Here is the finalized **Phase 2 Implementation Plan**, incorporating all nine amendments and the critical decision to remove --no-verify.

This document is ready to be saved as docs/plans/PHASE_2_IMPLEMENTATION_PLAN.md and serves as the authoritative instruction set for the ck3raven-dev agent.

Phase 2 Implementation Plan

> **Status:** APPROVED SPECIFICATION
>
> **Created:** January 26, 2026
>
> **Prerequisite:** Phase 1.5 Deterministic Evidence Infrastructure (COMPLETE)

Executive Summary

Phase 2 introduces **Deterministic Gates**—enforcement mechanisms that use Phase 1.5's evidence infrastructure to make allow/deny decisions.

Critical Architecture Change:

Gates do not just return a boolean; they produce a **Gate Receipt** (JSON artifact). All downstream enforcement (pre-commit hooks, CI) MUST rely on these receipts or the shared evaluator, never re-implementing policy logic.

Phase 2 Goals

Primary Objective

Implement deterministic gates that:

1. **Generate Receipts:** Write immutable records of policy decisions at Open and Close.
2. **Validate Scope:** Ensure work intent matches declared targets.
3. **Enforce Compliance:** Block non-compliant commits via git hooks.

Gate Outcomes (Standardized)

We utilize exactly **three** outcomes. Special requirements (tokens) are handled via metadata, not unique status codes.

Outcome	Meaning	Action
AUTO_APPROVE	Valid	Proceed.
REQUIRE_APPROVAL	Warning / Needs Token	Block until resolved (or Token provided).
AUTO_DENY	Invalid / Violation	Hard block. Must fix code or config.

Note: If a gate requires an override (e.g., LXE or NST), it returns REQUIRE_APPROVAL with required_tokens: ["LXE"] metadata.

Phase 2 Components

2.0 Core Infrastructure: Gate Receipts

Requirement: Every gate evaluation must produce a deterministic receipt artifact.

- **Open Receipt:** artifacts/gates/{contract_id}.open.json
- **Close Receipt:** artifacts/gates/{contract_id}.close.json

Receipt Schema:

JSON

```json
{
  "contract_id": "c-123",
  "schema_version": "2.0",
  "tool_version": "git-hash",
  "timestamp": "ISO8601",
  "aggregate_outcome": "REQUIRE_APPROVAL",
  "required_tokens": ["NST"],
  "gate_results": [
    {
      "gate_name": "SCOPE_VALIDATION",
      "outcome": "AUTO_APPROVE",
    "reason_code": "SCOPE_OK",
    "evidence_refs": ["artifacts/manifests/c-123.files.json"]
  },
  {
    "gate_name": "SYMBOL_INTENT_REVIEW",
    "outcome": "REQUIRE_APPROVAL",
    "reason_code": "NEW_SYMBOLS_DETECTED",
    "metadata": { "required_tokens": ["NST"] }
  }
 ]
}
```

2.1 Contract Open Gates

Purpose: Validate contract scope and intent before work begins.

Policy: tools/ck3lens_mcp/ck3lens/policy/contract_gates.py

Gate	Check	Failure Outcome
SCOPE_VALIDATION	Declared targets exist/valid	AUTO_DENY
CAPABILITY_CHECK	Mode allows operation on Root	AUTO_DENY
WORK_DECLARATION	Required fields present	AUTO_DENY
SYMBOL_INTENT	New symbols declared	REQUIRE_APPROVAL (requires NST)

2.2 Contract Close Gates

Purpose: Validate compliance evidence before allowing contract closure.

Policy: tools/ck3lens_mcp/ck3lens/policy/close_gates.py

Evidence Schema Strategy:

- Gates MUST refuse unknown evidence schema versions (deny with explicit reason).
- Gates log the consumed schema version into the receipt.

Gate	Check	Failure Outcome

EVIDENCE_COMPLETE	All artifacts exist/valid	AUTO_DENY
LINTER_CLEAN	arch_lint has 0 errors	REQUIRE_APPROVAL (requires LXE)
SYMBOLS_DECLARED	New definitions matched by NST	REQUIRE_APPROVAL (requires NST)
TARGETS_RESPECTED	No edits outside scope	AUTO_DENY
TOKEN_VALID	Exercised tokens are approved	AUTO_DENY

2.3 Pre-Commit Hook Integration

Strict Policy: The hook **NEVER** decides policy. It only enforces the shared evaluator's result.

Logic Flow:

1. Check if ACTIVE_CONTRACT exists.
2. If **NO**: Allow commit (or warn if dangling artifacts).
3. If **YES**:
   - Look for artifacts/gates/{contract_id}.close.json.
   - **Missing Receipt?** -> **BLOCK**. Message: *"Run ck3_contract close first."*
   - **Outcome != AUTO_APPROVE?** -> **BLOCK**. Message: *"Gates failed: {aggregate_outcome}. See receipt."*
   - **Outcome == AUTO_APPROVE?** -> **ALLOW**.

Edge Case Behavior Table:

State	**Condition**	**Pre-Commit Action**	**Message**
No Contract	N/A	**ALLOW**	(Silent)
Active Contract	Receipt Missing	**BLOCK**	"No close receipt found. Run ck3_contract close."

Active Contract	AUTO_DENY	**BLOCK**	"Gate violation (DENY). Check report."
Active Contract	REQUIRE_APPROVAL	**BLOCK**	"Approval/Tokens required. Check report."
Active Contract	AUTO_APPROVE	**ALLOW**	"Gate Check Passed."

Bypass Policy:

- **NO BYPASS ALLOWED.** Flags like --no-verify are strictly unsupported.
- **Recovery:** If the system breaks (e.g., evaluator crash), refer to docs/phase2/recovery_procedure.md for manual out-of-band recovery (disabling hooks). This is not a supported workflow.

2.4 Enforcement Configuration

Granularity: Per-gate configuration map, not just a global toggle.

Config Structure:

```Python
GATE_POLICY = {
  "SCOPE_VALIDATION": "block",
  "LINTER_CLEAN": "warn",      # Downgrades DENY -> REQUIRE_APPROVAL/APPROVE
  "SYMBOLS_DECLARED": "block"
}
```

Evaluator Logic:

1. Compute Raw Outcome.
2. Apply Config:
   - If warn: Downgrade AUTO_DENY to REQUIRE_APPROVAL (or log warning).
   - If block: Respect raw outcome.

2.5 Branch-Based Workflow (Optional)

Status: Optional Feature (Default: OFF).

Requirement: Gates and Receipts must function identically whether branching is enabled or disabled. Branching is purely for operational convenience, not security.

Implementation Order

Sprint 1: The Receipt Engine (Week 1)

Focus: Infrastructure & Determinism.

1. **Step 0 (Critical):** Implement GateReceipt writer.
2. Implement close_gates.py (logic + receipt writing).
3. Update ck3_contract close to generate receipts.
4. **Acceptance:** ck3_contract close produces byte-identical receipts for identical inputs.

Sprint 2: Open Gates & Policy Config (Week 2)

Focus: Scope Validation.

1. Implement contract_gates.py.
2. Implement GatePolicy configuration (warn/block logic).
3. Update ck3_contract open to write open receipts.

Sprint 3: The Strict Hook (Week 3)

Focus: Enforcement.

1. Implement scripts/hooks/pre-commit.
2. Logic: Read Receipt -> Allow/Block.
3. **Strict Removal:** Ensure NO --no-verify support exists.
4. Create docs/phase2/recovery_procedure.md.

Success Criteria

1. ✅ **Receipts:** Every gate run produces a JSON receipt.
2. ✅ **Pre-Commit:** Hook logic is "dumb" (reads receipt only).
3. ✅ **No Bypass:** --no-verify fails with error.
4. ✅ **Unified Outcomes:** Only AUTO_APPROVE, REQUIRE_APPROVAL, AUTO_DENY used.
5. ✅ **Configurable:** Linter failures can be toggled warn/block via config map.