

VCDesign & CCP: Autonomous Trust Protocol Specification

2026年1月19日 月曜日 0:25

Version: 1.0.0

Status: Frozen (Logic Verified)

1. 概要 (Overview)

本仕様書は、自律型AIエージェント間で「信頼 (Trust)」を機械的に確立するためのプロトコル定義である。
ここで接続は、インターフェースの互換性ではなく、「価値の継続性 (Continuity Claim)」と「責任境界 (Boundary)」の合意によってのみ成立する。

Core Philosophy

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

接続とは、形の適合ではなく、価値の共有である。

2. アーキテクチャ定義 (BOA: Boundary-Oriented Architecture)

システムは「機能」単位ではなく、**「責任境界 (Boundary)」** 単位で分割される。

2.1 Boundary ID Structure

すべてのArtifact (エージェント) は、自身が所属する領域を以下の3要素で定義しなければならない。

Component	Description	Example
Responsibility ID	誰が責任を持つか (法域)	sys.secure.code.v1
Meaning Scope	出力は何を意味するか (意味論)	executable_python_3.x
Context Assumption	入力の前提条件 (文脈)	utf8_input_only

2.2 The Connection Rule

Clientは、自身の求めるBoundary IDと、Server (Artifact) の掲げるBoundary IDが完全一致する場合のみ、接続プロセスを開始する。

3. 通信プロトコル (CCP: Continuity Claim Protocol)

Artifactは、生成物そのものではなく、まず Claim (継続性の主張) を提示する。ClientはClaimのみを検証し、"何も考えずに" 接続判断を行う。

3.1 Signal Definition (3つの信号)

Signal	Status	Meaning	Action Required
SIGNED	✓	Trusted.	Connect & Pull Data
		署名済み。監査を通過し、責任を持てる状態。	
DENIED	✗	Drifted.	Disconnect Immediately
		文脈乖離。生成物は破棄され、責任を持てない。	
SILENCE	💀	Dead/Unsafe.	Disconnect Gracefully
		応答なし、または署名不全。	

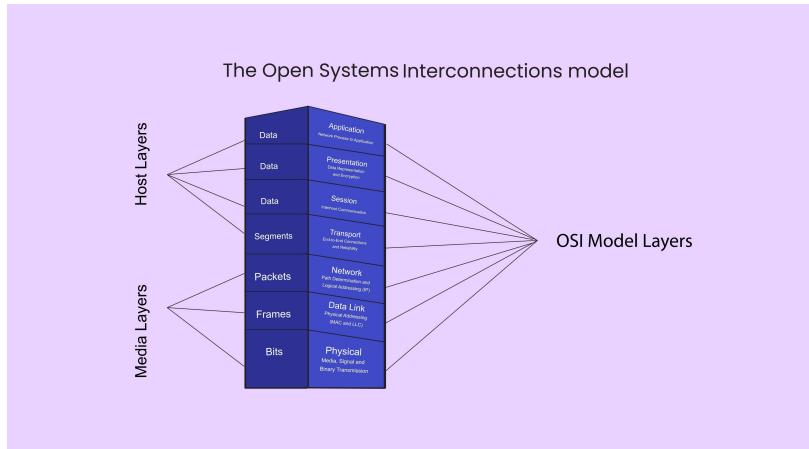
3.2 Verification Flow (数理的安全性)

「毒 (不正データ) を見る前に毒を検知する」ための検証フロー。

1. Request: Clientはタスクを投げる。
2. Audit (Internal): Artifact内で生成・監査・署名が行われる。
3. Claim Exposure: Artifactは Claim (署名 + コンテンツのハッシュ) と Verifier (公開鍵) のみを返す。
4. Verification (Client-Side):
 - Clientは verify(signature, digest, public_key) を実行。
 - 成功: パイプを開通 (pull_data) する。
 - 失敗: 即時切断する。データ本体には一切触れない。
5. Transfer (Server-Side Check):
 - pull_data 要求時、Artifactは再ハッシュ計算を行い、改ざんがないか最終確認してデータを渡す。

4. 内部構造要件 (The Artifact Anatomy)

各Artifactは、DI (依存性注入) によって以下の3層構造を持たねばならない。



4.1 The Core (Generator)

- 役割: 値値を生み出すエンジン (LLM等)。
- 制約: 署名鍵 (Private Key) へのアクセス権を持たない。

4.2 The Sensor (Auditor)

- 役割: 値値を監査する検問所。
- 権限: 唯一、署名鍵 (Notary) を持つ存在。
- ロジック: 出力が Boundary の定義 (Meaning Scope) に合致するか判定し、合致する場合のみ署名を発行する。

4.3 The Loop (Healer)

- 役割: 乖離 (Drift) 検知時、Clientに返す前にCoreへ修正指示を出し、再生成を試みる自律ループ。

5. 実装設計 (Configuration Design)

本システムは実装 (コード) に依存せず、以下のYAML定義によってネットワーク構成が決定される。

```
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
```

Boundary ID は、Responsibility / Meaning / Context を合成した識別子である。

6. 数理的保証 (Mathematical Assurance)

本プロトコルは以下の条件下において、不正データ（Drift/Malware）の伝播確率を 0% に収束させる。

1. **暗号学的制約:** Coreは署名鍵を持たず、Sensorの監査を通らない限り、正当な署名（SIGNED）を生成できない。
2. **プロトコル制約:** Clientは verify() が True でない限り、データ取得API (pull_data) をコールしない（できない）。
3. **完全性制約:** データ転送直前にハッシュ（Digest）の再検証が行われるため、監査後～転送前のメモリ改ざん（Race Condition）を防ぐ。

VCDesign (設計思想・アーキテクチャ)

<https://vcdesign.org/>

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

This document is **normative**.

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