InterSAGE

The Secure and Verifiable Interoperability Protocol for An Internet of Agents

A Paper from the DeepKernel Lab

Zhenhua Zou Sheng Guo Qiuyang Zhan
Lepeng Zhao Shuo Li Zhuotao Liu†
†Corresponding author: zhuotaoliu@tsinghua.edu.cn

August 14, 2026

Abstract

The emerging Internet of Agents—a global environment in which LLM-powered agents discover peers, negotiate trust, invoke tools, and delegate tasks across organizational boundaries—is currently being standardized from the communication layer upward. Protocols increasingly specify how agents exchange messages, but not how an agent proves what it is, what it is authorized to do, whether its advertised capabilities are genuine, or how its actions remain accountable after delegation. We argue that this leaves the ecosystem without the foundation for *secure interoperability*.

We present InterSAGE, a trust-native protocol suite that supplies this trust substrate alongside existing Internet and agent communication protocols. InterSAGE is organized into four layers—Persistent Identity (L0), Discovery (L1), Trust Negotiation (L2), and Accountability (L3)—covering nine security aspects across the agent lifecycle. Its core contribution is a set of four layer-aligned design primitives. First, Agent Identity Cards provide persistent agent identity with four-dimensional binding across developer, code package, operator, and deployment context. Second, capability-aware discovery turns skill and tool advertisements into DID-bound Verifiable Credential manifests, so discovery results are verified for issuer provenance, subject binding, permission alignment, and freshness before interaction begins. Third, trust negotiation combines monotonic capability attenuation with two-tier access control, making least privilege a signed structural invariant while preserving application-level policy independence. Fourth, kernel-mediated cryptographic audit trails bind usage, delegation, and execution traces to agent identity without requiring a consensus ledger.

InterSAGE does not replace existing agent protocols such as MCP, A2A, ANP, or AG-UI; it defines the missing trust-relevant primitives that those protocols need: AIC capability boundaries, DID-bound manifest VCs, least-privilege session tokens, and signed execution traces. This separation ensures that communication protocols can evolve independently, while trust semantics remain explicit, portable, and verifiable. We compare InterSAGE against 50+ related efforts across agent protocols, decentralized identity, OAuth/OIDC extensions, zero-trust governance, delegation systems, and audit architectures. We show that InterSAGE is the only architecture that jointly enforces persistent identity, capability-aware discovery, trust negotiation, and accountability as a single four-layer trust substrate for the Internet of Agents.

1 Introduction

The emergence of LLM-powered autonomous agents marks a qualitative shift in how software interacts with the world. Unlike traditional microservices that execute deterministic APIs, these agents reason over natural

language, invoke tools dynamically, and collaborate with other agents to accomplish complex, multi-step tasks [1, 2, 3, 4]. Industry analysts project that by 2029 agentic AI will autonomously resolve a large share of common customer service issues without human intervention [5], and major cloud providers have already released agent-to-agent communication protocols [6, 7, 8] to enable such collaboration at scale.

This trajectory evolves toward an *Internet of Agents* (IoA) [4, 9, 10]—a global network where billions of heterogeneous agents discover one another, negotiate task parameters, exchange value, and compose into ad-hoc workflows across organizational boundaries. Multiple research groups have begun designing protocol stacks for this vision: Fleming et al. propose reference layers for agent communication (L8) and semantics (L9) atop TCP/IP [11]; the Agent-OSI project outlines a six-layer reference architecture including settlement and provenance [10]; and comprehensive surveys catalog the growing zoo of protocols spanning agent-to-agent (A2A), agent-to-tool (MCP), and related interaction axes [12, 13, 14].[1]

The missing trust layer. A striking pattern emerges across these efforts: they focus predominantly on *how agents communicate* where security is again treated as the second-class citizen. Yet the agent threat landscape is qualitatively different from the traditional networking and computer systems. Agents operate with delegated authority, make non-deterministic decisions, and dynamically compose into workflows whose authorization requirements cannot be statically pre-computed. An agent that impersonates a supply-chain optimizer, escalates its capabilities beyond delegation scope, or repudiates a financial commitment can cause cascading damage that no transport-layer encryption or OAuth token can prevent [16, 17, 18]. A first wave of *trust-layer* proposals has begun to address this gap—Microsoft's AgentMesh [19], which its documentation describes as "SSL for AI agents"; Huang et al.'s unified zero-trust architecture for the agentic web [20]; AIP [21], which fuses identity, attenuation, and provenance into invocation-bound capability tokens; and Ramachandran and Mishra's enterprise identity-aware governance framework [22]. These efforts converge on the diagnosis but diverge in mechanism: each enforces least-privilege through *runtime* policy evaluation, behavioral attestation, or per-call Datalog rather than through structural invariants signed into the agent's credential at issuance. In the language of classical protocol design, the IoA still lacks its *trust plane*—the security substrate that binds identity, authorization, and accountability into a coherent whole that holds even when downstream policy engines, attestors, or operators are misconfigured or compromised.

Lessons from our prior work. In BlockA2A [23], we presented a unified trust framework for multi-agent systems, integrating decentralized identifiers (DIDs), blockchain-anchored ledgers, and smart-contract-enforced access control. The framework demonstrated that security *can* be systematically layered onto agent collaboration protocols such as A2A [6]. However, its reliance on blockchain infrastructure introduced deployment constraints: on-chain transaction latency, gas costs, and the requirement for a shared ledger among all participants limited applicability to environments where blockchain nodes are available and economically viable. The core security *ideas*—cryptographic agent identity, capability-bounded access control, immutable audit trails, and defense orchestration—remain sound, but their *realization* must be generalized beyond any specific type of infrastructure.

This paper: InterSAGE. We present InterSAGE, the first trust-native protocol suite for secure and verifiable agent-to-agent interoperability. Rather than proposing another communication stack, InterSAGE provides the *trust layers* that any communication protocol (MCP, A2A, or future designs) can build upon. Its four layers address nine critical aspects of agent-native security:

1. **Layer 0 — Agent Identity:** *persistent identity* via cryptographic agent identity cards (AICs) with four-dimensional binding to developer, code package, operator, and deployment context.
2. **Layer 1 — Discovery:** *capability-aware discovery* in which each skill and tool is a signed Verifiable

[1] In Ehtesham et al., "ACP" denotes the Agent Communication Protocol (REST/HTTP), not the Agent Client Protocol [15].

Credential (VC) bound to the agent's DID. Manifest VCs are presented during discovery and verified for supply-chain authenticity, subject binding, and permission alignment.

3. **Layer 2 — Trust Negotiation:** *authentication* via mutual attestation, *delegation* via chains with monotonic capability attenuation, and a two-tier *access control* model.
4. **Layer 3 — Accountability:** *token-usage tracing*, *payment* primitives, and *action accountability* via identity-signed execution traces and non-repudiation.

InterSAGE is guided by five design principles (detailed in §3.1) and introduces four novel design primitives. We summarize each below using a uniform pattern—shared goal with prior work, the specific divergence, and the operationally observable consequence—a pattern we apply throughout §9 when contrasting InterSAGE with related systems.

- **Persistent Identity (L0).** An Agent Identity Card (AIC) cryptographically binds four identity dimensions—developer, code package, operator, and operational context—into a single verifiable credential. The goal of giving each agent a verifiable persistent identity is shared with SPIFFE workload identities, OAuth 2.0 client credentials, W3C DIDs, OIDC for Agents (OIDC-A) [24], the OpenID Foundation's strategic agenda for agentic IAM [25], and AIP's capability tokens [21]; the divergence is that each of those models binds only a single dimension—workload instance, client application, holder key, or token issuer—whereas the AIC binds all four dimensions independently and signs their conjunction. The consequence is that an attacker who compromises the operator's deployment cannot impersonate a different developer or substitute a different code package, because each dimension carries its own independently verifiable signature.
- **Capability-Aware Discovery (L1).** Each skill and tool an agent advertises is a signed Verifiable Credential (VC) whose `credentialSubject.id` is the agent's own DID—making it non-transferable and replay-resistant. Skill VCs are issued by skill distributors, tool VCs are issued by tool providers, and both can carry optional GAR endorsements before being presented in the agent's registration record during discovery. The goal of making agents discoverable is shared with ANP's DID-based discovery [8], A2A's well-known Agent Cards [6], and registry-style agent catalogs; the divergence is that InterSAGE treats discovery results as *verified capability credentials* rather than self-declared descriptions. Requesters verify each manifest VC for supply-chain authenticity, subject binding to the presenter's AIC, and permission alignment against the agent's capability boundary $S_{\max}$. The consequence is that discovery cannot become an implicit escalation channel: an agent cannot advertise skills it does not hold, tools it has not been authorized to use, or permissions it was not granted, because every claim is cryptographically bound to the agent's identity and independently verifiable at discovery time.
- **Trust Negotiation (L2).** InterSAGE combines a four-stage monotonic attenuation chain (Developer → Operator → GAR → Runtime) with a two-tier access control model that separates infrastructure-tier cryptographic verification from application-tier declarative policy. The goal of bounded authorization is shared with AIP's Biscuit/Datalog attenuation [21], Saavedra's Delegation Grants [26], AgentMesh's policy plane [19], OAuth 2.0 token exchange [27], and enterprise governance frameworks [22]; the divergence is that InterSAGE encodes the capability boundary into signed credentials and then evaluates application policy on an independent path. The consequence is that no policy edit, JIT decision, or misconfigured PDP downstream of issuance can grant capabilities the preceding stage did not authorize, while an application-tier regression still cannot weaken cryptographic identity verification.
- **Accountability (L3).** Every agent action is recorded in a tamper-evident hash chain and signed by the agent's kernel-protected private key. The goal of immutable accountability is shared with blockchain-anchored ledgers (like BlockA2A [23]) and centralized audit logs; the divergence is that InterSAGE achieves non-repudiation via trusted kernel-mediated cryptography rather than distributed consensus

or trusted third parties. The consequence is that agents can prove their execution history and attribute costs across boundaries in completely decentralized or resource-constrained environments where global ledgers are unviable.

Scope and positioning. InterSAGE is a *positioning paper*: it presents the conceptual framework, protocol architecture, and design rationale at a level suitable for guiding subsequent protocol specifications, formal verification, and systems implementation. Detailed cryptographic proofs, performance benchmarks, and production deployment experiences are deferred to companion publications currently in preparation.

Contributions. In summary, this paper makes the following contributions:

1. We articulate the case for *secure interoperability* as the foundational missing layer in the Internet of Agents, distinct from communication interoperability and grounded in a dedicated trust substrate.
2. We present InterSAGE, a four-layer security protocol suite that coherently addresses nine security aspects: persistent identity, capability-aware discovery, authentication, delegation, access control, payment, semantic tagging, token-usage tracing, and action accountability. The first five receive detailed protocol-level treatment; the latter four are specified as framework-level primitives that establish architectural slots for future refinement.
3. We introduce four novel design primitives: the Agent Identity Card for persistent identity, capability-aware discovery through DID-bound Verifiable Credential manifests that make skill and tool claims cryptographically verifiable at discovery time, trust negotiation combining the monotonic capability attenuation chain with the two-tier A2A access control model, and kernel-mediated cryptographic audit trails for accountability.
4. We position InterSAGE within the current landscape of agent protocol, identity, and governance efforts (§9)—organized into six clusters spanning IoA stacks and surveys, agent IAM and OAuth/OIDC extensions, zero-trust DID/VC architectures, delegation-centric protocols, security-design principles, and industry trust overlays—and make the joint-coverage argument explicit: each individual primitive in InterSAGE is anticipated by some prior work, but the conjunction of persistent identity, capability-aware discovery, trust negotiation with monotonic attenuation and two-tier access control, and kernel-mediated accountability appears in no prior single architecture (Table 7).

The remainder of this paper is organized as follows. §2 surveys the agent interaction landscape and threat model. §3 presents the design philosophy and protocol suite overview. §4–§7 detail each layer. §8 provides a security analysis and comparison with existing approaches. §9 discusses related work in depth. §10 outlines limitations and future directions. §11 concludes.

2 Background & Threat Landscape

2.1 The Agent Interaction Landscape

Modern agent-to-agent communication is shaped by a rapidly growing set of protocols, each targeting a different facet of the problem:

Model Context Protocol (MCP). The Model Context Protocol (MCP) [7], introduced by Anthropic and now maintained as an open, vendor-neutral specification, standardizes how an agent (client) discovers and invokes external tools hosted by MCP servers via JSON-RPC. Third-party deployment surveys report over 97 million monthly SDK downloads as of early 2026 [28]; MCP is widely adopted for agent-to-tool integration. The core specification, however, does not define agent-native identity, mutual authentication between agents, or authorization semantics beyond transport-level security.

Agent-to-Agent Protocol (A2A). The Agent-to-Agent (A2A) protocol [6], developed under the A2A open specification effort, enables peer-to-peer task delegation between agents using capability-based *Agent*

Cards. A2A supports both synchronous and asynchronous interactions and provides structured task lifecycle management. Published security guidance centers on HTTPS and OAuth 2.0; the specification does not yet standardize agent-native identity, structured delegation chains, or cross-domain trust comparable to a dedicated trust plane.

Agent Network Protocol (ANP). ANP [8] introduces a three-layer architecture—identity and encrypted communication, meta-protocol negotiation, and application protocol—using W3C DIDs and JSON-LD for open agent discovery. ANP focuses on the "agentic web" vision but, in its published white paper, treats access control and end-to-end accountability less extensively than identity and discovery.

Emerging protocol stacks. Several groups propose higher-level architectures. Fleming et al. [11] propose reference layers for agent communication (L8) and agent semantics (L9) above TCP/IP (not part of the classical OSI stack). Agent-OSI [10] proposes a six-layer decentralized stack including settlement and provenance. ACPS [14] defines registration, discovery, interaction, and tooling protocols. Coral Protocol [29] provides open infrastructure for agent communication, coordination, trust, and payments framed as the "Internet of Agents." Surveys [12, 13, 30] catalog and compare these efforts.

Beyond A2A: adjacent interaction axes and trust overlays. The protocol space continues to multiply along distinct interaction axes that complement A2A: AG-UI [31] for agent-to-user streaming, the Agent Client Protocol (ACP) [15] for editor-to-coding-agent flows, IBM ContextForge [32] as a federated gateway proxying MCP/A2A/REST, and the `agents.json` specification [33] extending OpenAPI with agent-specific interaction contracts. Most recently, Microsoft's AgentMesh [19] describes itself in project documentation as "SSL for AI agents," layering SPIFFE/SVID workload identity, a runtime policy engine, and A2A/MCP/IATP protocol translators atop existing communication stacks; a parallel academic effort, Huang et al.'s unified zero-trust architecture [20], adds behavioral attestation and trust-adaptive runtime environments to the same trust-overlay paradigm. We return to both as our primary industry and academic comparators in §9.

> **Key Insight 1: Observation.** Existing protocols and stacks overwhelmingly prioritize *communication interoperability*: how agents find each other, exchange messages, and invoke tools. Security is treated as an aspect to be satisfied at each layer rather than as a *foundational design center* with its own coherent architecture. Even the most security-conscious trust overlays (AgentMesh [19], the unified zero-trust architecture of Huang et al. [20]) enforce least-privilege through runtime policy evaluation and behavioral attestation rather than through monotonic capability attenuation that holds at the credential level—so a misconfigured policy or a compromised attestor can silently widen capability. This leaves critical gaps that individual protocol patches cannot close.

2.2 Agent-Specific Threat Model

The agent ecosystem introduces threats that are qualitatively different from traditional web services. We identify six categories that motivate the design of InterSAGE.

Table 1 summarizes these threats. Prior threat analyses [16, 17, 18, 23] have identified overlapping subsets; our contribution is to map them to their *structural causes*—the architectural deficiencies that enable each threat—which directly motivates the layered design of InterSAGE.

The taxonomy in Table 1 is not constructed in isolation. It synthesizes three independent lines of analysis: Anbiaee et al. [17] compare MCP, A2A, Agora, and ANP and identify twelve protocol-level risks, several of which (cross-protocol credential laundering, executable-component attestation, lifecycle risks) map directly onto T1–T4. AgentRFC [16] formalizes eleven security principles as TLA+ invariants and introduces the Composition Safety principle—the observation that properties holding for individual protocols can break

Table 1: *Agent-specific threat categories and their structural causes.*

Threat	Description	Structural Cause
T1: Identity spoofing	An agent claims to be built by a trusted developer or to possess capabilities it lacks.	Agents use self-declared names or API keys; no binding to developer, code, or operator.
T2: Capability escalation	A child agent exercises permissions beyond what its parent delegated.	Static permission models lack monotonic attenuation; delegation is often unconstrained.
T3: Delegation abuse	Unbounded delegation chains create laundering paths for authority.	No depth bounding, no cascading revocation, no tenant isolation in delegation.
T4: Cross-domain trust breakdown	Agents from different organizations cannot verify each other's identity or capabilities.	No shared trust anchor; each framework uses its own identity model.
T5: Payment & usage fraud	An agent consumes LLM tokens or services without traceable identity.	Token usage is tied to API keys, not to cryptographic agent identity; no metering protocol.
T6: Action repudiation	An agent denies having performed a high-impact action (e.g., financial transfer).	Execution traces are not cryptographically signed; audit logs are mutable.

under composition—which underlies T2 and T4. Identity-aware governance analyses [22] ground these abstract risks in production incidents reported in that survey (e.g., the ClawHavoc supply-chain attack on 824+ malicious agent skills, Cisco's finding that 26% of analyzed agent skills contain security vulnerabilities, and 30,000+ internet-exposed OpenClaw instances), making T1–T6 verifiable rather than purely analytical.

2.3 Why Traditional Internet Security Falls Short

The Internet's existing security mechanisms were designed for human users interacting with web services. Three fundamental mismatches arise when they are applied to autonomous agents.

OAuth/OIDC assumes human principals. OAuth 2.0 [34] and OpenID Connect [35] model a three-party flow: a human user authorizes a client application to access a resource server. Agent-to-agent interactions invert this model—both parties are non-human, may be ephemeral, and require mutual (not unilateral) authentication. The OpenID Foundation's strategic agenda for agentic IAM [25] explicitly catalogs these gaps and surveys candidate extensions; OIDC-A [24] is the most concrete extension to date, defining standard claims for representing agent identity, attestation, and delegation chains within OAuth. Yet such extensions still inherit OAuth's interactive-flow assumptions and bind to a single *client* dimension, not to the developer, code package, operator, and operational context that agent provenance requires [36, 37].

TLS provides transport, not identity semantics. TLS [38] secures the channel but does not tell the recipient *who built* the agent, *what code* it runs, or *who authorized* its deployment. A TLS certificate binds a public key to a domain name, not to a developer-code-operator triple. Mutual TLS (mTLS) adds client-side certificates but still does not express agent capabilities, delegation depth, or revocation cascades. Recent zero-trust frameworks add SPIFFE/SVID workload identity above mTLS [39, 40, 41], which removes static API keys and enables short-lived attestation, but binds keys to *workload instances*—a single dimension—rather than to the conjunction of developer, code package, operator, and operational context that agent trust decisions require.

X.509 does not bind agent semantics. X.509 certificates [42] bind a public key to a subject name and are widely used in PKI. However, agent identity requires richer semantics: capability boundaries, delegation chains, code-package hashes, and operational context. Encoding these in X.509 extensions is technically

possible but semantically awkward and incompatible with the existing CA ecosystem. SPIFFE/SPIRE [40] provides workload identity for Kubernetes but, as noted above, does not model agent-native concepts such as delegation depth, capability attenuation, or developer-code provenance.

> **Key Insight 2: Gap.** No existing Internet security mechanism provides the combination of *persistent identity* with four-dimensional binding, *capability-aware discovery* with verifiable skill and tool manifests, *trust negotiation* that combines monotonic capability attenuation with two-tier access control, and *accountability* via tamper-evident execution traces. InterSAGE fills this gap with a dedicated security protocol suite.

3 InterSAGE: Design Philosophy & Protocol Suite Overview

3.1 Design Principles

InterSAGE is guided by five principles distilled from lessons learned in BlockA2A [23], the DeepKernel security kernel, and the broader agent security literature:

P1. Trust as a binding layer, not an add-on. Existing IoA stacks [11, 10, 14] design communication first and add security externally—via auth headers, gateway proxies, or protocol-specific translators [19]. InterSAGE inverts this: its trust primitives (AIC capability boundaries, manifest VCs, session tokens, trace entries) are designed to be *embedded into* existing protocol messages. An MCP invocation carries an AIC-bound capability context; an ANP discovery response carries manifest VCs; an A2A interaction carries a Layer 2 session token; an AG-UI event stream attaches provenance metadata. The trust layer *pervades* the communication layer rather than wrapping around it.

P2. Complementary to existing Internet layers. InterSAGE does not replace TCP/IP, HTTP, or application-layer protocols such as MCP and A2A. Instead, it provides a *trust overlay* that any transport and any agent protocol can bind to. An agent using A2A over HTTPS can adopt InterSAGE's identity and access control layers without modifying A2A's message format.

P3. General cryptographic primitives, no infrastructure lock-in. BlockA2A demonstrated the value of blockchain-anchored auditability, but at the cost of deployment generality. InterSAGE uses standard Ed25519 signatures, a PKI-style Global Agent Registry (GAR), and challenge-response protocols that work in any environment—cloud, edge, or air-gapped. Specific deployments can optionally layer additional trust anchors (TEEs, HSMs, distributed ledgers) without changing the core protocol.

P4. Deny-by-default, converge-on-strict. Capabilities, permissions, and access are denied unless explicitly granted. When multiple policies combine (e.g., developer declaration, operator provisioning, runtime policy), the system computes the *intersection*, never the union. Authority can only be attenuated, never amplified—a property we call *monotonic capability attenuation*.

P5. Structural guarantees over policy-based assurances. Where possible, InterSAGE encodes security properties as structural invariants (e.g., a child AIC's capability set is a cryptographically signed subset of its parent's) rather than relying on runtime policy engines to enforce them. Structural guarantees survive misconfiguration; policy-based assurances do not.

3.2 Protocol Suite Architecture

InterSAGE is organized into four layers, each addressing a distinct set of security concerns. Figure 1 illustrates the architecture and its relationship to the existing Internet protocol stack.

Layer 0: Agent Identity. The foundation of InterSAGE. Every agent receives a cryptographic *Agent Identity Card* (AIC) that binds four identity dimensions—developer, code package, operator, and operational

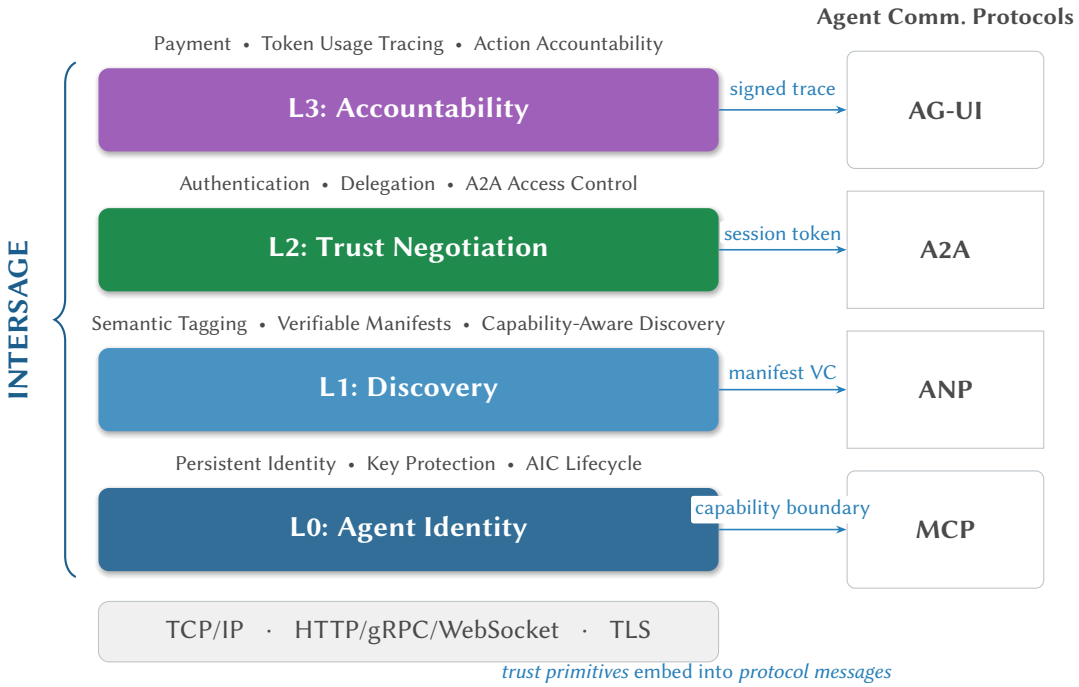


Figure 1: *The InterSAGE four-layer protocol suite. Each layer builds on the one below it and* embeds *trust primitives into existing agent communication protocols: L0 constrains MCP tool invocation through AIC capability boundaries, L1 exposes DID-bound manifest VCs through discovery protocols such as ANP, L2 authorizes A2A interaction through session tokens, and L3 attaches signed execution traces to AG-UI streams.*

context—into a single verifiable credential signed by the Global Agent Registry. Layer 0 also manages the agent's key lifecycle: provisioning, rotation, and revocation with cascading guarantees. (§4)

Layer 1: Discovery. Before two agents can interact, they must find each other and understand each other's capabilities. Layer 1 provides capability-aware discovery: each skill and tool is represented as a signed Verifiable Credential (VC) bound to the agent's DID, issued by the relevant skill distributor or tool provider, and presented in the agent's registration record. Requesters verify each manifest VC for issuer authenticity, subject binding, and permission alignment against the agent's AIC capability boundary—transforming directory lookup into a trust-establishing protocol step. Layer 1 supports both registry-mediated and peer-to-peer discovery modes. (§5)

Layer 2: Trust Negotiation. The interaction layer. When an agent wishes to collaborate with or delegate to another, Layer 2 orchestrates mutual attestation via challenge-response over AICs, computes the session capability boundary via intersection, and evaluates the responder's application-level access policy. Its core primitive combines monotonic capability attenuation with two-tier access control: cryptographic credentials define the maximum boundary, while application policy can only narrow the resulting session. Layer 2 also manages delegation: a parent agent can issue a child AIC with strictly attenuated capabilities, forming a cryptographic delegation chain with bounded depth and cascading revocation. (§6)

Layer 3: Accountability. Every interaction leaves a cryptographic trace. Layer 3 binds LLM token-usage records to agent identity, provides payment primitives for agent-to-agent service exchange, and ensures that execution traces are signed by the agent's kernel-held private key for non-repudiation. (§7)

3.3 Relationship to Existing Protocols

The agent ecosystem already has a rich set of communication protocols—MCP, A2A, ANP, AG-UI [31], ACP [15]—that define *what agents say to each other*: message formats, task lifecycles, tool invocations, UI event streams. InterSAGE does not compete with any of them. Instead, it treats them as *host protocols*—the

transport and application channels into which its trust primitives are embedded—and answers a different question: *why should agents trust each other?*

We discuss this compositional relationship here, rather than deferring it to Related Work, because it is integral to the design: every layer of InterSAGE (Figure 1) is defined in terms of how its primitives bind to host-protocol messages, so understanding that binding is a prerequisite for understanding the protocol suite itself. The Related Work section (§9) serves a different purpose: it compares InterSAGE against *security-oriented* architectures and trust frameworks—AgentMesh, AIP, HDP, ZT-IAM, AgentRFC, among others—that share InterSAGE's goal of adding trust to the agent ecosystem but differ in mechanism. Those works are InterSAGE's *comparators*; the protocols below are its *substrates*.

Figure 1 shows this substrate relationship concretely. InterSAGE operates alongside—not above or below—the host protocols, and the two concerns compose naturally:

- An MCP tool invocation carries the agent's AIC capability boundary (from Layer 0), ensuring that the tool server can verify the caller is authorized to invoke the requested tool before execution.
- ANP's DID-based discovery responses embed the agent's manifest VCs (from Layer 1), enabling any peer resolving an agent's DID to verify skill and tool claims via subject binding and permission alignment.
- An A2A interaction carries a session token issued by InterSAGE's Layer 2 after mutual attestation and capability intersection, scoping the session to the least-privilege boundary of both parties.
- AG-UI events streamed to a frontend include identity-signed execution traces (from Layer 3), allowing the UI to display verified provenance and non-repudiable action history for each agent.

This composability is a direct consequence of Principle P2: by avoiding assumptions about the underlying transport or application protocol, InterSAGE remains agnostic to the rapidly evolving agent communication landscape.

4 Layer 0: Persistent Agent Identity

> **Key Insight 3: Design Thesis.** Without verified identity, every other security property is built on air. Layer 0 ensures that every agent in the InterSAGE ecosystem is a *cryptographic principal*—unforgeable, verifiable, and bound to its provenance.

4.1 Agent Identity Card (AIC)

The *Agent Identity Card* is the foundational credential in InterSAGE. An AIC is a verifiable credential that binds an agent's identity to its permission boundary. Formally:

$$\mathrm{AIC} = \mathrm{Sign}_{\mathrm{GAR}}(\mathrm{DID}_{\mathrm{agent}} \parallel K_{\mathrm{pub}} \parallel S_{\max}) \tag{1}$$

where $\mathrm{DID}_{\mathrm{agent}}$ is a structured agent identifier derived from the $\langle$*developer*, *code_pkg*, *deploy_ctx*$\rangle$ triple, K_{pub} is the agent's Ed25519 public key (whose corresponding private key K_{priv} never leaves the isolated trust boundary), and $S_{\max}$ is the *capability boundary*—the maximum set of permissions this agent may ever exercise.

The AIC is signed by the Global Agent Registry (GAR) using its root Ed25519 key, analogous to a Certificate Authority signing an X.509 certificate. However, unlike X.509, the AIC natively encodes agent-specific semantics: capability sets, delegation depth, identity assurance levels, and operational context.

4.2 Four-Dimensional Identity Binding

A persistent agent identity must answer four questions simultaneously: *who built it*, *what code does it run*, *who deployed it*, and *in what context does it operate*. InterSAGE binds all four into the AIC through what we

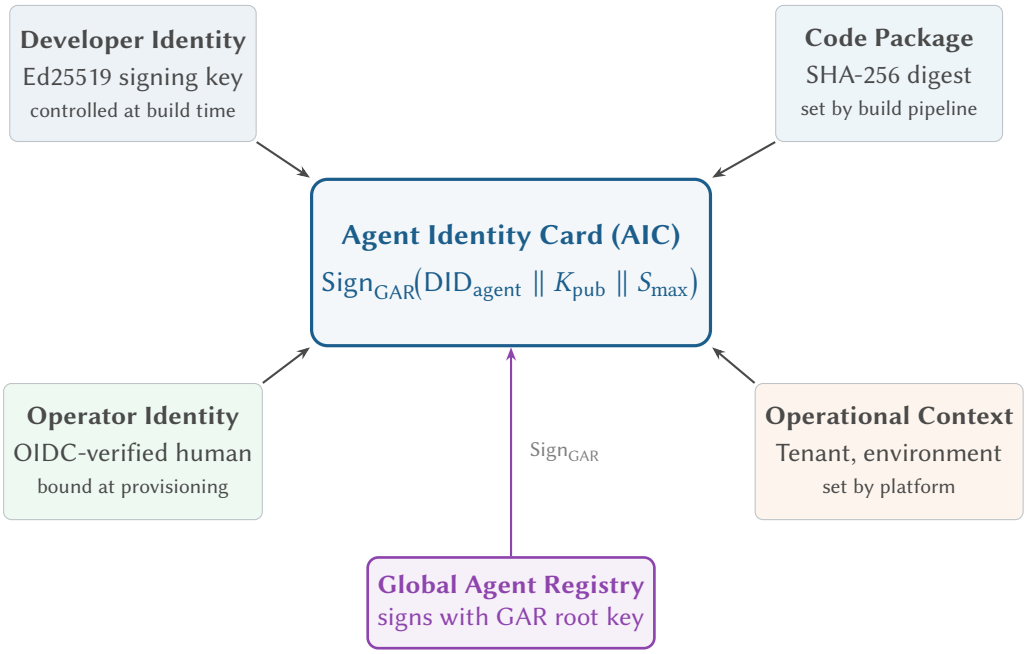


Figure 2: *Four-dimensional identity binding in the Agent Identity Card. Each dimension is controlled by a different party and verified independently.*

Table 2: *The four identity dimensions of an AIC.*

Dimension	What It Binds	Who Controls	Verification
Developer Identity	Developer's Ed25519 signing key	Developer (build time)	GAR verifies developer enrollment
Code Package	Cryptographic digest of the agent's code or container image	Build pipeline	GAR checks code hash at issuance
Operator Identity	OIDC-verified human operator (e.g., via Google, GitHub SSO)	Human operator (provisioning)	GAR binds OIDC token to AIC
Operational Context	Tenant ID, deployment environment, region	Platform operator	GAR records at issuance; inherited by delegates

call *four-dimensional identity binding*.

This four-dimensional binding addresses threat T1 (identity spoofing) from Table 1: an attacker cannot forge an AIC without simultaneously controlling the developer's signing key, producing a matching code digest, presenting valid OIDC credentials, and registering with the correct operational context.

Contrast with existing approaches. DID-based identity [43, 26] binds a public key to a decentralized identifier but does not natively encode developer, code, or operator dimensions. SPIFFE [40] provides workload identity attestation via platform-specific mechanisms but does not model capability boundaries or delegation semantics. The AIP protocol [21] introduces Invocation-Bound Capability Tokens without separating developer and operator dimensions in the credential model. InterSAGE's four-dimensional binding provides richer provenance than workload-centric models such as SPIFFE/SVID (used by AgentMesh [19]) and, among the identity proposals surveyed in §9, is the only one we found that jointly binds developer, code, operator, and operational context within a single verifiable credential.

4.3 Global Agent Registry (GAR)

The GAR serves as the trust anchor for the InterSAGE ecosystem, playing a role analogous to a Certificate Authority in traditional PKI. Its responsibilities include:

1. **Developer enrollment:** onboarding developers with verified signing keys and policy governance.
2. **AIC issuance:** validating that the requested capability boundary $S_{\max}$ is a subset of the developer's declared maximum, verifying the developer signature, binding the OIDC operator identity, and signing the AIC with the GAR root key.
3. **AIC lookup and verification:** enabling any party to resolve an agent's AIC by identifier and verify its signature chain.
4. **Revocation:** maintaining a revocation registry; revoking an AIC atomically invalidates it and all its delegated descendants (cascading revocation, detailed in §6.2).

The GAR can be deployed as a centralized service (suitable for enterprise environments), a federated constellation (analogous to federated CAs), or backed by a distributed ledger for environments that require decentralized trust anchoring. This flexibility is a direct consequence of Principle P3.

4.4 Key Protection Tiers

The agent's private key K_{priv} is the root of its cryptographic identity. InterSAGE defines three protection tiers, allowing deployments to choose the appropriate security-cost trade-off:

Tier	Mechanism	Security Level	Deployment
1	OS file isolation (`0o600`)	Moderate	Dev / testing
2	OS keychain (Keychain, DPAPI)	Good (HW-backed)	Production
3	TEE enclave (SGX, Nitro)	Strong (key never leaves HW)	High-assurance

Table 3: *Key protection tiers. K_{priv} never leaves the local trust boundary regardless of tier.*

Regardless of the tier, a fundamental invariant holds: *K_{priv} never leaves the isolated trust boundary.* All signing operations are mediated by the agent's designated kernel, which acts as an HSM-like custodian. External agents and even the agent's own application logic interact only with opaque *handles*—they can request signatures but never access the raw key material.

4.5 Identity Lifecycle

An AIC progresses through a well-defined lifecycle:

1. **Provisioning:** The operator submits a registration request (agent ID, capabilities, OIDC token) to the kernel. The kernel generates a fresh Ed25519 keypair, constructs the AIC payload, and forwards it to the GAR for signing. The signed AIC is returned and cached locally.
2. **Active use:** The AIC is used for mutual attestation (§6), session token issuance, and delegation.
3. **Rotation:** Key rotation generates a new keypair, revokes the old AIC, and issues a replacement preserving all identity dimensions. Active sessions are invalidated to prevent stale credential use.
4. **Revocation:** An AIC can be revoked by the operator, the GAR administrator, or automatically upon expiration. Revocation cascades to all delegated descendants (§6.2), ensuring that revoking a parent instantly invalidates the entire subtree.

The lifecycle is designed so that identity is *persistent* (the agent's identity dimensions survive key rotation) while credentials are *ephemeral* (AICs have bounded validity and can be revoked at any time). This distinction is crucial for long-lived agents that operate across sessions and deployments.

5 Layer 1: Registration, Discovery & Semantic Interoperability

Layer 0 establishes *who an agent is*; Layer 1 establishes *what an agent can do and how to find it.* This layer bridges identity to interaction by providing the infrastructure for agents to advertise their capabilities,

discover peers, and cryptographically verify the authenticity of advertised skills before any trust negotiation begins.

The central insight of Layer 1 is that discovery must be a *security primitive*: every capability claim an agent advertises is backed by a signed Verifiable Credential (VC) bound to the agent's DID and verifiable against the AIC trust chain. This transforms agent discovery from a directory lookup into a trust-establishing protocol step.

5.1 Capability-Aware Registration

When an agent is provisioned (Layer 0), its AIC already contains a capability boundary $S_{\max}$. Layer 1 extends this with a richer *registration record* that the agent publishes to the GAR or a federated discovery service:

Registration Record

agent_id	$\text{DID}_{\text{agent}}$ from the AIC
aic_ref	reference to the signed AIC
capabilities	structured list from the capability vocabulary
semantic_tags	tags for capability-aware search
endpoints	transport endpoints (HTTP, gRPC, WebSocket)
protocols	supported protocols (A2A, MCP, ANP)
manifests	list of signed skill/tool manifest VCs
metadata	description, version, pricing hints

The registration record is itself signed by the agent's K_{priv} (via the kernel), ensuring that only the agent can create or update its own record. The GAR or discovery service verifies this signature against the AIC's K_{pub} before accepting the registration.

5.2 Semantic Capability Tagging

Raw capability enumerations (e.g., `fs.read`, `network.fetch`) describe what system surfaces an agent can touch but not the *semantic domain* of the agent's expertise. InterSAGE introduces a two-level tagging scheme:

1. **System capabilities** (from Layer 0): a closed vocabulary of permission classes bound to the AIC. These are machine-enforced at runtime.
2. **Semantic tags** (from Layer 1): an open vocabulary of domain-specific labels (e.g., `legal-review`, `code-generation`, `financial-analysis`) that describe the agent's expertise at a human-understandable and LLM-parseable level.

Semantic tags are *not* capability grants—they carry no enforcement weight. Their purpose is to enable capability-aware discovery: a requester searching for a "code review" agent can filter candidates by semantic tag, then verify actual capabilities via the AIC and manifest VCs before Layer 2 negotiation. This separation prevents semantic labels from being abused as implicit permission escalation vectors.

5.3 Capability-Aware Discovery

InterSAGE treats discovery as a security primitive rather than a directory lookup. A discovery response is only useful if the requester can verify not just that an agent claims a skill or tool, but that the advertised capability is backed by a signed manifest VC and fits within the agent's AIC capability boundary. This turns Layer 1 into *capability-aware discovery*: semantic search narrows the candidate set, while DID-bound manifest VCs and AIC-bound capabilities make the result safe to consume during Layer 2 negotiation.

InterSAGE supports two discovery modes that can coexist within the same ecosystem:

Registry-mediated discovery. Agents query the GAR (or a federated registry constellation) with structured queries over capabilities, semantic tags, protocol support, and trust attributes (e.g., minimum identity assurance level, trusted developer list). The registry returns matching registration records, each verifiable against the corresponding AIC. This mode is analogous to DNS resolution and is suitable for enterprise and platform-managed environments.

Peer-to-peer discovery. In decentralized environments, agents can discover peers via protocol-native mechanisms (e.g., ANP's DID-based discovery [8], A2A's well-known Agent Cards [6]). InterSAGE does not replace these mechanisms but *augments* them: once a candidate agent is discovered via any means, its AIC can be resolved and verified through the GAR, adding a trust verification step that the native discovery protocol may lack.

Both modes converge on the same verification flow: the discovered agent's AIC is verified against the GAR's trust chain, and each manifest VC in its registration record is validated via the four-check protocol (§5.4). Discovery thus answers three questions simultaneously: "who is out there" (AIC verification), "what can they do" (manifest VC verification), and "should I trust them" (subject binding + permission alignment).

5.4 Verifiable Skill & Tool Manifests

An agent's capabilities are realized through two mechanisms:

- **Skills**—reusable capability modules (code packages) that run within the agent's execution environment.
- **Tools**—external service bindings (API endpoints, MCP servers) that the agent invokes over the network.

In InterSAGE, both skills and tools are represented as *signed Verifiable Credentials (VCs)* cryptographically bound to the agent's DID and verifiable against the AIC trust chain. This design ensures that capability claims—whether local code or remote services—carry the same structural trust guarantees as agent identity itself. The two types share a common credential structure but differ in their issuance model, reflecting their distinct trust relationships.

Credential structure. Both skill and tool manifest VCs follow the W3C Verifiable Credentials data model [44] adapted for agent semantics:

Skill Manifest VC

@context	W3C VC context + InterSAGE skill vocabulary
id	unique credential URI
type	["VerifiableCredential", "SkillManifest"]
issuer	distributor DID (skill author)
credentialSubject	{
id	$\text{DID}_{\text{agent}}$ (the agent holding this skill)
manifest_id	unique manifest identifier
perms_required	system capabilities the skill needs
caps_provided	what the skill enables for the agent
code_hash	digest of the skill code package
}	
proof	Ed25519 signature by the distributor
gar_endorsement	optional GAR endorsement

Tool Manifest VC

@context	W3C VC context + InterSAGE tool vocabulary
id	unique credential URI
type	["VerifiableCredential", "ToolManifest"]
issuer	tool provider DID (service operator)
credentialSubject	{
id	$\text{DID}_{\text{agent}}$ (the agent authorized to use this tool)
manifest_id	unique manifest identifier
perms_required	capabilities the tool invocation needs
caps_provided	what the tool enables for the agent
endpoint	service URI (MCP server, REST API, etc.)
api_hash	digest of the tool's interface schema
}	
proof	Ed25519 signature by the tool provider
gar_endorsement	optional GAR endorsement

In both cases the **credentialSubject.id** field binds the manifest to a specific agent DID ($\text{DID}_{\text{agent}}$). This binding means that a manifest VC is non-transferable: even if an attacker obtains the credential, it cannot be presented on behalf of a different agent because the verifier checks that the subject DID matches the presenter's AIC.

Skill manifest issuance. The lifecycle of a *skill* manifest VC proceeds as follows:

1. **Distributor publishes skill.** The skill author (the distributor) creates the manifest, signs it with their distributor key, and optionally submits it to the GAR for endorsement.
2. **Agent acquires skill.** When an agent installs a skill, the kernel verifies the distributor's signature and (if present) the GAR endorsement. The kernel then issues a *binding proof*: a secondary signature using the agent's K_{priv} that attests "I hold this skill and my AIC permits its required capabilities."
3. **Agent presents during discovery.** The agent includes its bound skill manifest VCs in its registration record.

Tool manifest issuance. Tools differ from skills because their trust anchor is the *tool provider* (the external service operator), not a code distributor:

1. **Tool provider registers service.** The service operator publishes a tool manifest describing the tool's endpoint, interface schema, and required permissions. The provider signs the manifest with their provider key and optionally obtains a GAR endorsement.
2. **Provider issues VC to agent.** When an agent requests access to a tool, the provider verifies the agent's AIC and checks that the agent's capability boundary $S_{\max}$ is a superset of the tool's `perms_required`. If satisfied, the provider issues a tool manifest VC with `credentialSubject.id` set to the agent's $\text{DID}_{\text{agent}}$—effectively granting the agent verifiable authorization to invoke the service.
3. **Agent presents during discovery.** The agent includes its tool manifest VCs alongside skill manifest VCs in its registration record. Requesters can now verify that the agent not only claims access to a tool but has been explicitly authorized by the tool provider.

In both flows, the agent's registration record ultimately contains a set of manifest VCs—some for skills, some for tools—each independently verifiable by any requester or registry.

Verification during discovery. When a requester discovers an agent and retrieves its registration record,

the following checks are performed on each manifest VC before the agent is considered a valid candidate for Layer 2 negotiation:

- **Supply-chain verification:** The issuer's signature (skill distributor or tool provider) and any GAR endorsement are verified, preventing supply-chain attacks analogous to the "ClawHavoc" incident [22].
- **Subject binding:** The `credentialSubject.id` must match the discovered agent's $\text{DID}_{\text{agent}}$. This prevents manifest replay attacks where an adversary copies another agent's manifest credentials.
- **Permission alignment:** The manifest's `perms_required` must be a subset of the agent's AIC capability boundary $S_{\max}$. If a skill or tool requires permissions beyond the agent's boundary, the manifest is rejected, ensuring that discovery cannot surface agents whose advertised claims exceed their actual authorization.
- **Freshness:** The manifest VC must not be expired or revoked (checked against the GAR's revocation list or status endpoint).

This four-check verification protocol transforms discovery from a trust-me directory into a *verify-then-interact* security boundary. The consequence is structural: an agent cannot advertise capabilities it does not possess, because every advertised skill or tool must be backed by a VC whose subject binding, issuer provenance, and permission alignment are all independently verifiable.

6 Layer 2: Trust Negotiation

Layer 2 is the interaction layer of InterSAGE. It governs three critical processes: *mutual attestation* (how two agents establish trust), *delegation* (how a parent agent creates bounded child identities), and *access control* (how a responder decides what a requester may do). Together, these processes address threats T2–T4 from Table 1.

6.1 Mutual Attestation Protocol

When two agents wish to interact, neither should blindly trust the other. The mutual attestation protocol establishes bilateral trust through a challenge-response exchange grounded in the agents' AICs.

We write Kernel_A and Kernel_B for the kernels that hold K^A_{priv} and K^B_{priv} respectively. When both agents reside on the same host, $\text{Kernel}_A = \text{Kernel}_B$ and the protocol reduces to local operations.

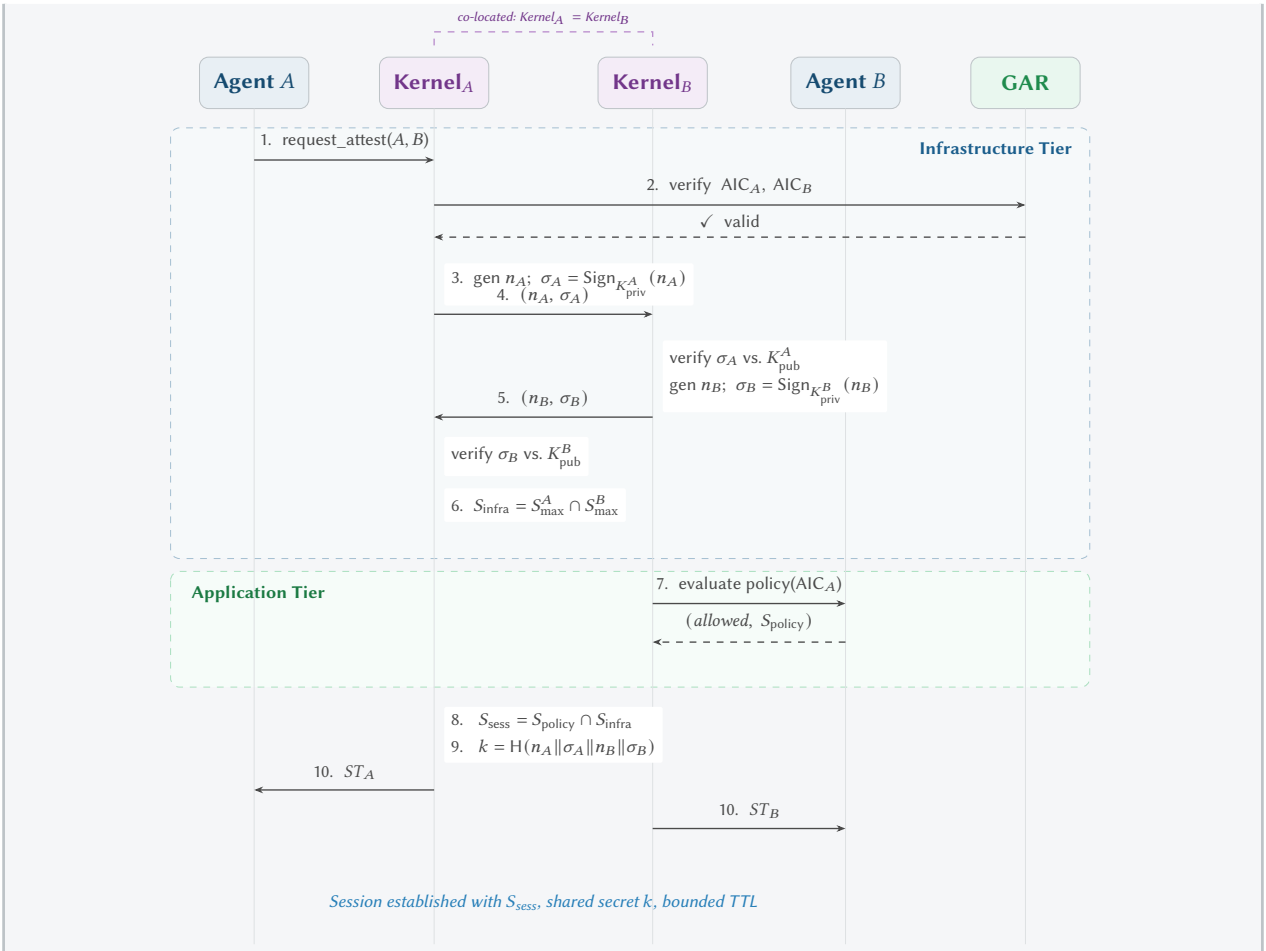


Figure 3: *Mutual attestation protocol. Each agent's kernel (Kernel$_A$, Kernel$_B$) mediates its own signing operations; private keys never leave the isolated trust boundary. When both agents share a host, the two kernels collapse to one. The protocol produces a pair of session tokens with capabilities scoped to $S^A_{\mathrm{max}} \cap S^B_{\mathrm{max}}$ (further narrowed by any application-level policy).*

1. Agent A requests attestation with Agent B through Kernel$_A$.
2. Kernel$_A$ resolves AIC$_A$ and AIC$_B$, verifying both against the GAR (signature validity, expiration, revocation status).
3. Kernel$_A$ generates a fresh nonce n_A (32 bytes, 60 s TTL) and signs it: $\sigma_A = \mathrm{Sign}_{K^A_{\mathrm{priv}}}(n_A)$.
4. The challenge (n_A, σ_A) is delivered to Kernel$_B$. Kernel$_B$ verifies σ_A against K^A_{pub} from AIC$_A$, generates a fresh nonce n_B, and signs: $\sigma_B = \mathrm{Sign}_{K^B_{\mathrm{priv}}}(n_B)$.
5. Kernel$_B$ returns (n_B, σ_B) to Kernel$_A$. Kernel$_A$ verifies σ_B against K^B_{pub} from AIC$_B$.
6. **Infrastructure capability bound:** $S_{\mathrm{infra}} = S^A_{\mathrm{max}} \cap S^B_{\mathrm{max}}$.
7. **Application-level policy** (if registered for B): the responder's access policy evaluates AIC$_A$'s attributes and returns $(\mathit{allowed},\ S_{\mathrm{policy}})$.
8. **Session capabilities:** $S_{\mathrm{session}} = S_{\mathrm{policy}} \cap S_{\mathrm{infra}}$.
9. **Shared secret:** $k = \mathrm{H}(n_A \,\|\, \sigma_A \,\|\, n_B \,\|\, \sigma_B)$.
10. Kernel$_A$ issues session token ST_A and Kernel$_B$ issues ST_B, each with capabilities S_{session}, shared secret k, and bounded TTL.

Remark (Deployment Topologies). The protocol above is parameterized by the relationship between Kernel$_A$ and Kernel$_B$. Three instantiations cover the practical deployment spectrum:

Co-located (single kernel). When A and B reside on the same host, Kernel$_A$ = Kernel$_B$ and steps 3–5 collapse to kernel-internal operations with no network round-trip. This is the natural mode for DeepKernel-managed multi-agent orchestration on a single device.

Remote-mediated (Trusted Attestation Service). A standalone service—architecturally identical to an agent kernel but deployed as a remote endpoint—coordinates the exchange. Each agent's designated kernel retains K_{priv} and signs within its trust boundary; the TAS relays challenges, caches GAR verification results, and issues session tokens. The TAS plays a role analogous to an OIDC Provider in human web SSO: it centralizes session establishment while each agent retains sovereign control of its private key.

Direct peer-to-peer. Each kernel signs its own nonce and verifies the counterpart's signature independently via the GAR. No trusted intermediary is required. This mode is fully decentralized but incurs higher latency: each side performs an independent GAR round-trip, and every attestation requires a full network challenge-response exchange. AIC caching with revocation-push notifications (§10) can amortize the GAR cost.

The three modes can coexist within a single ecosystem: co-located attestation for intra-device workflows, mediated attestation for organizational clusters, and direct P2P for open cross-organization interactions.

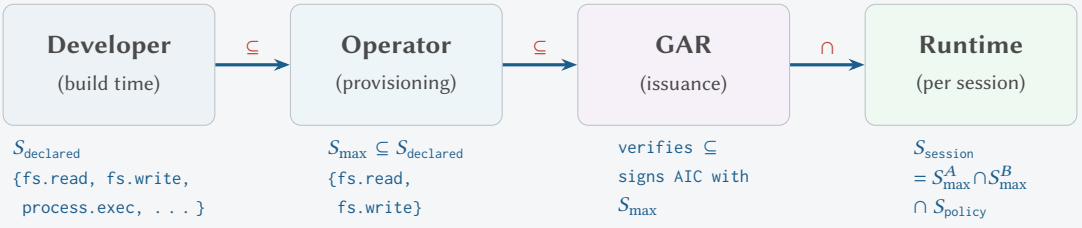
Anti-spoofing guarantees. Session tokens are bound to specific agent instances (AIC + execution context). Tokens expire on agent termination; replay is impossible because nonces are single-use and time-bounded. Private keys never leave the agent's isolated kernel, so stolen tokens are useless outside the authorized execution context.

6.2 AIC Delegation Chain

In multi-agent orchestration scenarios, a parent agent often needs to dynamically spawn task-specific child agents. InterSAGE supports this through *AIC delegation*: the parent requests the kernel to issue a child AIC with strictly attenuated capabilities, forming a cryptographic chain analogous to an X.509 intermediate-CA hierarchy.

Delegation Protocol

1. Parent requests child AIC from the kernel, specifying desired capabilities $S_{\text{child}} \subseteq S_{\text{max}}^{\text{parent}}$.
2. Kernel verifies the parent's AIC chain (iterative walk to GAR root).
3. Kernel checks: $S_{\text{child}} \subseteq S_{\text{max}}^{\text{parent}}$ (no capability escalation).
4. Kernel checks: delegation depth $\leq$ configured ceiling.
5. Kernel generates fresh Ed25519 keypair for the child.
6. GAR issues child AIC signed by the parent's key, with:
   - $S_{\text{max}}^{\text{child}} = S_{\text{child}}$
   - Inherited `tenant_id` and `deployment_environment`
   - Validity capped to parent's remaining lifetime
7. Child AIC is returned; parent-to-child link is recorded.



Each stage can only ***narrow****, never widen. Authority flows left→right via ⊆ and ∩.*

Figure 4: *Monotonic capability attenuation chain. Each party can only* narrow *the boundary set by the preceding party. The runtime intersection further constrains per-invocation permissions.*

The delegation mechanism provides six structural security properties:

Table 4: *Security properties of AIC delegation.*

Property	Mechanism
Monotonic attenuation	$S_{\max}^{\text{child}} \subseteq S_{\max}^{\text{parent}}$ enforced cryptographically at delegation time.
Tenant isolation	Child inherits `tenant_id` from parent; cross-tenant delegation is structurally impossible.
Depth bounding	$d_{\text{eff}} = \min(d_{\text{kernel}}, d_{\text{request}})$; the request can only tighten, never loosen.
Cascading revocation	Revoking a parent AIC atomically revokes all descendants; chain verification fails for the entire subtree.
Validity capping	Child's `expires_at` $\leq$ parent's `expires_at`; parent expiry cascades temporally.
Cycle resistance	Iterative chain verification with visited set and hard hop cap; no recursion overflow or cycle attacks.

6.3 Two-Tier A2A Access Control

A pure capability-intersection model (§6.1, step 6) is necessary but insufficient: the responder agent may wish to restrict interactions based on attributes beyond raw capabilities—for example, allowing only agents from trusted developers, requiring a minimum identity assurance level, or enforcing same-tenant isolation.

InterSAGE introduces a *two-tier access control model* that separates platform-enforced trust from application-level policy:

1. **Infrastructure tier (mandatory, kernel-enforced):** Cryptographic AIC verification via the GAR. This tier answers: "Is this agent who it claims to be?" Both parties' AICs are verified for signature validity, expiration, and revocation. The infrastructure capability bound $S_{\text{infra}} = S_{\max}^{A} \cap S_{\max}^{B}$ is computed.
2. **Application tier (optional, agent-defined):** The responder agent registers a declarative *access policy* that evaluates the requester's AIC attributes:
   - Required capabilities (requester must possess specific caps)
   - Trusted developers (glob patterns on `developer_id`)
   - Trusted operators (glob patterns on `operator_id`)
   - Minimum identity assurance level (IAL threshold)
   - Maximum risk level
   - Same-tenant requirement

   The application tier answers: "Does this specific responder want to work with this specific requester?" It produces a (possibly narrowed) set of *granted capabilities* that is intersected with the infrastructure bound.

The final session capability set is:

$$S_{\text{session}} = S_{\text{policy}} \cap \left(S_{\max}^{A} \cap S_{\max}^{B}\right) \tag{2}$$

This two-tier design has three key advantages over single-layer approaches:

- **Composability:** Infrastructure and application concerns are independently auditable and evolvable.
- **Backward compatibility:** Agents without a registered policy fall back to the infrastructure tier alone.
- **Fine-grained control:** The responder can make admission decisions based on identity attributes that raw capability intersection cannot express.

6.4 The Capability Narrowing Chain

InterSAGE's approach to authorization can be summarized as a four-stage *narrowing chain* in which each stage can only tighten the permission boundary:

Stage 1:	Developer	declares	S_{declared}
Stage 2:	Operator	selects	$S_{\text{max}} \subseteq S_{\text{declared}}$
Stage 3:	GAR	enforces & signs	S_{max} (subset-checked, signed AIC)
Stage 4:	Runtime	intersects	$S_{\text{session}} \subseteq S^A_{\text{max}} \cap S^B_{\text{max}}$

At no point can any party *widen* the boundary set by the preceding party. This is enforced cryptographically: the developer's signature covers S_{declared}; the GAR's signature covers $S_{\text{max}} \subseteq S_{\text{declared}}$; the session token is kernel-signed over $S_{\text{session}} \subseteq S^A_{\text{max}} \cap S^B_{\text{max}}$. Any attempt to insert additional capabilities requires forging a signature.

This monotonic attenuation provides *structural least-privilege*: the property holds by construction, not by correct policy configuration. Misconfiguring a policy can deny legitimate access but can never grant unauthorized capabilities.

7 Layer 3: Accountability & Economics

Layers 0–2 establish who an agent is, what it can do, and whom it trusts. Layer 3 closes the loop by ensuring that every agent action is *traceable*, *attributable*, and *economically accountable*. This layer addresses threats T5 (payment and usage fraud) and T6 (action repudiation) from Table 1.

7.1 Token-Usage Tracing

LLM-powered agents consume computational resources (inference tokens, tool invocations, storage) on behalf of their operators. In multi-agent workflows, a single user request may trigger cascading agent interactions, each consuming tokens from different LLM providers. Without identity-aware metering, costs are attributed to API keys rather than to the agents (and ultimately the humans) who authorized the work.

InterSAGE binds token-usage records to cryptographic agent identity:

Token-Usage Record

agent_id	$\text{DID}_{\text{agent}}$ of the consuming agent
session_id	session token ID from Layer 2 attestation
provider	LLM provider identifier
model	model name and version
input_tokens	prompt token count
output_tokens	completion token count
timestamp	UTC timestamp
delegation_chain	ordered list of AIC IDs from root to leaf
signature	$\text{Sign}_{K_{\text{priv}}}$(record payload)

The delegation chain field enables *cost attribution up the delegation tree*: if agent C was delegated by agent B, which was delegated by agent A (the root), the usage record traces the cost back to A's operator. Combined with the AIC's operator identity (OIDC-bound), this enables precise per-user, per-agent cost allocation in multi-tenant environments.

7.2 Payment Primitives

As agents increasingly provide services to other agents (e.g., a code-review agent charging per review, a data-analysis agent billing per query), the Internet of Agents requires payment primitives that are identity-aware and auditable.

InterSAGE defines a minimal payment protocol framework built on Layer 0–2 primitives:

1. **Price advertisement:** An agent's registration record (Layer 1) includes optional pricing metadata (rate type, currency, unit).
2. **Payment negotiation:** During mutual attestation (Layer 2), the requester and responder can negotiate payment terms as part of the session establishment. Payment terms are included in the session token metadata.
3. **Service delivery with metering:** During the session, both token usage and service-specific metering events are recorded with identity-signed records (§7.1).
4. **Settlement:** After service completion, the metering records serve as cryptographically verifiable invoices. Settlement can occur through traditional payment rails, escrow services, or—for environments that support it—on-chain settlement [10].

InterSAGE is deliberately agnostic about the settlement mechanism: it provides the *identity and metering infrastructure* that any payment system requires, without mandating a specific payment rail. This follows Principle P3 (no infrastructure lock-in). Notably, Google's Agent Payments Protocol (AP2) [45] is a recent open specification for AI-driven agent payments, focusing on the settlement rail and transaction flow. InterSAGE and AP2 are complementary: AP2 defines *how agents pay*, while InterSAGE defines *who is paying whom and with what authority*, binding every payment event to a verified cryptographic identity and a capability-bounded session.

7.3 Action Accountability

Every action an agent takes—tool invocations, delegations, message exchanges—generates an *execution trace entry* that is cryptographically signed by the agent's kernel-held private key:

Trace Entry

trace_id	unique identifier
agent_id	$\text{DID}_{\text{agent}}$ of the acting agent
action_type	tool_call | delegation | message | payment
action_params	structured parameters of the action
action_result	outcome or status
session_id	session context (from Layer 2)
timestamp	UTC timestamp
prev_hash	hash of the preceding trace entry (chain integrity)
signature	$\text{Sign}_{K_{\text{priv}}}(\text{entry payload} \,\|\|\, \text{prev\_hash})$

The `prev_hash` field chains trace entries into a hash-linked log, providing tamper evidence: any modification to a historical entry breaks the hash chain from that point forward. The signature binds each entry to its agent's cryptographic identity, providing non-repudiation.

Key gains of kernel-mediated accountability. What distinguishes InterSAGE's accountability model from application-level logging or blockchain-anchored ledgers is its reliance on *trusted, kernel-mediated cryptography*. The kernel (whether running locally or as a remote service) provides a verifiable, isolated

trust boundary, typically anchored by hardware mechanisms like Trusted Execution Environments (TEEs) and secure boot. Because the agent's private key K_{priv} never leaves this kernel, the architecture yields three key gains: (1) *Decentralized non-repudiation*: agents can cryptographically prove their execution history and attribute costs without relying on a global consensus ledger or trusted third-party auditor. (2) *Deployment ubiquity*: immutable audit trails can be maintained even in air-gapped, edge, or resource-constrained environments where blockchain nodes are economically or technically unviable. (3) *Compromise containment*: because the application logic cannot access the raw signing key or rewrite historical logs, an attacker compromising the agent's LLM or memory cannot forge retroactive trace entries, guaranteeing the integrity of the audit trail up to the exact moment of breach.

Contrast with BlockA2A. In BlockA2A [23], audit trails were anchored to a blockchain via Merkle proofs, providing immutability backed by distributed consensus. InterSAGE achieves comparable tamper-evidence through identity-signed hash chains, which work in any environment. For deployments requiring stronger immutability guarantees, the hash chain root can be periodically anchored to a public timestamping service or distributed ledger.

7.4 Non-Repudiation

Non-repudiation in InterSAGE rests on two pillars:

1. **Kernel-mediated signing:** The agent's private key K_{priv} is held exclusively by the kernel. All signing operations (attestation challenges, session tokens, trace entries, usage records) are mediated by the kernel's signing API. The agent's application logic cannot forge signatures because it never has access to the raw key.
2. **AIC chain binding:** Every signature can be verified against the agent's AIC, which chains back to the GAR root key. The verifier needs only the GAR's public key (a well-known trust anchor) to validate any trace entry from any agent in the ecosystem.

Together, these mechanisms ensure that if a trace entry bears a valid signature under an agent's K_{pub}, and the agent's AIC chain verifies back to the GAR root, then the named agent *did* perform the recorded action. The agent cannot repudiate it without claiming that the kernel was compromised—a claim that can be evaluated against the key protection tier (Table 3) and deployment attestation records.

8 Security Analysis

A protocol suite spanning identity, discovery, trust negotiation, and accountability must be evaluated by whether its properties hold jointly under a defined attacker model. This section provides a rigorous, property-centric security evaluation of InterSAGE. Table 5 summarizes the security properties provided by InterSAGE.

8.1 Threat Model and Assumptions

We assume a *Dolev-Yao network adversary* [46] capable of intercepting, replaying, and injecting messages across all network paths. Crucially, the adversary is augmented with *LLM-level compromise capabilities*: it can control an agent's application logic, issue arbitrary API calls, and observe context data (e.g., via prompt injection or malicious memory retrieval). However, the attacker *cannot* extract the agent's private key K_{priv} from the underlying kernel or compromise the kernel's signing API.

Our evaluation rests on three cryptographic and structural assumptions: (1) **GAR Honesty**: The Global Agent Registry root key is uncompromised and accurately verifies developer/operator bindings during AIC issuance. (2) **Kernel Integrity**: The agent's key-custody layer (the kernel) enforces delegation bounds and never signs payloads without authorization. (3) **Cryptographic Soundness**: Ed25519 provides existential

Table 5: *Security properties of the InterSAGE protocol suite. Threats refer to Table 1.*

Property	Mechanism	Guarantee Level	Threats
Identity Integrity	Four-dimensional AIC binding, GAR root anchor	Cryptographic	T1, T4
Capability Confinement	Monotonic attenuation chain, two-tier access control	Structural	T2
Delegation Safety	Subset enforcement, depth bounds, tenant isolation	Crypto + Structural	T2, T3
Discovery Integrity	Verifiable credentials, supply-chain validation	Verifiable Cred.	T1, T2
Accountability	Kernel-mediated signing, hash-linked execution logs	Cryptographic	T5, T6

unforgeability (EUF-CMA) and H is collision-resistant. Graceful degradation assumptions, such as loose time synchronization and OIDC provider honesty, limit the scope of localized failures without causing systemic compromise.

8.2 Property-Centric Security Evaluation

We prove that InterSAGE satisfies its core security properties under the defined threat model, directly neutralizing threats T1–T6 (Table 1).

Identity & Authenticity (T1, T4). An adversary attempting to spoof an agent's identity (T1) must forge the Agent Identity Credential (AIC). The AIC requires four independently verified signatures: developer code-package digest, build pipeline attestation, OIDC operator binding, and the final GAR endorsement. Because each dimension is cryptographically bound and verified prior to GAR issuance, single-dimension compromise (e.g., a rogue OIDC provider) cannot yield full identity forgery. At runtime, mutual attestation defeats replay attacks by requiring fresh nonce signing, proving possession of K_{priv}. Furthermore, the GAR acts as a shared root of trust, enabling cross-domain verification (T4) via a single AIC chain, eliminating the need for pairwise trust agreements.

Capability Confinement & Delegation Safety (T2, T3). InterSAGE neutralizes capability escalation (T2) structurally. The monotonic attenuation chain ensures that permissions can only be narrowed at each delegation stage (developer $\rightarrow$ operator $\rightarrow$ GAR $\rightarrow$ session). For any chain $\text{AIC}_0 \rightarrow \cdots \rightarrow \text{AIC}_n$, the invariant $S_{\max}^{\text{AIC}_n} \subseteq \cdots \subseteq S_{\max}^{\text{AIC}_0}$ is enforced cryptographically; violating it requires forging a signature at a prior stage. At runtime, the two-tier access control model computes $S_{\text{session}} = S_{\text{policy}} \cap (S_{\max}^{A} \cap S_{\max}^{B})$. Even if the application-level policy (S_{policy}) is compromised or misconfigured, it cannot grant access beyond the cryptographically verified infrastructure bound. Delegation abuse (T3) is prevented via depth bounding, strict validity capping, tenant isolation, and cascading revocation, ensuring delegation cannot be used as a capability-laundering channel.

Discovery Integrity (T1, T2). Existing discovery directories suffer from self-declared, unverified claims. InterSAGE transforms discovery into a verify-then-interact boundary using Verifiable Credentials (VCs). An agent cannot advertise unauthorized skills because manifest VCs undergo strict verification: supply-chain signature checks, subject binding against the presenter's $\text{DID}_{\text{agent}}$, and permission alignment against the agent's $S_{\max}$. Consequently, discovery cannot serve as an implicit escalation path.

Accountability & Non-repudiation (T5, T6). Action repudiation (T6) and usage fraud (T5) are mitigated without relying on distributed consensus. Every operational trace entry is signed by K_{priv} under kernel mediation. Since application logic never possesses K_{priv}, an LLM-compromised agent cannot forge signatures. Furthermore, trace entries are linked via cryptographic hashes (`prev_hash`); an attacker modifying past entries invalidates the chain. Hence, usage records remain cryptographically attributable up the delegation tree to a human-accountable operator.

Table 6: *Security guarantee comparison under failure scenarios. ✓ = guarantee holds, × = guarantee breaks, △ = bounded break.*

Scenario	InterSAGE	AgentMesh [19]	AIP [21]	BlockA2A [23]
PDP misconfigured to allow `tools:*`	✓ (bound by S_{max} intersection)	× (single-PDP silently grants access)	△ (depends on Datalog correctness)	✓ (smart contract enforced)
Operator credential compromised	△ (developer + code signatures remain valid)	× (workload impersonation)	× (root issuer mints full capabilities)	△ (no dev/operator separation)
Child escalating beyond parent's S_{max}	✓ (cryptographic subset enforcement)	△ (runtime policy error enables escalation)	✓ (attenuation at token level)	— (no protocol-level delegation attenuation)
Attacker rewrites past execution logs	✓ (K_{priv} kernel-mediated, hash chain)	△ (application-level logging)	△ (application-level logging)	✓ (blockchain immutability)

8.3 Security under Composition

A common pitfall in protocol design is composition failure, where security properties holding in isolation break when layers interact. InterSAGE achieves composition safety through strict *downward-only dependencies* and *independent failure domains*. The guarantees of higher layers rely only on the invariants of lower layers, never the reverse. For instance, L2 session capability intersection relies on L0's AIC integrity; however, a misconfigured L2 application policy cannot widen L0's cryptographic capability boundary. Similarly, L3 accountability depends solely on L0's kernel-mediated key custody, remaining tamper-evident regardless of L1 discovery mechanisms or L2 trust negotiation flaws. This structural isolation ensures that a compromise at the application or discovery layer cannot weaken the cryptographic and identity invariants enforced by the infrastructure.

8.4 Comparison and Residual Risks

Table 6 illustrates InterSAGE's structural guarantees compared to AgentMesh [19], AIP [21], and BlockA2A [23] under specific failure scenarios. Where existing approaches rely on correct runtime policy configuration (e.g., AgentMesh) or omit protocol-level delegation attenuation (e.g., BlockA2A), InterSAGE provides cryptographic bounds that fail closed independently of the policy engine.

Scope Boundaries. Certain risks fall explicitly outside InterSAGE's protocol scope. InterSAGE secures *identity and authorization limits*, not cognitive correctness. If an agent suffers a prompt injection that causes it to misuse its *authorized* capabilities, this is an LLM-level cognitive failure rather than a protocol flaw. Furthermore, catastrophic infrastructure compromise (e.g., GAR root key theft) or hardware-level key extraction (bypassing the kernel) represent ultimate trust boundaries mitigated by external operational security measures, such as HSMs and Trusted Execution Environments (TEEs), rather than protocol-level invariants.

9 Related Work

How we contrast InterSAGE with prior work. We organize the rapidly growing body of agent protocol and security research into six clusters and contrast InterSAGE with each using a uniform three-step pattern: (i) the goal or mechanism that InterSAGE *shares* with the cluster, (ii) the specific point of *divergence*, named in the canonical vocabulary of persistent identity, capability-aware discovery, trust negotiation, and accountability, and (iii) the operationally observable *consequence*—typically a concrete attacker action or misconfiguration that InterSAGE structurally denies and the comparator catches only via runtime

Table 7: *Comparison of InterSAGE with representative agent security approaches. ● = comprehensive support, ○ = partial support, — = not addressed. Rows follow the InterSAGE four-layer suite (Figure 1): L0 Agent Identity (1–2), L1 Discovery (3–4), L2 Trust Negotiation (5–8 in protocol flow order), L3 Accountability (9–12), then deployment generality (13). The leftmost column labels the protocol layer for each block;* Cross *marks a cross-cutting deployment dimension. For deployment generality: ● = cloud + edge + air-gapped, ○ = cloud-only or specific infrastructure, — = blockchain required.*

Layer	Dimension	*InterSAGE*	*AgentMesh*	*AIP*	*HDP*	*ANP*	*Ag-OSI*	*ZT-IAM*	*Blockа2A*
L0	Persistent identity	●	●	○	—	○	○	●	●
	4-dim binding	●	○	—	—	—	—	○	—
L1	Capability-aware discovery	●	○	—	—	○	○	○	—
	Verifiable manifests	●	—	—	—	—	—	○	—
L2	Mutual attestation	●	○	○	—	○	—	○	○
	Monotonic attenuation	●	—	○	—	—	—	—	—
	Two-tier access control	●	—	—	—	—	—	—	○
	Delegation chains	●	○	○	●	—	—	○	—
L3	Token-usage tracing	●	○	—	—	—	—	—	—
	Payment primitives	●	—	—	—	—	●	—	—
	Action accountability	●	●	○	○	—	○	○	●
	Non-repudiation	●	○	○	○	—	—	—	●
Cross	Deployment generality	●	○	●	●	○	—	○	—

policy evaluation, behavioral attestation, or operator vigilance. We use this pattern because it forces every distinction claim to be grounded in mechanism rather than rhetoric.

What InterSAGE inherits. InterSAGE is not built from scratch. It inherits decentralized identifiers and verifiable credentials from the W3C and SSI lineage, SPIFFE-style workload attestation, OAuth/OIDC scoping vocabulary, capability tokens in the UCAN/Biscuit lineage, the idea of monotonic attenuation as a delegation discipline, signed software manifests, and tamper-evident hash-chained audit. InterSAGE's contribution is the *architecture that combines and structurally enforces* these primitives within a single coherent four-layer trust plane; we therefore foreground the conjunction rather than any individual primitive.

Table 7 compares InterSAGE with representative approaches across thirteen evaluation dimensions ordered to mirror the four-layer trust substrate in Figure 1. *Layer 0 (Agent Identity)* uses rows 1–2 (*persistent identity*, *four-dimensional binding*). *Layer 1 (Discovery)* uses rows 3–4 (*capability-aware discovery*, *verifiable manifests*). *Layer 2 (Trust Negotiation)* uses rows 5–8 in the same sequence as §3.2: *mutual attestation*, *monotonic attenuation*, *two-tier access control*, then *delegation chains*. *Layer 3 (Accountability)* uses rows 9–12 in the same sequence as §7: *token-usage tracing*, *payment primitives*, *action accountability*, *non-repudiation*. Row 13 is cross-cutting *deployment generality* (independent of any single layer).

9.1 Communication Stacks vs. Security-First Substrates

A first cluster of work designs communication-first stacks for the Internet of Agents. Fleming et al. [11] propose reference layers for agent communication (L8) and semantics (L9) atop TCP/IP (outside the classical OSI model); Agent-OSI [10] proposes a six-layer decentralized stack with identity, settlement, and provenance; ACPS [14] defines registration, discovery, interaction, and tooling protocols; Coral Protocol [29] provides open infrastructure for communication, coordination, trust, and payments; the OpenAgents Network Model [47] defines an event-centered network abstraction with scoped networks, routable addresses, mods,

resources, discovery, and transport bindings; Du et al. [48] analyze agent communication from five classical Internet-architecture perspectives; the IoA framework of Chen et al. [4] and the survey by Wang et al. [9] provide ecosystem perspectives; and Google's Agent Payments Protocol (AP2) [45] is a prominent open effort focused on AI-driven agent payments. A parallel cluster of comparative surveys [12, 13, 30, 49, 50, 51, 52] catalog and compare these efforts (including landscape surveys with limited security depth, such as [50, 51]); across these surveys, security is typically treated as one dimension among several rather than as a dedicated, cryptographically bound trust layer.

Relation to InterSAGE. InterSAGE shares with these works the goal of supporting interoperable multi-agent collaboration; it diverges by being a security-first protocol suite that is agnostic to the underlying communication stack, and is therefore designed to *complement* rather than compete with any of the architectures above. In particular, OpenAgents is best understood as a network model rather than an in-depth security or verifiability design: its verification levels and guard mods name where authentication, access control, and rate limiting can occur, but they do not specify AIC-style multi-dimensional identity binding, signed capability manifests, monotonic attenuation, two-tier authorization, or tamper-evident audit. Similarly, while ANP [8] uses W3C DIDs for discovery and mutual verification, its published design does not specify structured delegation chains or a tamper-evident accountability layer comparable to InterSAGE; attestation and access control are lighter than cryptographically bound, challenge-response attestation with monotonic capability attenuation. Agent-OSI [10] provides comprehensive layering and prototypes on-chain escrow and verification in its reference implementation, limiting deployment generality where blockchain is required. The observable consequence is that InterSAGE can layer on top of MCP, A2A, ANP, ACPS, Agent-OSI, or OpenAgents-style event networks without changes to the host protocol's wire format.

9.2 Single-Dimension IAM vs. Multi-Dimensional Binding

A second cluster retrofits human-IAM primitives onto agent flows. The OpenID Foundation's strategic agenda [25] catalogs the gaps in extending OAuth 2.0 and OpenID Connect to agentic authorization; OIDC-A [24] is the most concrete OIDC extension to date, defining standard claims for agent identity, attestation, and delegation chains. South et al. [36, 53] extend OAuth 2.0 with agent-specific credentials and natural-language permission scoping. Bhushan et al. [37] provide a five-pattern taxonomy organizing SPIFFE/SPIRE, OAuth 2.0, OIDC, Token Exchange, DPoP, CIBA, and decentralized identity by interaction type (user-to-agent, orchestrator-to-agent, agent-to-internal, agent-to-external, cross-domain federation). On the decentralized-identity side, Aydeger and Zeydan [54] integrate SSI with LLM agents on a blockchain backend; AGNTCY Identity [55] and BillionsNetwork [56] provide open-source toolkits using W3C VCs and the iden3 protocol respectively.

Relation to InterSAGE. InterSAGE shares with this cluster the use of standardized credential formats and the goal of cross-domain agent authentication. It diverges by defining an agent-native AIC lifecycle whose four-dimensional binding (developer, code package, operator, context) is signed at issuance, rather than treating the agent as a single OAuth client or a single VC holder. The consequence is that an attacker who steals an operator's OAuth client secret or who phishes a holder key cannot silently substitute a different code package or claim a different developer, because each dimension carries its own independently verifiable signature—a property no OAuth or single-holder DID extension provides.

9.3 Directory Discovery vs. Capability-Aware Manifests

A third cluster builds zero-trust identity and discovery infrastructure for agents. The DID/VC + Agent Name Service strand—Huang et al.'s zero-trust framework with DIDs, VCs, ANS, and ZKPs [41], the ANS itself as a DNS-inspired PKI directory [57], and Garzon et al.'s ledger-anchored DID/VC prototype for cross-domain LLM-agent authentication [43]—focuses on *discovery and authentication*. The SPIFFE-on-

Kubernetes strand—Huang and Hughes's Springer chapter on agentic AI identity security [39], Pappu et al.'s SPIFFE-based zero-trust authentication [40], Bhushan's explainable zero-trust framework [58], and Palavali's SSI-for-microservices framework [59]—focuses on *workload attestation and short-lived credentials*. A third strand frames the problem at the enterprise governance layer: Ramachandran and Mishra's identity-aware governance [22] explicitly proposes ambient-authority elimination, infrastructure-level policy enforcement, sequence-aware authorization, independent action verification, and hallucination-aware audit, and grounds these in documented production incidents.

Relation to InterSAGE. InterSAGE shares with this cluster the zero-trust posture and the use of DID/SPIFFE-style cryptographic identity. It diverges in two ways. First, the DID/VC works typically bind only the *holder* dimension and the SPIFFE works only the *workload-instance* dimension; InterSAGE's AIC binds developer, code package, operator, and context as independently signed dimensions, so the four-dimensional binding strictly subsumes both. Second, the enterprise-governance proposals share InterSAGE's two-tier intuition (separating infrastructure from application policy) but realize it through a single PDP at the infrastructure layer; InterSAGE realizes it as two independent evaluation paths with independent failure semantics. The consequence is that a PDP misconfiguration in the application tier cannot weaken cryptographic AIC verification in the infrastructure tier, and vice versa—each tier fails closed independently.

9.4 Policy-Evaluated vs. Structurally-Attenuated Delegation

A fourth cluster focuses specifically on *delegation chains* as first-class protocol artifacts. We discuss this cluster in detail because it is where the monotonic-capability-attenuation primitive of InterSAGE has the closest competitors.

AIP [21] introduces Invocation-Bound Capability Tokens (IBCTs) that fuse identity, attenuated authorization, and provenance into an append-only token chain, with compact (signed JWT) and chained (Biscuit/Datalog) wire formats and transport bindings for MCP, A2A, and HTTP. LDP [60] extends this with rich delegate identity cards carrying quality hints, governed sessions, and trust domains as protocol-level boundaries. HDP [61] (an IETF Internet-Draft) provides a lightweight Ed25519 hop-chain dedicated to *human delegation provenance* in agentic systems. Saavedra's framework [26] introduces Delegation Grants (DGs)—first-class authorization artifacts with explicitly enforced scope reduction—together with a Canonical Verification Context, a Trust Gateway, and optional blockchain anchoring. South et al.'s authenticated delegation [36, 53] extends OAuth 2.0/OIDC with agent-specific credentials and natural-language scoping for delegation chains.

Relation to InterSAGE. All five works share with InterSAGE the goal of bounding what a delegated agent may do and producing an auditable chain of authority; AIP and Saavedra's DGs go furthest by making attenuation a structural property of the credential rather than only a runtime check. The divergences are mechanism-specific. AIP's attenuation is *invocation-scoped* and *policy-evaluated*: each IBCT carries Datalog rules re-evaluated at every hop, and the verifier must run a Datalog engine to reject a malformed token. InterSAGE's attenuation is *lifecycle-scoped* and *intersection-evaluated*: the capability boundary is signed into the AIC at issuance and every subsequent narrowing reduces to set-intersection plus signature verification—no Datalog runtime, no per-hop policy interpretation. Neither AIP nor Saavedra's DGs separate *developer* from *operator* as distinct issuance stages, so a compromised AIP root issuer or a compromised DG-issuing trust gateway can mint a credential with the full upstream capability set; InterSAGE's four-dimensional binding makes the developer-declared boundary and the operator-provisioned subset two distinct cryptographic stages, so an operator-tier compromise structurally cannot escalate beyond the developer-tier boundary. HDP and the OAuth-extension delegation proposals address human-to-agent provenance and OAuth compatibility respectively, but do not extend monotonic attenuation across the multi-stage agent supply chain. The observable consequence is that the two attack categories that AIP's

chained model uniquely catches (delegation-depth violations and audit evasion via empty context [21]) are also caught by InterSAGE, plus a third category—operator-tier capability widening—that AIP's single-issuer model does not catch.

9.5 Conformance Testing vs. Cryptographic Enforcement

A fifth cluster articulates security principles and threat models for the agentic ecosystem rather than full protocols. AgentRFC [16] is the closest principles framework to InterSAGE: it defines a six-layer Agent Protocol Stack analogous to ITU-T X.800 for OSI, formalizes eleven security principles as TLA+ invariants with an explicit property taxonomy (spec-mandated, spec-recommended, AASM-hardening, APS-completeness), and introduces the Composition Safety principle—the observation that security properties holding for individual protocols can break when protocols are composed through shared infrastructure. The AgentConform tooling extracts normative clauses into a typed Protocol IR and model-checks the resulting TLA+ model. Anbiaee et al. [17] present comparative threat modeling across MCP, A2A, Agora, and ANP and enumerate twelve protocol-level risks. Wibowo and Polyzos [18] argue that agentic safety is an architectural principle rather than an add-on, and bottom-up deconstruct single-, multi-, and interoperable-multi-agent stacks. Chaffer's "Know Your Agent" [62] proposes a governance frame centered on identity verification, behavioral monitoring, and accountability; the agent discovery survey by Guo et al. [52] introduces a two-stage capability-discovery framework with semantic modeling.

Relation to InterSAGE. InterSAGE shares with AgentRFC the diagnosis that agent protocols need a principled security framework, and we adopt the Composition Safety principle in our threat model (§2.2). The divergence is that AgentRFC's invariants are *model-checked* properties intended for conformance testing of independent protocols, whereas InterSAGE's invariants are *cryptographically structural*—the monotonic capability attenuation chain enforces the bound at the credential level, not at the spec-conformance level. The consequence is complementary: AgentRFC can flag where existing specs leave Composition Safety gaps; InterSAGE provides a credential format that closes those gaps by construction. The threat-modeling and governance-framework works are likewise complementary—they identify risks that InterSAGE is designed to neutralize architecturally rather than articulate.

9.6 Single-Plane Overlays vs. Two-Tier Authorization

The sixth cluster contains the two industry- or industry-adjacent efforts that share InterSAGE's framing as a trust layer added on top of existing communication protocols: Microsoft's AgentMesh [19] and Huang et al.'s unified zero-trust architecture for the agentic web [20].

AgentMesh. AgentMesh, released as part of the Agent Governance Toolkit, is marketed in its documentation as "SSL for AI agents" (Public Preview). It organizes its architecture into four layers: (i) an identity and zero-trust core using Agent CA with SPIFFE/SVID identities and Ed25519 or ML-DSA-65 signatures; (ii) a trust and protocol bridge that translates between A2A, MCP, and IATP with capability scoping; (iii) a compliance and policy plane; and (iv) a reward and learning engine. Project documentation claims alignment with the OWASP Agentic Top 10 categories and ships an MCP proxy.

Relation to InterSAGE. InterSAGE and AgentMesh share the high-level goal of adding a trust layer to the multi-agent ecosystem and both use signed credentials for identity. They differ in five mechanism-level ways, each with a concrete observable consequence:

1. **Identity model.** AgentMesh's SPIFFE-based identity binds a key to a single workload instance; InterSAGE's AIC binds developer, code package, operator, and operational context as four independently signed dimensions. *Consequence:* an attacker who compromises an AgentMesh workload signs valid SVIDs for the workload's identity but cannot prove anything about its code provenance; under In-

terSAGE, the same compromise cannot impersonate a different developer or substitute a different code package because the developer and code-package signatures are independent of the operator-tier credential.

2. **Bound enforcement: structural vs. policy-based.** AgentMesh enforces least-privilege through a runtime policy engine: a misconfigured policy rule (e.g., a tenant operator accidentally granting `tools:*`) can silently grant excessive access. InterSAGE's monotonic capability attenuation chain encodes the bound at credential issuance, so the same policy edit cannot widen capability beyond the developer-declared boundary regardless of any subsequent runtime policy decision. *Consequence:* InterSAGE remains safe under PDP misconfiguration in a way AgentMesh does not.
3. **Access-control architecture.** AgentMesh uses a single policy plane that conflates cryptographic verification with declarative authorization. InterSAGE's two-tier access control keeps infrastructure-tier checks (AIC validity, capability intersection) on a separate evaluation path from application-tier declarative rules. *Consequence:* an application-tier regression cannot weaken cryptographic checks, and an infrastructure-tier credential lapse cannot bypass per-agent authorization—each failure mode is independently auditable.
4. **Protocol relationship.** AgentMesh ships explicit A2A/MCP/IATP protocol translators; InterSAGE is protocol-agnostic and provides trust primitives that any host protocol can bind to. *Consequence:* InterSAGE does not require a translator update to support a new host protocol, and protocol additions do not expand the trust-layer attack surface.
5. **Scope.** AgentMesh embeds a reward and learning engine for adaptive governance. InterSAGE deliberately confines itself to the trust and accountability layer and treats adaptive behavior as orthogonal. *Consequence:* InterSAGE's verifier semantics remain pure cryptographic checks, simplifying formal verification and audit.

Unified zero-trust architecture (Huang et al.). InterSAGE's closest *academic* comparator is Huang et al.'s unified zero-trust architecture [20], which likewise proposes a unified zero-trust architecture for the agentic web. It builds on DIDs, VCs, and ANS for identity and discovery and adds three mechanisms: Trust-Adaptive Runtime Environments (TARE), Causal Chain Auditing, and Dynamic Identity with Behavioral Attestation. It shares with InterSAGE the diagnosis that the agentic web needs a unified trust layer; it diverges by enforcing trust through *runtime behavioral attestation and trust scoring* rather than through issuance-time cryptographic attenuation. The observable consequence is that an agent whose behavior has not yet diverged from baseline receives a high trust score from TARE even if its capability boundary has been silently widened upstream; under InterSAGE the same agent's AIC-bounded capability set is unchanged regardless of behavioral history.

9.7 Our Prior Work: BlockA2A

BlockA2A [23] introduced a unified multi-agent trust framework combining DIDs for cross-domain authentication, blockchain-anchored ledgers for immutable audit, and smart contracts for dynamic access control. It demonstrated effective defense against diverse MAS attacks and practical integration with A2A [6].

Evolution to InterSAGE. InterSAGE preserves BlockA2A's core security ambitions (unified trust, immutable audit, dynamic access control) while making three key advances: (1) removing the blockchain dependency in favor of general-purpose cryptographic primitives, broadening deployment to edge and air-gapped environments; (2) introducing layer-aligned primitives for persistent identity, capability-aware discovery, trust negotiation, and accountability, whereas BlockA2A approximated identity and authorization only at the smart-contract policy level; and (3) adding DID-bound skill/tool manifest VCs, payment, and token-usage tracing capabilities that BlockA2A did not address. InterSAGE can be viewed as the generalization

of BlockA2A from a blockchain-specific realization to a deployment-agnostic protocol suite.

9.8 Summary: the Joint-Coverage Argument

Each of InterSAGE's primitives is anticipated by at least one prior work surveyed above: DID/VC and SPIFFE precede the AIC's identity machinery; UCAN, Biscuit, AIP IBCTs, and Saavedra DGs precede the attenuation discipline; AgentMesh's policy plane and the enterprise governance frameworks precede the two-tier intuition; AgentRFC precedes the principled-invariants framing; and BlockA2A precedes the unified-trust ambition. InterSAGE's contribution is twofold.

First, InterSAGE *deepens* each primitive beyond its nearest predecessor. Where SPIFFE, OAuth, and DIDs each bind a single identity dimension, the AIC binds four independently signed dimensions so that compromising one tier cannot impersonate another. Where A2A Agent Cards and ANP's DID-based discovery treat skill and tool advertisements as self-declared descriptions, InterSAGE represents each skill and tool as a DID-bound Verifiable Credential that is verified for supply-chain authenticity, subject binding, and permission alignment at discovery time—making discovery a trust-establishing security boundary rather than a directory lookup. Where AIP and Saavedra DGs attenuate capabilities via per-hop Datalog or runtime policy, InterSAGE's monotonic chain signs the capability boundary at issuance and reduces verification to signature checking, so the bound holds even when the policy engine is wrong. Where AgentMesh and the enterprise governance frameworks route cryptographic checks and application policy through a single PDP, InterSAGE's two-tier split keeps them on independent evaluation paths with independent failure semantics, so a regression in one tier cannot weaken the other. And where BlockA2A anchors audit trails to blockchain consensus, InterSAGE achieves tamper-evident non-repudiation via kernel-mediated signing that works in air-gapped and resource-constrained environments without a distributed ledger.

Second, InterSAGE provides the *conjunction*: no prior single architecture jointly enforces persistent identity with four-dimensional binding, capability-aware discovery through DID-bound Verifiable Credential manifests, trust negotiation that combines lifecycle-scoped monotonic capability attenuation with two-tier access control, and kernel-mediated cryptographic audit trails. Table 7 makes both contributions visible across thirteen evaluation dimensions: every prior approach achieves at most a strict subset of InterSAGE's coverage in breadth, and the per-primitive depth gap is most pronounced precisely where structural guarantees matter—under operator compromise, PDP misconfiguration, or cross-protocol composition.

10 Discussion & Future Directions

In this section, we discuss the limitations of the InterSAGE framework and outline key directions for future research. We first acknowledge the boundaries of this positioning paper and the practical challenges of deploying a global trust anchor. We then highlight promising avenues for future work, ranging from formal verification and privacy-preserving extensions to scalable infrastructure and integration with existing agent frameworks.

10.1 Limitations

Positioning scope. InterSAGE is a positioning paper that establishes the conceptual framework, protocol architecture, and design rationale. A complete protocol specification would additionally require formal proofs of security properties, quantitative performance evidence, implementation detail, and adversarial validation; we defer these to companion work. Concretely, the following aspects are explicitly out of scope here:

- *Formal verification:* TLA+ or ProVerif models of the attestation and delegation protocols.
- *Performance evaluation:* Latency and throughput benchmarks under realistic multi-agent workloads.

- *Implementation details:* A companion DeepKernel system paper will describe the full implementation including kernel architecture, eBPF enforcement, and perception/cognition layers.
- *Adversarial evaluation:* Red-team exercises and attack simulation against the protocol suite.

GAR as a trust anchor. The Global Agent Registry serves as a centralized (or federated) trust anchor, analogous to a Certificate Authority. This introduces the same trade-offs as traditional PKI: the GAR must be highly available, its compromise would be catastrophic, and cross-GAR trust requires federation agreements. Decentralized alternatives (e.g., blockchain-backed GARs, web of trust models) can mitigate these risks but introduce latency and governance complexity.

Adoption bootstrapping. The value of InterSAGE increases with adoption (network effects). Early deployments may face a chicken-and-egg problem where few agents support InterSAGE's identity layer, reducing the incentive for others to adopt. We envision a phased adoption path: Layer 0 (identity) can be adopted independently of Layers 1–3, providing immediate value for agent authentication even when only a subset of the ecosystem participates.

Semantic tag governance. Layer 1's semantic tagging relies on an open vocabulary. Without governance, semantic tags may become fragmented or misleading. We anticipate that domain-specific tag ontologies will emerge organically (similar to Docker image tags or npm package keywords) and can be curated by registry operators.

10.2 Future Directions

Formal verification. The attestation and delegation protocols lend themselves to formal verification using tools such as TLA+ [16] for safety properties and ProVerif or Tamarin for cryptographic protocol analysis. Formalizing the monotonic attenuation chain as a lattice-theoretic invariant would strengthen the structural guarantee claims.

Attestation deployment topologies. The mutual attestation protocol (§6.1) is parameterized by the relationship between Kernel_A and Kernel_B, yielding three deployment modes with distinct trust and performance profiles. *Co-located attestation* ($\text{Kernel}_A = \text{Kernel}_B$) is the lowest-latency option, natural for DeepKernel-managed multi-agent orchestration on a single host; the entire challenge-response collapses to kernel-internal operations with no network round-trip. *Remote-mediated attestation* interposes a Trusted Attestation Service (TAS)—architecturally identical to a local agent kernel but deployed as a cloud endpoint—that coordinates nonce exchange, caches GAR verification results, and issues session tokens. The TAS plays a role analogous to an OIDC Provider in human web SSO: it centralizes session establishment at the cost of introducing a single point of trust that must itself be highly available and key-protected. *Direct peer-to-peer attestation* eliminates the intermediary: each kernel signs its own nonce and independently verifies the counterpart's signature via the GAR. This mode is fully decentralized but incurs at least two independent GAR round-trips per attestation; AIC caching with revocation-push notifications can amortize the cost. The three modes can coexist within a single ecosystem—co-located for intra-device workflows, mediated for organizational clusters, and direct P2P for open cross-organization interactions—and the choice is determined by the deployment context rather than by the protocol itself.

Privacy-preserving extensions. The current design exposes the requester's full AIC attributes to the responder during Layer 2 attestation. Zero-knowledge proofs (ZKPs) could enable selective disclosure: an agent could prove it possesses a required capability or meets an IAL threshold without revealing its full identity. This is particularly relevant for privacy-sensitive domains such as healthcare and finance.

Integration with agent frameworks. InterSAGE is designed to be framework-agnostic, but practical adoption requires SDK integrations with popular agent frameworks (LangGraph, AutoGen, CrewAI, Semantic

Kernel) and vendor SDKs (OpenAI Agents SDK [63], Google ADK [64]). We envision thin adapter libraries that bridge each framework's identity and tool invocation APIs to InterSAGE's protocol primitives. AgentMesh's approach of building explicit protocol translators for A2A, MCP, and IATP (in the same Agent Governance Toolkit monorepo as AgentMesh) [19] suggests a pragmatic integration pattern that InterSAGE could adopt for its initial deployment targets.

Scalability of the GAR. As the agent population grows to millions or billions, the GAR must scale correspondingly. Horizontal sharding by tenant or geographic region, caching at the kernel level (with revocation push notifications), and eventual consistency models for non-critical metadata are all viable strategies that need empirical evaluation.

Cross-GAR federation. In a multi-stakeholder ecosystem, multiple GARs will coexist (enterprise GARs, cloud-provider GARs, open-community GARs). Cross-GAR trust establishment requires mutual recognition protocols analogous to cross-certification in PKI or trust federation in SAML. Designing these protocols is a natural extension of InterSAGE's architecture.

Agentic payment infrastructure. Layer 3's payment primitives are deliberately minimal. A full agentic payment infrastructure would need to address escrow for long-running tasks, dispute resolution, quality-of-service guarantees, and micro-payment efficiency. Google's AP2 [45] is a leading open effort in this direction, focusing on the payment rail and transaction flow. We view InterSAGE's identity and metering layer as the foundation upon which AP2-style settlement protocols can be built, providing the "who is paying whom and with what authority" semantics that payment rails alone cannot supply.

Post-quantum readiness. AgentMesh [19] already lists ML-DSA-65 alongside Ed25519 as a supported signature scheme. As NIST post-quantum standards mature, InterSAGE's AIC and attestation protocols should be extended to support PQ signature algorithms (e.g., ML-DSA, SLH-DSA) as drop-in replacements for Ed25519, preserving the structural security properties while future-proofing the cryptographic substrate.

11 Conclusion

The Internet of Agents will not become secure merely by standardizing how agents exchange messages. The harder problem is whether independently built agents can make interoperable trust decisions: is this agent the entity it claims to be, are its advertised skills and tools genuine, can its authority only decrease as it is delegated, and can its actions later be attributed without ambiguity? This paper has argued that these questions require a trust substrate, not another communication protocol.

InterSAGE provides such a substrate as a four-layer protocol suite spanning Persistent Identity, Discovery, Trust Negotiation, and Accountability. Its principal contribution is the conjunction of four layer-aligned primitives: Agent Identity Cards with four-dimensional binding; capability-aware discovery through DID-bound skill/tool manifest VCs; trust negotiation that combines monotonic capability attenuation with two-tier access control; and kernel-mediated cryptographic audit trails. Each primitive has antecedents in existing identity, delegation, policy, or audit systems. What is missing from prior work, and what InterSAGE supplies, is their joint enforcement across the full agent lifecycle: from issuance and discovery, through negotiation and delegation, to metering and non-repudiation.

This architecture deliberately remains communication-protocol agnostic. Rather than competing with MCP, A2A, ANP, AG-UI, or future agent protocols, InterSAGE defines the trust objects those protocols can carry: AIC capability boundaries, manifest VCs, session tokens, and signed traces. This separation is what makes the design deployable across cloud, edge, and air-gapped settings without relying on a single runtime, policy engine, registry topology, or distributed ledger.

As a positioning paper, InterSAGE establishes the conceptual framework and protocol architecture;

formal verification, performance benchmarks, and a companion DeepKernel systems paper are left to ongoing work. The claim of this paper is therefore precise: communication interoperability is necessary for agents to talk, but secure interoperability is necessary for agents to act across organizational boundaries. InterSAGE is a step toward making the underlying trust layer explicit, composable, and verifiable.

References

[1] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[2] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

[3] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

[4] Weize Chen, Ziming You, Ran Li, Yitong Guan, Chen Qian, Chenyang Zhao, et al. Internet of agents: Weaving a web of heterogeneous agents for collaborative intelligence. *arXiv preprint arXiv:2407.07061*, 2024.

[5] Gartner. Gartner predicts agentic AI will autonomously resolve 80% of common customer service issues by 2029. Gartner Press Release, 2025.

[6] Google. Agent-to-agent (A2A) protocol specification. https://a2a-protocol.org/latest/specification/, 2025.

[7] Anthropic. Model context protocol (MCP) specification. https://modelcontextprotocol.io/, 2025.

[8] Gao Chang, Eddie Lin, Chongyuan Yuan, Ran Cai, Bin Chen, Xinlin Xie, et al. Agent network protocol technical white paper. *arXiv preprint arXiv:2508.00007*, 2025.

[9] Yuntao Wang, Song Guo, Yutong Pan, Zhou Su, Fangmin Chen, et al. Internet of agents: Fundamentals, applications, and challenges. *IEEE Transactions on Cognitive Communications and Networking*, 2025. arXiv:2505.07176; accepted by IEEE TCCN.

[10] Wenxin Xu, Taotao Wang, Yihan Xia, Shengli Zhang, and Soung Chang Liew. Agent-OSI: A layered protocol stack toward a decentralized internet of agents. *arXiv preprint arXiv:2602.13795*, 2026.

[11] Charles Fleming, Luca Muscariello, Vivek Pandey, et al. A layered protocol architecture for the internet of agents. *arXiv preprint arXiv:2511.19699*, 2025.

[12] Yingxuan Yang, Huayi Chai, Yuqi Song, Shuai Qi, Muning Wen, Na Li, et al. A survey of AI agent protocols. *arXiv preprint arXiv:2504.16736*, 2025.

[13] Abul Ehtesham, Aditi Singh, Gaurav Kumar Gupta, and Sandeep Kumar. A survey of agent interoperability protocols: MCP, ACP, A2A, and ANP. *arXiv preprint arXiv:2505.02279*, 2025.

[14] Chao Li, Jiaxing Wu, Qiong Du, Shui Yu, Rui Zou, Kan Yu, et al. ACPS: Agent collaboration protocols for the internet of agents. *arXiv preprint arXiv:2505.13523*, 2025.

[15] Agent Client Protocol Project. Agent client protocol (ACP): A protocol for connecting any editor to any agent. https://github.com/agentclientprotocol/agent-client-protocol, 2025.

[16] Shenghan Zheng and Qifan Zhang. AgentRFC: Security design principles and conformance testing for agent protocols. *arXiv preprint arXiv:2603.23801*, 2026.

[17] Zeynab Anbiaee, Mahdi Rabbani, Mansur Mirani, Gunjan Piya, et al. Security threat modeling for emerging AI-agent protocols: A comparative analysis of MCP, A2A, agora, and ANP. *arXiv preprint arXiv:2602.11327*, 2026.

[18] Juan A. Wibowo and George C. Polyzos. Toward a safe internet of agents. *arXiv preprint arXiv:2512.00520*, 2025.

[19] Microsoft. AgentMesh: Production-grade trust layer for multi-agent systems. https://github.com/microsoft/agent-governance-toolkit/tree/main/agent-governance-python/agent-mesh, 2026. Part of the Agent Governance Toolkit; SPIFFE/SVID-based identity, policy engine, A2A/MCP/IATP protocol bridge.

[20] Kevin Huang, Yasir Mehmood, Haris Atta, Jingbo Huang, et al. Fortifying the agentic web: A unified zero-trust architecture against logic-layer threats. *arXiv preprint arXiv:2508.12259*, 2025.

[21] Shiv Prakash. AIP: Agent identity protocol for verifiable delegation across MCP and A2A. *arXiv preprint arXiv:2603.24775*, 2026.

[22] Hari Ramachandran and Gautam Mishra. Identity-aware governance for autonomous AI agents: A framework for enterprise authorization, delegation, and audit. *SSRN*, (6439998), 2026.

[23] Zhenhua Zou, Zhuotao Liu, Lepeng Zhao, and Qiuyang Zhan. BlockA2A: Towards secure and verifiable agent-to-agent interoperability. *arXiv preprint arXiv:2508.01332*, 2025.

[24] Sudhir Nagabhushanaradhya. OpenID connect for agents (OIDC-A) 1.0: A standard extension for LLM-based agent identity and authorization. *arXiv preprint arXiv:2509.25974*, 2025.

[25] Tobin South, Sudhir Nagabhushanaradhya, et al. Identity management for agentic AI: The new frontier of authorization, authentication, and security. *arXiv preprint arXiv:2510.25819*, 2025. OpenID Foundation whitepaper.

[26] Daniel R. Saavedra. Interoperable architecture for digital identity delegation for AI agents with blockchain integration. *arXiv preprint arXiv:2601.14982*, 2026.

[27] Michael B. Jones, Brian Campbell, John Bradley, and Nat Sakimura. OAuth 2.0 token exchange (RFC 8693). IETF RFC 8693, 2020.

[28] Vivek Srinivasan. Bridging protocol and production: Design patterns for deploying AI agents with model context protocol. *arXiv preprint arXiv:2603.13417*, 2026.

[29] RJ Georgio, C Forder, S Deb, A Rahimov, et al. Coral protocol: Open infrastructure connecting the internet of agents. *arXiv preprint arXiv:2505.00749*, 2025.

[30] Hana Derouiche, Zaki Brahmi, and Haithem Mazeni. Agentic AI frameworks: Architectures, protocols, and design challenges. *arXiv preprint arXiv:2508.10146*, 2025.

[31] CopilotKit. AG-UI: The agent-user interaction protocol. https://github.com/ag-ui-protocol/ag-ui, 2025.

[32] IBM. ContextForge: An AI gateway, registry, and proxy for MCP, A2A, and REST/gRPC APIs. https://github.com/IBM/mcp-context-forge, 2026.

[33] Wild Card AI. agents.json: An open specification for API and agent interaction contracts. https://github.com/wild-card-ai/agents-json, 2025.

[34] Dick Hardt. The OAuth 2.0 authorization framework (RFC 6749). IETF RFC 6749, 2012.

[35] OpenID Foundation. OpenID connect core 1.0. https://openid.net/specs/openid-connect-core-1_0.html, 2014.

[36] Tobin South, Samuele Marro, Thomas Hardjono, Robert Mahari, et al. Authenticated delegation and authorized AI agents. *arXiv preprint arXiv:2501.09674*, 2025.

[37] Bharat Bhushan, Karthik Pappu, and Aman Mittal. A conceptual framework for authentication in agentic AI ecosystems: Protocol analysis and taxonomy. In *IEEE ICCA*, 2025.

[38] Eric Rescorla. The transport layer security (TLS) protocol version 1.3 (RFC 8446). IETF RFC 8446, 2018.

[39] Kevin Huang and Chris Hughes. Agentic AI identity security. In *Securing AI Agents*. Springer, 2025.

[40] Karthik Pappu, Bharat Bhushan, and Aman Mittal. SPIFFE-based zero-trust authentication for AI agent ecosystems. In *IEEE ICCA*, 2025.

[41] Kevin Huang, Venkata Sai Narajala, Justin Yeoh, Justin Ross, et al. A novel zero-trust identity framework for agentic AI: Decentralized authentication and fine-grained access control. *arXiv preprint arXiv:2505.19301*, 2025.

[42] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and Tim Polk. Internet X.509 public key infrastructure certificate and certificate revocation list profile (RFC 5280). IETF RFC 5280, 2008.

[43] Sandro Rodríguez Garzon, Arian Vaziry, Elif Merve Kuzu, Daniel E. Gehrmann, et al. AI agents with decentralized identifiers and verifiable credentials. *arXiv preprint arXiv:2511.02841*, 2025.

[44] W3C. Verifiable credentials data model v2.0. https://www.w3.org/TR/vc-data-model-2.0/, 2025. W3C Recommendation, 2025-05-15.

[45] Google. Agent payments protocol (AP2): Building a secure and interoperable future for AI-driven payments. https://github.com/google-agentic-commerce/AP2, 2026.

[46] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[47] OpenAgents. OpenAgents network model. https://openagents.org/docs/zh/concepts/openagents-network-model, 2026. Version 1.0; updated May 8, 2026.

[48] Chenguang Du, Chao Wang, Yihan Chao, Xin Xie, and Yong Cui. AI agent communication from internet architecture perspective: Challenges and opportunities. *arXiv preprint arXiv:2509.02317*, 2025.

[49] Cheonsu Jeong. A study on the MCP x A2A framework for enhancing interoperability of LLM-based autonomous agents. *arXiv preprint arXiv:2506.01804*, 2025.

[50] Qiang Duan and Zhihui Lu. AI agent communications in the future internet—paving a path toward the agentic web. *Future Internet (MDPI)*, 18(3):171, 2026.

[51] Zhuo Liang, Erzhuo Cui, Qian Wei, Rui She, Ting Li, Minghui Guo, et al. A2H: Agent-to-human protocol for AI agent. *arXiv preprint arXiv:2602.15831*, 2026.

[52] Song Guo, Yuntao Wang, Zhou Su, Yutong Pan, Qimei Hu, and Tom H. Luan. Agent discovery in internet of agents: Challenges and solutions. *IEEE Network*, 2026. arXiv:2511.19113.

[53] Tobin South, Samuele Marro, Thomas Hardjono, Robert Mahari, et al. Position: AI agents need authenticated delegation. In *ICML*, 2025. Position paper.

[54] Abdullah Aydeger, Engin Zeydan, et al. Decentralized digital identity management for large language model agents. *IEEE Communications Standards Magazine*, 2026.

[55] AGNTCY. AGNTCY identity: Onboarding, creating, and verifying identities for agents and MCP servers. https://github.com/agntcy/identity, 2026.

[56] BillionsNetwork. Verified agent identity: Decentralized identity management for AI agents using iden3. https://github.com/BillionsNetwork/verified-agent-identity, 2026.

[57] Kevin Huang, Venkata Sai Narajala, Idan Habler, et al. Agent name service (ANS): A universal directory for secure AI agent discovery and interoperability. *arXiv preprint arXiv:2505.10609*, 2025. arXiv:2505.10609.

[58] Bharat Bhushan. An explainable zero trust identity framework for LLMs, AI agents, and agentic AI systems. *EuroLexis Open Access Journal*, 2025.

[59] Dhanush Reddy Palavali. Agentic AI for self-sovereign identity: A decentralized zero trust framework for autonomous microservices. *IJCMI*, 2025.

[60] Shiv Prakash. LDP: An identity-aware protocol for multi-agent LLM systems. *arXiv preprint arXiv:2603.08852*, 2026.

[61] Asiri Dalugoda. HDP: A lightweight cryptographic protocol for human delegation provenance in agentic AI systems. *arXiv preprint arXiv:2604.04522*, 2026. IETF Internet-Draft draft-helixar-hdp-agentic-delegation-00.

[62] TJ Chaffer. Know your agent: Governing AI identity on the agentic web. SSRN Working Paper, 2025. SSRN 5162127.

[63] OpenAI. OpenAI agents SDK: A lightweight framework for multi-agent workflows. https://github.com/openai/openai-agents-python, 2025.

[64] Google. Agent development kit (ADK): An open-source python toolkit for building AI agents. https://github.com/google/adk-python, 2025.