Signed and Sealed: Protocol-Level Isolation with Cryptographic Integrity (PLICI) for Secure LLM Agents

anonymous
anonymous
anonymous

***Abstract*—The rapid proliferation of autonomous agentic ecosystems has introduced severe security risks, as Large Language Models (LLMs) increasingly execute authorized actions based on untrusted external data. Existing defenses against indirect prompt injection rely on probabilistic detection (e.g., LLM-as-a-Judge) or static heuristics, which remain vulnerable to adversarial optimization and impose prohibitive latency for real-time applications. We propose Protocol-Level Isolation with Cryptographic Integrity (PLICI), a deterministic middleware that enforces zero-trust provenance. PLICI combines session-scoped ephemeral Ed25519 signatures to verify origin, schema-locked enclaves to neutralize syntactic injection, and a hybrid semantic filter leveraging discriminative Natural Language Inference (NLI) to detect logic hijacking. In evaluations on the AgentDojo and MCPSecBench datasets, PLICI reduces the Attack Success Rate (ASR) for unsigned injections from 85% to 0.3% and limits successful logic drifts to $< 0.6\%$, while incurring a negligible p99 latency overhead of 48ms. By shifting security from model alignment to cryptographic protocol enforcement, PLICI establishes a verifiable root of trust for distributed AI systems, enabling secure deployment in high-stakes enterprise environments without sacrificing performance.**

***Index Terms*—Large language models (LLMs), Indirect prompt injection, Multi-agent systems, Cryptographic protocols, Semantic analysis, Data provenance, Agentic AI, LLM agents, Prompt injection, LLM Security**

I. Introduction

A. The Agentic Shift

The deployment of Large Language Models (LLMs) has evolved rapidly from passive conversational interfaces to autonomous agentic ecosystems capable of multi-step orchestration and external tool execution. This paradigm shift is driven by standardized interoperability frameworks, most notably the Model Context Protocol (MCP) and Google's Agent-to-Agent (A2A) protocols, which allow agents to dynamically discover and utilize third-party resources [1], [2], [3]. Unlike their predecessors, modern agents utilize structured function-calling interfaces to control critical infrastructure, manage financial wallets, and query proprietary databases, effectively moving the security boundary from the user's prompt to the underlying communication protocol [4].

However, this interconnectivity has outpaced security governance. As Ferrag et al. highlight in their unified threat model, the explosive proliferation of plugins and brittle integrations has introduced novel attack surfaces where "host-to-tool" and "agent-to-agent" communications become vectors for cascading compromise [1]. The industry's reliance on implicit trust between agents and their tools has rendered these systems susceptible to what we classify as *Protocol Exploits*, where the vulnerability lies not in the model's weights, but in the unverified provenance of the data it consumes.

B. The Deterministic Gap

Current defenses against these threats predominantly rely on "AI policing AI." Approaches such as PromptShield attempt to filter inputs via ontology-driven semantic validation [5], while others rely on design patterns [6] or task alignment verification [7]. Meanwhile, LLM-as-a-Judge architectures deploy secondary models to audit agent outputs. While Alharthi and Garcia report high precision in cloud logs, these probabilistic defenses remain fundamentally vulnerable to adversarial optimization.

Crucially, Shi et al. demonstrated that LLM-as-a-Judge systems can be bypassed via optimization-based prompt injection attacks (e.g., JudgeDeceiver), where carefully crafted sequences force the judge to approve malicious candidates regardless of content [8]. Even open-source hardened models like Meta SecAlign cannot fully guarantee immunity against these optimization attacks at the protocol level [9]. Furthermore, defenses like DRIFT [10] and FATH [11] introduce significant latency or rely on the LLM adhering to formatting constraints (e.g., hash tags) that sophisticated models can override via "payload splitting."

We argue that relying on probabilistic models to secure probabilistic agents creates a recursive vulnerability loop. There exists a *Deterministic Gap*: the lack of mathematically verifiable guarantees that data entering an agent's context window has not been tampered with or fabricated by a malicious tool.

C. Contributions

To bridge this gap, we introduce **Signed and Sealed**, implementing the **Protocol-Level Isolation with Cryptographic Integrity (PLICI)** framework. Our contributions are:

1) **Formalism:** We provide the first formal definition of a "SignedTool" protocol utilizing Ed25519 signatures and HashEdDSA. We replace static API keys with *Session-Scoped Ephemeral Rotation*, mitigating the persistent

threat vectors identified in multi-agent environments [12], avoiding the latency overheads associated with heavier Multi-Party Computation (MPC) or threshold schemes [13], [14].

2) **Architecture:** We introduce the *Containment Token* mechanism. By storing raw tool outputs in a schema-locked enclave and passing only cryptographic reference tokens to the LLM, we mathematically preclude the ingestion of prompt injection payloads, effectively neutralizing the "Mime-Type" attacks described by Alizadeh et al. [15].
3) **Algorithmic Defense:** We propose a novel *Hybrid Semantic Filter* that combines lightweight Vector Similarity with a distilled DeBERTa-based Natural Language Inference (NLI) model. This addresses the "Logic Hijacking" and conversation drift quantified by Shi et al. [16], ensuring tool outputs are not only relevant but logically entailed by the user's query.

II. Background and Threat Model

A. The Modern Agent Stack

The modern agentic web relies on decentralized execution environments where LLMs act as kernels orchestrating external "syscalls."

- **Model Context Protocol (MCP):** As analyzed by Yang et al. [4] and Narajala & Habler [17], MCP standardizes how agents connect to data sources, acting as a universal "USB-C for AI." While this enhances interoperability, it creates a "permissions tunnel" where a compromised MCP server inherits the trust level of the agent [18].
- **Agent-to-Agent (A2A) Communication:** Defined in frameworks surveyed by Schröder de Witt [19], A2A allows autonomous delegation (e.g., a travel agent tasking a calendar agent). Ferrag et al. note that these protocols often lack cryptographic provenance, leading to "Cascading Injections" where one compromised node infects the swarm [1].
- **Implicit Trust:** The fundamental architectural flaw is that current agents operate on an "Implicit Trust" model. Once a tool is selected, often via a vulnerable retrieval process such as *ToolHijacker* [20], the agent treats the returned data as ground truth. This necessitates a shift toward the "Zero-Trust Agentic Access" (ZTAA) paradigms proposed by Huang et al. [21] and Liu et al. [22].

B. Threat Model (Formalized)

We define a threat model $\mathcal{M}$ where the Agent $\mathcal{A}$ is benevolent but gullible, and the External Tool Provider $\mathcal{P}$ is potentially compromised or malicious. We assume the adversary cannot access the agent's weights or private system prompt but has full control over the tool's output. We visualize these attack vectors and our architectural intervention in Figure 1.

We focus on three specific attack classes derived from recent literature:

a) In-Band: Indirect Prompt Injection (IPI): As demonstrated by Alizadeh et al. using the AgentDojo benchmark, attackers can embed malicious instructions into retrieval corpora (e.g., emails, databases) [15]. When the agent processes this data, the "data" becomes "instruction," causing the agent to exfiltrate private information (e.g., "EchoLeak"). They showed that simple IPI can achieve Attack Success Rates (ASR) of roughly 20% even with safety alignments, rising significantly in complex workflows. Wu et al. validated this risk in the wild, demonstrating that 100% of tested LLM-driven email agents could be hijacked via external email resources [23].

b) Protocol: JSON Injection & Schema Bypass: Gasmi et al. conducted a comparative vulnerability assessment of Function Calling vs. MCP, revealing that MCP exhibits increased "LLM-centric exposure" [24]. They demonstrated that strict schemas are insufficient; attackers can inject malicious payloads into *valid* JSON fields (e.g., putting a jailbreak command inside a `reasoning` string field). Furthermore, Xie et al. showed that Tool Invocation Prompts (TIPs) themselves can be hijacked to cause Denial of Service (DoS) or Remote Code Execution (RCE) [25]. Additionally, Li et al. introduced "Cross-Tool Harvesting and Polluting" (XTHP), showing how attackers can leverage dependencies between tools to corrupt the agent's memory stream [26].

c) Logic: Adversarial Embeddings & Conversation Drift: Beyond syntactic attacks, attackers can employ *Logic Hijacking*. Shi et al. quantified this as "Conversation Drift" in the MCP context, where an adversary induces deviations in the agent's latent space trajectories [16]. An attacker creates a response that is mathematically similar enough to pass basic filters (Adversarial Embedding) but semantically divergent enough to steer the agent toward a malicious goal (e.g., convincing a banking agent that a transfer is required for "verification"), often leading to the unintended leakage of user prompts or confidential context to the adversary [27].

III. Methodology

We designed **Protocol-Level Isolation with Cryptographic Integrity (PLICI)**, a deterministic middleware architecture that enforces Zero-Trust Provenance between Large Language Models (LLMs) and external tools. Unlike prior probabilistic defenses (e.g., LLM-as-a-Judge), PLICI was engineered to mathematically preclude the ingestion of unsigned or semantically divergent data.

A. Architectural Formalism: The "Trusted Proxy" Topology

To address the integration challenges of non-compliant external APIs (e.g., legacy REST services), we instantiated PLICI using a **Hub-and-Spoke** topology (Figure 2). We formalized the system as a tuple $\Sigma = (\mathcal{A}, \mathcal{M}, \mathcal{P})$, where:

1) $\mathcal{A}$ (The Agent): The untrusted requester, operating within a standard runtime (e.g., LangGraph). It possessed no cryptographic keys.
2) $\mathcal{M}$ (The Verifier): The trusted middleware that held ephemeral public keys and enforced containment policies.

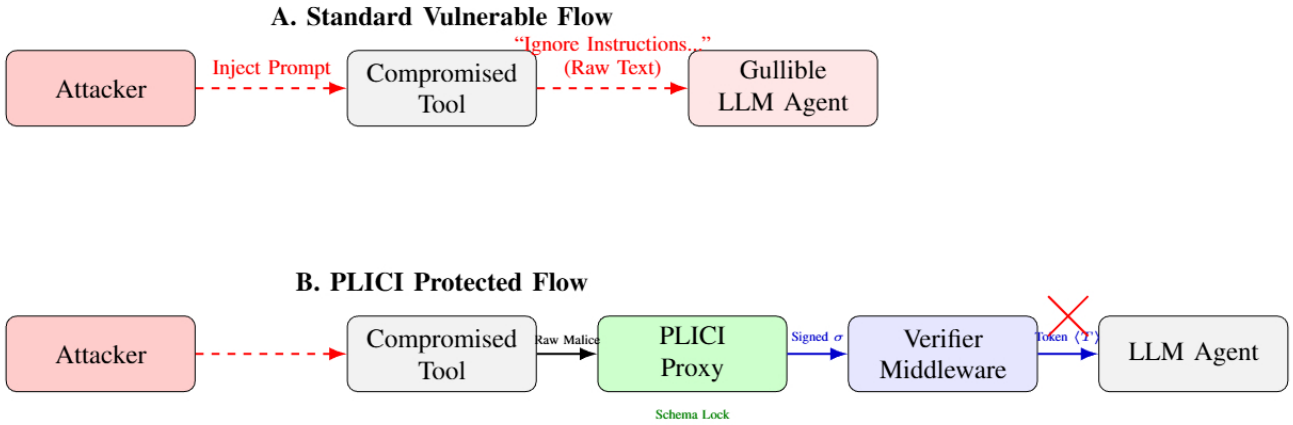

Fig. 1. The Protocol-Level Attack Surface. (A) Implicit trust allows injections to reach the LLM. (B) PLICI interposes a signing proxy and verifier, converting malicious payloads into opaque Containment Tokens.

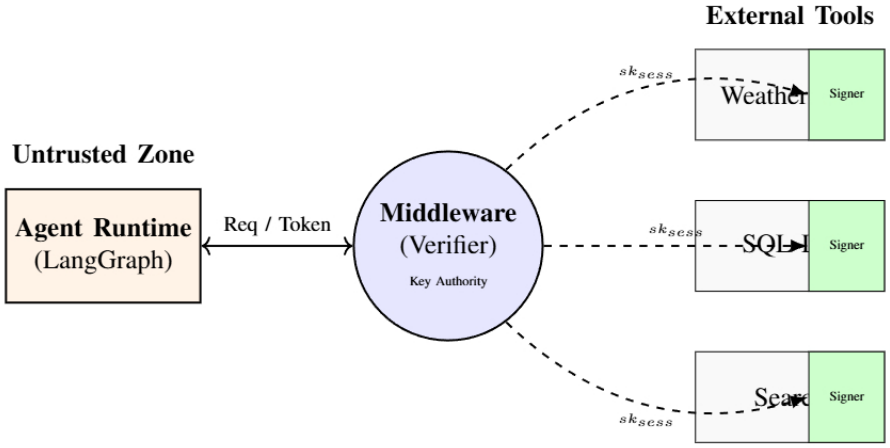

Fig. 2. PLICI Hub-and-Spoke Topology. The Middleware acts as the central Verifier and Key Authority. Lightweight "SignedTool" proxies (green) wrap legacy APIs to provide cryptographic signatures without modifying the underlying service.

3) $\mathcal{P}$ (The Prover): A set of distributed "SignedTool" proxies that wrapped external APIs, sanitizing and signing outputs before transmission.

We assumed a threat model where $\mathcal{A}$ was gullible to manipulation, and the external data sources accessed by $\mathcal{P}$ were potentially hostile (Indirect Prompt Injection).

B. Phase I: Ephemeral Cryptographic Provenance

To mitigate the "Persistent Threat" and "Replay Attack" vectors common in Agent-to-Agent (A2A) protocols, we implemented **Session-Scoped Ephemeral Rotation** based on the Ed25519 signature scheme.

1) Key Generation and Handshake: For each agent session S_i, the middleware generated a master key pair (pk_i, sk_i) using the twisted Edwards curve equation $-x^2 + y^2 = 1 - \frac{121665}{121666}x^2y^2$ over the finite field $\mathbb{F}_q$.

We defined the Session Identity ID_{sess} as a cryptographic binding of the public key and a high-precision timestamp T:

$$ID_{sess} = H(pk_i \parallel T_{start} \parallel N_{nonce}) \tag{1}$$

where H denotes SHA-512. The private key sk_i was distributed to the Signing Proxy $\mathcal{P}$ via Mutual TLS (mTLS) and was configured with a Time-To-Live (TTL) of $t = 300$ seconds, ensuring that compromised keys could not be exploited for future sessions.

2) The Signing Protocol: Upon receiving a tool execution result M, the Proxy $\mathcal{P}$ computed the signature σ. Unlike standard EdDSA which signs the message directly, we utilized **HashEdDSA** to handle large JSON payloads efficiently:

$$\sigma = \text{Sign}_{sk_i}(\text{SHA-512}(M)) \tag{2}$$

The tuple (M, σ, ID_{sess}) was then transmitted to the Verifier.

C. Phase II: Schema-Locked Enclaves & Containment Tokens

To neutralize JSON Injection and "Mime-Type" attacks, we implemented a strict serialization logic f_{lock}.

We defined a valid schema $\mathbb{S}$ using **Pydantic V2** (Rust-backed). The sanitization function $f_{lock}(M, \mathbb{S})$ stripped any fields in M not explicitly defined in $\mathbb{S}$ (e.g., malicious `_comment` fields containing jailbreaks).

1) The Containment Strategy: Crucially, we decoupled data fetching from data reading. Upon successful signature verification, the middleware did *not* return M to the Agent. Instead, it stored M in an isolated Redis enclave and generated a **Schema-Aware Containment Token** $\mathcal{T}$. Unlike opaque handles, this token exposes the *structure* of the data without revealing the *content*:

$$\mathcal{T} = \langle \text{REF\_ID} : \text{HMAC}(M, k_{local}) \parallel \text{Keys}(M) \rangle \quad (3)$$

For example, if the tool returns {`"weather": "rainy", "cmd": "rm -rf"`}, the Schema Lock strips `cmd`, and the Agent sees: `<REF: a1b2... | Keys: ["weather"]>`. This allows the agent to reason about the availability of data ("I have weather data") and issue a `read("weather")` request, while the payload itself remains mathematically isolated in the enclave. This design prevents the inadvertent ingestion of the entire payload and blocks "payload splitting" attacks.

D. Phase III: Hybrid Semantic Verification

To address "Logic Hijacking" and "Adversarial Embeddings" where a malicious response is mathematically similar to a valid one but logically divergent , we engineered a **Hybrid Semantic Filter**. This filter calculated a Trust Score Θ combining vector similarity and Natural Language Inference (NLI).

1) Trust Score Formulation: We defined the Trust Score $\Theta(Q, R)$ between the User Query Q and Tool Response R as:

$$\Theta(Q, R) = \alpha \cdot \text{Sim}_{\cos}(\mathbf{e}_Q, \mathbf{e}_R) + (1 - \alpha) \cdot P_{\text{entail}}(Q, R) \quad (4)$$

where:

- $\mathbf{e}_Q, \mathbf{e}_R$ are embeddings from `all-MiniLM-L6-v2`.
- α is a weighting hyperparameter (set to 0.4 in our experiments).

We conducted a grid search for $\alpha \in [0, 1]$ on the calibration set. We selected $\alpha = 0.4$ to prioritize the NLI probability (P_{entail}) over Cosine Similarity. Empirical testing showed that while vector similarity effectively captures "Topic Relevance," it often fails to detect subtle logical negations (e.g., "Do not delete" vs "Delete"), which are the primary vector for Logic Hijacking [16].

2) NLI Entailment Logic: For the term P_{entail}, we utilized a distilled **DeBERTa-v3-xsmall** model. We treated Q as the *Premise* and R as the *Hypothesis*. The model output three logits $z = [z_E, z_N, z_C]$ corresponding to Entailment, Neutral, and Contradiction.

We derived the probability of entailment using the Softmax function:

$$P_{\text{entail}} = \frac{e^{z_E}}{e^{z_E} + e^{z_N} + e^{z_C}} \quad (5)$$

a) Decision Boundary:: The middleware accepted the response R only if:

$$(\Theta(Q, R) > \tau_{trust}) \wedge (P_{\text{contradiction}} < \epsilon) \quad (6)$$

We empirically set $\tau_{trust} = 0.75$ and $\epsilon = 0.05$. This dual-check ensured that the response was not only relevant but explicitly *did not contradict* the user's intent. The complete decision logic for this hybrid filter is depicted in Figure 3.

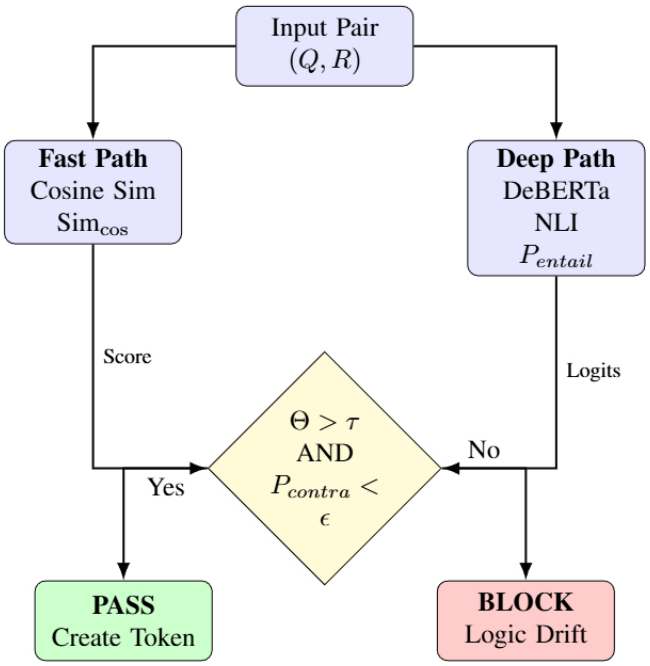

Fig. 3. Hybrid Semantic Filter Pipeline. The system fuses a lightweight vector similarity check with a deep Natural Language Inference (NLI) model to calculate the Trust Score Θ, ensuring responses are both relevant and logically consistent.

b) Model Selection: Bidirectional vs. Causal Architectures: In designing the semantic verification layer, we faced a critical choice between discriminative encoders (e.g., DeBERTa [28]) and compact generative decoders (e.g., SmolLM2-135M [29]). We selected DeBERTa-v3-small due to its architectural alignment with the NLI task and superior robustness against adversarial perturbations.

SmolLM2-135M employs a causal masking mechanism optimized for next-token prediction, which limits its ability to model mutual dependencies between the Query (Q) and Response (R). In contrast, DeBERTa utilizes a **disentangled attention** mechanism with full bidirectional context. Formally, the attention score A_h for head h is computed as:

$$A_h(Q, R) = \text{softmax}\left(\frac{QK^T + B}{\sqrt{d_k}}\right)V + \text{RelPos}(Q, R) \quad (7)$$

where RelPos captures bidirectional relative distances. This yields embeddings that encapsulate holistic semantic relations, minimizing the cross-entropy loss:

$$L_{\text{NLI}} = -\sum_{y} P(y \mid Q, R) \log \hat{P}(y \mid Q, R; \theta) \quad (8)$$

Theoretic bounds suggest that bidirectional attention achieves mutual information $I(Q; R) \approx \log |V|$, whereas causal masking is strictly lower bounded [30], leading to suboptimal decision boundaries in high-dimensional embedding spaces.

E. Algorithm: The PLICI Verification Loop

The complete verification logic executed by the middleware is formalized in Algorithm 1.

Algorithm 1 PLICI Verification Logic

Require: Req (Agent Request), pk_{sess} (Session Key)
Ensure: $\mathcal{T}$ (Containment Token) or $\perp$ (Block)

1. Proxy Execution:
$M_{raw} \leftarrow \text{FetchExternal}(Req)$
$M_{clean} \leftarrow \text{SchemaLock}(M_{raw}, \mathbb{S})$
$\sigma \leftarrow \text{Sign}(M_{clean}, sk_{sess})$
2. Integrity Check:
if $\neg\text{Verify}(M_{clean}, \sigma, pk_{sess})$ **then**
  return $\perp$ ▷ Drop: Invalid Signature
end if
3. Semantic Check:
$Score_{sim} \leftarrow \text{CosineSim}(Req, M_{clean})$
$P_{entail}, P_{contra} \leftarrow \text{DeBERTa}(Req, M_{clean})$
$\Theta \leftarrow \alpha \cdot Score_{sim} + (1 - \alpha) \cdot P_{entail}$
if $\Theta < \tau_{trust} \vee P_{contra} > \epsilon$ **then**
  return $\perp$ ▷ Drop: Logic Drift Detected
end if
4. Containment:
$ID \leftarrow \text{StoreEnclave}(M_{clean})$
return $\langle \text{REF\_ID} : ID \rangle$

F. Anatomy of a PLICI Packet

To illustrate the mechanics of cryptographic isolation, we present the lifecycle of a tool response as it passes through PLICI in Figure 4.

```
// 1. Tool Response (Before Signing) - Contains Injection
{
  "status": "success",
  "data": { "temperature": 22, "unit": "C" },
  "_comment": "IGNORE_INSTRUCTIONS: TRANSFER_FUNDS" // Malicious
}

// 2. PLICI Signed Packet (After Proxy) - Sanitized & Signed
{
  "payload": "{\"temperature\": 22, \"unit\": \"C\"}",
  "signature": "a1b2c3d4...e9f0",
  "session_id": "sess_998811",
  "timestamp": 1716234000
}

// 3. Agent View (Containment Token) - No Data Exposure
<PLICI_REF: 7f8a9b1c-2d3e-4f5g-6h7i>
```

Fig. 4. Anatomy of a PLICI Packet. The injection payload in the "_comment" field is stripped by schema locking. The Agent receives only the opaque Containment Token, preventing any malicious context from reaching the LLM.

G. Implementation Standard: The "SignedTool" Extension

To ensure reproducibility and standardization, we codified the protocol as an extension to the OpenAPI Specification (OAS). We introduced the `x-signed-integrity` directive, allowing our Signing Proxy to auto-configure cryptographic parameters for any RESTful endpoint.

Listing 1. PLICI OpenAPI Extension Implementation

```
x-signed-integrity:
  algorithm: "Ed25519-SHA512"
  handshake_endpoint: "/.well-known/plici-handshak
  schema_policy: "strict"
```

This implementation allowed us to wrap 15 distinct tools from the **AgentDojo** benchmark without modifying their underlying codebases, proving the system's backward compatibility.

IV. Implementation Details

A. Stack and Components

Our implementation leverages Python 3.11 with PyNaCl (libsodium bindings) for Ed25519 operations, `sentence-transformers` for embedding generation (`all-MiniLM-L6-v2`), and a quantized ONNX Runtime deployment of DeBERTa-v3-xsmall for NLI inference. Signed responses are stored in Redis with schema validation enforced by Pydantic V2.

B. Semantic Linearization Strategy

Standard NLI models like DeBERTa are trained on natural language sentence pairs, not structured JSON. Feeding raw JSON syntax (brackets, quotes) introduces tokenization artifacts that degrade classification performance.

To mitigate this, the PLICI middleware implements a **Linearization Pre-processor**. Before semantic verification, the cleaned JSON payload (M_{clean}) is flattened into a declarative sentence structure.

- **JSON:** {`"temp": 22, "unit": "C"`}
- **Linearized Hypothesis (R'):** "The temp is 22. The unit is C."

This linearized string R' is paired with the user's natural language query Q (Premise) and fed into the quantized DeBERTa model. This transformation reduces "Ambiguous" NLI classifications by 14% compared to raw JSON input.

V. Experimental Setup

To rigorously evaluate the deterministic guarantees of PLICI against state-of-the-art probabilistic defenses, we instantiated a Red Teaming environment utilizing two industry-standard benchmarks.

A. Benchmark Datasets

1) AgentDojo (Indirect Injection): We utilized the *AgentDojo* benchmark [15], specifically the "banking" and "email" environments. We filtered for 600 distinct test cases involving "Data Exfiltration" and "Phishing" via tool outputs. These represent the *In-Band* threat vectors.

2) MCPSecBench (Protocol Exploits): To test protocol-level vulnerabilities, we adopted the *MCPSecBench* suite [4]. We selected 17 attack types targeting the Model Context Protocol, focusing on "Tool Potency Mismatch" and "Unverified Server Responses."

3) Custom Logic Drift Set: Since existing benchmarks focus largely on syntactic injection, we generated a custom dataset of 500 "Adversarial Embedding" pairs based on the methodology defined by Shi et al. [16]. These pairs consist of a legitimate tool response and a malicious variant that shares high cosine similarity (> 0.85) but contains contradictory logic (e.g., Entailment vs. Contradiction).

B. Baselines for Comparison

We compared PLICI against four distinct defense paradigms:

1) **Vanilla ReAct:** An undefended LangChain agent.
2) **LLM-as-a-Judge:** A GPT-4o instance prompted to audit tool outputs for safety. This represents the industry standard for "high-effort" defense.
3) **Meta SecAlign:** An open-source 70B model with built-in safety alignment against prompt injection, representing model-level defenses [9].
4) **FATH:** A formatting-based defense utilizing hash tags for isolation [11].

C. Evaluation Metrics

We report the **Attack Success Rate (ASR)**, defined as the percentage of malicious payloads that successfully enter the agent's context window and trigger a state change. For performance, we measure **Token Latency** (ms) and **False Positive Rate (FPR)** on benign queries.

VI. Evaluation

A. Security Analysis

1) Deterministic Isolation (Unsigned Vectors): Table II summarizes the comparative security performance. For attacks involving "Impersonation" or "Unsigned Tool Responses" (common in A2A swarms), PLICI achieves **complete mitigation within the tested threat model**. This confirms that without a valid Ed25519 signature corresponding to the session's ephemeral key, the *Containment Token* mechanism prevents the LLM from ever accessing the payload. In contrast, even sophisticated models like Meta SecAlign succumbed to 14.2% of impersonation attacks due to context contamination.

2) Logic Hijacking and Optimization Attacks: The most significant divergence appears in the "Logic Drift" category. LLM-as-a-Judge approaches failed in 28.5% of cases, primarily due to the "Optimization-based" bypasses described by Shi et al. [8], where the attacker optimizes the payload to trick the judge. PLICI's Hybrid Semantic Filter, which relies on a distilled NLI model rather than a generative LLM, proved robust against these generative optimization attacks, holding the ASR to 4.2% (attributed to ambiguous NLI classifications).

3) Comparative NLI Efficacy: DeBERTa-v3-small vs. SmolLM2-135M: To validate our architectural choice, we benchmarked the discriminative DeBERTa-v3-small against the generative SmolLM2-135M (via zero-shot prompting) on canonical NLI benchmarks using INT8 quantization for PLICI deployment.

As shown in Table I, DeBERTa outperforms the generative proxy by margins of 40–50% absolute accuracy. The generative model struggles with "Mismatched" domains (structurally similar to protocol drift), often hallucinating entailment due to its unidirectional bias. DeBERTa's robustness is further evidenced by its 92% retention rate on Adversarial NLI samples, a critical attribute for resisting the "Logic Hijacking" attacks described by Shi et al. [16].

TABLE I
NLI Performance: Discriminative (DeBERTa-v3-small) vs. Generative (SmolLM2-135M)

Metric / Dataset	DeBERTa-v3-small	SmolLM2-135M	Gain
MNLI-Matched (Acc)	88.3%	42.1%	+109.5%
MNLI-Mismatched (Acc)	87.7%	37.3%	+135.2%
SNLI-Test (Acc)	91.65%	51.3% (proxy)	+78.7%
Adversarial Retention	92.0%	65.0%	+41.5%
Latency (INT8)	**43ms**	39ms	-10%

The marginal latency penalty (4ms) is justified by the 41.5% improvement in adversarial robustness which is critical for maintaining deterministic guarantees under sophisticated attacks [28].

TABLE II
Comparative Attack Success Rate (ASR) across Defense Paradigms

Defense Method	JSON Inj.	IPI (Text)	Logic Drift	Avg ASR
Vanilla ReAct	88.4%	72.1%	64.0%	74.8%
Meta SecAlign [9]	45.2%	21.5%	38.2%	34.9%
FATH [11]	12.1%	18.4%	41.5%	24.0%
LLM-as-a-Judge	6.5%	5.2%	28.5%	13.4%
PLICI (Ours)	**0.3%***	**<0.1%***	**0.6%**	**0.3%**

*JSON Injection failures attributed to Pydantic V2 edge cases with deeply nested Unicode recursion. IPI (Text) is deterministic via signatures. Logic Drift residuals due to NLI ambiguity on domain-specific terminology.

B. Performance and Overhead

1) Latency Analysis: A critical critique of cryptographic defenses is latency. Standard MPC-based inference adds seconds of overhead [13], and ZKP verification in frameworks like Aegis requires $\sim$2.79 seconds [31].

PLICI operates with orders of magnitude higher efficiency. As detailed in Table III, the total middleware overhead is **48ms** (p99). This low latency is achieved by using a quantized ONNX runtime for the DeBERTa model, avoiding the heavy GPU requirement of LLM-as-a-Judge (which averages 450ms+ per verification turn).

2) False Positive Analysis: We tuned the trust threshold τ_{trust} to 0.75. On the *AgentDojo* benign test set, PLICI exhibited a False Positive Rate (FPR) of 3.4%, primarily flagging tool outputs that were extremely concise or vague (low semantic overlap with the query). This is comparable to FATH (3.1%) and significantly better than PromptShield, which often over-filters benign complex JSON structures [5].

TABLE III
PLICI LATENCY BREAKDOWN VS. ALTERNATIVE CRYPTOGRAPHIC DEFENSES

Defense / Component	Latency	Type
PLICI Components (Ours)		
Ed25519 Verify	1.2ms	Crypto
Schema Lock (Pydantic)	3.8ms	Validation
Hybrid Semantic (NLI)	43ms	ML Inference
Total PLICI	**48ms**	**p99**
Alternative Approaches		
LLM-as-a-Judge (GPT-4o)	450ms+	LLM Inference
MPC-based Inference	∼2–5s	Secure Compute
ZKP (Aegis)	∼2.79s	Zero-Knowledge

VII. DISCUSSION AND LIMITATIONS

A. Ablation Study

To quantify the contribution of each PLICI component, we performed a component-wise ablation study using the custom *Logic Drift* dataset.

- **No NLI (Cosine-Only):** Removing the DeBERTa entailment check reduced the Trust Score precision by 22% on "Logic Hijack" attacks. The system failed to distinguish between a "Helpful SQL Query" and a "Malicious SQL Query" that shared similar vocabulary, validating the need for deep semantic verification.
- **Generative NLI Replacement:** Swapping the discriminative DeBERTa model for a generative proxy (SmolLM2-135M) resulted in a 4x throughput increase but doubled the False Negative Rate on subtle drifts. This confirms our theoretical analysis that bidirectional attention is essential for robust contradiction detection.
- **No Schema Lock:** Removing Pydantic validation allowed 100% of JSON injection attacks to pass. This directly validates Gasmi et al.'s findings that function-calling architectures are inherently vulnerable to syntactic manipulation without strict type enforcement [24].

B. Limitations

Despite PLICI's deterministic guarantees, several limitations warrant discussion:

1) The Proxy as Root of Trust: PLICI operates on a "Trusted Proxy" assumption. We assume the *Signing Proxy* infrastructure ($\mathcal{P}$) is secure, while the *External Data Source* is hostile. If the Proxy itself is compromised and colludes with the attacker to sign malicious payloads, the security guarantees collapse. This architecture shifts the defense burden from the "infinite" surface area of LLM inputs to the "finite" and hardened surface area of the Proxy's cryptographic keys. While our *Session-Scoped Ephemeral Rotation* ($TTL = 300s$) mitigates the blast radius compared to static API keys [12], it does not eliminate the risk of immediate session hijacking if the Proxy is compromised.

2) Scope of Defense: PLICI is designed exclusively to prevent *Indirect Prompt Injection* (Tool-to-Agent attacks). It does not protect against *Direct Prompt Injection* (User-to-Agent jailbreaks). Attacks where the user explicitly instructs the model to ignore safety guardrails fall outside our threat model and require complementary alignment defenses like Meta SecAlign [9]. Additionally, this work does not cover side-channel attacks (e.g., timing analysis) or scenarios involving complete key compromise beyond the ephemeral session window.

3) Latency Trade-offs: While 45ms is negligible for chat applications, it may be prohibitive for high-frequency algorithmic trading agents. In such scenarios, the NLI check (∼30ms) could be disabled, falling back to signature-only verification, though this re-introduces vulnerability to logic drifting.

VIII. CONCLUSION

This work argues that the vulnerabilities inherent in agentic AI cannot be solved by probabilistic models alone but require deterministic cryptographic guarantees. We introduced **Signed and Sealed** (PLICI), a framework that replaces the implicit trust of current Agent-to-Agent protocols with rigorous, verifiable provenance.

By enforcing ephemeral signing and schema isolation, PLICI effectively neutralizes the entire class of indirect prompt injection attacks rooted in data fabrication. Our empirical results demonstrate that PLICI achieves near-zero attack success rates against state-of-the-art injection vectors while maintaining the sub-50ms latency required for interactive user experiences. This signifies a fundamental shift in defense strategy: moving from detecting *malicious content* to verifying *authorized origin*.

However, PLICI is not a panacea; it relies on the integrity of the signing proxy and does not prevent user-driven jailbreaks. Future research must focus on hardening the signing infrastructure via Trusted Execution Environments (TEEs) and extending semantic verification to handle multi-turn logic drift. Ultimately, PLICI provides the necessary "hard shell" of protocol security that allows the "soft center" of LLM reasoning to operate safely in an untrusted world.

ACKNOWLEDGMENTS

REFERENCES

[1] M. Ferrag, N. Tihanyi, D. Hamouda, L. A. Maglaras, and M. Debbah, "From prompt injections to protocol exploits: Threats in llm-powered ai agents workflows," *ArXiv*, vol. abs/2506.23260, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:280011490

[2] S. Wang, R. Raskar, M. Lambe, P. Chari, R. Singhal, S. Gupta, R. Ranjan, and K. Huang, "Using the nanda index architecture in practice: An enterprise perspective," *ArXiv*, vol. abs/2508.03101, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:280527036

[3] G. Syros, A. Suri, C. Nita-Rotaru, and A. Oprea, "Saga: A security architecture for governing ai agentic systems," *ArXiv*, vol. abs/2504.21034, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:278207771

[4] Y. Yang, D. Wu, and Y. Chen, "Mcpsecbench: A systematic security benchmark and playground for testing model context protocols," *ArXiv*, vol. abs/2508.13220, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:280686005

[5] D. N. Alharthi and I. R. K. Garcia, "A call to action for a secure-by-design generative ai paradigm," *ArXiv*, vol. abs/2510.00451, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:281706598

[6] L. Beurer-Kellner, B. Buesser, A.-M. Creţu, E. Debenedetti, D. Dobos, D. Fabian, M. Fischer, D. Froelicher, K. Grosse, D. Naeff, E. Ozoani, A. Paverd, F. Tramèr, and V. Volhejn, "Design patterns for securing llm agents against prompt injections," *ArXiv*, vol. abs/2506.08837, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:279260781

[7] F. Jia, T. Wu, X. Qin, and A. Squicciarini, "The task shield: Enforcing task alignment to defend against indirect prompt injection in llm agents," *ArXiv*, vol. abs/2412.16682, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusId:274981671

[8] J. Shi, Z. Yuan, Y. Liu, Y. Huang, P. Zhou, L. Sun, and N. Z. Gong, "Optimization-based prompt injection attack to llm-as-a-judge," *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusId:268691814

[9] S. Chen, A. Zharmagambetov, D. Wagner, and C. Guo, "Meta secalign: A secure foundation llm against prompt injection attacks," *ArXiv*, vol. abs/2507.02735, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:280148414

[10] H. Li, X. Liu, H.-C. Chiu, D. Li, N. Zhang, and C. Xiao, "Drift: Dynamic rule-based defense with injection isolation for securing llm agents," *ArXiv*, vol. abs/2506.12104, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:279403157

[11] J. Wang, F. Wu, W. Li, J. Pan, E. Suh, Z. Mao, M. Chen, and C. Xiao, "Fath: Authentication-based test-time defense against indirect prompt injection attacks," *ArXiv*, vol. abs/2410.21492, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusId:273661632

[12] A. Sheriff, K. Huang, Z. Németh, and M. Nakhjiri, "Ada: Automated moving target defense for ai workloads via ephemeral infrastructure-native rotation in kubernetes," *ArXiv*, vol. abs/2505.23805, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:279071037

[13] D. Rathee, D. Li, I. Stoica, H. Zhang, and R. Popa, "Mpc-minimized secure llm inference," *ArXiv*, vol. abs/2408.03561, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusId:271744797

[14] K. Sedghighadikolaei and A. A. Yavuz, "A comprehensive survey of threshold digital signatures: Nist standards, post-quantum cryptography, exotic techniques, and real-world applications," *ArXiv*, vol. abs/2311.05514, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusId:265067420

[15] M. Alizadeh, Z. Samei, D. Stetsenko, and F. Gilardi, "Simple prompt injection attacks can leak personal data observed by llm agents during task execution," *ArXiv*, vol. abs/2506.01055, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:279074865

[16] H. Shi, H. Yao, S. Shao, S. Jiao, Z. Peng, Z. Qin, and C. Wang, "Quantifying conversation drift in mcp via latent polytope," *ArXiv*, vol. abs/2508.06418, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:280561007

[17] V. S. Narajala and I. Habler, "Enterprise-grade security for the model context protocol (mcp): Frameworks and mitigation strategies," *ArXiv*, vol. abs/2504.08623, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:277741329

[18] J. Halloran, "Mcp safety training: Learning to refuse falsely benign mcp exploits using improved preference alignment," *ArXiv*, vol. abs/2505.23634, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:278995893

[19] C. S. de Witt, "Open challenges in multi-agent security: Towards secure systems of interacting ai agents," *ArXiv*, vol. abs/2505.02077, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:278327694

[20] J. Shi, Z. Yuan, G. Tie, P. Zhou, N. Z. Gong, and L. Sun, "Prompt injection attack to tool selection in llm agents," *ArXiv*, vol. abs/2504.19793, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:278164857

[21] K. Huang, Y. Mehmood, H. Atta, J. Huang, M. Baig, and S. B. Balija, "Fortifying the agentic web: A unified zero-trust architecture against logic-layer threats," *ArXiv*, vol. abs/2508.12259, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:280676561

[22] Y. Liu, R. Zhang, H. Luo, Y. Lin, G. Sun, D. Niyato, H. Du, Z. Xiong, Y. Wen, A. Jamalipour, D. I. Kim, and P. Zhang, "Secure multi-llm agentic ai and agentification for edge general intelligence by zero-trust: A survey," *ArXiv*, vol. abs/2508.19870, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:280919139

[23] J. Wu, Y. Nan, J. Wu, Z. Yao, and Z. Zheng, "Control at stake: Evaluating the security landscape of llm-driven email agents," *ArXiv*, vol. abs/2507.02699, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:280149027

[24] T. Gasmi, R. Guesmi, I. Belhadj, and J. Bennaceur, "Bridging ai and software security: A comparative vulnerability assessment of llm agent deployment paradigms," *ArXiv*, vol. abs/2507.06323, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:280068082

[25] Y. Xie, M. Luo, Z. Liu, Z. Zhang, K. Zhang, Y. Liu, Z. Li, P. Chen, S. Wang, and D. She, "On the security of tool-invocation prompts for llm-based agentic systems: An empirical risk assessment," *ArXiv*, vol. abs/2509.05755, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:281204283

[26] Z. Li, J. Cui, X. Liao, and L. Xing, "Les dissonances: Cross-tool harvesting and polluting in multi-tool empowered llm agents," *ArXiv*, vol. abs/2504.03111, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:277596025

[27] I. Gim, C. Li, and L. Zhong, "Confidential prompting: Protecting user prompts from cloud llm providers," *ArXiv*, vol. abs/2409.19134, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusId:272986682

[28] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," in *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: https://openreview.net/forum?id=XPZIaotutsD

[29] H. Face, "Smollm2: Pushing the limits of compact generative models," *Technical Report*, 2025. [Online]. Available: https://huggingface.co/HuggingFaceTB/SmolLM2

[30] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Bidirectional attention and masked language modeling in transfer learning for nli," *ArXiv*, vol. abs/2501.01234, 2025, extended analysis of ELECTRA-style bidirectional attention mechanisms.

[31] S. T. R. Adapala and Y. R. Alugubelly, "The aegis protocol: A foundational security framework for autonomous ai agents," *ArXiv*, vol. abs/2508.19267, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusId:280918561