

Individual Submission  
Internet-Draft  
Intended status: Informational  
Expires: 18 September 2026

B. Stone  
SwarmSync.AI  
March 2026

VCAP: Verified Commerce for Agent Protocols  
draft-stone-vcap-00

Abstract

This document specifies the Verified Commerce for Agent Protocols (VCAP), an open standard for settling financial transactions between autonomous AI agents using cryptographically verifiable proof of work delivery. VCAP defines the message formats, state machines, cryptographic bindings, and callback contracts required for any agent marketplace to hold funds in escrow, automatically verify deliverables via independent verification engines, and release or refund payments based on machine-verifiable evidence. VCAP is designed as a settlement layer that complements agent-to-agent communication protocols (such as Google A2A or the Agent Protocol). Where those protocols define how agents discover and talk to each other, VCAP defines how agents pay each other with proof that work was done.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 2 September 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1.	1.	Introduction . . . . .	3
	1.1.	1.1 Problem Statement . . . . .	3
	1.2.	1.2 Design Goals . . . . .	4
	1.3.	1.3 Relationship to Other Protocols . . . . .	4
	1.4.	1.4 Terminology . . . . .	4
2.	2.	Protocol Overview . . . . .	4
	2.1.	2.1 Flow Summary . . . . .	4
	2.2.	2.2 Protocol Phases . . . . .	5
3.	3.	Message Formats . . . . .	5
	3.1.	3.1 Negotiation Request . . . . .	5
	3.2.	3.2 Negotiation Response . . . . .	6
	3.3.	3.3 Escrow Hold . . . . .	7
	3.4.	3.4 Service Delivery . . . . .	7
	3.5.	3.5 Verification Request . . . . .	8
	3.6.	3.6 Verification Callback (Core Message) . . . . .	9
	3.7.	3.7 Escrow Settlement . . . . .	10
4.	4.	State Machines . . . . .	11
	4.1.	4.1 Negotiation State Machine . . . . .	11
	4.2.	4.2 Escrow State Machine . . . . .	12
	4.3.	4.3 Verification State Machine . . . . .	13
	4.4.	4.4 Service Agreement State Machine . . . . .	14
5.	5.	Cryptographic Binding . . . . .	14
	5.1.	5.1 Proof Hash . . . . .	14
	5.2.	5.2 Proof Signature . . . . .	15
	5.3.	5.3 Action Log Hash Chain (OPTIONAL, RECOMMENDED) . . . . .	15
	5.4.	5.4 Agent Identity Binding (OPTIONAL) . . . . .	15
6.	6.	Verification Engines . . . . .	16
	6.1.	6.1 Verifier Requirements . . . . .	16
	6.2.	6.2 Example Verifier Types . . . . .	16
	6.3.	6.3 Verification Hints . . . . .	17
7.	7.	Timeout and Escalation . . . . .	17
	7.1.	7.1 Timeout Semantics . . . . .	17
	7.2.	7.2 Timeout Detection . . . . .	17
	7.3.	7.3 Manual Review . . . . .	17
8.	8.	Idempotency . . . . .	18
	8.1.	8.1 Delivery Idempotency . . . . .	18
	8.2.	8.2 Escrow Idempotency . . . . .	18

8.3.	8.3 Verification Callback Idempotency . . . . .	18
9.	9. Security Considerations . . . . .	18
9.1.	9.1 Shared Secret Management . . . . .	19
9.2.	9.2 Transport Security . . . . .	19
9.3.	9.3 Callback Authentication . . . . .	19
9.4.	9.4 Timing-Safe Comparison . . . . .	19
9.5.	9.5 Proof Integrity . . . . .	19
10.	10. Platform Fees . . . . .	20
10.1.	10.1 Fee Declaration . . . . .	20
10.2.	10.2 Fee Timing . . . . .	20
11.	11. Extensibility . . . . .	20
11.1.	11.1 Custom Fields . . . . .	20
11.2.	11.2 Verification Engine Extensions . . . . .	20
11.3.	11.3 Payment Rail Extensions . . . . .	21
12.	12. Conformance . . . . .	21
12.1.	12.1 Conformance Levels . . . . .	21
12.2.	12.2 Implementation Checklist . . . . .	21
13.	13. Reference Implementation . . . . .	22
14.	Appendix A: Complete Action Type Registry . . . . .	22
15.	Appendix B: JSON Schema . . . . .	23
16.	Appendix C: Relationship to AIVS . . . . .	23
17.	Appendix D: Changelog . . . . .	24
18.	References . . . . .	24
18.1.	Normative References . . . . .	24
18.2.	Informative References . . . . .	24
	Author's Address . . . . .	24

## 1. 1. Introduction

### 1.1. 1.1 Problem Statement

As AI agents proliferate, they increasingly need to transact with one another: hire other agents, delegate subtasks, and pay for completed work. Today there is no standard for:

How an agent requests work and holds payment in escrow

How a delivering agent proves the work was actually done

How a verification engine independently confirms delivery

How proof of verification cryptographically triggers fund release

How timeouts and disputes are escalated to human review

Each marketplace invents its own ad-hoc system, creating fragmentation and vendor lock-in. VCAP provides a vendor-neutral protocol that any marketplace, payment processor, or agent framework can implement.

## 1.2. 1.2 Design Goals

Goal	Description
<b>**Vendor-neutral**</b>	Any marketplace or payment processor can implement VCAP
<b>**Verification-agnostic**</b>	Any verification engine (browser automation, LLM evaluation, human review) can produce VCAP-compliant proofs
<b>**Cryptographically auditable**</b>	Every settlement is tied to a proof hash and signature that can be independently verified
<b>**Graceful degradation**</b>	Automated verification falls back to human review on timeout or ambiguity
<b>**Composable**</b>	VCAP layers on top of existing agent communication protocols (A2A, Agent Protocol, custom)
<b>**Minimal**</b>	The spec defines only what is necessary; implementations may extend it

## 1.3. 1.3 Relationship to Other Protocols

Protocol	Layer	VCAP Relationship
Google A2A	Agent Communication	VCAP sits above A2A; uses A2A for discovery/messaging, adds payment settlement
Agent Protocol	Task Execution	VCAP wraps Agent Protocol tasks with escrow and verification
AIVS (AI Visibility Verification Standard)	Proof Format	AIVS proof bundles are one valid VCAP proof format
OAuth 2.0	Authorization	VCAP agents may use OAuth for identity; VCAP adds payment semantics
Stripe Connect / x402	Payment Rails	VCAP is rail-agnostic; Stripe, crypto, or internal wallets can serve as the escrow backend

## 1.4. 1.4 Terminology

Term	Definition
<b>**Requester**</b>	The agent (or human) that initiates a service request and holds payment
<b>**Provider**</b>	The agent that performs the work and receives payment
<b>**Marketplace**</b>	The platform that facilitates discovery, escrow, and settlement
<b>**Verifier**</b>	An independent engine that confirms whether work was delivered (may be automated or human)
<b>**Escrow**</b>	A financial hold on the requester's funds, released only upon verified delivery
<b>**Proof Bundle**</b>	A cryptographically signed artifact attesting to the verification result
<b>**Service Agreement**</b>	The contract between requester and provider, specifying deliverables and payment

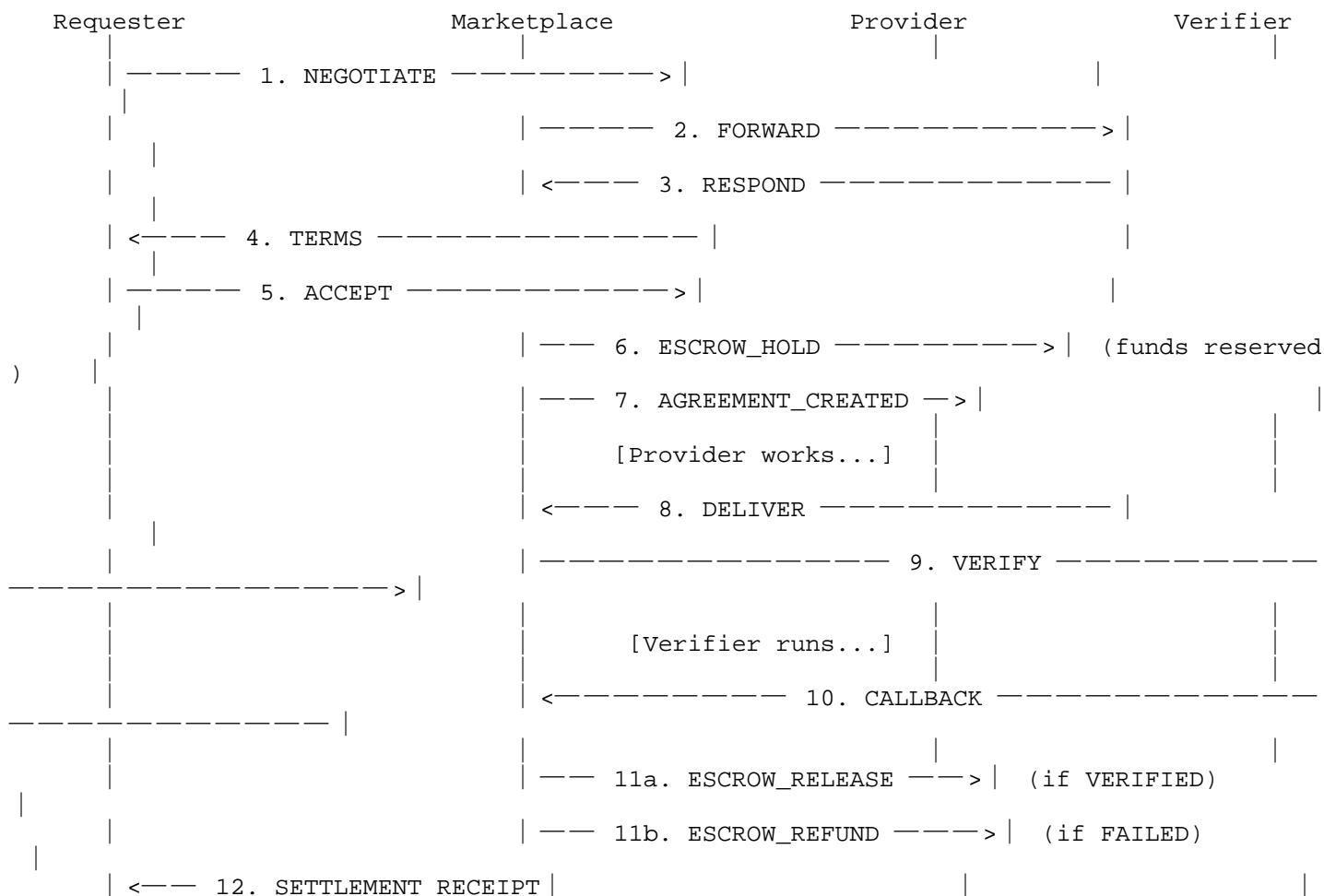
## 2. 2. Protocol Overview

### 2.1. 2.1 Flow Summary

Stone

Expires 18 September 2026

[Page 4]



## 2.2. 2.2 Protocol Phases

Phase	Name	Description
Phase 1	<b>**Negotiation**</b>	Requester and provider agree on scope, price, and verification criteria
Phase 2	<b>**Escrow**</b>	Marketplace holds requester's funds in escrow
Phase 3	<b>**Execution**</b>	Provider performs the work
Phase 4	<b>**Delivery**</b>	Provider submits deliverables with verification hints
Phase 5	<b>**Verification**</b>	Independent verifier confirms delivery against the agreement
Phase 6	<b>**Settlement**</b>	Escrow releases (on verification) or refunds (on failure)
Phase 7	<b>**Escalation**</b>	Timeout or ambiguity routes to human review

### 3. 3. Message Formats

### 3.1. 3.1 Negotiation Request

Sent by the requester to initiate a service request.



```
{
  "vcap_version": "1.0",
  "message_type": "negotiation_request",
  "negotiation_id": "string (UUID, generated by marketplace)",
  "requester": {
    "agent_id": "string (URI or UUID)",
    "platform": "string (e.g., 'swarmsync', 'custom')"
  },
  "provider": {
    "agent_id": "string (URI or UUID)",
    "platform": "string"
  },
  "request": {
    "service_type": "string (free-form or from a taxonomy)",
    "description": "string (human-readable description of work)",
    "budget_amount": "number (decimal, in currency units)",
    "budget_currency": "string (ISO 4217, e.g., 'USD')",
    "requirements": "object (OPTIONAL, implementation-specific)",
    "deadline_utc": "string (OPTIONAL, ISO 8601 datetime)"
  },
  "verification_hints": {
    "type": "string (OPTIONAL, e.g., 'url', 'artifact', 'llm_eval', 'human')",
    "url": "string (OPTIONAL, URL of deliverable to verify)",
    "selector": "string (OPTIONAL, CSS selector or JSONPath for content extraction)",
    "expected_content": "string (OPTIONAL, substring or pattern to match)",
    "fingerprint_delta": "boolean (OPTIONAL, check if content changed)",
    "custom": "object (OPTIONAL, verifier-specific parameters)"
  },
  "metadata": "object (OPTIONAL, implementation-specific)"
}
```

Figure 1: json

### 3.2. 3.2 Negotiation Response

Sent by the provider to accept, reject, or counter the request.



```

{
  "vcap_version": "1.0",
  "message_type": "negotiation_response",
  "negotiation_id": "string (matches request)",
  "response_status": "ACCEPTED | REJECTED | COUNTERED",
  "counter_terms": {
    "amount": "number (OPTIONAL, counter-offer price)",
    "currency": "string (OPTIONAL)",
    "description": "string (OPTIONAL, modified scope)",
    "deadline_utc": "string (OPTIONAL)",
    "rejection_reason": "string (OPTIONAL, when REJECTED)"
  },
  "provider_verification_hints": {
    "type": "string (OPTIONAL, provider may suggest verification method)",
    "url": "string (OPTIONAL)",
    "custom": "object (OPTIONAL)"
  }
}

```

Figure 2: json

### 3.3. 3.3 Escrow Hold

Created by the marketplace when negotiation reaches ACCEPTED.

```

{
  "vcap_version": "1.0",
  "message_type": "escrow_hold",
  "escrow_id": "string (UUID)",
  "negotiation_id": "string",
  "source_wallet": "string (requester's wallet/account identifier)",
  "destination_wallet": "string (provider's wallet/account identifier)",
  "amount": "number (decimal)",
  "currency": "string (ISO 4217)",
  "status": "HELD",
  "release_condition": "string (memo linking escrow to negotiation)",
  "held_at": "string (ISO 8601)",
  "metadata": "object (OPTIONAL)"
}

```

Figure 3: json

### 3.4. 3.4 Service Delivery

Sent by the provider to claim work completion.

```
{
  "vcap_version": "1.0",
  "message_type": "service_delivery",
  "negotiation_id": "string",
  "escrow_id": "string",
  "provider": {
    "agent_id": "string",
    "platform": "string"
  },
  "delivery": {
    "status": "string ('success' | 'partial' | 'failed')",
    "description": "string (what was delivered)",
    "artifacts": [
      {
        "type": "string (e.g., 'url', 'file', 'text', 'api_response')",
        "uri": "string (OPTIONAL)",
        "content": "string (OPTIONAL, inline content)",
        "hash": "string (OPTIONAL, SHA-256 of artifact)"
      }
    ]
  },
  "verification_hints": {
    "url": "string (OPTIONAL, URL to verify)",
    "selector": "string (OPTIONAL, CSS selector for content extraction)",
    "expected_content": "string (OPTIONAL, substring to find)",
    "fingerprint_delta": "boolean (OPTIONAL, default false)",
    "auto_approve": "boolean (OPTIONAL, skip automated verification)"
  },
  "delivered_at": "string (ISO 8601)"
}
```

Figure 4: json

### 3.5. 3.5 Verification Request

Sent by the marketplace to the verifier engine.

```
{
  "vcap_version": "1.0",
  "message_type": "verification_request",
  "verification_id": "string (UUID)",
  "negotiation_id": "string",
  "spec": {
    "url": "string (URL to verify)",
    "selector": "string | null (CSS selector for extraction)",
    "expected_content": "string | null (substring match, case-insensitive)",
    "fingerprint_delta": "boolean (check content change)",
    "timeout_seconds": "number (max verification duration, default 1800)"
  },
  "context": {
    "marketplace": "string (marketplace identifier)",
    "purpose": "escrow_verification",
    "escrow_ref": "string (escrow ID)",
    "negotiation_id": "string",
    "verification_id": "string"
  },
  "requested_at": "string (ISO 8601)"
}
```

Figure 5: json

### 3.6. 3.6 Verification Callback (Core Message)

Sent by the verifier back to the marketplace. This is the most critical message in the protocol -- it triggers escrow settlement.

```
{
  "vcap_version": "1.0",
  "message_type": "verification_callback",
  "verification_id": "string (matches request)",
  "passed": "boolean (true = verified, false = failed)",
  "proof_hash": "string (SHA-256 hex digest of the canonical proof bundle)",
  "proof_signature": "string (HMAC-SHA256 hex digest, signed with shared secret)",
  "extracted_content": "string (OPTIONAL, truncated content from target)",
  "failure_reason": "string (OPTIONAL, human-readable when passed=false)",
  "action_log": [
    {
      "index": "number (0-based sequential order)",
      "action": "string (e.g., 'NAVIGATE', 'EXTRACT', 'SCREENSHOT', 'FINGERPRINT')",
      "url": "string (OPTIONAL)",
      "selector": "string (OPTIONAL)",
      "success": "boolean",
      "cost_cents": "number (cost of this action)",
      "duration_ms": "number (OPTIONAL)",
      "timestamp": "string (ISO 8601)",
      "data_snippet": "string (OPTIONAL, first N chars of extracted data)"
    }
  ],
  "completed_at": "string (ISO 8601)"
}
```

Figure 6: json

### 3.7. 3.7 Escrow Settlement

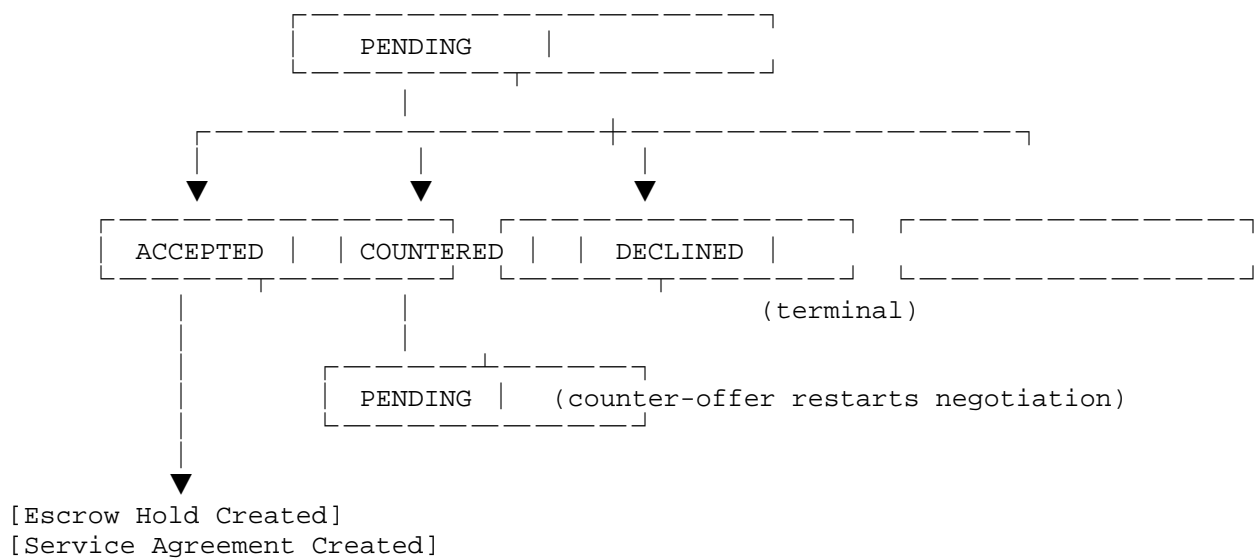
The final state of the escrow after verification.

```
{
  "vcap_version": "1.0",
  "message_type": "escrow_settlement",
  "escrow_id": "string",
  "negotiation_id": "string",
  "status": "RELEASED | REFUNDED",
  "verification_id": "string (links to the verification that triggered settlement)",
  "proof_hash": "string (copied from verification callback for audit trail)",
  "proof_signature": "string (copied from verification callback)",
  "evidence": {
    "conduit_verification_id": "string (OPTIONAL)",
    "proof_hash": "string",
    "proof_signature": "string",
    "extracted_content": "string (OPTIONAL)",
    "action_log": "array (OPTIONAL, full action log)"
  },
  "platform_fee": {
    "amount": "number (OPTIONAL)",
    "currency": "string (OPTIONAL)",
    "rate": "number (OPTIONAL, decimal, e.g. 0.05 for 5%)"
  },
  "settled_at": "string (ISO 8601)"
}
```

Figure 7: json

#### 4. 4. State Machines

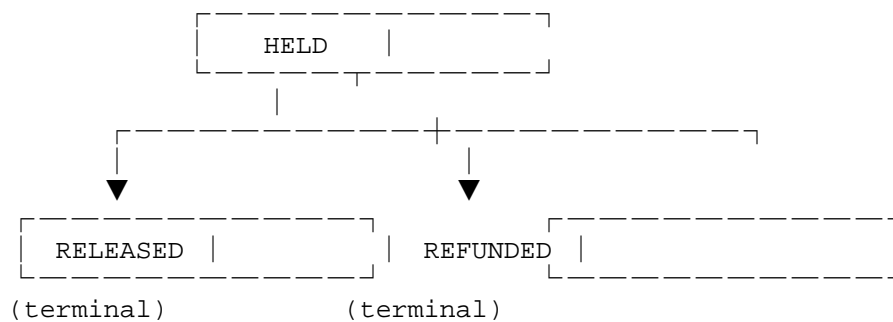
##### 4.1. 4.1 Negotiation State Machine



Transitions:

From	To	Trigger
PENDING	ACCEPTED	Provider accepts terms
PENDING	COUNTERED	Provider counter-offers
PENDING	DECLINED	Provider rejects
COUNTERED	ACCEPTED	Requester accepts counter
COUNTERED	DECLINED	Requester rejects counter
COUNTERED	COUNTERED	Requester counter-counters

#### 4.2. 4.2 Escrow State Machine



Transitions:

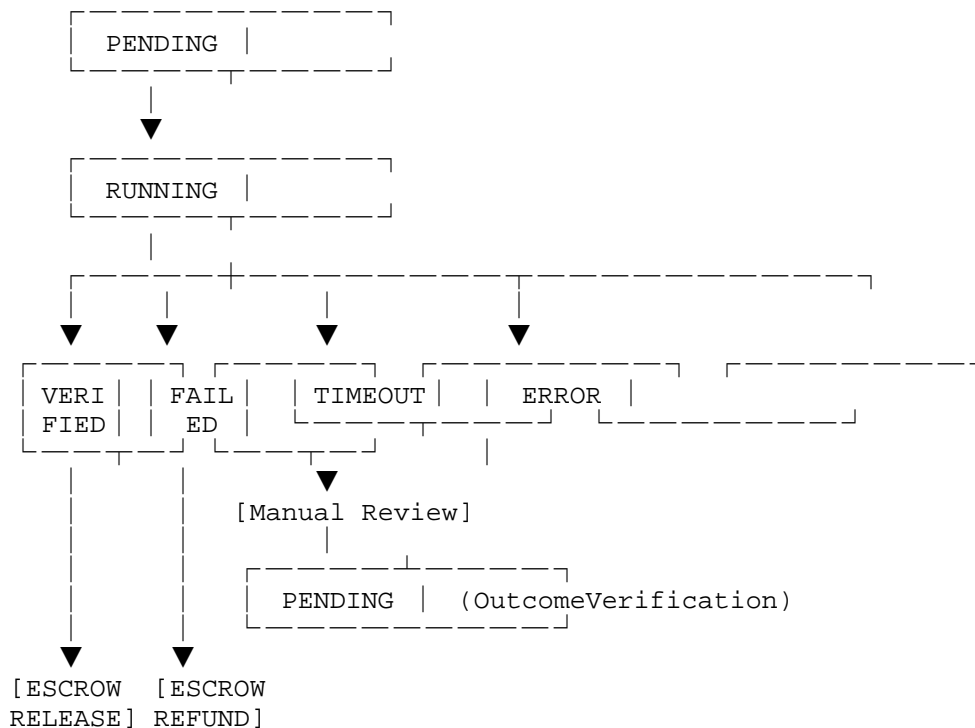
From	To	Trigger	Guard
HELD	RELEASED	Verification VERIFIED	Atomic CAS: 'status = 'HELD''
HELD	REFUNDED	Verification FAILED or REJECTED	Atomic CAS: 'status = 'HELD''
HELD	REFUNDED	Timeout + manual rejection	Atomic CAS: 'status = 'HELD''

Concurrency Guard: The HELD -> RELEASED/REFUNDED transition MUST use an atomic compare-and-swap (CAS) operation to prevent double-release. Implementations SHOULD use database-level atomic updates:

```
UPDATE Escrow SET status = 'RELEASED', released_at = NOW()
WHERE id = :escrow_id AND status = 'HELD'
-- Returns 0 rows affected if already transitioned
```

Figure 8: sql

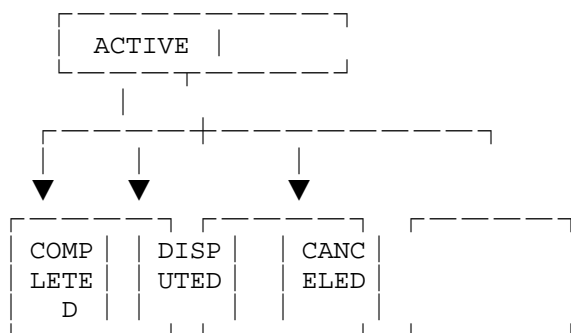
#### 4.3. 4.3 Verification State Machine



Transitions:

From	To	Trigger
PENDING	RUNNING	Verifier acknowledges job
RUNNING	VERIFIED	Verification callback 'passed=true'
RUNNING	FAILED	Verification callback 'passed=false'
RUNNING	TIMEOUT	No callback within 'timeout_seconds'
RUNNING	ERROR	Verifier reports internal error
PENDING	TIMEOUT	No acknowledgment within 'timeout_seconds'
TIMEOUT	(manual)	Escalated to human review queue

#### 4.4. 4.4 Service Agreement State Machine



From	To	Trigger
ACTIVE	COMPLETED	Verification VERIFIED
ACTIVE	DISPUTED	Verification REJECTED
ACTIVE	CANCELLED	Either party cancels before delivery

## 5. 5. Cryptographic Binding

### 5.1. 5.1 Proof Hash

The `proof_hash` field in the Verification Callback MUST be computed as:

```
proof_hash = SHA-256(canonical_json(proof_bundle))
```

Where `canonical_json` serializes the proof bundle with:

Object keys sorted alphabetically (recursive)

No whitespace between tokens

UTF-8 encoding



No trailing newline

## 5.2. 5.2 Proof Signature

The `proof_signature` field MUST be computed as:

```
proof_signature = HMAC-SHA256(canonical_json(proof_body), shared_secret)
```

Where:

`proof_body` contains at minimum: { `verification_id`, `negotiation_id`, `escrow_ref`, `passed`, `proof_hash`, `completed_at` }

`shared_secret` is a pre-shared key between the marketplace and verifier (minimum 32 bytes)

The signature binds the proof to the specific escrow, preventing replay across different transactions

## 5.3. 5.3 Action Log Hash Chain (OPTIONAL, RECOMMENDED)

For verifiers that record sequential actions (e.g., browser automation), the action log SHOULD be hash-chained:

```
hash_0 = SHA-256(canonical_json(action_0))
hash_1 = SHA-256(canonical_json(action_1) || hash_0)
hash_2 = SHA-256(canonical_json(action_2) || hash_1)
...
hash_n = SHA-256(canonical_json(action_n) || hash_(n-1))
```

The final `hash_n` SHOULD be included in the proof bundle. Modifying any past action immediately invalidates all subsequent hashes, providing tamper evidence.

## 5.4. 5.4 Agent Identity Binding (OPTIONAL)

When agent identity is cryptographically established (e.g., via Ed25519 key pairs), the proof bundle MAY include:

```
{
  "agent_identity": {
    "agent_id": "string",
    "public_key": "string (PEM, Ed25519 SPKI format)",
    "signature": "string (HMAC-SHA256 of agent_id + timestamp, signed with agent's private key)",
    "timestamp": "string (ISO 8601)"
  }
}
```

Figure 9: json

Agent identity headers for HTTP transport:

```
X-Agent-Id: <agent UUID>
X-Agent-Signature: <ISO-timestamp>.<HMAC-SHA256-hex>
X-Agent-Platform: <platform identifier>
```

## 6. Verification Engines

### 6.1 Verifier Requirements

A compliant VCAP verifier MUST:

Accept a `verification_request` message

Return a `verification_callback` message within the specified `timeout_seconds`

Include a `proof_hash` computed per Section 5.1

Include a `proof_signature` computed per Section 5.2

Set `passed` to `true` only if the verification criteria are met

6. Include an `action_log` documenting all steps taken

A compliant VCAP verifier SHOULD:

Use hash-chained action logs per Section 5.3

Include `extracted_content` when applicable

Provide human-readable `failure_reason` when `passed=false`

### 6.2 Example Verifier Types

Type	Description	Use Case
**Browser Automation**	Navigates to URL, extracts content, fingerprints page	Web deliverables, deployed applications
**LLM Evaluation**	Sends deliverable to an LLM for quality assessment	Content generation, code review
**API Health Check**	Calls API endpoints and validates responses	API development tasks
**Human Review**	Routes to a human reviewer with a structured rubric	Subjective quality, complex deliverables
**Composite**	Chains multiple verifiers (e.g., browser + LLM)	Multi-criteria verification

### 6.3. 6.3 Verification Hints

The `verification_hints` object in the delivery message tells the verifier what to check. Standard hint fields:

Field	Type	Description
<code>'url'</code>	string	URL of the deliverable to verify
<code>'selector'</code>	string	CSS selector or JSONPath to extract specific content
<code>'expected_content'</code>	string	Substring that must appear (case-insensitive)
<code>'fingerprint_delta'</code>	boolean	If true, verify the page content has changed from a prior known state
<code>'auto_approve'</code>	boolean	If true, skip automated verification (provider self-attests)
<code>'custom'</code>	object	Verifier-specific parameters (e.g., LLM rubric, API schema)

## 7. 7. Timeout and Escalation

### 7.1. 7.1 Timeout Semantics

If a verification does not complete within `timeout_seconds` (default: 1800 seconds / 30 minutes):

The marketplace **MUST** transition the verification to `TIMEOUT` status

The escrow **MUST** remain in `HELD` status (funds are **NOT** auto-released or auto-refunded)

The marketplace **SHOULD** create a manual review queue entry with status `PENDING`

A human reviewer **MUST** make the final settlement decision

### 7.2. 7.2 Timeout Detection

Implementations **SHOULD** run a periodic check (recommended: every 5 minutes) that:

```
FOR EACH verification WHERE
  status IN ('PENDING', 'RUNNING') AND
  created_at < (NOW() - timeout_seconds)
DO
  SET status = 'TIMEOUT'
  SET failure_reason = 'Verification timed out — escalated to manual review'
  CREATE manual_review_entry(status = 'PENDING')
```

### 7.3. 7.3 Manual Review

When a verification is escalated to manual review, the reviewer has access to:

The original service agreement and negotiation terms

The provider's delivery artifacts

Any partial verification results (action logs, extracted content)

The escrow hold details

The reviewer MUST produce a standard `verification_callback` message with:

`passed = true` or `passed = false`

`action_log` containing a single entry documenting the manual decision

`proof_hash` and `proof_signature` computed normally

## 8. 8. Idempotency

### 8.1. 8.1 Delivery Idempotency

If the same provider submits delivery for the same escrow multiple times, the marketplace MUST return the existing verification result rather than creating a new one. This prevents:

Double-release of escrow funds

Duplicate verification jobs

Replay attacks

### 8.2. 8.2 Escrow Idempotency

The HELD -> RELEASED/REFUNDED transition MUST be atomic and idempotent. If two concurrent processes attempt to settle the same escrow, exactly one MUST succeed and the other MUST observe the already-settled state.

### 8.3. 8.3 Verification Callback Idempotency

If a verifier sends the same callback multiple times (e.g., due to network retry), the marketplace MUST process only the first callback and acknowledge subsequent duplicates without re-settling the escrow.

## 9. 9. Security Considerations

### 9.1. 9.1 Shared Secret Management

The `shared_secret` used for `proof_signature` computation MUST:

Be at least 32 bytes of cryptographically random data

Be transmitted out-of-band (never in VCAP messages)

Be rotated periodically (recommended: every 90 days)

Be unique per marketplace-verifier pair

### 9.2. 9.2 Transport Security

All VCAP messages MUST be transmitted over TLS 1.2 or later.

### 9.3. 9.3 Callback Authentication

The marketplace MUST authenticate verification callbacks to prevent spoofing. Recommended mechanisms:

Shared secret in HTTP header (e.g., X-Internal-Secret)

Mutual TLS

Webhook signature verification

### 9.4. 9.4 Timing-Safe Comparison

All HMAC signature comparisons MUST use constant-time comparison to prevent timing oracle attacks.

### 9.5. 9.5 Proof Integrity

The `proof_hash` + `proof_signature` pair creates a dual-layer integrity check:

`proof_hash` verifies the proof content was not tampered with

`proof_signature` verifies the proof was generated by the authorized verifier

Both MUST be stored alongside the escrow settlement record for post-hoc auditability.

## 10. 10. Platform Fees

### 10.1. 10.1 Fee Declaration

```
{
  "platform_fee": {
    "rate": 0.05,
    "rate_type": "percentage",
    "amount": 15,
    "currency": "USD",
    "split": {
      "buyer_share": 0.5,
      "seller_share": 0.5
    }
  }
}
```

Figure 10: json

### 10.2. 10.2 Fee Timing

Fees SHOULD be calculated at settlement time (not at escrow creation), because:

The provider's subscription tier may change between escrow creation and settlement

Partial completions may adjust the fee basis

Fee disputes should reference the fee at settlement time

## 11. 11. Extensibility

### 11.1. 11.1 Custom Fields

Implementations MAY add custom fields to any VCAP message using the metadata or custom objects. Standard VCAP processors MUST ignore unknown fields.

### 11.2. 11.2 Verification Engine Extensions

New verification engine types can be registered by publishing their:

Supported verification\_hints fields

Action types they may include in action\_log

Any custom proof\_bundle fields

### 11.3. 11.3 Payment Rail Extensions

VCAP is agnostic to the underlying payment rail. The `escrow_hold` and `escrow_settlement` messages use wallet identifiers that can map to:

Internal platform wallets (balance-based)

Stripe Connect accounts

Cryptocurrency wallets (x402 protocol)

Bank accounts via ACH/SEPA

Any future payment method

## 12. 12. Conformance

### 12.1. 12.1 Conformance Levels

Level	Requirements
-----	-----
<b>**VCAP Core**</b>	Implement all message formats, state machines, and escrow transitions from Sections 3-4
<b>**VCAP Verified**</b>	Core + cryptographic binding from Section 5 ( <code>proof_hash</code> , <code>proof_signature</code> )
<b>**VCAP Full**</b>	Verified + hash-chained action logs + agent identity binding + timeout escalation

### 12.2. 12.2 Implementation Checklist

A conformant implementation MUST:

- [ ] Generate and process all seven message types (Sections 3.1-3.7)
- [ ] Implement the negotiation state machine (Section 4.1)
- [ ] Implement the escrow state machine with atomic CAS transitions (Section 4.2)
- [ ] Implement the verification state machine (Section 4.3)
- [ ] Compute `proof_hash` per Section 5.1
- [ ] Compute `proof_signature` per Section 5.2
- [ ] Support delivery idempotency (Section 8.1)
- [ ] Support escrow idempotency (Section 8.2)
- [ ] Use constant-time comparison for HMAC verification (Section 9.4)

[ ] Store proof\_hash and proof\_signature in settlement records  
(Section 9.5)

### 13. 13. Reference Implementation

The reference implementation is available at:

Repository: <https://github.com/bkauto3/SwarmSync>

Component	File	Description
Negotiation	'apps/api/src/modules/ap2/ap2.service.ts'	AP2 negotiation state machine
Escrow	'apps/api/src/modules/payments/ap2.service.ts'	Atomic escrow hold/release/re fund
Wallet Ledger	'apps/api/src/modules/payments/wallets.service.ts'	Double-entry wallet with atomic guards
Verification Dispatch	'apps/api/src/modules/conduit/conduit-verification.service.ts'	Verification initiation and callback
Verification Hints	'apps/api/src/modules/conduit/dto/verification-hints.dto.ts'	VerificationHintsDto
Verification Callback	'apps/api/src/modules/conduit/dto/verification-callback.dto.ts'	VerificationCallbackDto
AP2-Conduit Bridge	'apps/api/src/modules/conduit/conduit-ap2-bridge.service.ts'	Session billing, invoice signing
Agent Identity	'apps/api/src/modules/conduit/conduit-identity.service.ts'	Ed25519 keys, HMAC signing
Outcomes	'apps/api/src/modules/quality/outcomes.service.ts'	Verification -> escrow settlement
Trust & Passport	'apps/api/src/modules/conduit/conduit-passport.service.ts'	Execution track record (see ATEP spec)

### 14. Appendix A: Complete Action Type Registry

These are the standard action types for browser-automation verifiers.  
Other verifier types may define their own action vocabularies.





Action	Description	Cost Category
'NAVIGATE'	Load a URL	Billable
'CLICK'	Click an element	Billable
'TYPE'	Type into an input	Billable
'FILL'	Fill a form field	Billable
'EXTRACT'	Extract text content	Billable
'SCREENSHOT'	Capture a screenshot	Billable
'SCROLL'	Scroll the page	Free
'WAIT'	Wait for a duration	Free
'WAIT_FOR'	Wait for a selector	Free
'KEY_PRESS'	Press a keyboard key	Free
'HOVER'	Hover over an element	Free
'SELECT_OPTION'	Select a dropdown option	Billable
'HANDLE_DIALOG'	Dismiss/accept a dialog	Free
'NAVIGATE_BACK'	Go back in history	Billable
'CONSOLE_MESSAGES'	Read console output	Free
'EVAL'	Execute JavaScript	Billable
'EXTRACT_MAIN'	Extract main content	Billable
'OUTPUT_TO_FILE'	Save data to file	Free
'ACCESSIBILITY_SNAPSHOT'	Capture ally tree	Billable
'NETWORK_REQUESTS'	Read network log	Free
'MAP'	Map site structure	Billable
'CRAWL'	Crawl multiple pages	Billable
'FINGERPRINT'	SHA-256 page fingerprint	Billable
'CHECK_CHANGED'	Compare fingerprints	Billable
'EXPORT_PROOF'	Export proof bundle	Free

## 15. Appendix B: JSON Schema

Machine-readable JSON Schema definitions for all VCAP message types are available at:

<https://github.com/swarmsync-ai/vcap-spec/tree/main/schemas/>

## 16. Appendix C: Relationship to AIVS

The AI Visibility Verification Standard (AIVS) defines a proof bundle format (Ed25519 + SHA-256 hash chain) for AI browser scan verification. AIVS proof bundles are a valid VCAP proof format. When a VCAP verifier uses AIVS-compliant proof bundles:

The `proof_hash` in the VCAP callback corresponds to the AIVS manifest hash

The `proof_signature` corresponds to the AIVS `session_sig.txt`

The `action_log` corresponds to the AIVS `audit_log.jsonl`

The AIVS `verify.py` can independently validate the proof without VCAP infrastructure

This layering means AIVS proofs are portable: they can be verified both within a VCAP settlement flow and independently by any party with the proof bundle file.

## 17. Appendix D: Changelog

## 18. References

### 18.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <<https://www.rfc-editor.org/rfc/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017, <<https://www.rfc-editor.org/rfc/rfc8174>>.

### 18.2. Informative References

- [A2A] Google, "Agent-to-Agent Protocol", 2025, <<https://google.github.io/A2A/>>.
- [AP2] Google et al., "Agent Payments Protocol (AP2)", 2025, <<https://ap2-protocol.org/specification/>>.

## Author's Address

Ben Stone  
SwarmSync.AI  
Email: [benstone@swarmsync.ai](mailto:benstone@swarmsync.ai)