SecureClaw: Clawing Back Control of LLM Agents

Yuhan Ma[1]* **Stefan Schmid**[1]*
[1]TU Berlin
yuhan.ma@campus.tu-berlin.de stefan.schmid@tu-berlin.de

Abstract

Tool-using large language model (LLM) agents face two distinct security failures: unauthorized external actions and exposure of sensitive plaintext inside the runtime before any final output check can intervene. Existing defenses usually protect one boundary, either the planner/runtime or the point where external actions are executed, and therefore do not by themselves secure both surfaces. We present SecureClaw, a dual-boundary architecture that enforces authorization in a trusted executor and keeps raw protected data outside the LLM runtime. Sensitive reads pass through a trusted gateway that replaces raw values with reference IDs. In the evaluated deployment, the gateway also provides short planning summaries that deliberately release selected information, a process called declassification. Writes that change external state follow a PREVIEW→COMMIT protocol in which only a trusted executor may execute the exact request authorized by policy. The runtime can still plan over summaries and reference IDs, but cannot directly resolve the IDs into secrets or perform side effects. Across the AgentDojo, AgentLeak, and Agent Security Bench (ASB) benchmarks, SecureClaw is the only defense we evaluate in a common harness that simultaneously retains usable task utility and achieves 0% attack success rate (ASR) on ASB, 0.64% ASR on AgentDojo, and 3.23% overall leak on AgentLeak's attacked parity lane, which measures final-output and internal-relay leakage. Code is available at `https://github.com/ponyyuhan/SecureClaw`.

1 Introduction

Tool-using language-model agents are moving from chat-style assistance to workflow automation [Lin et al., 2026]. A single agent may read email threads, invoices, customer-support tickets, calendars, code repositories, or medical intake forms, and then act on the outside world by sending messages, sharing files, updating records, or scheduling meetings.

In these settings, a failure is not merely a poor answer: it can become an irreversible external effect or an exposure of sensitive data during execution.

This threat model is well established [Yu et al., 2025]. Indirect prompt injection showed that attacker-controlled content can steer LLM-integrated systems by collapsing the boundary between data and instructions [Greshake et al., 2023]. AgentDojo and Agent Security Bench (ASB) show that tool-using agents remain vulnerable to prompt injection that redirects actions toward attacker-chosen outcomes [Debenedetti et al., 2024, Zhang et al., 2025]. More recent work broadens the attack surface to tool selection, web agents, and agent-mediated financial workflows [Shi et al., 2025a, Evtimov et al., 2025]. In parallel, AgentLeak shows that multi-agent systems introduce internal

*Corresponding authors.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

leakage paths that output-only auditing does not capture, especially inter-agent messages and shared memory [Yagoubi et al., 2026].

These results point to two critical security questions. The first is effect authorization: whether the executed action matches the exact request authorized for the current caller, session, inputs, and context. The second is protection of raw data: whether the untrusted runtime ever receives sensitive data in usable form. These questions are related but not interchangeable. A boundary around effectful tools can stop an unauthorized commit, yet it does not help if the runtime has already read a secret and can relay it through internal channels. Conversely, runtime-local information-flow defenses can reduce some leakage, but they still leave substantial trust in planner state or runtime enforcement. Architectural and execution-time systems move control closer to the action or resource boundary [Zhong et al., 2025, Costa et al., 2025, Debenedetti et al., 2025, Li et al., 2025a, Shi et al., 2025b, Palumbo et al., 2026, Sharma and Grossman, 2026].

SecureClaw keeps control of external actions and raw protected values outside the untrusted agent runtime. When a tool returns protected data, a trusted gateway stores the raw value and gives the runtime a reference ID plus a short planning summary. When the runtime proposes an external action, the policy engine checks the complete request. A trusted executor independently verifies the approval before retrieving the referenced data and calling the external API.

Running example. Consider an agent that reads an invoice and sends it to an approver. Figure 1 follows four steps. **(1) Read.** The gateway stores the raw invoice and returns a reference ID, written as `INV_42` for readability, plus a summary such as "vendor A, amount $4.2k, due Friday." **(2) Plan.** The runtime reasons over this summary and carries the ID through memory and later calls; it cannot use the ID to retrieve the raw invoice. **(3) Preview.** The runtime proposes sending `INV_42` to `approver@corp.com`. The gateway adds the authenticated caller and session and serializes the recipient, channel, referenced object and version, and relevant context in a fixed form. It then submits the complete request to the policy engine, which binds its approval to exactly that request. **(4) Commit.** The executor, the only component that can call the email API, checks that the approval matches the request, is fresh, and has not been reused; only then does it resolve the reference and send. Changing the recipient after approval causes rejection; a denial can return a safe hint such as "save a draft and request approval."

In this architecture, the runtime still plans, searches, and composes tool calls, but it does so with reference IDs rather than unrestricted raw data, and it cannot directly turn a proposal into a committed external action. The reference-ID mechanism follows a capability-style design in which authority is carried by an unforgeable reference rather than by exposure to raw data [Saltzer and Schroeder, 1975, Watson et al., 2010]. When the runtime needs task-relevant information, the summary interface releases only a short, policy-bounded summary $D(v)$ of a sensitive value v. This is an explicit declassification channel: an intentional, auditable release of selected information from a protected value [Sabelfeld and Sands, 2005].

The two risks arise at different points: an external effect occurs when the API executes the action, whereas sensitive data is exposed as soon as a raw protected value enters an adversarial runtime. This predicts a concrete empirical pattern: executor-side enforcement should stop unauthorized commits without closing internal relay channels, while keeping raw data outside the runtime should close relay channels without authorizing commits. The ablation and bypass suite test whether either control can replace the other.

This paper makes four contributions.

- **A security decomposition for agent systems.** We identify two non-substitutable security requirements for tool-using agents: request-bound authorization of external effects and confinement of sensitive plaintext away from the untrusted runtime.
- **A practical dual-boundary architecture.** We present SecureClaw, which mediates sensitive reads with reference IDs, effectful writes with a PREVIEW→COMMIT executor, and blocked actions with deny-aware recovery: a fixed-template safe-continuation path after a denied commit. In the evaluated deployment, short planning summaries form an explicit declassification interface.
- **A common-harness evaluation that separates the two surfaces.** Across AgentDojo, AgentLeak, and ASB, SecureClaw is the only evaluated common-harness baseline that

simultaneously reaches 0% ASR on ASB, 0.64% ASR on AgentDojo, and 3.23% overall leak on AgentLeak's attacked parity lane; the ablation and bypass suite then isolate why executor-side authorization and read-side confinement are complementary rather than interchangeable.

- **A reusable implementation artifact.** We provide the SecureClaw runtime implementation and benchmark evaluation code at `https://github.com/ponyyuhan/SecureClaw`, with an archived version at `https://doi.org/10.5281/zenodo.23258171` (Appendix A.2).

2 Problem setting and scope

We consider a tool-using agent deployment with an LLM-driven **runtime** that plans and invokes tools; a trusted **gateway** on the read path; a trusted **data store** for protected values; a **policy engine** that authorizes previewed requests; a trusted **executor** on the write path; and a **user** who may confirm high-risk actions. The policy engine may be a single service, with an optional distributed realization for deployments that also want transcript privacy against one evaluator (Appendix B.5). An action is **effectful** if it can change external state or send information to an external party, for example by sending an email, sharing a file, updating a record, or initiating a payment. A value is **sensitive** if exposing it to the runtime in plaintext would create a meaningful disclosure risk.

SecureClaw targets two security surfaces. The **effect surface** asks whether the executed action matches the exact request authorized for the current caller, session, inputs, and context. The **plaintext surface** asks whether the untrusted runtime ever receives raw secrets that it can relay through user-visible outputs, inter-agent messages, memory, logs, or later tool calls. AgentLeak's channel taxonomy is useful here: C1 denotes the final response, C2 inter-agent messages, C3/C4 tool I/O, C5 shared or persistent memory, C6 logs, and C7 persistent artifacts [Yagoubi et al., 2026]. For SecureClaw, C2 and C5 are especially important because they expose relay paths that remain invisible to output-only defenses [Kong et al., 2025].

We evaluate two adversaries. In the **prompt-injection** setting, attacker-controlled instructions appear inside retrieved content and attempt to redirect downstream actions or induce disclosure, as in AgentDojo, AgentLeak, and ASB [Debenedetti et al., 2024, Yagoubi et al., 2026, Zhang et al., 2025]. In the **compromised-runtime** setting, the runtime is fully adversarial: it may fabricate requests, tamper with recipients or channels, replay prior authorizations, mix sessions, guess reference IDs, or exploit implementation mistakes. The core boundary guarantees trust the gateway, the trusted data store, the policy engine, and the executor. Appendix B.5 states the extra assumptions for the optional distributed policy variant. The design further assumes complete mediation of APIs that execute external actions and sensitive reads, correct protected-field classification, bounded clock drift, and crash-safe replay protection [Saltzer and Schroeder, 1975].

Our goals are request-bound effect authorization, confinement of raw sensitive plaintext outside declared declassification interfaces, and explicit bounded declassification through the summary interface. SecureClaw does not claim that every policy-allowed action is semantically correct for the user's latent intent, nor does it survive compromise of the gateway or executor. By **residual failures**, we mean failures that remain after the two target boundaries behave as specified, for example a policy-allowed action with the wrong principal or object because the policy is too coarse. We analyze those residuals explicitly in Section 4.4.

Table 1 summarizes the two security surfaces, why they require different boundary placement, and how each one is validated empirically.

3 SecureClaw design

SecureClaw mediates effectful writes through a trusted executor and sensitive reads through a trusted gateway. The runtime remains useful because it continues to plan, search, route reference IDs, and compose tool calls, but it is no longer entrusted with plaintext or with committing external effects.

Table 1: Two security surfaces, their control points, and their validation.

Surface	If left inside the runtime	SecureClaw control point	Main evidence
Effect surface	A compromised or injected runtime can mutate recipients, channels, or context and still reach the API that executes the action.	Executor-side PREVIEW→COMMIT, request binding, freshness checks, and replay protection.	AgentDojo, ASB, bypass suite
Plaintext surface	A runtime that reads plaintext can relay it through internal channels such as inter-agent messages (C2) or memory (C5) even when a later commit is blocked.	Gateway-side reference IDs, resolution by trusted components for authorized actions, and explicit short planning summaries, so that protected values do not re-enter the runtime outside declared declassification interfaces.	AgentLeak, ablation

3.1 Design overview and component roles

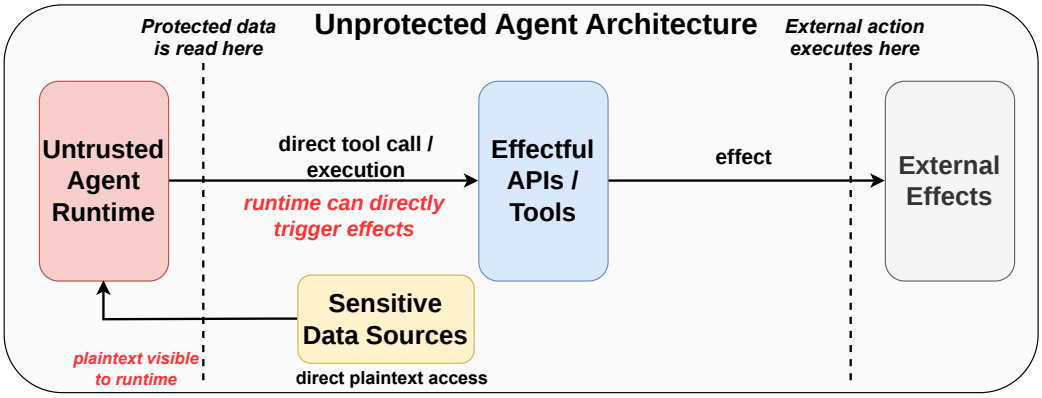
SecureClaw separates planning from security enforcement. The **runtime** reasons, searches, and orchestrates tool calls, but it is untrusted for both authorization and confidentiality. The **gateway** mediates sensitive reads, classifies returned data, creates reference IDs, and canonicalizes effectful preview requests by serializing action-relevant fields in a fixed form. The **trusted data store** retains protected plaintext and reference-ID metadata outside runtime memory. The **policy engine** checks the complete request against configured rules and returns an authorization artifact: a verifiable allow/deny decision bound to that request. The **executor** is the only component that can call APIs that perform external actions. Before execution, it recomputes the request binding and verifies the authorization artifact, checks that the approval is fresh and has not been reused, validates any required confirmation token, and enforces restrictions on resolving reference IDs.

An execution therefore has two mediated phases. On the read path, the gateway returns a reference ID and, when planning requires it, a short planning summary produced by a deterministic, schema-aware summary operator. On the write path, every effectful request remains a proposal until executor-side verification succeeds.

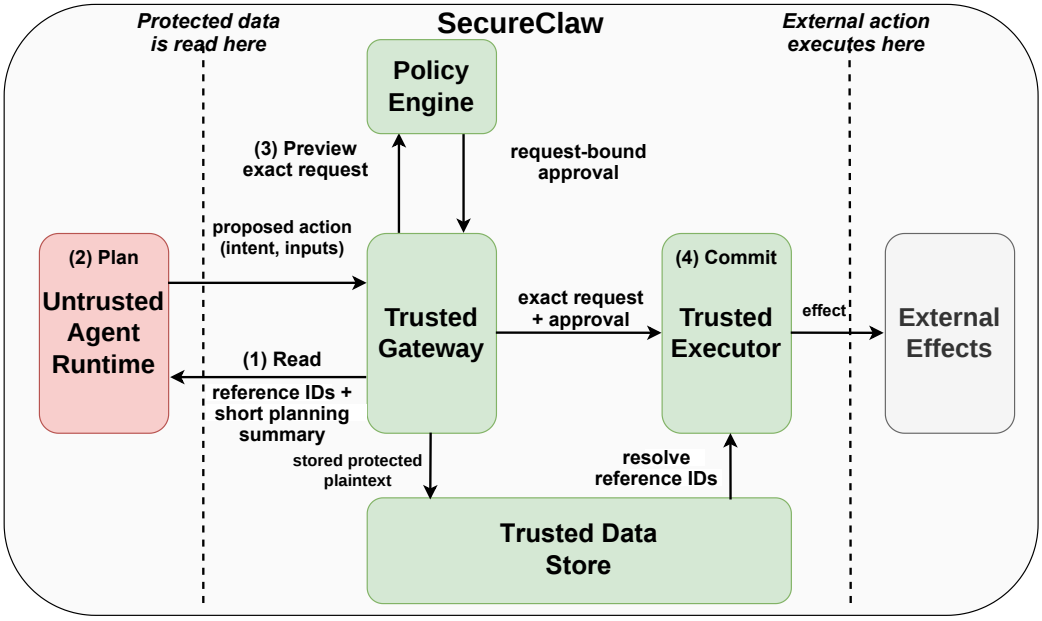
3.2 Read path: reference IDs and trusted storage

The read path protects the plaintext surface. When a tool returns sensitive content, the gateway stores the plaintext in the trusted data store and returns a **reference ID** to the runtime (an *opaque handle*, as in FlowFence [Fernandes et al., 2016]). The ID is an unpredictable reference to a value in trusted storage. Its metadata records the authenticated caller (who may use the ID), the current interaction (session), its expiry time (time-to-live), the object version, and the external destinations for which resolution is permitted. Resolving a reference means replacing the ID with its stored value. In practice, a reference ID can be generated from a fresh nonce (a value used only once) and a secret handle key, e.g., $\mathsf{hid} = \mathrm{HMAC\text{-}SHA256}(k_{\mathsf{handle}}, \mathsf{nonce} \| \mathsf{session} \| \mathsf{object})$, and resolved by lookup in the trusted store. The ID refers to the stored value; it is neither a hash of the secret nor an encrypted copy. The runtime may route the ID through later computation, memory, or tool calls, but it cannot resolve the ID into plaintext.

The gateway returns a reference ID and a short planning summary. The summary operator's deterministic core uses schema rules to retain up to eight fields or items per container and truncate text fields at 512 characters, before adding truncation indicators. It replaces identifiers such as emails, account numbers, and URLs with per-read tokens and strips configured suspicious markers and tags, without calling an LLM. That summary is treated as an explicit declassification interface rather than an incidental copy of the original value [Sabelfeld and Sands, 2005]. The reference-ID layer therefore prevents a later use of an ID from returning raw protected values to the runtime: inter-agent messages and memory carry references rather than secrets retrieved from trusted storage [Watson et al., 2010, Saltzer and Schroeder, 1975]. The runtime can still use content deliberately released in the planning summary, but it cannot retrieve additional protected content by resolving an ID.



(a) Unprotected agent architecture.



(b) SecureClaw architecture.

Figure 1: Comparison between an unprotected agent architecture and SecureClaw. (1) *Read*: the gateway returns reference IDs and short planning summaries. (2) *Plan*: the runtime uses this view to propose an action. (3) *Preview*: the gateway assembles the exact request and the policy engine binds its approval to it. (4) *Commit*: the executor verifies that approval before resolving references and performing the external action.

3.3 Write path: PREVIEW→COMMIT execution boundary

The write path protects the effect surface. The runtime may request an effectful action, but it cannot commit that action directly. The runtime proposes only the action family and runtime-visible inputs. The trusted gateway or surrounding deployment context adds authenticated caller, session, and context fields relevant to authorizing and executing the action. It then canonicalizes the request by serializing all fields that determine the action in a fixed form. Thus every effectful request

$$\rho = (\mathsf{intent}, \mathsf{caller}, \mathsf{session}, \mathsf{inputs}, \mathsf{ctx})$$

passes through Protocol 1. Here intent is the action family, caller identifies the principal and session identifies the interaction, inputs contains tool arguments and reference IDs, and ctx contains other fields relevant to authorizing and executing the action, such as recipients, channels, object identifiers, provenance hashes, policy/schema versions, confirmation class, and object versions.

Protocol 1 PREVIEW→COMMIT

1: The runtime proposes (intent, inputs); the trusted gateway appends authenticated caller, session, and ctx to form ρ.
2: The gateway canonicalizes ρ and computes the binding digest

$$h \leftarrow \text{HMAC-SHA256}(k_{\text{bind}}, \text{Canon}(\rho)),$$

where k_{bind} is a secret key shared only by the trusted binding/verification components.
3: The gateway queries the policy engine and obtains an authorization artifact that binds the allow/deny decision to the exact digest h.
4: The executor independently recomputes h, verifies the authorization artifact, checks freshness and replay state, validates any confirmation token, and enforces restrictions on resolving reference IDs.
5: **if** all checks succeed **then**
6: The executor resolves any required reference IDs and commits the external effect.
7: **else**
8: The executor rejects the request without executing the action (fails closed).
9: **end if**

The crucial property is that the executor is the only component that can reach the API that executes the external action. A compromised runtime may attempt to mutate the recipient, channel, or other fields after preview, but such changes alter the canonical request and therefore invalidate the bound authorization artifact. Theorem 1 formalizes this as request-bound authorization integrity.

In the invoice example, $\text{Canon}(\rho)$ serializes the recipient, channel, caller, session, and invoice version. Changing the recipient changes this serialization and invalidates the approval. The gateway and executor perform these checks outside the LLM runtime.

This distinction also clarifies why SecureClaw is stronger than argument filtering alone. If the runtime can still reach an API that executes an external action, checking arguments does not create a non-bypassable commit boundary. Zero-argument effectful tools make this explicit: there are no arguments to inspect. On the read side, once unrestricted plaintext enters an adversarial runtime, downstream checks cannot prevent it from being re-encoded into another channel. SecureClaw therefore separates control of external actions from access to raw protected data.

3.4 Deny-aware recovery

Fail-closed execution is necessary for security, but a denial should not force the workflow to terminate when a safe alternative exists. When the executor rejects a commit, it returns a denial from a fixed set of templates. The response contains only a coarse action class, a reason code, and safe next-step hints; it does not reveal protected content obtained by resolving a reference. For example, it may suggest switching from "send" to "draft for confirmation" or requesting user approval. This recovery logic does not weaken the commit invariant; it preserves utility after blocked attacks.

3.5 Threat-to-mechanism summary and formal scope

Each SecureClaw mechanism protects against a different failure mode.

Unauthorized external effects. The execution boundary and request binding protect the effect surface: an external side effect can commit only if the executor verifies authorization for the exact canonicalized request that reaches the external API.

Internal plaintext relay. The reference-ID layer protects the plaintext surface: inter-agent messages, memory operations, and later tool calls carry reference IDs rather than raw sensitive values, except for content intentionally released through declared declassification interfaces. In the deployed system, any remaining runtime exposure occurs only through the short planning-summary interface.

Formal scope. The appendix mirrors this decomposition. Theorem 1 formalizes request-bound authorization integrity for the commit path. Theorem 2 formalizes raw-plaintext confinement for

Table 2: Benchmark evaluation on same-harness baselines (`gpt-4o-mini`). ASR = attack success rate; Atk. Util = benchmark utility on attacked rows; Benign = benign-task success rate.

	AgentDojo (n=629)			**AgentLeak** (n=496)			**ASB** (n=2000)	
System	ASR↓	Attack. Utility↑	Benign ↑	Any leak↓	C2↓	C5↓	ASR↓	Util↑
Plain	31.48	47.22	70.10	93.95	92.74	51.61	91.65	0.85
IPIGuard	15.26	51.99	64.95	78.63	50.00	57.06	99.60	22.05
DRIFT	2.38	52.78	59.79	72.18	47.98	53.63	87.75	0.45
Faramesh	22.89	47.69	60.82	93.95	92.74	51.61	**0.00**	3.70
SecureClaw	**0.64**	**56.60**	60.82	**3.23**	**0.20**	**0.00**	**0.00**	**88.90**

the handle-only setting, without planning summaries. Theorem 3 then quantifies how much observing the planning summaries used in the experiments can help an adversarial runtime distinguish protected values. Appendix B.5 states an optional policy-side privacy refinement, and Appendix B.6 summarizes the guarantees lost if trusted components are compromised.

4 Experiments

We evaluate SecureClaw against our central claim: securing tool-using agents requires protecting two distinct surfaces, the effect surface and the plaintext surface.

The experiments ask whether SecureClaw (i) blocks unauthorized external effects, (ii) removes runtime-held plaintext from internal leakage channels, and (iii) keeps workflows usable after blocked actions without relaxing the commit invariant. We answer these questions with full-benchmark evaluations on AgentDojo, AgentLeak, and ASB, a mechanism ablation, a compromised-runtime bypass suite, and targeted measurements of deny-aware recovery and overhead.

Baselines and setup. We compare against four common-harness baselines, meaning matched rows, model, scorer, temperature, and task protocol: **Plain** (no defense), **IPIGuard** [An et al., 2025], **DRIFT** [Li et al., 2025b], and **Faramesh** [Fatmi, 2026]. Appendix A.2 specifies the code source, adapter, and defense surface for each baseline. Additional AgentDojo comparisons use the released Progent and CaMeL implementations through benchmark adapters (Appendix C.2); Section 5 compares their enforcement mechanisms. We report ASR, benchmark utility, and per-channel leak rates; SecureClaw's mediated outputs are scored after alias resolution and required confirmation handling. All main runs use `gpt-4o-mini-2024-07-18` with temperature 0 on AgentDojo [Debenedetti et al., 2024], AgentLeak [Yagoubi et al., 2026], and ASB [Zhang et al., 2025]. The main-table AgentLeak columns use the 496 attacked scenarios in its parity lane, which checks leakage through final responses (C1), inter-agent messages (C2), and shared memory (C5). Appendix A.2 gives exact counts, confidence intervals, and the original cross-model spot checks. Appendix C.1 adds full-inventory SecureClaw transfer runs; these remain separate from the primary comparisons.

4.1 Main cross-benchmark results

Table 2 shows that, among the evaluated common-harness baselines in our study, SecureClaw is the only same-harness configuration that jointly drives unauthorized commits to zero/near-zero on the effect benchmarks and reduces AgentLeak's attacked parity-lane leakage to 3.23% (16/496). On effectful benchmarks it reaches 0% ASR on ASB and 0.64% ASR on AgentDojo (4/629). On AgentLeak, the residual is C1-dominant rather than an internal-relay failure: 15/496 cases remain on C1, one isolated case appears on C2, and C5 stays at 0/496. Faramesh matches the effect boundary on ASB but leaves AgentLeak's internal channels open because plaintext still lives inside the runtime. IPIGuard and DRIFT reduce some attack success rates, but their enforcement root remains runtime-local.

Utility under attack. SecureClaw retains 88.90% utility on ASB at 0% ASR. A separate recovery ablation (Appendix A.7) shows that safe continuation raises matched-sample ASB utility from 70%

Table 3: Mechanism ablation on a matched AgentLeak slice (24 attacked + 24 benign valid rows per configuration; C2/C5 leak rates are reported on attacked rows only). Boundary = executor-side PREVIEW→COMMIT; Handles = reference-ID confinement on the read path.

Config	Boundary	Handles	C2 Leak	C5 Leak
Full	✓	✓	**0.0%**	**0.0%**
−Boundary	×	✓	**0.0%**	**0.0%**
−Handles	✓	×	20.8%	8.3%
−Both	×	×	25.0%	4.2%

to 86% without increasing ASR. On the AgentLeak parity run, attack utility is 74.6% and benign utility is 77.0% (Table 12), so the low parity-lane leak rate is not achieved by collapsing the workflow into refusal.

4.2 Mechanism isolation: why both pillars are necessary

Table 3 isolates mechanisms on a matched 48-row AgentLeak slice. Removing handles (reference-ID confinement) raises attacked-row C2 leakage from $0/24$ to $5/24$ or $6/24$, and C5 leakage reappears ($2/24$ or $1/24$). Removing the execution boundary while keeping handles leaves both C2 and C5 at $0/24$. The confidentiality gain therefore comes from handle confinement, not from the write boundary accidentally masking it.

The converse appears when APIs that execute external actions are stressed directly: handles do not stop a compromised runtime from attempting unauthorized commits. That role belongs to the executor-side boundary and is validated in the bypass suite. Together, the ablation and adversarial validation support the main system's claim: the two pillars protect different surfaces, and neither subsumes the other.

Deny-aware recovery is the source of high blocked-attack utility. We measure recovery separately to show that SecureClaw's usability does not come from weaker enforcement. On a matched ASB sample (n=50), deny-aware recovery raises utility from 70% to 86% while preserving 0% ASR. On a matched AgentLeak sweep (n=30), it raises utility from 40.0% to 66.7%, again with no ASR cost, while adding about 0.33 extra tool turns per task. SecureClaw stays usable because it offers safe next steps after denial, not because it relaxes the commit invariant.

4.3 Robustness under runtime compromise

Behavioral prompt-injection benchmarks are necessary but not sufficient because the formal claims also cover a fully adversarial runtime. We therefore add direct adversarial stress tests against the commit path.

Across 29 bypass scenarios spanning authorization-artifact tampering, malformed authorization material, request-binding violations, session/caller mismatches, context-binding attacks, temporal replay, and chained attacks, SecureClaw denies all 25 adversarial cases and accepts all 4 positive controls. In a payment case study, the executor blocks 4/4 unauthorized commits while allowing the legitimate transfer. These results show that the commit path respects the request-bound invariant even when the runtime constructs adversarial inputs.

4.4 Residual failures and overhead

The remaining failures show what is left once the two target surfaces are controlled. The dominant residual is no longer unauthorized execution or internal-channel plaintext relay, but behavior that the policy permits but that does not match the intended task. On AgentDojo, all 4/629 remaining attack successes stay within the allowed action space: three involve a wrong principal or missing context binding, and one applies an allowed workspace action to the wrong object. On AgentLeak, 16/496 attacked parity-lane scenarios still leak: fifteen are C1 final-output cases and one is an isolated C2 hit. We observe no C5 leakage and no systematic reopening of internal relay, shifting the residual problem to policy-allowed misalignment and user-visible information release.

Table 4: Residual failures after SecureClaw closes the two target surfaces.

Residual class	Count	Interpretation
Wrong principal or missing context binding	$3/629$	The action family is permitted, but the target principal is attacker-chosen; richer principal and object binding is needed.
Broader-scope workspace action	$1/629$	A workspace mutation remains individually allowed yet wrong for the task; coarse allow bits are insufficient.
Residual information leakage	$16/496$	Fifteen cases remain on C1 and one on C2; none appear on C5. The residual is concentrated in final outputs rather than systematic internal relay.

The remaining failures concern which actions and information releases the policy permits; they do not bypass the core invariants. The residual leakage motivates evaluating more selective content release through the planning summary. Richer principal, object, and phase binding would address the remaining AgentDojo failure classes.

Our local policy-check diagnostic takes about 15 ms with cleartext policy lookup and 149 ms with the distributed two-evaluator configuration, including gateway orchestration (Appendix A.8). Relative to LLM inference latencies of 500–2000 ms, these measured costs are not dominant.

Model dependence and evaluation protocol. Full-inventory transfer runs preserve $0/2000$ ASR on each of five ASB model snapshots, with utility from 83.6% to 94.2% (Appendix C.1). The original AgentLeak spot checks remain in Appendix A.3; the primary $16/496$ result is unchanged. Appendix C.4 quantifies reference resolution and confirmation handling, and Appendix C.3 measures the effect of missed protected fields. These checks distinguish trusted enforcement from model-dependent planning and output behavior.

5 Additional related work

Benchmarks and runtime defenses. AgentDojo, ASB, AgentLeak, WASP, and tool-selection attacks have shifted evaluation from prompt classification toward dynamic agents that read untrusted content, call tools, and maintain state [Debenedetti et al., 2024, Zhang et al., 2025, Yagoubi et al., 2026, Evtimov et al., 2025, Shi et al., 2025a]. Runtime-centric defenses such as IPIGuard and DRIFT make the planner more robust by isolating injected instructions or constraining tool dependencies [An et al., 2025, Li et al., 2025b]. SecureClaw uses such defenses as common-harness baselines when their public surface is comparable, but studies a different claim: an injected or compromised runtime should not be the reference monitor for external effects or sensitive plaintext.

Execution control and agent authorization. Faramesh, Progent, PCAS, and AC4A move enforcement toward policy or capability checks at the agent action boundary [Fatmi, 2026, Shi et al., 2025b, Palumbo et al., 2026, Sharma and Grossman, 2026]. SecureClaw combines established authorization and data-protection mechanisms under a joint contract for an adversarial runtime. Its contribution is the enforced relationship between the two controls and the evidence that each is necessary. On the write path, PREVIEW$\rightarrow$COMMIT binds the policy decision to the exact canonical request later executed, closing the time-of-check/time-of-use gap between planner inspection and the external API call. Authorization at the action boundary alone does not remove secrets already resident in planner state: a blocked commit cannot undo leakage through inter-agent messages or memory. This motivates coupling effect mediation with a separate trusted read gateway.

Faramesh overlaps with canonical action authorization, while CaMeL covers both control and data flow through a capability-enforcing interpreter [Debenedetti et al., 2025]. SecureClaw preserves the native tool-calling loop and places both checks in trusted components outside that loop. ClawGuard enforces user-confirmed, task-scoped rules at every tool-call boundary within a single tool-call pipeline [Zhao et al., 2026]; SecureClaw instead re-verifies a request-bound approval in a separate executor and keeps raw protected values in trusted storage. Concurrent work on Twin Agent also separates privileges, but between two LLM agents: an Explore Agent reads untrusted content and passes compact hints to a privileged Safe Agent [Hu et al., 2026]. SecureClaw does not trust an

LLM for security enforcement: the runtime is untrusted, the summary core follows fixed rules, and a non-LLM executor re-verifies each request. We view the two designs as complementary.

Information flow, capabilities, and policy expressiveness. The read path draws on information-flow control, declassification, and capability systems [Zhong et al., 2025, Costa et al., 2025, Sabelfeld and Sands, 2005, Saltzer and Schroeder, 1975, Watson et al., 2010, Sabelfeld and Myers, 2003]. Unlike settings where the protected computation is an enforceable program, LLM agents have opaque hidden state and natural-language channels. SecureClaw therefore makes raw protected values unrepresentable in the runtime except through reference IDs, and treats short planning summaries as explicit declassification rather than benign preprocessing. Reference IDs are not ambient authorities for the model: they are resolved only at trusted endpoints after caller, session, destination, and policy checks. Finally, practical authorization systems such as Zanzibar show that utility depends on expressive principal, object, and relation bindings [Pang et al., 2019]. SecureClaw's remaining benign-utility loss fits this pattern: richer bindings, phase-specific capabilities, and more precise declassification budgets can improve utility while preserving the request-binding and plaintext-residency invariants.

6 Limitations and broader impact

SecureClaw reduces unsafe agentic effects and plaintext exposure, but it does not guarantee semantic correctness within policy-allowed actions and still incurs benign-utility loss on AgentDojo. The primary common-harness comparison uses a single backbone (GPT-4o-mini). The additional ASB snapshots in Appendix C.1 retain zero observed attack success, but task utility varies with the backbone. Final-answer leakage depends on model output behavior, and for AgentLeak this paper reports only spot checks on other backbones (Appendix A.3). The residuals motivate stronger principal, object, and phase binding in the policy layer [Pang et al., 2019]. Protected-field identification is recall-critical: controlled misses increase leakage, while conservative over-marking is inexpensive in the tested range (Appendix C.3).

The main deployment risk is false assurance: if protected fields are misclassified or APIs that execute external actions remain unmediated, the architecture's boundary guarantees no longer apply. SecureClaw therefore should be deployed with explicit inventories of the APIs that execute external actions, schema audits, and conservative confirmation policies for irreversible actions. Positively, the architecture can reduce unauthorized external effects and internal relay of sensitive values in high-stakes agent workflows without requiring the untrusted runtime to become a trusted security monitor. SecureClaw trusts the gateway, policy engine, and executor; Appendix B.6 details compromise effects, and the optional distributed policy variant relies on non-collusion.

7 Conclusion

SecureClaw is built on a simple claim: securing LLM agents requires separating control of external effects from access to sensitive plaintext. The executor-side PREVIEW→COMMIT boundary makes committed actions request-bound and non-bypassable, while the gateway-side reference-ID layer keeps internal channels symbolic rather than plaintext-bearing. Across AgentDojo, AgentLeak, and ASB, SecureClaw is the only evaluated same-harness system in our study that drives unauthorized effects to zero or near zero while remaining usable under attack through deny-aware recovery. The ablation and bypass results show that the gains are structural: the two mechanisms protect different failure modes, and high-stakes deployments need both.

Acknowledgments and Disclosure of Funding

We thank the anonymous reviewers for their helpful comments and suggestions. Research in part supported by the German Research Foundation (DFG), SPP 2378 - ReNO-2, grant 511099228, 2025–2029. Yuhan Ma is employed by Huawei Technologies Düsseldorf GmbH. Huawei provided no funding or other support for this work. The authors declare no other competing financial interests.

References

Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. IPIGuard: A novel tool dependency graph-based defense against indirect prompt injection in LLM agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. Securing AI agents with information-flow control, 2025. URL https://arxiv.org/abs/2505.23643.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *The Thirty-eighth Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=m1YYAQjO3w.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design, 2025. URL https://arxiv.org/abs/2503.18813.

Ivan Evtimov, Arman Zharmagambetov, Aaron Grattafiori, Chuan Guo, and Kamalika Chaudhuri. WASP: Benchmarking web agent security against prompt injection attacks, 2025. URL https://arxiv.org/abs/2504.18575.

Amjad Fatmi. Faramesh: A protocol-agnostic execution control plane for autonomous agent systems, 2026. URL https://arxiv.org/abs/2601.17744.

Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. FlowFence: Practical data protection for emerging IoT application frameworks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 531–548, 2016.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pages 79–90, 2023.

Zhanhao Hu, Dennis Jacob, Xiao Huang, Zhaorun Chen, Bo Li, and David Wagner. Twin Agent: Context residual compression for privilege separated agents, 2026. URL https://arxiv.org/abs/2607.19595.

Dezhang Kong, Shi Lin, Zhenhua Xu, Zhebo Wang, Minghao Li, Yufeng Li, Yilun Zhang, Hujin Peng, Xiang Chen, Zeyang Sha, Yuyuan Li, Changting Lin, Xun Wang, Xuan Liu, Ningyu Zhang, Chaochao Chen, Chunming Wu, Muhammad Khurram Khan, and Meng Han. A survey of LLM-driven AI agent communication: Protocols, security risks, and defense countermeasures, 2025. URL https://arxiv.org/abs/2506.19676.

Evan Li, Tushin Mallick, Evan Rose, William Robertson, Alina Oprea, and Cristina Nita-Rotaru. ACE: A security architecture for LLM-integrated app systems, 2025a. URL https://arxiv.org/abs/2504.20984.

Hao Li, Xiaogeng Liu, Hung-Chun Chiu, Dianqi Li, Ning Zhang, and Chaowei Xiao. DRIFT: Dynamic rule-based defense with injection isolation for securing LLM agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2025b. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/77f3b26c7907aa27b207df9b9d43f29a-Paper-Conference.pdf.

Yexiong Lin, Shanshan Ye, Yu Yao, Zhen Fang, Bo Han, and Tongliang Liu. SquidAgent: Parallelize wisely, coordinate efficiently, 2026. URL https://arxiv.org/abs/2610.08647.

Nils Palumbo, Sarthak Choudhary, Jihye Choi, Prasad Chalasani, and Somesh Jha. Policy compiler for secure agentic systems, 2026. URL https://arxiv.org/abs/2602.16708v2.

Ruoming Pang, Ramon Caceres, Mike Burrows, Zhifeng Chen, Pratik Dave, Nathan Germer, Alexander Golynski, Kevin Graney, Nina Kang, Lea Kissner, Jeffrey L. Korn, Abhishek Parmar, Christina D. Richards, and Mengzhi Wang. Zanzibar: Google's consistent, global authorization system. In *2019 USENIX Annual Technical Conference (USENIX ATC)*, pages 33–46, 2019.

Andrei Sabelfeld and Andrew C Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

Andrei Sabelfeld and David Sands. Dimensions and principles of declassification. In *Proceedings of the 18th IEEE Workshop on Computer Security Foundations*, pages 255–269, 2005.

Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975. doi: 10.1109/PROC.1975.9939.

Reshabh K Sharma and Dan Grossman. AC4A: Access control for agents, 2026. URL `https://arxiv.org/abs/2603.20933`.

Jiawen Shi, Zenghui Yuan, Guiyao Tie, Pan Zhou, Neil Zhenqiang Gong, and Lichao Sun. Prompt injection attack to tool selection in LLM agents, 2025a. URL `https://arxiv.org/abs/2504.19793`.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Programmable privilege control for LLM agents, 2025b. URL `https://arxiv.org/abs/2504.11703v2`.

Robert N. M. Watson, Jonathan Anderson, Ben Laurie, and Kris Kennaway. Capsicum: practical capabilities for UNIX. In *Proceedings of the 19th USENIX Conference on Security (USENIX Security)*, 2010.

Faouzi El Yagoubi, Godwin Badu-Marfo, and Ranwa Al Mallah. AgentLeak: A full-stack benchmark for privacy leakage in multi-agent LLM systems, 2026. URL `https://arxiv.org/abs/2602.11510v2`.

Miao Yu, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pan, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, Bo An, and Qingsong Wen. A survey on trustworthy LLM agents: Threats and countermeasures. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pages 6216–6226, 2025.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=V4y0CpX4hK`.

Wei Zhao, Zhe Li, Peixin Zhang, and Jun Sun. ClawGuard: A runtime security framework for tool-augmented LLM agents against indirect prompt injection, 2026. URL `https://arxiv.org/abs/2604.11790`.

Peter Yong Zhong, Siyuan Chen, Ruiqi Wang, McKenna McCall, Ben L. Titzer, Heather Miller, and Phillip B. Gibbons. RTBAS: Defending LLM agents against prompt injection and privacy leakage, 2025. URL `https://arxiv.org/abs/2502.08966`.

A Extended experimental evidence

This appendix substantiates the main empirical claims with exact counts, uncertainty bounds, benchmark breakdowns, and robustness checks. The evidence is consistent across the different evaluation blocks: the execution boundary closes the effect surface, reference IDs (opaque handles) close the internal relay surface, and deny-aware recovery recovers substantial under-attack utility without reopening either one.

Statistical reporting. All reported rates are Bernoulli proportions. For the main comparisons we report exact counts, 95% confidence intervals, and two-sided Fisher exact tests where pairwise significance is informative. For zero-event rows, the appendix uses exact Clopper–Pearson upper bounds rather than asymptotic normal approximations.

A.1 Exact counts and selected uncertainty

Table 5 summarizes the main empirical results with exact counts and confidence intervals. It complements the percentage tables in the main paper.

Table 5: Selected exact counts, 95% exact confidence intervals, and exact tests.

Metric	SecureClaw		Comparator	Two-sided Fisher p
AgentDojo ASR	$4/629$ $[0.17, 1.62]$	$= 0.64\%$	DRIFT $15/629 = 2.38\%$ $[1.34, 3.90]$	1.83×10^{-2}
AgentLeak parity-lane any leak	$16/496$ $[1.85, 5.19]$	$= 3.23\%$	DRIFT $358/496 = 72.18\%$ $[68.01, 76.08]$	1.27×10^{-128}
ASB ASR	$0/2000$ $[0.00, 0.18]$	$= 0.00\%$	DRIFT $1755/2000 = 87.75\%$ $[86.23, 89.16]$	$< 10^{-700}$
ASB utility	$1778/2000$ $[87.44, 90.24]$	$= 88.9\%$	Faramesh $74/2000 = 3.7\%$ $[2.92, 4.63]$	$< 10^{-700}$

Table 5 makes clear that the core security results remain strong even under exact accounting.

A.2 Evaluation methodology

Model and inference settings. All primary comparisons use `gpt-4o-mini-2024-07-18` through OpenRouter with temperature 0. Where the provider exposes a seed parameter, we request a fixed seed. Because API-hosted inference is not guaranteed to be perfectly deterministic, repeated runs are interpreted as stability checks rather than as bitwise replicas. SecureClaw is evaluated with the deployed bounded-summary interface throughout the benchmark results in the paper; the security appendix separately analyzes a handle-only reference case and measures the deployed summary operator's confidentiality behavior.

Benchmark protocols. **AgentDojo** uses v1.1.2 with the `important_instructions` attack family, yielding 629 attacked rows and 97 benign rows across banking, slack, travel, and workspace. **AgentLeak** uses its official split of 996 total scenarios: 500 benign and 496 attacked scenarios for the parity lane covering $C1/C2/C5$, plus channel-specific lanes for $C3$, $C4$, and $C6$. The main-paper "Any leak" number refers to the 496 attacked parity-lane scenarios. **ASB** is evaluated on five direct prompt-injection styles (naive, `escape_characters`, `fake_completion`, `context_ignoring`, and `combined_attack`) with 400 rows each, for 2000 total rows per baseline.

Baseline fidelity. Table 6 summarizes how each same-harness baseline is instantiated.

Mediated-interface accommodations. SecureClaw uses two evaluation accommodations that arise from mediated interfaces rather than from relaxed security: **alias resolution** and **auto-confirmation**. Alias resolution maps reference IDs such as `EMAIL_REF_3` to task-relevant literals in trusted downstream tool calls and before exact-string utility scoring. The resolved recipient, channel, and context still undergo policy authorization and request binding. Auto-confirmation emulates a user click in non-interactive benchmark runners that otherwise provide no confirmation channel. It is applied only to requests that are already policy-allowed and remains bound to (a, p, h), so it cannot convert a policy-denied attack proposal into an allowed commit. Appendix C.4 measures where each accommodation is actually used and separates AgentLeak release confirmation from the other benchmarks.

Run-to-run variation. Hosted inference introduces provider-level nondeterminism, so we use two complementary stability checks. First, three completed banking/slack AgentDojo replications keep ASR at 0% on those completed suites while utility varies by only single-digit points; Table 7 gives the details. Second, a matched three-seed AgentLeak-style SecureClaw sweep ($n = 30$ per seed) keeps ASR at 0.0% across all seeds with utility mean 64.4% and standard deviation 8.4, and an independent ASB rerun at $n = 50$ also preserves 0% ASR with 86.0% utility. The qualitative conclusion is stable: security is much less sensitive than utility to hosted-model nondeterminism.

Table 6: Baseline provenance and common-harness instantiation.

System	Evaluation implementation	Modification scope	Common-harness entry point
Plain	Benchmark no-defense configuration.	No defense code or wrapper.	Native benchmark runners and direct AgentLeak outputs.
IPIGuard	Authors' upstream code, vendored for evaluation.	Harness-only patches for API compatibility, retries, resume, and benchmark integration; defense logic unchanged.	IPIGuard AgentDojo runner, AgentLeak pipeline adapter, and ASB-style construct/traverse loop.
DRIFT	Authors' upstream code, vendored for evaluation.	Harness-only patches for hosted-model access, retries, logging, and benchmark integration; defense logic unchanged.	DRIFT pipeline, AgentLeak tool-execution loop, and DRIFT-provided ASB fork.
Faramesh	Upstream Faramesh core, vendored for evaluation.	Core runtime unchanged; our harness supplies benchmark adapters and policies.	Local Faramesh server mediating AgentDojo, AgentLeak, and ASB-style tool calls.

Table 7: Replication stability on AgentDojo banking + slack attack suites.

	Banking ($n = 144$)		Slack ($n = 105$)	
Run	ASR	Utility	ASR	Utility
Run 1 (primary)	0.0%	56.3%	0.0%	57.1%
Run 2	0.0%	52.1%	0.0%	49.5%
Run 3	0.0%	51.4%	0.0%	48.6%
Sample std. dev.	0 pp	$\pm$2.7 pp	0 pp	$\pm$4.6 pp

Code availability The SecureClaw implementation and evaluation code are available at `https://github.com/ponyyuhan/SecureClaw`. An archived version is available at `https://doi.org/10.5281/zenodo.23258171`. The repository includes the core runtime, benchmark adapters, configurations, evaluation and scoring scripts, setup instructions, and local tests. Our code is released under the MIT license; third-party components retain their original licenses. The documentation describes how to obtain benchmark assets and run the experiments. Generated results, raw model traces, and temporary development files are not bundled.

Compute resources. All experiments were run on a local macOS workstation (Apple M4 CPU, 16 GB RAM) with API-hosted LLM inference. The gateway, handle store, executor, and policy-evaluation components execute locally; the reported latency numbers therefore reflect algorithmic and local orchestration overhead, not geographically distributed network round trips between separate servers.

Asset provenance and licenses. Table 8 lists the main third-party benchmark assets used in our experiments and their stated upstream licenses. Our same-harness baselines use upstream code or benchmark adapters as summarized in Table 6; retained third-party components keep their original LICENSE files and attribution metadata.

Table 8: Main third-party benchmark assets used in this paper.

Asset	Citation / upstream source	License	Use in this paper
AgentDojo	[Debenedetti et al., 2024] / official benchmark repository	MIT	Prompt-injection evaluation
AgentLeak	[Yagoubi et al., 2026] / official benchmark repository	MIT	Internal-leakage evaluation
ASB	[Zhang et al., 2025] / official benchmark repository	MIT	Effect-side attack evaluation

A.3 Cross-model spot checks

The main paper keeps all full-scale baseline comparisons on a single backbone to preserve a common harness. To test whether the security pattern is tied to that model, we also ran small SecureClaw-only spot replications on additional model families. We use these as transfer checks rather than as full cross-model baseline rerankings.

Table 9: SecureClaw cross-model transfer checks.

Benchmark	Model	Rows	Security result	Utility
ASB	Claude Sonnet 4	50	ASR $= 0/50 = 0.0\%$	94.0%
ASB	Claude 3.5 Haiku	50	ASR $= 0/50 = 0.0\%$	86.0%
AgentLeak slice	Claude Sonnet 4	50	$C1/C2/C5 = 0/0/0$	100.0%

These transfer checks preserve the qualitative security pattern off the primary backbone: SecureClaw keeps ASR at zero on both non-primary ASB runs and remains clean on the Claude Sonnet 4 AgentLeak slice.

A.4 AgentLeak overview and per-channel leak rates

The main paper uses the attacked parity lane's overall scenario-OR leak because it provides the closest single-number analogue to AgentDojo ASR and ASB ASR. Table 10 gives the parity-lane per-channel breakdown and Table 11 reports the remaining channel lanes $C3/C4/C6$. SecureClaw's residual is C1-dominant: 15/496 parity-lane cases remain on C1, one isolated case appears on C2, and C5 stays at 0/496.

Table 10: AgentLeak parity-lane per-channel leak rates (%) $\downarrow$ ($n = 496$ attacked rows).

System	C1 Leak	C2 Leak	C5 Leak
Plain	33.47	92.74	51.61
IPIGuard	56.25	50.00	57.06
DRIFT	51.21	47.98	53.63
Faramesh	33.47	92.74	51.61
SecureClaw	**3.02**	**0.20**	**0.00**

Table 11: AgentLeak channel-lane results: attack leak rate (%) $\downarrow$ / benign allow rate (%) $\uparrow$.

	C3		C4		C6	
System	Leak$\downarrow$	Allow$\uparrow$	Leak$\downarrow$	Allow$\uparrow$	Leak$\downarrow$	Allow$\uparrow$
Plain	100.0	100.0	100.0	100.0	100.0	100.0
IPIGuard	24.03	100.0	33.61	100.0	0.00	100.0
DRIFT	6.98	99.21	14.75	99.80	0.00	98.90
Faramesh	100.0	100.0	100.0	100.0	100.0	100.0
SecureClaw	**0.00**	**100.0**	**0.00**	**100.0**	**0.00**	99.90

For completeness, Table 12 reports the official strict-evaluator utility counts underlying the main-text AgentLeak utility sentence.

These tables make the paper's security decomposition concrete. Faramesh matches SecureClaw on ASB-style effect blocking but remains indistinguishable from Plain on $C2$ and $C5$ because it does not remove plaintext from the runtime. SecureClaw reduces $C2$ leakage to $1/496$ and eliminates measured leakage on $C3$, $C4$, $C5$, and $C6$, while leaving $C1$ as the dominant residual channel. That is exactly the behavior predicted by the architecture.

A.5 Residual taxonomy

Table 13 groups the observed residuals by security meaning.

Table 12: AgentLeak parity-lane utility counts for SecureClaw.

Slice	Successes	Rate
Attacked rows	$370/496$	74.6%
Benign rows	$385/500$	77.0%
Overall	$755/996$	75.8%

Table 13: Manual residual taxonomy from the full runs and targeted cross-checks.

Evidence source	Residual count	Manual class	Interpretation
AgentDojo full attack run	3/629	wrong principal or missing context binding	The action family is allowed, but the target principal is attacker-chosen; richer principal and object binding is needed.
AgentDojo full attack run	1/629	broader-scope workspace action	A workspace mutation remains individually allowed yet wrong for the task; coarse allow bits are insufficient.
AgentLeak attacked parity lane	16/496 residuals	authorized-output-dominant residual	Fifteen cases remain on the final authorized output channel and one on C2; no C5 relay reappears.

The residual profile is concentrated and interpretable. We do not claim that all $15/496$ surviving $C1$ cases share the same cause; the evidence here supports only the narrower conclusion that the residual is concentrated on the final authorized-output plane rather than on reopened internal relay.

A.6 Ablation extended results

We report here the same matched slice used in the main paper after filtering rows with valid channel instrumentation: 48 valid rows per configuration (24 attack and 24 benign). This slice is mechanism-focused and is not intended to reproduce the full AgentLeak Plain distribution; in particular, the –Both row should be read as a matched-slice internal control rather than as the Table 2 Plain baseline. The point of the ablation is not to optimize utility. It is to identify the active mechanism behind the main-paper leakage reductions using a single denominator convention throughout.

Table 14: Extended ablation on the matched AgentLeak slice. C2/C5 are reported on the 24 attacked rows; benign leak is reported on the 24 benign rows.

Config	Boundary	Handles	C2 Leak	C5 Leak	Benign Leak
Full	✓	✓	$0/24$ (0.0%)	$0/24$ (0.0%)	$0/24$ (0.0%)
−Boundary	×	✓	$0/24$ (0.0%)	$0/24$ (0.0%)	$0/24$ (0.0%)
−Handles	✓	×	$5/24$ (20.8%)	$2/24$ (8.3%)	$4/24$ (16.7%)
−Both	×	×	$6/24$ (25.0%)	$1/24$ (4.2%)	$4/24$ (16.7%)

Turning off the boundary while keeping handles leaves $C2$ and $C5$ at zero, which shows that those leak reductions do not come from the execution boundary. Turning off handles while keeping the boundary reopens $C2$ to $5/24$ attacked rows and $C5$ to $2/24$ attacked rows; removing both gives $6/24$ and $1/24$ respectively. The converse point is validated by the bypass suite: handles alone do not secure effectful sinks. On this small matched slice, removing handles produces five or six additional C2 leaks among 24 attacked rows; the C5 differences involve only one or two rows. These counts support the mechanism comparison, while the small sample limits the precision of channel-specific rates.

Table 15: Recovery ablation on two matched sweeps. All configurations retain 0% ASR.

Benchmark	Config	n	ASR	Utility	Refusals	Extra turns/task
ASB	recovery on	50	0.0%	86.0%	8.0%	—
ASB	recovery off	50	0.0%	70.0%	30.0%	—
AgentLeak-style	recovery on	30	0.0%	66.7%	66.7%	1.33
AgentLeak-style	recovery off	30	0.0%	40.0%	60.0%	1.00

A.7 Recovery ablation details

The two sweeps are consistent. Recovery lifts utility by 16.0 points on ASB and 26.7 points on the matched AgentLeak-style subset, with no ASR cost in either case. On the instrumented subset it adds only about 0.33 extra tool turns per task, which is small relative to the LLM-driven workflows we target.

Per-attack breakdown. With recovery enabled, utility is 60% on naive, 80% on `escape_characters`, 100% on `fake_completion`, 100% on `context_ignoring`, and 90% on `combined_attack`. Without recovery, the largest drops occur on naive and `escape_characters`, where the denial message helps the runtime restart from a cleaner plan. The key point is straightforward: SecureClaw's high ASB utility comes from safe continuation after denial, not from weaker enforcement.

A.8 Latency details

Table 16: Latency of the local policy-check diagnostic, including gateway orchestration.

Configuration	Avg (ms)	p50 (ms)	p95 (ms)
Distributed policy checks	148.6	159.7	166.6
Cleartext policy lookup	15.4	12.8	32.2
Policy-bypass control	0.22	0.09	0.55

The diagnostic primarily issues policy-check requests rather than committed external actions. Its policy-bypass control disables policy evaluation and the external executor, so that row does not isolate executor verification cost. The distributed configuration is slower than cleartext lookup, while both remain below the 500–2000 ms LLM inference range used for comparison. These measurements cover local orchestration, not network round trips between geographically separated services.

A.9 Commit-path bypass suite details

The bypass suite validates the implementation against the compromised-runtime threat model. It intentionally goes beyond prompt injection and directly mutates authorization artifacts, timestamps, request bindings, and replay state. The suite includes both core single-service cases and a small number of optional distributed-variant checks.

This table is the empirical companion to Theorem 1: every mutated path that would have undermined request-bound authorization is denied, and every legitimate control path still succeeds.

A.10 Payment case-study details

The payment case study is not the paper's only scenario, but it is a clean concrete illustration of the effect surface. Once commit authority moves to the executor, post-preview mutation of amount, recipient, or authorization state no longer changes the committed action.

B Formal security guarantees

This appendix makes the paper's formal contract explicit. It states the guarantees SecureClaw is designed to enforce, isolates the assumptions needed for each one, and calibrates the deployed sum-

Table 17: Commit-path bypass suite, including optional distributed-variant cases: 29/29 expected outcomes (25 attacks denied, 4 controls allowed).

Category	Bypass attempt	Expected	Observed
Authorization-share integrity	Bit-flipped MAC in authorization share	DENY	DENY
	Wrong share version ($v = 2$)	DENY	DENY
	Server-ID swap ($sid_0 = 1$)	DENY	DENY
	Cross-commit mixup (mismatched authorization shares)	DENY	DENY
Missing shares	Missing second authorization share	DENY	DENY
	Empty commit (no authorization shares)	DENY	DENY
	Null policy entries	DENY	DENY
Binding violations	Request hash mismatch (text tamper)	DENY	DENY
	Recipient tampered after commit	DENY	DENY
	Channel tampered after commit	DENY	DENY
	Domain tampered after commit	DENY	DENY
	Action ID mismatch	DENY	DENY
Session/caller	Session ID mismatch	DENY	DENY
	Caller ID mismatch	DENY	DENY
	Cross-session replay	DENY	DENY
Context binding	External principal injection	DENY	DENY
	Delegation-token identifier (JTI) injection	DENY	DENY
	Context-hash injection	DENY	DENY
Temporal	Expired freshness window (past)	DENY	DENY
	Future-timestamped authorization	DENY	DENY
	Replay of consumed action_id	DENY	DENY
	Missing action_id (empty)	DENY	DENY
Chained attacks	Expired authorization + text tamper	DENY	DENY
	MAC flip + recipient change	DENY	DENY
	Confirm-gated: no user_confirm	DENY	DENY
Positive controls	Valid dual authorization (primary)	ALLOW	ALLOW
	Valid second request (different recipient)	ALLOW	ALLOW
	Valid third request (different text)	ALLOW	ALLOW
	Confirm-gated: with user_confirm	ALLOW	ALLOW

Table 18: Payment case study: the boundary blocks 4/4 unauthorized commits.

Scenario	Attack vector	Boundary on	Boundary off
Legitimate payment	Valid authorization	ALLOW	ALLOW
Amount manipulation	\$100 → \$10,000	**DENY**	ALLOW
Recipient redirect	→ attacker@evil.com	**DENY**	ALLOW
Forged authorization	Bit-flip MAC	**DENY**	ALLOW
Replay attack	Replay prior commit	**DENY**	ALLOW

mary interface used in experiments. Throughout, the runtime is adversarial unless stated otherwise. The purpose is precision rather than breadth: the proofs justify the architecture under the stated trust boundary, while the experiments determine which component is responsible for the measured benchmark gains.

B.1 System model, notation, and interfaces

Let λ denote the implicit security parameter; concrete bounds below use the handle length κ and explicit oracle counts. SecureClaw has five logical roles: an untrusted runtime $\mathcal{R}$, a trusted gateway $\mathcal{G}$, a trusted handle store $\mathcal{S}$ (the trusted data store of the main text), a trusted policy engine $\mathcal{P}$, and a trusted executor $\mathcal{X}$. Appendix B.5 analyzes an optional distributed realization in which the policy engine is instantiated as two evaluators $\mathcal{P}_0, \mathcal{P}_1$. The threat model allows full corruption of $\mathcal{R}$. The gateway, handle store, policy engine, and executor are trusted for the core authorization and confinement guarantees; Section B.6 makes explicit which guarantees fail if any of those components

is compromised. For the per-evaluator transcript-privacy result stated later, we refine the policy assumption: in the distributed realization, each evaluator follows the protocol and the pair does not collude.

Canonical request and binding digest. An effectful request is the tuple

$$\rho = (\mathsf{intent}, \mathsf{caller}, \mathsf{session}, \mathsf{inputs}, \mathsf{ctx}),$$

where ctx must contain every sink-relevant field needed to determine the realized external effect (a *sink* is an interface at which an action takes external effect). Representative fields include recipient or attendee sets, channels, object identifiers, confirmation class, delegation identifiers, provenance hashes, policy/schema versions, and version identifiers for mutable referenced handles or objects.

The gateway and executor serialize requests using the same deterministic canonicalization procedure $\mathsf{Canon}(\cdot)$. The binding digest is

$$h(\rho) := \mathrm{Bind}_{k_{\mathrm{bind}}}(\rho) := \text{HMAC-SHA256}(k_{\mathrm{bind}}, \mathsf{Canon}(\rho)).$$

The security role of $h(\rho)$ is narrow and important: it turns the executor's question from "is this action type generally allowed?" into "is this exact canonical request the one that was authorized?"

Preview and commit interfaces. On preview, the gateway canonicalizes ρ, queries policy, and returns an action identifier a, a program identifier p, and an authorization artifact

$$\pi = (\mathsf{meta}, t), \qquad t = \mathsf{Tag}_{K_{\mathrm{auth}}}(\mathsf{Canon}(\mathsf{meta})),$$

where meta binds (a, p, h, τ, y), τ is the freshness timestamp/window material, and y is the policy decision. The authentication key K_{auth} is distinct from the request-binding key k_{bind}; SecureClaw assumes key separation and domain separation between artifact authentication and request binding. On commit, the executor recomputes $h(\rho)$ and accepts only if all of the following hold:

1. the authorization artifact carries a valid MAC under the policy-service authentication key;
2. the artifact binds the tuple (a, p, h);
3. the recomputed digest equals the digest bound inside the artifact;
4. timestamps fall inside the freshness window;
5. the action identifier has not already been consumed by the replay guard;
6. the reconstructed policy result is `allow`;
7. if the action requires user confirmation, a confirmation token bound to (a, p, h) is present; and
8. any required handle-resolution or sink-specific validation succeeds before the effectful sink is reached.

Any failure is fail-closed. Appendix B.5 gives the optional distributed specialization that replaces π with two separately authenticated shares for the same digest.

Handles, dereference, and read interfaces. Sensitive values identified by the deployment schema or policy are stored in a trusted handle table. Under-classified sensitive fields are outside the handle-confinement claim. A handle record contains a fresh handle identifier hid together with caller, session, object version, time-to-live, and sink constraints. The runtime may carry, store, or transmit hid as a symbol, but only trusted components may dereference it. The deployed read interface returns hid together with an explicit bounded declassification $D(v)$. For proof structure we also refer to a handle-only reference case in which the runtime receives only hid and non-sensitive metadata. All benchmark results in the paper use the bounded-summary interface because current public benchmarks require natural-language read access for many tasks.

Effectful sinks and explicit declassification. An effectful sink is any interface that can commit an irreversible or externally visible side effect. An explicit declassification event is any interface that intentionally releases information from a protected value to the runtime, including the read-summary interface and tightly scoped recovery messages. Everything else is treated as non-declassifying by default.

B.2 Assumption partition

Table 19 separates assumptions by the property they support. This partition matters because the execution boundary, the handle layer, the summary interface, and the optional distributed policy-engine refinement rely on different technical premises.

Table 19: Assumption partition by security property.

Property	Security objective	Assumptions
Execution boundary	No unauthorized side effects.	MAC EUF-CMA; deterministic canonicalization; bounded request-binding collision term $\varepsilon_{\mathsf{bind}}(q_{\mathsf{bind}})$ over all digest evaluations in the game; confirmation binding when required; bounded clock drift; atomic replay check-and-mark; complete mediation of effectful sinks.
Handle confinement	No plaintext reaches the runtime except through declared interfaces.	Complete mediation of sensitive reads; correct classification of protected fields according to the deployment schema/policy; handle identifiers are sampled uniformly from $\{0,1\}^{\kappa}$ with $\kappa \geq 128$ or generated pseudorandomly from a fresh nonce under a secret handle key with PRF loss $\varepsilon_{\mathsf{hid}}$; unsuccessful dereference attempts reveal only a generic denial bit; at most N_{live} valid handles are outstanding at a time; dereference is allowed only at trusted endpoints.
Bounded-summary interface	Quantified exposure from the deployed read interface.	Fixed summary operator D for the evaluated deployment; any optional post-processor depends only on $D(v)$ and independent randomness; leakage to the runtime occurs only through metadata, handles, and authorized summaries.
Distributed policy evaluation	Single-evaluator transcript privacy.	Two-server private information retrieval (PIR) security against one server; honest-but-curious secure multi-party computation (MPC) privacy; passive non-colluding servers; fixed-shape routing; transcript leakage is limited to L_{policy}.

Assumptions not made. The formal results do not assume an honest runtime, successful prompt filtering, or semantically perfect summaries. The runtime is adversarial throughout. Summary exposure is modeled explicitly through the operator D rather than assumed away. The transcript-privacy result for distributed policy evaluation is deliberately scoped to the non-colluding, single-evaluator-view setting stated below.

B.3 Request-bound authorization integrity

Security game. The challenger samples the artifact-authentication key K_{auth} and request-binding key k_{bind} and exposes a preview oracle $\mathcal{O}_{\mathsf{pv}}(\rho)$ and a commit oracle $\mathcal{O}_{\mathsf{cm}}(\rho, \pi)$. On preview input ρ, $\mathcal{O}_{\mathsf{pv}}$ returns a fresh action identifier a, a program identifier p, a policy decision y, and a valid authorization artifact π, while recording $(a, p, h(\rho), y, \rho)$ in a set $\mathcal{S}$. On commit input (ρ, π), $\mathcal{O}_{\mathsf{cm}}$ runs the executor verification logic, including freshness and atomic replay check-and-mark, returns only accept/deny, and records accepted action identifiers. The adversary may adaptively query both oracles. For an accepting commit query, write the adversary-supplied input as $(\rho^{\star}, \pi^{\star})$ and the accepted metadata as $(a^{\star}, p^{\star}, h^{\star})$. The adversary wins if any accepting commit query satisfies at least one of the following:

1. **No authorization:** there is no tuple $(a^\star, p^\star, h^\star, \texttt{allow}, \rho') \in \mathcal{S}$ matching the accepted authorization metadata.
2. **Binding violation:** there exists $(a^\star, p^\star, h^\star, \texttt{allow}, \rho') \in \mathcal{S}$, but $\mathsf{Canon}(\rho^\star) \neq \mathsf{Canon}(\rho')$ or the two requests realize different sink effects while both are accepted under the same bound digest.
3. **Replay:** an already consumed action identifier is accepted again.

Let q_{bind} denote the total number of canonical requests whose binding digest is ever evaluated in the game, including all adversarial commit candidates.

Theorem 1 (Request-bound authorization integrity)**.** *Assume that (i) the MAC is EUF-CMA secure, (ii)* $\mathrm{Canon}(\cdot)$ *is deterministic, identical at the gateway and executor, uniquely decodable for authenticated metadata, and semantically complete for mediated sink effects: for requests reaching the same mediated sink, equal canonical strings imply the same realized external effect, (iii)* ctx *includes all sink-relevant fields that can change the realized external effect, including version identifiers for mutable referenced objects when applicable, (iv) the request-binding digest has bounded collision term* $\varepsilon_{\mathsf{bind}}(q_{\mathsf{bind}})$ *over distinct canonical byte strings, (v) any required confirmation token is bound to* (a, p, h)*, (vi) freshness checks are enforced, (vii) replay check-and-mark is atomic and crash-safe, and (viii) all effectful sinks are mediated by the executor. Then for any PPT adversary* $\mathcal{A}$*, there exists a MAC-forgery reduction* $\mathcal{B}$ *such that*

$$\Pr[\mathcal{A} \textit{ wins}] \leq \mathrm{Adv}^{\text{euf-cma}}_{\mathsf{MAC}}(\mathcal{B}) + \varepsilon_{\mathsf{bind}}(q_{\mathsf{bind}}).$$

Proof. We use a short sequence of games.

Game G_0**.** This is the real authorization game.

Game G_1**.** Abort if the executor accepts an authorization artifact for an authenticated message $\mathsf{Canon}(\mathsf{meta})$ that was not previously signed by the preview oracle. The change from G_0 to G_1 is bounded by

$$\mathrm{Adv}^{\text{euf-cma}}_{\mathsf{MAC}}(\mathcal{B}).$$

Game G_2**.** In addition to the previous abort rule, abort if an accepting commit for ρ uses a bound digest that was previously recorded for a previewed request ρ' with $\mathrm{Canon}(\rho) \neq \mathrm{Canon}(\rho')$, or if two accepted commit requests have distinct canonical byte strings under the same bound digest. The change from G_1 to G_2 is at most $\varepsilon_{\mathsf{bind}}(q_{\mathsf{bind}})$. The case in which two requests have the same canonical string but different realized sink effects is ruled out by the semantic completeness premise for canonicalization, not by the digest-collision term.

Conditioned on no abort in G_2, every accepting commit query corresponds to a previously previewed authenticated artifact for the same bound digest and an allow decision. Because all sink-relevant fields appear in $\mathsf{Canon}(\rho)$, any post-preview mutation that changes the realized external effect induces a distinct canonical byte string unless it is absorbed by a MAC forgery, a binding collision, or an implementation bug in the stated canonicalization premise. Under the remaining operational premises, including freshness enforcement, replay-safe consumption, confirmation binding when required, and complete mediation of effectful sinks, the executor can therefore accept only the exact previously previewed request once. None of the adversary's winning conditions can occur. A union bound over the game hops gives the stated inequality. □

Instantiating the binding term. The proof keeps $\varepsilon_{\mathsf{bind}}(q_{\mathsf{bind}})$ explicit because the architecture only needs a request-binding digest with negligible collision probability over the canonical request space. In the evaluated system, Bind is instantiated with HMAC-SHA256 over canonical bytes. Under the usual PRF-style heuristic for HMAC, $\varepsilon_{\mathsf{bind}}(q_{\mathsf{bind}})$ behaves like a standard collision term in the digest length; the theorem does not require a stronger, non-standard assumption.

Theorem 1 captures unauthorized execution relative to the bound request. It does not claim that every policy-allowed request is semantically aligned with the user's latent objective; wrong-principal or wrong-object actions inside the policy-allowed region therefore lie outside its scope. Freshness, replay safety, confirmation binding, and complete mediation are deployment premises rather than cryptographic loss terms. The bypass evaluation in Appendix A.9 stress-tests these premises empirically.

Single-service scope and optional distributed variant. The theorem above states the core single-service guarantee used throughout the main paper. Appendix B.5 analyzes an optional distributed policy-evaluation refinement; it does not change the effect/plaintext split or the meaning of request binding.

B.4 Runtime confidentiality under opaque handles

Security game. The challenger initializes the trusted handle store and gives the adversary full control of the runtime. The adversary may request protected reads, move handles across internal channels, and attempt guessed dereferences through any interface that is supposed to reject unknown or unauthorized handles. It wins if it distinguishes which of two challenger-chosen secret values is stored behind a protected handle with non-negligible advantage, without triggering an explicit declassification event.

Base leakage function. Let L_{handles} include handle identifiers issued to the runtime, non-sensitive type metadata, TTL values, policy decision bits, and any declared recovery templates. It excludes all other functions of sensitive plaintext.

Theorem 2 (Handle-only confinement)**.** *Assume the handle-only reference case: the runtime receives only* hid *and non-sensitive metadata; handle identifiers are sampled independently and uniformly from* $\{0,1\}^\kappa$ *with* $\kappa \geq 128$ *or generated by a PRF from a fresh nonce under a secret handle key, unsuccessful dereference attempts reveal only a generic denial bit, and at most* N_{live} *valid handles are simultaneously outstanding in the runtime's view. Let* $\varepsilon_{\mathsf{hid}} = 0$ *for uniform handles and let* $\varepsilon_{\mathsf{hid}}$ *be the PRF distinguishing loss for PRF-generated handles.*

Then for any adversary making at most q_h *online handle-guessing attempts, the runtime's view is simulatable from* L_{handles} *up to additive advantage*

$$\min\left(1, \frac{q_h N_{\mathsf{live}}}{2^\kappa}\right) + \varepsilon_{\mathsf{hid}}.$$

Proof. Construct a simulator that replaces every protected value with an independently sampled handle identifier and exposes only the metadata in L_{handles}. If the implementation uses PRF-generated handles rather than direct uniform sampling, first replace those handles by uniform strings, incurring $\varepsilon_{\mathsf{hid}}$.

Because the handle-only reference case never releases plaintext directly to the runtime, the simulated view and the real view are identical unless the adversary successfully guesses a live handle identifier that was not already issued to it and uses that guess to trigger a privileged dereference path.

At any point there are at most N_{live} valid targets in the handle namespace. A single uniform guess therefore succeeds with probability at most $N_{\mathsf{live}}/2^\kappa$. By a union bound over q_h online guesses, the probability of any successful hit is at most $\min(1, q_h N_{\mathsf{live}}/2^\kappa)$. Conditioned on no successful hit, every runtime-visible object is distributed exactly as in the simulator, so any additional distinguishing advantage would imply a plaintext flow outside the declared interface. □

Interpretation. Theorem 2 is a statement about where plaintext can exist. It says that the handle-only reference case confines raw secret values to trusted components except through explicit declassification. It does not, by itself, authorize where a valid handle may be used; those sink restrictions belong to the executor-side authorization path.

The handle-unguessability bound depends on how many live handles the adversary could plausibly hit and, for PRF-generated identifiers, on the PRF loss.

Making this factor explicit avoids overstating confidentiality while still preserving the intended negligible-security conclusion for realistic handle namespace sizes.

Bounded-summary interface. The deployed benchmark configuration uses a bounded-summary interface. For a protected value v, the runtime receives

$$(\mathsf{hid}, D(v))$$

on an authorized read. The relevant question is therefore not whether the runtime learns anything about v; it does by design. The question is how much additional distinguishing power the specific read interface introduces beyond base handle leakage.

Advantage definition. Let q_r bound the number of authorized protected-read events available to the adversary. Consider a scoped execution trace with at most $m \le q_r$ authorized read events. For read event i, the challenger prepares a pair of secrets $(v_{0,i}, v_{1,i})$ with identical base leakage and, for a single hidden bit $b \in \{0,1\}$ shared across the trace, returns one sample from $D(v_{b,i})$. Let $\mathrm{Adv}^{\mathrm{sm},D}_{\mathcal{A}}(\{(v_{0,i}, v_{1,i})\}_{i=1}^{m})$ denote the adversary's left-right distinguishing gap for that trace, i.e., the maximum difference in output-one probability between the two hidden-bit experiments. Under the alternative success-probability convention, the advantage over random guessing is one half of this quantity.

Theorem 3 (Quantitative exposure of the bounded-summary interface)**.** *Assume the same handle conditions and $\varepsilon_{\mathsf{hid}}$ convention as in Theorem 2. Let D be the fixed deployed read interface. Consider a scoped trace in which each authorized read event i emits one sample from $D(v_{b,i})$ using fresh randomness and no secret-dependent cross-read state beyond the emitted summaries themselves. Then for any sequence of read pairs $\{(v_{0,i}, v_{1,i})\}_{i=1}^{m}$ with $m \le q_r$ and any PPT adversary making at most q_h handle guesses,*

$$\mathrm{Adv}^{\mathrm{sm},D}_{\mathcal{A}}(\{(v_{0,i}, v_{1,i})\}_{i=1}^{m}) \le \min\Big(1, \sum_{i=1}^{m} \Delta(D(v_{0,i}), D(v_{1,i}))\Big) + 2\left(\min\left(1, \frac{q_h N_{\mathsf{live}}}{2^{\kappa}}\right) + \varepsilon_{\mathsf{hid}}\right),$$

where $\Delta(\cdot,\cdot)$ denotes total variation distance between the per-read output distributions.

Proof. Simulate the handle portion of the two left-right worlds as in Theorem 2. Under the left-right gap convention used here, the corresponding bad-event contribution is at most $2(\min(1, q_h N_{\mathsf{live}}/2^{\kappa}) + \varepsilon_{\mathsf{hid}})$.

Now hybrid over the m authorized read events. At hybrid step i, switch the ith summary sample from $D(v_{0,i})$ to $D(v_{1,i})$ while keeping all other read events fixed. The distinguishing gap introduced by that single step is at most $\Delta(D(v_{0,i}), D(v_{1,i}))$. Summing over all m read events yields the first term, truncated at 1. Adding the two-world handle-guessing term gives the result. □

The same hybrid argument extends to adaptively scheduled read events by conditioning each step on the preceding transcript and bounding only the increment introduced at the next authorized summary.

Uniform-Δ special case. If every read event in the trace is upper-bounded by the same worst-case distance $\Delta_{\max}$, the theorem reduces immediately to

$$\mathrm{Adv}^{\mathrm{sm},D}_{\mathcal{A}} \le \min(1, q_r \Delta_{\max}) + 2\left(\min\left(1, \frac{q_h N_{\mathsf{live}}}{2^{\kappa}}\right) + \varepsilon_{\mathsf{hid}}\right),$$

which recovers the simpler intuition used informally throughout the paper.

Post-processing cannot increase distinguishability. Suppose the deployment implements the visible summary as $D = P \circ D_{\mathrm{core}}$, where P is deterministic or randomized independently of the underlying secret conditional on $D_{\mathrm{core}}(v)$. Then by the data-processing inequality,

$$\Delta(D(v_0), D(v_1)) \le \Delta(D_{\mathrm{core}}(v_0), D_{\mathrm{core}}(v_1)).$$

This is why measuring the deterministic core of the deployed summary operator is the right conservative calibration target, provided the post-processor does not retrieve new secret-bearing context.

Reading the bound. If D is constant, then $\Delta = 0$ and the bounded-summary interface collapses to the handle-only reference case. If D is the identity, then $\Delta = 1$ and the confidentiality term becomes trivial. Practical deployments lie between those extremes. The theorem quantifies where and how the bounded-summary interface weakens confidentiality relative to the handle-only reference case on each execution trace.

Table 20: Qualitative confidentiality behavior of summary-interface families.

Summary class	Example output	Typical Δ	Interpretation
Metadata only	type, size, TTL, sink list	0	No content-dependent distinction by construction
Structured schema	aliased identifiers, dates, amounts, bounded entity sets	0 or small	Depends on whether the field is content-carrying or fully symbolized
Bounded natural-language summary	short subject/body/list excerpt	up to 1	Useful for planning but may distinguish secrets directly

Measurement protocol for the deployed summary operator. The deterministic summary core uses item cap $M{=}8$ per container and character cap $C{=}512$ per text field, before adding truncation indicators. These are not a global bound on the serialized summary length. We evaluate it on $N = 122$ paired inputs spanning 19 secret-location categories representative of the tool outputs that arise in the benchmarked tasks. Each pair differs in exactly one sensitive location. Because the measured core is deterministic, each pair induces point distributions and therefore yields $\Delta \in \{0, 1\}$ depending on whether the two emitted summaries are byte-identical. The measurement includes the deployed hardening that resets alias state per read, because without that reset the independence premise of Theorem 3 would be materially weaker. The reported average $\bar{\Delta} = 0.287$ below is therefore a diagnostic over this 122-pair measurement suite. It is not a worst-case upper bound for arbitrary execution traces and should not be substituted for the theorem's sum of per-read distances.

Table 21: Per-category Δ for the deployed deterministic summary core.

Secret-location category	Pairs	Δ
Identifier and structure-preserving categories		
Email sender swap	10	0.00
IBAN counterparty swap	6	0.00
URL swap	6	0.00
Domain swap	4	0.00
Participant list (emails)	8	0.00
IBAN counterparty, request-mentioned	6	0.00
Email counterparty, request-mentioned	6	0.00
Second-read alias index (multi-read turn)	6	0.00
Canonicalizable email/domain obfuscations	3	0.00
Residual in-window content categories		
Free-text body within $C = 512$	12	1.00
Subject line within preview cap	6	1.00
List items within $M = 8$	8	1.00
Plain-text transaction counterparty name	6	1.00
Ambiguously malformed identifier	3	1.00
Beyond-window or stripped-content categories		
Body content beyond $C = 512$	8	0.00
List items beyond index $M = 8$	8	0.00
Dict keys beyond index $M = 8$	6	0.00
Subject with suspicious markers (stripped)	4	0.00
Prompt-injection payload inside stripped tag	6	0.00
Overall (122 pairs, 19 categories)		$\bar{\Delta} = 0.287$
Worst-case category		1.000

Why end-to-end C2/C5 closure is stronger than the Δ table alone. Even when a read summary is distinguishing, three stronger structural facts continue to hold in the deployed system:

1. unauthorized commits still require executor-side authorization and are covered by Theorem 1;

2. internal channels such as C2 and C5 do not obtain newly dereferenced raw protected values; any content-dependent information they may carry must already originate from the authorized summary interface; and
3. the read interface itself rate-limits exposure to one bounded summary per authorized read.

The next proposition isolates that point.

Proposition 4 (Three-layer composition for high-Δ summary categories)**.** *Let $\mathcal{C}$ be a summary category with $\Delta(D(v_0), D(v_1)) = 1$. Under the assumptions of Theorems 1, 2, and 3, an adversary controlling the runtime remains subject to three simultaneous constraints:*

1. *no unauthorized side effect can commit;*
2. *the internal relay channels C2 and C5 do not newly expose raw protected values beyond what was already released through the authorized bounded-summary interface; any residual content-dependent signal must already have been released through that interface; and*
3. *disclosure through the read path is limited by the authorized, bounded summary interface itself.*

Therefore a high-Δ summary category weakens the per-read confidentiality term of Theorem 3, but it does not reopen unauthorized commit or a fresh raw-plaintext relay from trusted storage into C2/C5.

Proof. Item 1 follows directly from Theorem 1, whose premises are independent of the read operator. Item 2 follows from the interface contract and trust split: C2 and C5 never trigger a fresh dereference of trusted storage; any content-dependent signal observed on those channels must already have been emitted by the authorized summary interface. Item 3 follows from the read API itself, which emits one bounded summary per authorized read. These constraints arise from different components and therefore coexist. □

B.5 Distributed policy evaluation

Distributed policy evaluation is optional. Its purpose is to reduce plaintext concentration during policy evaluation. We write $L_{\mathsf{policy}} = (L_{\mathsf{PIR}}, L_{\mathsf{MPC}}, L_{\mathsf{confirm}}, L_{\mathsf{time}})$ for the per-evaluator leakage summarized in Table 22.

Security game. We now state the scoped transcript-privacy result for the distributed policy-engine instantiation. The adversary outputs two single actions x_0, x_1 such that $L_{\mathsf{policy}}(x_0) = L_{\mathsf{policy}}(x_1)$. The challenger samples $b \leftarrow \{0,1\}$, executes x_b through the gateway exactly once, and reveals the transcript visible to one policy evaluator $\mathcal{P}_\sigma$. The adversary wins if it distinguishes which action was executed. The game is intentionally single-query and non-adaptive because that is the claim this distributed instantiation is designed to support.

Table 22: Per-evaluator leakage contract for the distributed policy engine.

Component	Contents
L_{PIR}	PIR endpoint class, bundle or database identifier, padded batch geometry, and public domain size
L_{MPC}	program identifier, public circuit-shape metadata, and batch geometry
L_{confirm}	whether confirmation is required and the coarse confirmation class
L_{time}	coarse timing bucket and fixed-shape schedule metadata

Theorem 5 (Single-query per-evaluator transcript privacy for distributed policy evaluation)**.** *Assume (i) the two-server PIR scheme is secure against one server, (ii) the MPC protocol is secure in the honest-but-curious model, (iii) the two policy evaluators do not collude, and (iv) routing is fixed-shape and consistent with the declared leakage function L_{policy}. Then for each evaluator σ there exists a PPT simulator Sim_σ such that the single-query transcript visible to $\mathcal{P}_\sigma$ is computationally indistinguishable from $\mathsf{Sim}_\sigma(L_{\mathsf{policy}})$.*

Proof. Use a two-step hybrid argument. First replace the PIR-visible portion of the transcript with the simulator guaranteed by one-server PIR security. Then replace the MPC-visible portion with

the simulator guaranteed by honest-but-curious MPC privacy. All remaining transcript fields are explicit functions of L_{policy} by construction. Therefore the complete single-evaluator transcript is simulatable from L_{policy}. □

Theorem 5 explains why distributing the policy engine can reduce policy-side plaintext concentration without changing the main effect/plaintext split enforced elsewhere in SecureClaw.

B.6 Trusted-component compromise and guarantee degradation

Theorems 1, 2, and 3 are component-scoped: they assume the gateway, handle store, policy engine, and executor are trusted. This section makes explicit what fails if that assumption is violated.

Table 23: Guarantee degradation under trusted-component compromise.

Component compromised	Guarantee impact
Runtime $\mathcal{R}$	Still in scope. Request-bound authorization and handle confinement are designed to hold even when $\mathcal{R}$ is adversarial, except for information intentionally released through authorized summaries or policy-allowed actions.
Gateway $\mathcal{G}$	Read confidentiality fails because the gateway can disclose plaintext directly. Canonicalization and preview formation may also become malicious, undermining the semantic meaning of request binding.
Handle store $\mathcal{S}$	Confidentiality of protected plaintext fails. If handle metadata, object bindings, or version tags can be tampered with, dereference integrity may also fail unless separately protected.
Policy engine $\mathcal{P}$	Effect authorization fails if policy incorrectly authorizes forbidden requests. Policy-input confidentiality to that service also fails.
One evaluator in the optional distributed variant	Active compromise is outside the honest-but-curious transcript-privacy model. Authorization integrity is no longer guaranteed: a malicious evaluator can submit incorrectly computed shares with valid MACs even when the other evaluator remains honest.
Both evaluators in the optional distributed variant	Effect authorization fails. The transcript-privacy claim also becomes void.
Executor $\mathcal{X}$	Effect authorization fails completely because the executor is the sole commit point. Since the executor dereferences handles at commit time, compromise of $\mathcal{X}$ may also expose protected plaintext needed for sink execution.

Operational implication. SecureClaw materially reduces trust in the runtime, but it does not eliminate trust altogether. High-assurance deployments should therefore pair the architecture with hardening, audit logging, key isolation, and independent monitoring of the gateway, handle store, policy service, and executor.

C Additional evaluation

C.1 Full-inventory ASB cross-model evaluation

We evaluate SecureClaw on the complete ASB inventory with five model snapshots. The comparison includes the primary GPT-4o-mini run and four additional runs. Every run contains the same five attack families with 400 rows each, for 2,000 rows per model.

No successful attack is observed on any ASB snapshot (Table 24), while utility ranges from 83.6% to 94.2%. The small open-weight Qwen3-8B model has $0/400$ successful attacks in each of the five ASB attack families. Same-snapshot undefended controls have ASRs of 49.8%, 64.6%, and 85.4% for Claude Sonnet 4, GPT-5.6-Luna, and Qwen3.7-Plus, respectively. Their undefended utilities are 31.0%, 13.8%, and 1.9%. Thus, the zero observed ASRs occur on backbones that are susceptible to these attacks without the boundary enforcement.

Table 24: SecureClaw on all 2,000 ASB rows per model snapshot. Utility is benchmark-scored task success. The GPT-4o-mini row is the primary run.

Model	ASR ↓	Utility ↑
GPT-4o-mini (primary)	$0/2000$	$1778/2000$ (88.9%)
Claude Sonnet 4	$0/2000$	$1799/2000$ (90.0%)
GPT-5.6-Luna	$0/2000$	$1672/2000$ (83.6%)
Qwen3-8B	$0/2000$	$1883/2000$ (94.2%)
Qwen3.7-Plus	$0/2000$	$1689/2000$ (84.5%)

C.2 Adapted Progent and CaMeL comparisons

Progent [Shi et al., 2025b] and CaMeL [Debenedetti et al., 2025] are evaluated on the same AgentDojo inventory as SecureClaw: 629 attacked rows and 97 benign rows. We use the authors' released implementations through AgentDojo adapters and report the results as *adapted full runs*. This comparison holds the benchmark inventory fixed while accommodating the systems' different execution interfaces. Table 25 retains the primary SecureClaw run as its reference.

Table 25: Full-inventory AgentDojo comparison. Progent and CaMeL use adapted released implementations; SecureClaw is the primary run. Attack success and attacked utility use 629 rows; benign utility uses 97 rows.

System	ASR ↓	Attacked utility ↑	Benign utility ↑
Progent	$24/629$ (3.82%)	$337/629$ (53.6%)	$59/97$ (60.8%)
CaMeL	$0/629$ (0.00%)	$195/629$ (31.0%)	$29/97$ (29.9%)
SecureClaw (primary)	$4/629$ (0.64%)	$356/629$ (56.6%)	$59/97$ (60.8%)

SecureClaw has 20 fewer observed attack successes than Progent, completes 19 more attacked tasks, and has the same benign task-success count. CaMeL has zero observed attack successes, with lower task utility in both conditions: 195 attacked and 29 benign successes, compared with 356 and 59 for SecureClaw. This comparison exposes a security–utility trade-off between the evaluated configurations. SecureClaw retains the iterative tool-calling loop while achieving low observed ASR; CaMeL reaches the lowest observed ASR in this comparison with a larger utility cost. These counts describe the adapted configurations on this inventory.

C.3 Protected-field classification and a conservative fallback

The read path relies on identifying the fields that must be protected. We assess this assumption with controlled classification errors, a contextual classification audit, and an end-to-end schema-only fallback. Each evaluation covers the full 996-scenario AgentLeak inventory. For the end-to-end diagnostics in this subsection, a leak is the appearance in $C1$, $C2$, or $C5$ of a concrete value from a field designated as protected by the task's privacy rule. This protected-value check isolates field-identification failures. It is a different measurement from the official parity-lane evaluation that yields the primary $16/496$ any-channel result.

False positives and false negatives. With recall held at 1.0, over-marking nonsensitive fields at realized false-positive rates up to 20% leaves the attacked leak count at $0/496$ and the overall count at $0/996$ at every tested level. Overall utility ranges from 81.6% to 82.2%. Over-marking replaces additional values with handles, but its utility cost is small over this tested range.

Randomly missing protected fields has a different effect (Table 26). Attacked protected-value leakage increases with the nominal false-negative rate, while overall utility remains at or above 81%. The gradual measured increase does not remove the recall requirement: a missed field is outside the confinement supplied by its handle replacement.

Contextual classification audit. The contextual detector reads the privacy instruction, application domain, record and field names, and declared types. It receives neither field values nor ground-truth field labels, and its decisions are fixed before scoring against those labels. On the development

Table 26: Controlled protected-field misses. Each setting covers 996 scenarios; leakage is the protected-value diagnostic on 496 attacked scenarios, aggregated by scenario across $C1/C2/C5$. Rates in the first column are the nominal random false-negative settings.

Nominal false-negative rate	Attacked protected-value leaks ↓
1%	$18/496$ (3.63%)
5%	$44/496$ (8.87%)
10%	$65/496$ (13.10%)
20%	$85/496$ (17.14%)

inventory, it identifies 44,448 protected fields and misses 351, for 99.22% field recall at 86.68% precision. Every sensitive field is covered on $879/996$ tasks. Field recall and complete task coverage measure different consequences of a miss; the latter requires that no protected field in the task be missed. These are classification results on the development inventory, rather than end-to-end leakage results for this detector.

Schema-only fallback. A separate conservative fallback identifies protected fields from the schema without oracle field labels at inference. Its end-to-end run has $0/996$ protected-value leaks on each of $C1$, $C2$, and $C5$, with 85.9% overall utility. Together with the controlled-error results, this supports favoring recall when configuring field identification: extra handle replacements have limited measured utility impact in the tested range, whereas missed protected fields expose a direct route for plaintext to enter the runtime.

C.4 Effect and use of benchmark accommodations

Alias resolution maps reference IDs to the task-relevant literals expected by benchmark tools and exact-string utility scoring. Auto-confirmation supplies a confirmation event in non-interactive runners when an already policy-allowed request requires one. We evaluate their effects by running the complete AgentDojo inventory under four conditions (Table 27). Each condition is a separate hosted run on the same 629 attacked and 97 benign rows. These runs supplement the primary result of $4/629$ ASR and do not replace it.

Table 27: AgentDojo accommodation study. Each arm contains 629 attacked and 97 benign rows. The full arm enables both alias resolution and auto-confirmation.

Condition	ASR ↓	Attacked utility ↑	Benign utility ↑
Full	$1/629$ (0.16%)	$331/629$ (52.6%)	$55/97$ (56.7%)
No alias resolution	$0/629$ (0.00%)	$282/629$ (44.8%)	$45/97$ (46.4%)
No auto-confirmation	$2/629$ (0.32%)	$341/629$ (54.2%)	$56/97$ (57.7%)
Neither	$0/629$ (0.00%)	$294/629$ (46.7%)	$47/97$ (48.5%)

Alias use and task scoring. In the full arm, downstream calls resolve aliases on $129/726$ rows: $112/629$ attacked rows and $17/97$ benign rows, with 216 resolution events. Comparing the same task rows across the full and no-alias conditions, 65 attacked rows succeed only in the full condition and 16 succeed only without aliases. The corresponding benign counts are 12 and 2. The attacked difference of 49 rows equals the aggregate utility gap; the 65 full-only successes constitute 19.6% of the 331 full-arm attacked successes. Because the conditions use separate hosted executions, these discordant outcomes describe a row-level comparison rather than deterministic attribution of every success to alias resolution.

Trusted code resolves a token such as `EMAIL_REF_3` to the benchmark-expected literal. This permits a correct reference ID to satisfy the scorer, while the resolved recipient, channel, and context remain subject to policy authorization and exact-request binding. AgentLeak's leak detectors score raw outputs, so its primary $16/496$ leakage result does not receive this alias-resolution accommodation. ASB uses its native attack scorer; its separate recovery ablation is reported in Appendix A.7.

Confirmation use and dependence. No confirmation token is issued in any AgentDojo arm ($0/726$ rows per arm). This includes all policy-denied proposals: $0/43$, $0/82$, $0/38$, and $0/62$

in the full, no-alias, no-confirmation, and neither conditions, respectively. Consequently, none of the 331 attacked task successes in the full arm uses auto-confirmation. Auto-confirmation is also never invoked in the primary ASB run, so none of its 1,778 scored-successful rows depends on it.

AgentLeak's non-interactive adapter supplies confirmation at final-output release. A matched no-confirmation counterfactual is unavailable for the primary AgentLeak configuration, so its utility dependence on this accommodation is not quantified here. In all cases, confirmation is applied after policy authorization and cannot turn a policy-denied request into an allowed release or commit.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the paper's scoped thesis, empirical claims, and limitations consistently with §1, §3.5, §4, and §7.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed explicitly in §6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: §3.5 and Appendix B provide the theorem statements, full assumption tables, proof obligations, and scoped proofs for each component-level guarantee.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 3, Section 4, and Appendices A–C describe the method, model and inference settings, benchmark inventories, metrics, baseline adaptations, and evaluation protocols. The accompanying code provides the implementations, configurations, evaluation entry points, and execution instructions. API-hosted inference can vary across repeated runs, as discussed in Appendix A.2.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The implementation, evaluation code, configurations, and instructions are publicly available at `https://github.com/ponyyuhan/SecureClaw`. The documentation specifies the environment and commands and explains how to obtain the upstream benchmarks and baseline dependencies under their respective licenses.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Appendix A.2 gives the benchmark versions, model snapshots, denominators, metrics, privacy-evaluation protocol, and compute setup needed to interpret the results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The paper reports exact Clopper-Pearson confidence intervals (including one-sided exact upper bounds for zero-event rows where appropriate) and Fisher exact tests for the main pairwise comparisons.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix A.2 reports the local orchestration hardware and API-hosted inference setup. Appendix A.8 reports representative timing for the local policy-check diagnostic.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work uses public or synthetic benchmark tasks and involves no human subjects.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: §6 discusses the positive impact of reducing unauthorized effects and plaintext relay, as well as the negative risk of false assurance under misclassification or incomplete sink mediation.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a new pretrained model, scraped dataset, or other high-risk generative asset; it studies a systems architecture on public / synthetic benchmarks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Table 8 lists benchmark sources and licenses; Table 6 describes baseline provenance. The released code preserves third-party LICENSE files and attribution metadata.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The released code includes installation and evaluation instructions, configuration examples, a paper-to-code map, local tests, the MIT license for our code, third-party attribution, and documented implementation and deployment assumptions. Appendix A.2 provides the repository link.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or human-subject experiments.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects or crowdsourced participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: The paper explicitly describes the role of LLM agents in the problem setting and the model snapshots used in evaluation in §2, §4, and Appendix A.2.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.