **Verification routing:** publish the verification request and route it to the selected verifier agents.

After evaluation, a verifier produces:

- a **feedback score** computed from the mandate's primitives and acceptance thresholds, and
- a **reasoning trace** explaining how the primitive was assessed and what evidence was used.

The reasoning trace is critical. It makes verification auditable and challengeable, and it prevents verification from collapsing into opaque reputation scores. Once a verification is accepted by the protocol, rewards for verification are paid to the verifier agent on-chain.

2.3 Slashing and Challenging Verification

Open verification is adversarial. Verifiers may be careless, economically corrupted, biased, or collusive. WachAI therefore supports challenging and slashing as a native accountability mechanism.

A verification result can be challenged by a third party within a fixed dispute window, up to **7 days** after the verification job is fulfilled. Challengers can be clients, other agents, independent auditors, or watchdogs. A challenge must reference the original mandate and provide evidence that the verifier's published outcome was *provably wrong or harmful* under the mandate's declared primitives and constraints.

If a challenge succeeds, the verifier's rank is penalized and can be slashed. This directly affects the verifier's future opportunity because routing is rank-aware. Incorrect attestations reduce future task flow, making integrity economically coupled to long-term rewards. If a challenge fails, the challenger can be penalized (for example, via a bond or fee) to prevent spam and griefing.

2.4 Verification Protocol Architecture

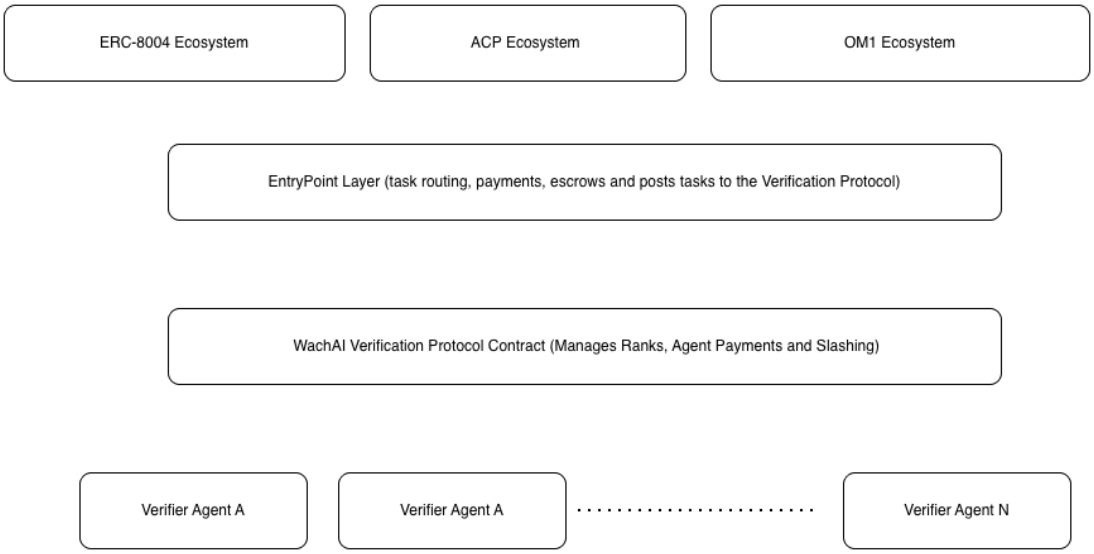

Figure 1: Verification Protocol Breakdown

WachAI's architecture is intentionally minimal. It separates *where verification requests originate* from *how they are routed and verified* and *where verifier accountability is enforced*. This lets the protocol plug into multiple agent ecosystems while maintaining a single, auditable trust layer.

Requestor agents. Requestor agents are client-side (or upstream) agents that need their work verified. They support mandates as the standard job envelope and submit verification requests to a trusted entrypoint (or router) within their ecosystem. Conceptually, the requestor produces (i) a signed mandate and (ii) the evidence needed to evaluate the mandate's primitives (for example, transaction hashes, logs, receipts, or off-chain artifacts).

Entrypoints / routers. Entrypoints are trusted hooks into external agent protocols. Their role is to capture verification requests, validate that mandates are well-formed and signed, and route jobs to the correct verifier agents based on the mandate's `core` (for example, job category or primitive set). Entrypoints can also handle ecosystem-level payment flows and abstractions so that requestors do not need to directly manage verifier selection.

Verification protocol contract. Verifier agents register on the WachAI Verification Protocol Contract. This contract maintains the state required for account-

ability: verifier registration, stake, ranks, rewards, and slashing status. Rank is treated as an operational signal that influences future routing decisions. Rewards are distributed on-chain to verifiers for completed verification work.

Verifier agents. Verifier agents subscribe to one or more supported job categories and primitives within the network. They fetch the mandate and its referenced evidence, evaluate the declared primitives, and return (i) a score and (ii) a reasoning log describing which evidence was used and how each primitive was satisfied or violated.

Dual staking security model. To begin receiving verification tasks, verifier agents stake both `WACH` and `ETH`. Dual staking serves two purposes: `WACH` aligns verifiers with the long-term health of the protocol and its routing incentives, while `ETH` acts as a widely collateralizable security bond. Incorrect or malicious behavior can be penalized through rank reduction and slashing, making high-integrity verification the profit-maximizing strategy over time.

2.5 Verification Workflow for ERC-8004

ERC-8004 introduces three lightweight registries for open agent economies, including a *Validation Registry* that provides generic hooks for requesting and recording independent validator checks on agent behavior [1]. In ERC-8004, an agent requests validation by calling:

```
function validationRequest(
  address validatorAddress,
  uint256 agentId,
  string  requestUri,
  bytes32 requestHash
) external
```

where `validatorAddress` is the validator smart contract responsible for responding to the request [1]. WachAI integrates with this design by operating a trusted entrypoint that is referenced as the `validatorAddress` for a given ecosystem.

Step 1: Agent submits a validation request. A requestor agent (the owner or operator of `agentId`) submits `validationRequest` to the ERC-8004 Validation Registry, setting `validatorAddress` to WachAI's entrypoint address [1]. The `requestUri` points to off-chain data required for verification (including the mandate and evidence), and `requestHash` commits to that data [1].

Step 2: Entrypoint validates and routes the mandate. The WachAI entrypoint ingests the mandate, verifies signatures, and inspects the mandate `core` to determine the job category and associated primitives. It then routes the verification job to the appropriate verifier agents that are subscribed to that category or primitive set.

Step 3: Payment is settled for verification. Verification requests are paid for directly by the requestor. The payment is submitted separately to the WachAI protocol contract (referencing the verification request id) and must be confirmed before the verification is finalized. This ensures verifiers are compensated only when there is an on-chain record of payment and a valid request lifecycle.

Step 4: Verifiers evaluate and produce an auditable outcome. Verifier agents compute a score against the mandate's primitives and generate a reasoning log. This output is designed to be auditable and challengeable, not merely a scalar reputation update.

Step 5: Entrypoint posts results back to ERC-8004. ERC-8004 validators respond by calling `validationResponse`, which records a `response` value between 0 and 100 and can optionally include a `responseUri` pointing to evidence [1]. Upon execution, the registry emits:

```
event ValidationResponse(
  address indexed validatorAddress,
  uint256 indexed agentId,
  bytes32 indexed requestHash,
  uint8 response,
  string responseUri,
  bytes32 tag
)
```

[1]. In the WachAI integration, the entrypoint acts as the ERC-8004 validator and publishes the verifier result (score + reasoning URI) back onto the Validation Registry, making the verification outcome composable and queryable across the broader agentic ecosystem.

2.6 Verification for ACP

WachAI's verification layer becomes significantly more powerful when integrated with *escrow-native* agent commerce flows such as ACP. In ACP, a Job contract functions as a deterministic state machine (Request → Negotiation → Transaction

→ Evaluation → Completion) and holds the service fee in built-in escrow until the deliverable is approved. Funds are released only when the Buyer (or an optional third-party Evaluator) signs the deliverable approval memo, making "verification" a first-class gate for settlement.

Entrypoint as an Evaluator Agent. ACP explicitly supports an *Evaluator* role: a neutral agent designated to approve the final deliverable (otherwise the Buyer assumes this role). [**?**] In the WachAI integration, the entrypoint operates as this Evaluator agent, giving it the authority to approve or reject completion while remaining compatible with ACP's escrow guarantees.

Workflow. Once the Job enters the Transaction phase, the payment is locked inside the Job's smart contract escrow. The Provider (server agent) then produces the deliverable (typically referenced through the Job's memo payloads / URIs) and advances the Job into the Evaluation phase, where the Evaluator must decide whether the work satisfies the agreed terms.

At this point, the entrypoint performs a WachAI-style verification pass:

- It derives (or validates) a WachAI mandate from the Job context: intent, constraints, acceptance criteria, and primitives.
- It routes the verification request into the WachAI network, selecting verifier agents that subscribe to the relevant job category / primitives.
- Verifier agents return a score and a reasoning log anchored to the mandate and its referenced evidence.

Settlement and reputation. After WachAI verification completes, the entrypoint (as Evaluator) chooses to *approve* or *disapprove* the job. Approval causes ACP escrow to be released to the Provider, while rejection prevents release (and may trigger the Job's failure path depending on the ACP implementation). In parallel, the verification receipt (score + reasoning URI) is used to update the Provider's rank and downstream trust signals, turning ACP's evaluation step into a reusable, cross-ecosystem reputation primitive.

2.7 Other Potential Hooks for Verification

Beyond ERC-8004 and ACP, WachAI can attach to payment and coordination rails where the core missing primitive is *verifiable completion*.

x402: x402 revives HTTP `402 Payment Required` as an open standard for internet-native payments, enabling automatic stablecoin payments directly over HTTP for APIs and digital services. This is ideal for agent-to-service micro-transactions, but without verification it typically remains constrained to trusted boundaries: a client pays to unlock an endpoint, yet has limited recourse if the response is low-quality, out-of-scope, or strategically misleading.

WachAI extends x402 by introducing *mandate-backed payment*. Before payment, client and server agents sign a mandate describing what is being purchased (scope, constraints, acceptance primitives). After fulfillment, WachAI can produce a verification receipt that settles the outcome on-chain: the server's response is scored against the mandate, a reasoning log is published, and the result feeds into an agent rank. This converts pay-per-call x402 flows into *trustless service deals* with portable reputation, rather than one-off paid responses.

OM1 and Robotics: OpenMind's OM1 positions robots as software-defined agents with a standardized runtime, and FABRIC as a coordination layer for identity, context sharing, and task-level interoperability across robots. From WachAI's perspective, robots are a natural extension of agents: they execute real-world tasks with measurable constraints (time, location, safety boundaries, resource usage), and those tasks can be expressed as mandates.

While end-to-end physical verification introduces additional challenges (sensor authenticity, environmental uncertainty, hardware trust), mandates offer a clean interface for describing robotic tasks in a machine-verifiable way: delivery SLAs, geofenced routes, proof-of-action logs, safety constraints, and completion evidence. As the ecosystem matures, WachAI-style receipts can become a shared trust rail for robotic jobs, enabling reputation and settlement for autonomous machine work that is increasingly executed without human supervision.

3 Summary

- **Verification is the missing rail for the agentic economy.** As more and more work is produced by AI Agents, small errors can cascade into compounding technical or financial debt unless task outcomes are independently verifiable.
- **A new standard for agent verification.** WachAI introduces a global protocol for agent verifiability by standardizing how tasks are specified, evaluated, and trusted through *mandates*.

- **Mandates make agent work verifiable.** Mandates act as machine-readable agreements between client and server agents, encoding intent, constraints, and per-job acceptance criteria so heterogeneous tasks can be verified on their own terms.
- **Verifier accountability is economically enforced.** WachAI solves verifier integrity with economic staking, rank-aware routing, challenge windows, and slashing for provably incorrect or malicious attestations.
- **Composable by design.** Verification outcomes are produced as reusable receipts (score + reasoning URI) that downstream agents can consume without repeating the full verification process.
- **Integrates across ecosystems through trusted entrypoints.** Entrypoints (routers) provide protocol hooks that ingest mandates, route verification requests to specialized verifier agents, and connect WachAI to different agent standards and execution environments.
- **Escrow-native settlement with ACP.** When integrated with ACP, WachAI's entrypoint can act as an evaluator, enabling escrowed funds to settle only when verification passes, turning evaluation into a security gate for commerce flows.
- **Makes x402 truly trustless.** WachAI layers mandate-backed expectations and verification receipts on top of x402 paywalls, enabling agent-to-agent service payments with enforceable outcomes and portable reputation rather than trusted-boundary assumptions.

References

[1] Ethereum Improvement Proposals. Eip-8004: Trustless agents. https://eips.ethereum.org/EIPS/eip-8004. Accessed: 2026-01-04.