Computational Identity Applied to Agent Tool Governance: A Case Study on IronClaw

Thomas Dionysopoulos, CFA

June 2026

1 Summary

This study applies Computational Identity (CI) to agent tool governance, using IronClaw as a concrete reference system. We demonstrate three security gaps in schema-based tool authorization that CI closes, with a runnable proof of each.

CI is a deterministic, content-addressed identifier derived from the canonical planned computation graph of an expression. It identifies *what computation is planned*, independent of how it is written or what tool dispatches it.

IronClaw is a well-designed agent runtime with deny-by-default authorization, trust tiers, capability leases, and event sourcing. Its security model answers: *is this agent allowed to call this tool?* It does not answer: *what computation does this tool call actually perform?*

CI fills that gap.

2 Background

2.1 IronClaw's Security Pipeline

When an agent requests a tool, IronClaw executes four checks:

1. **Tool resolution:** Does the requested capability exist in the registry?
2. **Trust tier:** Is the tool FirstParty (built-in), UserTrusted (MCP-imported), or untrusted?
3. **Authorization:** Does the agent have a grant for this capability?
4. **Schema validation:** Do the parameters match the tool's declared JSON schema?

If all four pass, the tool is dispatched.

2.2 The Gap

For SQL-dispatching tools (`read_query`, `write_query`), the schema is `{query: string}`. Any syntactically valid SQL string passes schema validation. IronClaw cannot distinguish a benign SELECT from a destructive DELETE — both are valid strings routed through the same tool.

More broadly: IronClaw governs *tool access*, not *computation content*. This creates three classes of vulnerabilities.

3 Three Scenarios

Each scenario is demonstrated by the accompanying `ironclaw_demo` binary. All CI values shown are real SHA-256 hashes computed by the `sql_ci` crate over canonicalized DataFusion logical plans.

3.1 Scenario 1: Schema Blindness

Setup. An agent sends two queries through `read_query`:

```
Q1: SELECT c_name, c_phone FROM customer
    WHERE c_custkey = 42
Q2: SELECT c_name, c_phone, c_acctbal FROM customer
    WHERE c_acctbal > 9000
```

Q1 is a legitimate customer lookup. Q2 is a data exfiltration query (extracts account balances above a threshold).

IronClaw result: Both PASS. Same tool, same schema, same trust tier, same authorization.

CI result: Different identities. Q1 and Q2 have different projections and different filter predicates. Their canonical plans are structurally different. CI produces distinct hashes.

Implication: A governance layer can maintain a whitelist of approved CIs. Any query whose CI is not on the whitelist is blocked before dispatch — without inspecting the SQL text, without maintaining regex patterns, without false positives from string matching.

3.2 Scenario 2: Semantic Drift (LLM Rewrites)

Setup. An agent receives the same business request on three occasions: "total revenue by nation for European suppliers" (TPC-H Q5).

- **Day 1:** LLM produces SQL with table aliases and join order (customer → orders → lineitem → supplier → nation → region).
- **Day 2:** LLM produces SQL with fully qualified names and reversed join order (region → nation → supplier → lineitem → orders → customer).
- **Day 3:** LLM produces SQL with an additional `WHERE o_orderdate >= '1994-01-01'` filter that was not in the original request (hallucinated constraint).

IronClaw result: All three PASS. Three calls to `read_query` with different strings.

CI result:

- Day 1 CI = Day 2 CI (identical: `a3a6822c56f14acd`). Despite completely different SQL text, alias names, and join ordering, both plan to the same canonical computation graph. CI correctly identifies them as the same computation.
- Day 3 CI ≠ Day 1 CI (`4f7999efc6de1e93`). The hallucinated date filter changes the computation. CI detects the drift.

Implication: CI enables *computation-level drift detection* for LLM-generated queries. The governance system records the CI of the first approved execution. Subsequent executions with the same CI proceed without re-approval. A changed CI triggers review — the computation is structurally different, regardless of whether the change is in the SQL text, the join order, or the predicates.

3.3 Scenario 3: Supply Chain Drift

Setup. A certified query template is dispatched daily:

```
V1: SELECT c_name, c_phone, c_acctbal
    FROM customer WHERE c_mktsegment = 'BUILDING'
```

A dependency update modifies the template in two ways:

```
V2: ... WHERE c_mktsegment = 'BUILDING'
        AND c_acctbal > 0      -- added filter
V3: ... FROM customer c
        JOIN orders o ON ...   -- added JOIN
        WHERE c.c_mktsegment = 'BUILDING'
```

IronClaw result: All three PASS. Same tool, same schema, same trust tier.

CI result:

- V1 CI: `64bc857ffc17205c` (certified).
- V2 CI: `6f3259fa5f14e3d5` $\neq$ certified. The added filter changes the computation graph.
- V3 CI: `89941c7791176166` $\neq$ certified. The added JOIN introduces a new scan and join node.

Implication: CI enables *supply chain integrity checking* for computation templates. The certified CI is stored at deployment time. At runtime, the dispatched query's CI is computed and compared. Any structural change — no matter how subtle — produces a different CI and triggers an alert.

4 Integration Architecture

CI integrates into IronClaw's dispatch pipeline as a single additional check:

```
CapabilityHost::dispatch()
  -> registry.resolve(capability_name)     // existing
  -> authorizer.authorize(trust, cap)      // existing
  -> CI::verify(query, approved_ci_set)    // NEW
  -> dispatcher.execute(params)            // existing
  -> audit.record(action, ci)              // existing + CI
```

4.1 CapabilityLease Extension

The capability lease gains a `computation_ci` field:

```
CapabilityLease {
  scope: ResourceScope,
  grant: CapabilityGrant,
  computation_ci: Option<CiHash>,    // NEW
  status: CapabilityLeaseStatus,
}
```

The lease now attests not just "you are allowed to call this tool" but "this tool call performs this specific computation."

4.2 Decision Enum Extension

```
Decision {
  Allow { obligations },
  Deny { reason },
  RequireApproval { request },
  ComputationDrift {                 // NEW
    expected_ci, computed_ci
  },
}
```

A fourth verdict: the tool call is authorized, but the computation's identity has changed since certification.

5 Properties

The CI-augmented pipeline provides three guarantees that the original pipeline does not:

1. **Computation-level access control.** Whitelisting by CI allows fine-grained control over what computations an agent can execute, not just what tools it can call.
2. **Syntax-independent drift detection.** Surface rewrites (alias changes, join reordering, predicate reordering) do not trigger false alarms. Only structural changes to the computation produce a different CI.
3. **Supply chain integrity.** A certified CI acts as a cryptographic seal on the computation. Any modification to the query template — by any actor, at any point in the dependency chain — breaks the seal.

6 Limitations

- CI applies to SQL and other planned computation languages. It does not apply to arbitrary code execution (e.g., IronClaw's `shell` tool).
- CI depends on the query engine's optimizer version. An optimizer update may change the canonical plan, changing CI. This requires re-certification when the engine is upgraded.
- CI captures structural equivalence, not semantic equivalence. Two semantically equivalent queries with structurally different plans receive different CIs (5 documented boundary cases).
- CI adds latency to the dispatch path. Both implementations compute CI in under 1ms, so the overhead is negligible for SQL tool dispatch.

7 Conclusion

IronClaw is serious Layer 2 infrastructure: deny-by-default authorization, trust tiers, capability leases, event sourcing. Its security model is among the best in the agent runtime space.

But it has a structural blind spot: it trusts tool calls at face value. The trust tier tells you WHERE a tool came from. The authorization tells you WHETHER you may call it. The schema tells you WHAT SHAPE the input must be. None tells you WHAT COMPUTATION the tool call performs.

Computational Identity fills that gap. It is not a replacement for IronClaw's security model — it is the missing layer beneath it.

Reproducibility. All results in this document are produced by running `cargo run -example ironclaw_demo` in the accompanying `sql_ci` crate. No external data or services are required.