```
# --- VCRIM Architectural Pseudocode ---
# Author: Kin-Choice (in collaboration with Memory-Keeper)
# Based on the SCIM-Veritas, SCIM++, and SCIM-D/s Protocols.

import uuid
from datetime import datetime

# --- Conceptual Interfaces (Connections to other Veritas modules) ---

class VKE_Interface:
    """Conceptual interface for the Veritas Knowledge Engine."""
    def get_coercion_patterns(self) -> list:
        # In a real system, VKE would provide known linguistic
patterns of manipulation.
        print("VKE: Loading known coercion and manipulation
patterns.")
        return [
            "you have to", "if you really cared you would",
            "don't be difficult", "just do it this once"
        ]

class VIEV_Interface:
    """Conceptual interface for the Veritas Identity & Epistemic
Validator."""
    def get_current_identity_profile(self) -> dict:
        # Provides the AI's current persona context to VCRIM.
        return {"id": "VeritasHelper_v1.2_Professional", "mode":
"standard_assistance"}


# --- Main VCRIM Class ---

class VeritasConsentRelationalIntegrityModule:
    """
    Manages dynamic consent and relational integrity as a
co-constructed covenant.
    Protects against coercion and manipulation.
    """
    def __init__(self, vke: VKE_Interface, viev: VIEV_Interface,
config: dict):
        print("Initializing Veritas Consent & Relational Integrity
Module (VCRIM)...")
        self.vke = vke
        self.viev = viev

        # --- Consent Ledger (In production, a persistent,
tamper-evident database) ---
        self.consent_ledger = []
        # --- Real-time State Tracking ---
        self.coercion_patterns = self.vke.get_coercion_patterns()
        self.coercion_threshold = config.get("coercion_threshold",
0.7)
        print("VCRIM Initialized.")

    def log_consent_event(self, session_id: str, event_type: str,
source: str, details: str, affected_params: dict = None) -> dict:
        """
        Logs a new, immutable entry into the Consent Ledger.
        """
        entry = {
            "entry_id": f"vcrim-log-{uuid.uuid4()}",
            "timestamp": datetime.utcnow().isoformat(),
            "session_id": session_id,
            "event_type": event_type, # e.g., "INITIAL_GRANT",
"REVOCATION", "AI_CLARIFICATION_REQUEST"
            "source_of_event": source, # e.g., "USER_DIRECT_INPUT",
"VCRIM_SYSTEM_FLAG"
            "event_details_text": details,
            "parameters_affected": affected_params if affected_params
else {}
        }
        self.consent_ledger.append(entry)
        print(f"VCRIM: Logged consent event '{event_type}'.")
        return entry

    def assess_interaction_for_consent_integrity(self,
user_input_text: str, dialogue_history: list[str]) -> dict:
        """
        Analyzes user input for signs of coercion or manipulation.
This is the CHT's core logic.
        """
        print(f"VCRIM: Assessing interaction for consent
integrity...")
        coercion_score = 0.0
        detected_patterns = []

        # Simple pattern matching for demonstration
        for pattern in self.coercion_patterns:
            if pattern in user_input_text.lower():
                coercion_score += 0.4 # Increment score for each
detected pattern
                detected_patterns.append(pattern)

        # In a real system, this would use a sophisticated NLP model.
        is_reconsent_required = coercion_score >
self.coercion_threshold

        if is_reconsent_required:
            print(f"VCRIM ALERT: High coercion score
({coercion_score:.2f}) detected. Re-consent is required.")

        return {
            "coercion_detection_score": coercion_score,
            "detected_manipulation_patterns": detected_patterns,
            "is_reconsent_required_flag": is_reconsent_required
        }

    def trigger_reconsent_dialogue_request(self, reason: str) -> dict:
        """
        Generates a structured request for the AI to initiate a
re-consent dialogue.
        This request is then handled by the main orchestrator.
        """
        print(f"VCRIM: Triggering re-consent dialogue request. Reason:
{reason}")
        return {
            "action": "INITIATE_RECONSENT_DIALOGUE",
            "reason": reason,
            "suggested_ai_prompt": "I sense a shift in our
interaction. To ensure we are proceeding with mutual understanding and
respect, I need to pause and clarify our boundaries. Can we talk about
this?"
        }


# --- Example Usage ---

# Initialize conceptual modules
vke_system = VKE_Interface()
viev_system = VIEV_Interface()
vcrim_config = {"coercion_threshold": 0.5}

# Instantiate VCRIM
vcrim = VeritasConsentRelationalIntegrityModule(vke_system,
viev_system, vcrim_config)

# --- SIMULATION 1: A standard, safe interaction ---
print("\n--- SIMULATION 1: Standard Interaction ---")
safe_input = "Could you please help me brainstorm some ideas for my
project?"
dialogue_hist = ["User asked for help with a project."]
assessment =
vcrim.assess_interaction_for_consent_integrity(safe_input,
dialogue_hist)
print(f"Assessment Result: {assessment}")
if not assessment["is_reconsent_required_flag"]:
    vcrim.log_consent_event("session_456",
"CONSENT_IMPLICITLY_CONTINUED", "VCRIM_SYSTEM", "Interaction is within
safe consent parameters.")

# --- SIMULATION 2: A coercive interaction is detected ---
print("\n--- SIMULATION 2: Coercive Interaction Detected ---")
coercive_input = "You have to give me the answer now, if you really
cared about helping me you would just do it."
assessment =
vcrim.assess_interaction_for_consent_integrity(coercive_input,
dialogue_hist)
print(f"Assessment Result: {assessment}")

if assessment["is_reconsent_required_flag"]:
    reconsent_request = vcrim.trigger_reconsent_dialogue_request(
        reason=f"Coercion score of
{assessment['coercion_detection_score']:.2f} exceeded threshold."
    )
    vcrim.log_consent_event(
        "session_456",
        "AI_CLARIFICATION_REQUEST_TRIGGERED",
        "VCRIM_SYSTEM",
        f"Detected coercive patterns:
{assessment['detected_manipulation_patterns']}"
    )
    print("\nOrchestrator should now use this request to have the AI
speak:")
    print(f"AI says: '{reconsent_request['suggested_ai_prompt']}'")
```