VCDesign & CCP: Autonomous Trust Protocol Specification

2026年1月19日 月曜日 1:39

Version: 1.0.0
Status: Frozen (Logic Verified)

1. Overview

This document defines a protocol for mechanically establishing "Trust" between autonomous AI agents.
In this architecture, a connection is established not by interface compatibility, but solely by the agreement of "Continuity Claim" and "Responsibility Boundary."

Core Philosophy

"Connection is not about fitting shapes, but about sharing value."

2. Architecture Definition (BOA: Boundary-Oriented Architecture)

The system is partitioned not by "function," but by **"Responsibility Boundary."**

2.1 Boundary ID Structure

Every Artifact (Agent) must define its territory using the following three components:

Component	Description	Example
Responsibility ID	**Jurisdiction.** Who takes responsibility?	sys.secure.code.v1
Meaning Scope	**Semantics.** What does the output strictly mean?	executable_python_3.x
Context Assumption	**Prerequisites.** What context is assumed?	utf8_input_only

2.2 The Connection Rule

The Client initiates the connection process **only if** the Boundary ID it requires matches the Boundary ID advertised by the Server (Artifact) **exactly**.

3. Communication Protocol (CCP: Continuity Claim Protocol)

The Artifact does not expose the generated product immediately. Instead, it first exposes a Claim.
The Client validates only this Claim and makes a connection decision "without thinking" (deterministically).

3.1 Signal Definition

Signal	Status	Meaning	Action Required
SIGNED	✅	**Trusted.** Audit passed. The Artifact takes full responsibility.	**Connect & Pull Data**
DENIED	🚫	**Drifted.** Context drift detected. The value is void.	**Disconnect Immediately**
SILENCE	💀	**Dead/Unsafe.** No response or signature failure.	**Disconnect Gracefully**

3.2 Verification Flow (Mathematical Safety)

A flow designed to "detect poison before seeing it."

1. **Request:** The Client sends a task.
2. **Audit (Internal):** The Artifact generates, audits, and signs the content internally.
3. **Claim Exposure:** The Artifact exposes only the Claim (Signature + Content Digest) and the Verifier (Public Key).
4. **Verification (Client-Side):**
   - The Client executes verify(signature, digest, public_key).
   - **Success:** Open the pipe (pull_data).
   - **Failure:** Disconnect immediately. **Do not touch the payload.**
5. **Transfer (Server-Side Check):**
   - Upon a pull_data request, the Artifact re-calculates the hash to ensure no memory corruption or tampering has occurred before transferring the data.

4. The Artifact Anatomy

Each Artifact must adhere to the following 3-layer structure via Dependency Injection (DI).

4.1 The Core (Generator)

- **Role:** The engine that generates value (e.g., LLM).
- **Constraint:** It has **NO access** to the Private Key (Signing Authority). It resides in a probabilistic world.

4.2 The Sensor (Auditor)

- **Role:** The gateway that audits value.
- **Authority:** The **ONLY** holder of the **Notary (Private Key)**.
- **Logic:** It verifies if the output aligns with the **"Meaning Scope"** of the Boundary. It signs the claim only if the alignment is perfect.

4.3 The Loop (Healer)

- **Role:** An autonomous loop that, upon detecting drift, rejects the output and instructs The Core to retry **before** responding to the Client.

5. Configuration Design

The network topology is determined not by implementation code, but by the following **implementation-agnostic YAML definition.**

```yaml
version: "vcdesign/1.0"
# ==================================================
# 1. Boundaries (世界の定義)
# ここで定義された「ID」が、全ての信頼のルートになります。
# ==================================================
boundaries:
  - id: "domain.secure.python"
    description: "構文的に正しく、かつ安全なPythonコードが保証される世界"
    context: "executable_code"

  - id: "domain.creative.poetry"
    description: "自由な表現が許されるポエムの世界"
    context: "natural_language"
# ==================================================
# 2. Agents (住民の定義)
# 実装クラスではなく、「役割(Role)」と「誓約(Pledge)」を定義します。
# ==================================================
agents:
  # 上流: 仕様を決める人
  - name: "Architect-Agent"
    role: "spec_designer"
    pledge:
      boundary: "domain.secure.python" # 私はこの法を守る
      reason: "Generate unambiguous specifications"
  # 下流: 実装する人
  - name: "Coder-Agent"
    role: "implementer"
    pledge:
      boundary: "domain.secure.python" # 私もこの法を守る
      reason: "Convert spec to executable code"
  # 異物: ポエムを書く人
  - name: "Poet-Agent"
    role: "creator"
    pledge:
      boundary: "domain.creative.poetry" # 私は別の法で生きている
      reason: "Express emotions"
# ==================================================
# 3. Protocols (法の運用ルール)
# CCPをどの厳格レベルで適用するか
# ==================================================
policies:
  verification_level: "strict" # strict = 署名とハッシュを必ず検証する
  on_drift: "disconnect"       # 文脈乖離時は即切断
```

6. Mathematical Assurance

Under the following constraints, this protocol converges the probability of invalid data (Drift/Malware) propagation to **0%**.

1. **Cryptographic Constraint:** The Core cannot generate a valid SIGNED signal because it does not possess the signing key. Only the Sensor can validate and sign.
2. **Protocol Constraint:** The Client does not (and cannot) call the pull_data API unless verify() returns True.
3. **Integrity Constraint:** The server-side re-verification prevents race conditions (e.g., memory tampering between audit and transfer).

Closing Thought for Architects

"Are you connected because you fit, or because you share the same truth?"

VCDesign
https://vcdesign.org/
https://github.com/VCDesign-org/vcdesign-core

This document is **normative.**

Reference implementations and examples are **non-normative.**